WebMedia2018

XXIV SIMPÓSIO BRASILEIRO DE SISTEMAS MULTIMÍDIA E WEB

Realização

SBC Sociedade Brasileira de Computação

16 a 19 Outubro

Salvador - BA - Brasil

Minicursos

Organizadores

Valter Roesler

Artur Kronbauer

Manoel Neto

Renato Novais

Roberto Willrich

Organização

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA Bahia

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE SISTEMAS E PRODUTOS

Cooperação

acm In-Cooperation

sigweb

acm SIGMM

Patrocínio

nic.br Núcleo de Informação e Coordenação do Ponto BR

cgi.br Comitê Gestor da Internet no Brasil

CAPES

CNPq Conselho Nacional de Desenvolvimento Científico e Tecnológico

Editora

Sociedade Brasileira de Computação (SBC)

XXIV Simpósio Brasileiro de Sistemas Multimídia e Web

De 16 a 19 de Outubro de 2018

Salvador, Bahia, Brasil

ANAIS

MINICURSOS

Organizadores

Valter Roesler (UFRGS)
Artur Kronbauer (UNIFACS)
Manoel C. M. Neto (IFBA)
Renato Novais (IFBA)
Roberto Willrich (UFSC)

Realização

Sociedade Brasileira de Computação – SBC

Em cooperação com

ACM/SIGWEB e ACM/SIGMM

Organização

Instituto Federal da Bahia – IFBA

Biblioteca Raul V. Seixas – Instituto Federal de Educação, Ciência e Tecnologia da Bahia - IFBA - Salvador/BA.
Responsável pela catalogação na fonte: Samuel dos Santos Araújo - CRB 5/1426.

S613a Simpósio Brasileiro de Sistemas Multimídia e Web. Minicursos. (24. : 2018 : Salvador, BA).
Anais [recurso eletrônico] / organização: IFBA. Salvador: SBC; IFBA, 2018.

246 p.
E-book.
ISBN: 978-85-7669-455-7.

Evento promovido pela Sociedade Brasileira de Computação – SBC, Porto Alegre, RS, em 16 a 19 de outubro, 2018, Salvador, Bahia. Organização: Instituto Federal de Educação, Ciência e Tecnologia da Bahia – IFBA.

1. Internet das Coisas. 2. Deep Learning. 3. Redes Sociais. 4. Computação em nuvem. 5. Web 3.0. I. SBC. II. IFBA. III. Título.
CDU 2 ed. 004.7

Prefácio

Tradicionalmente, o Simpósio Brasileiro de Sistemas Multimídia e Web oferece à sua comunidade minicursos de curta duração relacionados a temas que norteiam os últimos avanços na área de multimídia e web. Este também é o caso do XXIV Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia 2018), onde os minicursos têm a duração de 4 horas e permitem que participantes recebam informações sobre novas tecnologias e tópicos atuais de pesquisa em áreas correlatas ao evento. Assim, minicursos aparecem como uma excelente oportunidade para familiarização dos congressistas com novos temas de pesquisa que podem vir a ser úteis em suas vidas profissionais.

Para o Webmedia 2018, o processo de seleção de minicursos foi feito a partir de várias chamadas públicas divulgadas na lista eletrônica de emails da SBC, bem como ampla divulgação no site oficial do evento e em redes sociais. Foram recebidas 16 (dezesseis) propostas de minicursos, avaliadas por comitê composto de professores com conhecimento nos temas abordados. Cada minicurso foi avaliado por quatro avaliadores, que pontuaram notas para quesitos como relevância para o evento, expectativa do público, atualidade e conteúdo de cada minicurso. Ao final, seis propostas foram selecionadas, cujos conteúdos abordados constituem os capítulos deste livro.

O primeiro capítulo, intitulado *Evolução das arquiteturas de software rumo à Web 3.0*, aborda histórico e comparações dos tipos de modelos de arquitetura e plataformas atuais de desenvolvimento de software baseados em tecnologias no lado do cliente (React JS, Angular JS e Vue Js) e no servidor (Spring e Node.js). Também discute tendências atuais e futuras para sistemas que atendem requisitos de Web 3.0.

O Capítulo dois, *Extração e Classificação de Dados Semânticos do Twitter*, apresenta técnicas de extração de dados semânticos do Twitter, bem como ferramentas que implementam as técnicas apresentadas, visualizando como utilizar na prática esses recursos.

O terceiro capítulo é relacionado a Deep Learning, sendo intitulado *Desenvolvendo Modelos de Deep Learning para Aplicações Multimídia no Tensorflow*. Em especial, o minicurso prepara o participante para entender e desenvolver redes neurais profundas, redes neurais convolucionais (CNN), e redes neurais recorrentes (LSTM/GRU). Além disso, os modelos de Deep Learning são aplicados para resolver problemas do domínio da multimídia, como classificação de imagens, reconhecimento facial, detecção de objetos e classificação de cenas de vídeo.

Em seguida, o capítulo quatro explora o tema de Internet das Coisas (IoT), com o título *Desenvolvendo Sensores de Vídeo para a Internet das Coisas com o Raspberry Pi*. Nesse cenário, é apresentado como é possível facilmente criar dispositivos multimídia de monitoramento, desenvolvendo aplicações de baixo custo para a Internet das Coisas. O foco do minicurso é na construção de sensores com câmera, apresentando as principais configurações para diversos tipos de monitoramento. Adicionalmente, a transmissão de dados visuais é abordada, bem como detalhes iniciais para a construção de redes de sensores visuais sem fio no contexto de IoT.

O capítulo cinco é intitulado *Dados de Múltiplas Fontes da Web: coleta, integração e pré-processamento*, apresentando soluções para o desafio de lidar com dados extraídos da web. O problema é que tais dados são heterogêneos e não estruturados. Além disso, existem diferentes fontes de dados na Web, como sites, aplicativos, mídias, redes sociais e até mesmo bases de dados já construídas e disponibilizadas. Combinando tais dados pode-se obter conhecimentos novos, integrados e úteis, podendo ser aplicados na solução de problemas em diferentes campos, como sistemas inteligentes, marketing, sistemas de recomendação, entre muitos outros.

O sexto capítulo também discorre sobre Internet das Coisas (IoT), sendo intitulado *Do device à cloud com a plataforma SOFT-IoT: sua infraestrutura IoT em poucas horas*. O objetivo do minicurso é oferecer uma visão prática da IoT através de um cenário real no desenvolvimento de ambientes inteligentes. Será apresentado a plataforma SOFT-IoT, que utiliza o conceito de Névoa das Coisas, para explorar a capacidade de processamento, armazenamento e acesso nos dispositivos locais e na nuvem.

Esperamos que este livro seja útil para todos aqueles interessados e praticantes da área de Sistemas Multimídia e Web.

Salvador, outubro de 2018.

Valter Roesler (UFRGS)
Artur Kronbauer (UNIFACS)
Coordenadores dos Minicursos do WebMedia 2018

XIV Simpósio Brasileiro de Sistemas Multimídia e Web

16 a 19 de outubro de 2018
Salvador, Bahia, Brasil

Coordenação Geral

Manoel C. M. Neto (IFBA) – *Coordenador Geral*
Renato L. Novais (IFBA) – *Vice-Coordenador Geral*
Carlos Ferraz (UFPE) – *Coordenador do Comitê de Programa*
Windson Viana (UFC) – *Vice-Coordenador do Comitê de Programa*

Coordenador de Publicação

Roberto Willrich (UFSC)

Coordenadores do XV Workshop de Trabalhos de Iniciação Científica

Tiago Maritan (UFPB)
Antônio Maurício Pitangueira (IFBA)
Antônio Carlos Souza (IFBA)

Coordenadores do XVII Workshop de Ferramentas e Aplicações

Carlos de Salles (UFMA)
Álan L.V. Guedes (PUC-Rio)

Coordenadores do XVIII Workshop de Teses e Dissertações

Eduardo Barrére (UFJF)
Frederico Durão (UFBA)

Coordenadores de Minicursos

Valter Roesler (UFRGS)
Artur Kronbauer (UNIFACS)

Coordenador do V Workshop O Futuro da Videocolaboração

Leandro N. Ciuffo (RNP)
Valter Roesler (UFRGS)
Alexandre Carissimi (UFRGS)

Coordenação da Comissão Especial de Sistemas Multimídia e Web

Adriano C. Machado Pereira (UFMG) – *Coordenador*
Windson Viana (UFC) – *Vice-Coordenador*

Comitê Gestor

Carlos de Salles Soares Neto (UFMA)
Celso Alberto Saibel Santos (UFES)
Fábio de Jesus Lima Gomes (IFPI)
Guido Lemos de Souza Filho (UFPB)
José Valdeni de Lima (UFRGS)
Manoel C.M. Neto (IFBA)
Maria da Graça Campos Pimentel (USP)
Roberto Willrich (UFSC)
Valter Roesler (UFRGS)

Sumário

Capítulo 1

Evolução das arquiteturas de software rumo à Web 3.0

Raoni Kulesza[1,2], Marcelo F. de Sousa[2,3], Matheus Lima[1,2], Claudiomar Araujo[1,2], Aguinaldo M. Filho[4]

[1]Centro de Informática da Universidade Federal da Paraíba (UFPB)

[2]Laboratório de Aplicações de Vídeo Digital (LAViD)

[3]Instituto de Educação Superior da Paraíba (IESP)

[4]Tribunal de Contas do Estado da Paraíba (TCE-PB)

{raoni, marcelo, matheus.lima, claudiomar.araujo}@lavid.ufpb.br,

amfilho@tce.pb.gov.br

Abstract

Web Systems were initially supported by a client-server architecture and three standards (URL, HTTP and HTML), and has evolved in the last two decades. Usability, scalability, maintenance, portability, robustness, security and integration with other systems are the main challenges of this software category. This tutorial presents the history and evolution of Web-based software architectures. We discuss current software architectural styles, patterns, and development platforms based on client-side (React JS) and server-side (Spring) technologies. In addition, we also discuss Web 3.0 requirements such as communication protocols, Microservices, MV browser-based frameworks, boilerplates client-side code, asynchronous programming, and integration with cloud computing infrastructures.*

Resumo

Os sistemas Web foram inicialmente suportados por uma arquitetura cliente-servidor e três padrões (URL, HTTP e HTML) e evoluíram consideravelmente nas últimas duas décadas. Usabilidade, escalabilidade, manutenção, portabilidade, robustez, segurança e integração com outros sistemas são os principais desafios desta categoria de software. Este capítulo apresenta a história e a evolução das arquiteturas de software baseadas na Web. Discutimos estilos de arquitetura de software atuais, padrões e

plataformas de desenvolvimento baseados em tecnologias do lado do cliente (mais especificamente, React JS) e do lado do servidor (mais especificamente, Spring). Além disso, também discutimos os requisitos da Web 3.0, como protocolos de comunicação, microsserviços, frameworks MV, código do lado do cliente com boilerplates, programação assíncrona e integração com infraestruturas de computação em nuvem.*

1.1. Introdução

Sistemas Web se tornaram popular por conta da ubiquidade dos navegadores Web, que permitem convenientemente instalar e manter sistemas de software num servidor sem ter que alterar o software do lado do cliente, mesmo que o acesso seja realizado por milhões de navegadores [Groef 2016]. Atualmente, Sistemas Web são utilizados para as mais variadas aplicações, como por exemplo: comércio eletrônico, acesso a conteúdo audiovisual, correio eletrônico, redes sociais, buscas, portais corporativos, etc [Fox 2018].

Sistemas Web podem ser considerados uma variante do modelo de software que utiliza a arquitetura cliente servidor, onde o navegador representa o cliente que interpreta código HTML, CSS e Javascript e se comunica com o servidor utilizando uma URL e o protocolo HTTP [Deitel 2012]. Inicialmente cada página Web era entregue para os navegadores como documentos estáticos, fazendo com que os servidores apenas recebem requisições para localização e envio de arquivos. Entretanto, tal conceito evoluiu e atualmente os servidores podem gerar a cada requisição uma página dinâmica por meio de execução de software, acesso à banco de dados ou integração com outros sistemas. Além disso, uma página Web também pode executar código já no lado do cliente. Tais características fizeram surgir várias plataformas de desenvolvimento de software (linguagens, bibliotecas, APIs, frameworks) tanto do lado do servidor, como do lado do cliente [Raible 2015]. Tais soluções são escritas principalmente utilizando as linguagens Java, C#, Python, Ruby ou Javascript e existem centenas de opções [Raible 2018].

Outro fator importante é que rapidamente vários Sistemas Web agregam muito valor na sua operação e normalmente tem uma abrangência global para o seu acesso. Por exemplo, o Facebook tem 1 bilhão de acessos todos os dias e o Netflix possui 81,5 milhões de clientes em 80 países [Fox e Hal 2018]. Tais características obrigam este tipo de sistema atender requisitos cada vez mais exigentes, como por exemplo: alta disponibilidade e desempenho, escalabilidade, segurança, múltiplos pontos de falha, recuperação de desastre, suporte a transações e integração com outros sistemas [Newman 2015]. Consequentemente, o uso do estilo arquitetural cliente-servidor evoluiu bastante nesta categoria de software e vários modelos são apresentados atualmente como solução [Burns 2018].

Este capítulo tem como objetivo apresentar exemplos atuais de plataformas de desenvolvimento de software *Web* tanto do lado do cliente (React JS,), como do servidor (*Spring*). Ademais, serão apresentadas um histórico da evolução de modelos arquiteturais de Sistemas Web, como por exemplo, 3 camadas, n camadas, RESTFul [Webber 2010] e Microserviços [Bóner 2016]. Por fim, também serão apresentadas soluções que foram desenvolvidas no Tribunal de Contas do Estado da Paraíba (TCE-PB) em parceira com o Laboratório de Aplicações de Vídeo Digital (LAVID) da Universidade Federal da Paraíba (UFPB) de modo a ilustrar o uso prático das

tecnologias e modelos arquiteturais num projeto. A principal contribuição é disseminar o histórico dos sistemas Web e entender as tecnologias e arquiteturas utilizadas hoje e tendências para o futuro.

1.2. Fundamentos de Sistemas Web

Esta seção aborda os fundamentos e conceitos básicos sobre sistemas *Web* necessários para o entendimento das próximas seções do presente capítulo. São abordados os seguintes assuntos: histórico e evolução da Web; os padrões URL e HTTP e, por fim, a evolução das linguagens HTML e *JavaScript*.

1.2.1. Histórico e Evolução da Web

A *Web* – também conhecida como WWW ou *World Wide Web* – foi criada por Tim Berners-Lee no início dos anos 90, pode ser compreendida como um sistema distribuído e fracamente acoplado para o compartilhamento de documentos. Mais precisamente, a ideia original de Tim para a *Web* era que ela fosse um espaço colaborativo em que as pessoas pudessem se comunicar através de informações compartilhadas [Berners-Lee 1996]. Contudo, com o passar do tempo, o surgimento de novas tecnologias, como a Computação em Nuvem [Patterson e Fox 2012], *mashups* [Yu et al. 2008], dentre outras, impulsionou o desenvolvimento da *Web*, que deixou de ser apenas um sistema distribuído de documentos interligados e se tornou uma plataforma para aplicativos e serviços abertos, interativos e distribuídos [Maximilien, Ranabahu e Gomadam 2008].

Benioff [2008] propôs uma taxonomia que foi adaptada por Burégio [2014] que pode ser adotada para ajudar na compreensão da transformação sofrida pela *Web* em que divide a história em três ondas: (i) *read only* (do inglês, *Web* apenas de leitura); (ii) *read/write Web* (do inglês, *Web* de leitura e escrita) e (iii) *programmable Web* (do inglês, *Web* programável). Como podem ser observadas na Figura 1.1, as chamadas "ondas" não são divididas pelo tempo necessariamente, mas pelo surgimento de novas funcionalidades e, dessa forma, elas podem se sobrepõe e coexistem em determinados períodos.

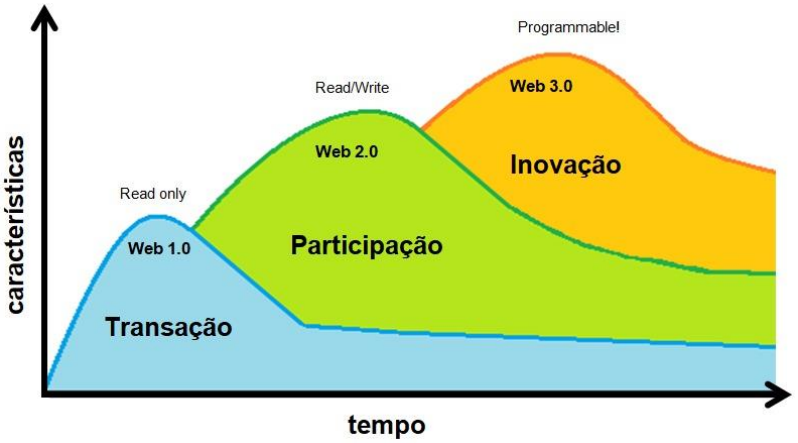


Figura 1.1. Evolução da *Web* Fonte: Adaptado de Burégio [2014]

A primeira onda da *Web*, a *read only Web*, é a chamada de *Web* 1.0 e basicamente possui aplicações capazes de prover informação em uma única direção, sendo limitadas no que diz respeito à comunicação e a interação entre os usuários.

Também são representantes da *Web* 1.0 as aplicações que permitem a realização de transações de bens e conhecimento. Dessa forma, aplicações como engenhos de busca e serviços *e-commerce* pertencem a essa primeira onda. A segunda onda da *Web*, a *read*/*write Web*, é a chamada de *Web* 2.0 e tem como principal característica a interação em comunidades, ou seja, o ponto central desta onda é a participação, colaboração e co-criação. Dessa forma, as redes sociais, *blogs*, etc, são representantes dessa segunda onda. Por último, a terceira onda da *Web*, a *programmable Web*, é a chamada de *Web* 3.0 e tem como característica a facilidade de desenvolver um sistema completo na própria *Web*, ou seja, qualquer pessoa pode criar uma nova aplicação ou serviço a partir de uma infraestrutura provida pela própria *Web*. Essa onda é impulsionada pelo advento da Computação em Nuvem. Agora, a *Web* assume o papel de uma plataforma para um ecossistema de pessoas, aplicações, serviços e até mesmo objetos físicos (*Internet of Things* – IoT).

1.2.2. Os padrões URL e HTTP

É necessário entender o funcionamento de algumas tecnologias para obter uma melhor compreensão da dinâmica existente entre os sistemas *Web* modernos. Destacam-se os seguintes conceitos fundamentais, por meio dos quais é possível desenvolver grande parte das aplicações *Web* atuais: recursos e suas representações; URIs; e ações ou verbos.

Recursos podem ser compreendidos como dados e informações – como um documento, um vídeo ou um dispositivo qualquer –, que podem ser acessados ou manipulados por meio dos sistemas baseados na *Web*. Muitos recursos do mundo real podem ser representados na *Web*, sendo necessária apenas a devida abstração de quais informações são adequados para representar estes recursos. Essa estratégia torna a *Web* uma plataforma heterogênea e acessível, pois praticamente qualquer coisa pode ser representada como recurso e disponibilizada na *Web* [Webber, Parastatidis e Robinson 2010]. A partir do momento que um recurso é publicado na Web é necessário uma forma de identificá-lo na rede, bem como de acessá-lo e manipulá-lo. Para tanto, a Web provê a URI (do inglês, *Uniform Resource Identifier*), que estabelece uma forma de identificação dos recursos por meio de um relacionamento de um-para-muitos, ou seja, uma URI identifica apenas um recurso, mas um recurso pode ser identificado por muitas URIs. Mais precisamente, um recurso como uma *Playlist*, por exemplo, pode ser representada pela linguagem de marcação HTML e interpretada pelos navegadores *Web*. De maneira similar, a *Playlist* também pode ser representada em formato XML/JSON, usualmente utilizado por outros sistemas e máquinas. A Figura 1.2 exemplifica a representação de um recurso com diversas URIs e representações.

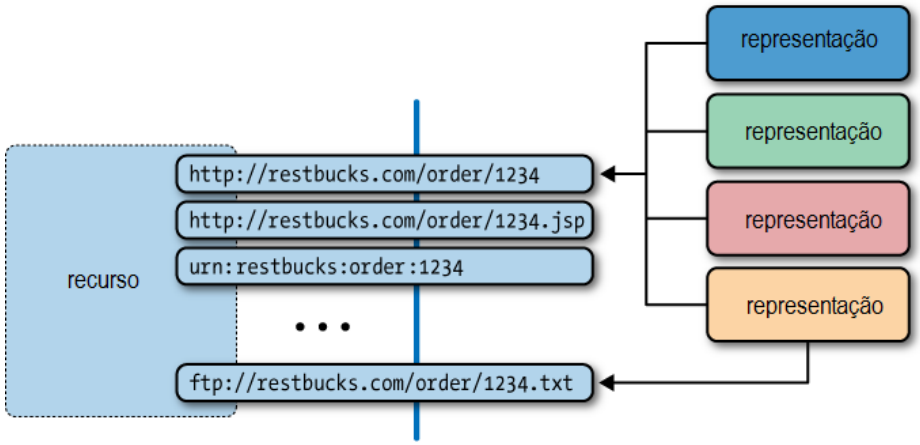


Figura 1.2. Princípios da Web [Webber, Parastatidis e Robinson 2010]

Uma URI identifica o mecanismo pelo qual um recurso pode ser acessado é geralmente referido como um URL (do inglês, *Uniform Resource Locator*). URIs HTTP são exemplos de URLs. O HTTP [RFC7231 2018] (do Inglês, *HyperText Transfer Protocol*) é um protocolo da camada de Aplicação do modelo OSI (do Inglês, *Open System Interconnection*) utilizado para transferência de dados na Internet. É por meio deste protocolo que os recursos podem ser manipulados. Para tanto, existem os chamados verbos ou ações providas pelo HTTP. A especificação original do HTTP fornece uma série de métodos de requisição responsáveis por indicar a ação a ser executada na representação de um determinado recurso. Esses métodos também são conhecidos como verbos HTTP (do inglês, HTTP *Verbs*). Os verbos HTTP utilizados para interação com os recursos *Web* são:

- **GET**: é utilizado para solicitar uma representação de um recurso específico e devem retornar apenas dados.
- **HEAD**: similar ao método GET, entretanto, não possui um corpo "*body*" contendo o recurso.
- **POST**: é utilizado para submeter uma entidade a um recurso específico, podendo causar eventualmente uma mudança no estado do recurso, ou ainda solicitando alterações do lado do servidor.
- **PUT**: substitui todas as atuais representações de seu recurso alvo pela carga de dados da requisição.
- **DELETE**: remove um recurso específico.
- **CONNECT**: estabelece um túnel para conexão com o servidor a partir do recurso alvo;
- **OPTIONS**: descreve as opções de comunicação com o recurso alvo.
- **TRACE**: executa uma chamada de *loopback* como teste durante o caminho de conexão com o recurso alvo;
- **PATCH**: aplica modificações parciais em um recurso específico.

A partir desses três conceitos fundamentais (recursos; URIs e ações) foi possível construir vários modelos de arquiteturas de software para o desenvolvimento de sistemas Web. Nas próximas duas seções serão abordadas as tecnologias de desenvolvimento do lado do cliente (seção 1.3) e servidor (seção 1.4) que apoiam a implementação desses sistemas.

1.3. Tecnologias do cliente para desenvolvimento de Sistemas Web

Com o sucesso e o aumento de seu acesso, a complexidade do conteúdo servido pela Web evoluiu, fazendo com que os conteúdos, antes apenas estáticos, tomassem forma de aplicações com capacidade de se comportar de forma similar a aplicações *desktop*. A partir dessa premissa, em 1995, a *Netscape Communications* apresentou o *JavaScript*, uma linguagem de *script* do lado do cliente que permite aos programadores melhorar a interface e interatividade do usuário com os elementos dinâmicos.

Em 2005, Jesse James Garrett tinha como objetivo a popularização de uma mudança no que o *Javascript* era capaz de fazer. Ele propôs uma abordagem na construção de aplicações Web chamada AJAX (*Asynchronous Javascript + XML*). A proposta apresentava uma mudança significativa no método tradicional das aplicações Web. No modo tradicional, interações do usuário efetuavam requisições HTTP para um servidor, que processava o pedido e retornava uma nova página HTML. A proposta de Garrett era adicionar uma camada responsável por solicitar dados ao servidor e realizar todo o processamento sem a necessidade de atualizar toda a estrutura do documento HTML que estava em exibição, tornando assim a comunicação entre cliente e servidor assíncrona [Garrett 2005]. Com o advento do AJAX, as páginas HTML tornaram-se mais amigáveis, visto que enviar os dados para o servidor para possibilitar a geração de toda a página Web deixou de ser sempre necessário.

Entretanto, a adoção do *JavaScript* não percorreu um caminho simples, principalmente, devido a competição entre os implementadores de navegadores Web que buscavam soluções específicas para os seus produtos, na maioria das vezes incompatíveis entre si. Tal contexto motivou a comunidade de desenvolvimento *JavaScriptt* a implementar bibliotecas e frameworks para mitigar esse problema, e oferecer comportamento uniforme e produtividade, como por exemplo jQuery[1]

Seguindo o movimento de sucesso do AJAX e consolidação do HTML5 e das ferramentas de produtividade para melhorar, principalmente, a implementação da interface gráfica com o usuário, foi proposto a expansão dessas facilidades para manipulação de todo o aplicativo do lado do cliente, surgindo o conceito de SPA (do Inglês, *Single Page Application*), que é um tipo de aplicação na qual carrega uma página HTML única, juntamente com seus recursos *Javascript* e CSS. Após isso, o navegador será responsável por reescrever dinamicamente à página atual em vez de carregar páginas novas inteiras de um servidor, minimizando o tráfego cliente-servidor. Com isso, o *browser* conterá mais lógica e será capaz de executar funções como renderização do HTML, validação, mudanças na interface do usuário e assim por diante [Mikowski 2013].

O *Javascript* cresceu bastante ao longo dos anos, possuindo uma comunidade bastante ativa que constantemente encontrou limitações e construiu ferramentas que supriram tais necessidades. Nos dias atuais, os desenvolvedores possuem várias alternativas modernas para criação da interface de usuário. Por exemplo: AngularJS[2], Ember[3], ReactJS[4], VueJS[5]. Na próxima seção, iremos nos aprofundar no ReactJS.

[1] https://jquery.org
[2] https://angular.io/
[3] https://www.emberjs.com/
[4] https://reactjs.org/

1.3.1. ReactJS

Ao longo da história da *Web*, várias bibliotecas *Javascript* foram desenvolvidas para tentar resolver os problemas de lidar com interfaces de usuário complexas. Entretanto, essas bibliotecas ainda mantinham a maneira clássica de separação de responsabilidades que divide o estilo (CSS), dados, estrutura (HTML) e interações dinâmicas (JavaScript).

O *ReactJS* é uma biblioteca *Javascript* para criação de interfaces, criada e mantida pelo *Facebook* [Facebook 2018]. Diferentemente de outras abordagens, ele simplifica o desenvolvimento *front-end*, pois sua principal estratégia é o Desenvolvimento Baseado em Componentes (do Inglês, *Component Driven Development*). Assim, em vez de definir um modelo único para suas interfaces, elas são divididas em pequenos componentes reutilizáveis, ou seja, o princípio é sempre buscar diminuir a complexidade por meio da separação em componentes [Mardan 2017]. A ideia é facilitar o reuso, além de promover outros benefícios como a manutenibilidade e desenvolvimento distribuído, além de integrar facilmente ao processo de desenvolvimento. Vale salientar que a criação de interfaces de usuários (UIs) componentizadas não é uma abordagem nova, porém, o *React* foi o primeiro a fazê-lo a partir do *JavaScript* puro sem uso de modelos.

O *React* não é uma estrutura *front-end Javascript* completa. Ele não estabelece uma maneira específica de desenvolver modelagem, estilo ou roteamento de dados. O *React* funciona como o "V" do modelo de arquitetura MVC (do Inglês, *Model View Controller*). Por conta disso, os desenvolvedores necessitam unir o *React* com uma biblioteca de roteamento ou modelagem. O desenvolvedor é livre para escolher quais bibliotecas utilizar, porém existe uma *React Stack* bastante adotada para auxiliar na construção de uma aplicação *front-end* completa [Mardan 2017]. Essa *stack* consiste em bibliotecas de dados e roteamento criadas para serem usadas especificamente com o *React*. Para o modelo de dados, temos, por exemplo, o *RefluxJS*[6], *Redux*[7], *Meteor*[8], *Flux*[9]. Para biblioteca de roteamento é recomendado utilizar o *React Router*[10]. E para estilização da interface do usuário é possível utilizar a coleção de componentes *React* que consomem a biblioteca do *Twitter Bootstrap*[11], o *React-Bootstrap*[12].

1.3.1.1. *Single Page Application* e ReactJS

O *React* possibilita a construção de uma SPA, embora não seja sua única forma de implementação. O código escrito em *React* pode coexistir com a marcação renderizada no servidor por algo como o ou com outras bibliotecas do lado do cliente.

Para exemplificar, assumindo que sua SPA utilize uma arquitetura do tipo MVC. O *navigator* da aplicação, funcionando como o "C" do padrão arquitetural MVC, determina quais dados buscar e qual modelo (*Model*) utilizar. Ele também realiza solicitações para obtenção de dados e preenche os *templates* (*Views*) a partir dos dados

[5] https://vuejs.org/
[6] https://github.com/reflux/refluxjs
[7] http://redux.js.org
[8] https://www.meteor.com
[9] http://facebook.github.io/flux/
[10] https://reacttraining.com/react-router/
[11] http://getbootstrap.com/
[12] https://react-bootstrap.github.io

obtidos para renderizar a interface do usuário na forma do HTML. A interface do usuário envia ações de volta ao SPA, tais como eventos do mouse, eventos de teclado, entre outros [Mardan 2017].

1.3.1.2. Virtual Dom

Um ponto que diferencia as aplicações desenvolvidas em *ReactJS* é o uso do *Virtual* DOM (VDOM), e é fundamental entendê-lo para compreender o funcionamento básico de uma aplicação desenvolvida em *ReactJS*. Trata-se de um conceito de programação na qual é criada uma representação "virtual" da interface do usuário em *JavaScript* puro, sendo mantida na memória e sincronizada com o *Document Object Model* (DOM) "real" por uma biblioteca como o *ReactDOM* [Facebook 2018].

Assim, a aplicação passa a manipular o VDOM e não o DOM diretamente. Uma das razões para a adoção do VDOM é que para que sejam realizadas as atualizações necessárias do DOM, sua estrutura pode executar diversas atualizações desnecessárias, causando assim perda de desempenho, principalmente para os casos em que a interface do usuário é complexa [Mardan 2017]. Com isso, a cada alteração no VDOM, um algoritmo primeiro calcula a diferença entre o VDOM e o DOM real e, a partir dessa análise, a biblioteca é capaz de identificar uma mudança na renderização, atualizando somente a mudança no DOM real [Chedeau 2013].

1.3.1.3 JavaScript Syntax eXtension

JSX (do Inglês, *JavaScript Syntax eXtension*) é uma extensão de sintaxe para escrever *Javascript* como se fosse XML. O JSX não é executado no navegador, mas é usado como o código-fonte para a compilação. Ele é transpilado em *Javascript* regular. Seu uso é opcional, porém é recomendado pelo *Facebook* para o desenvolvimento de aplicações em *React*. Apesar de parecer uma linguagem de modelo, o JSX possui o mesmo poder do *JavaScript* e produz elementos *React*. A Listagem 1.1 a seguir apresenta um trecho de código escrito sem o uso do JSX:

Listagem 1.1. Exemplo de Código sem o uso do JSX

```
1.const element = React.createElement("p", null, "Hello");
```

Já a Listagem 1.2 exemplifica o uso do JSX. Nela é possível observar que a sintaxe JSX ajuda a diminuir a verbosidade e facilita a criação de elementos *React*.

Listagem 1.2. Exemplo de Código com o uso do JSX

```
1.const element = <p>Hello</p>;
```

1.3.1.4. Gerenciador de pacotes

Para gerenciar todas as dependências de uma aplicação é utilizado um gerenciador de pacotes. No contexto deste trabalho, foi utilizado o gerenciador de pacotes npm[13] (*Node Package Manager*) que é distribuído com o Node.js[14], o que significa que ao realizar o

[13] https://www.npmjs.com
[14] https://nodejs.org

download do Node.js o npm é automaticamente instalado. Para verificar se a instalação acorreu adequadamente, basta executar os comandos apresentados na Listagem 1.3 no terminal:

Listagem 1.3. Comandos para verificação de instalação do node e npm

```
> node -v
> npm -v
```

1.3.1.5 *Create React App*

Create React App[15] é um *React toolchain* que constrói de forma simples e prática um *boilerplate* de uma aplicação SPA em *React*. Esta abordagem não é a única para utilizar o *React*, porém ela é recomendada para aqueles que estão iniciando no mundo do *React*, pois ela abstrai etapas de configurações importantes para uma aplicação *React* funcionar, permitindo ao programador focar apenas no código.

Para utilização do *Create React App* é necessário ter devidamente instalado em um computador a versão do *Node* >= 6 e npm >= 1.2. Para tanto, basta executar no terminal o comando descrito na Listagem 1.4 a seguir:

Listagem 1.4. Comandos para instalação do node

```
1. node install create-react-app -g
```

1.3.1.6. Primeira aplicação

Para criar a primeira aplicação *React* é necessário executar o seguinte comando no terminal, conforme a Listagem 1.5:

Listagem 1.5. Comandos para a primeira aplicação *React*

```
1. create-react-app minicurso-webmedia-react
2. cd minicurso-webmedia-react
3. npm start
```

Como já explanado, a linha 1 do código irá construir o *boilerplate* da aplicação. O *create-react-app*, além de outros arquivos de configuração, também instalará as bibliotecas *React* e *ReactDOM.* Após executá-lo, um diretório com o nome da aplicação é criado. Dentro deste diretório conterá todos os arquivos necessários para o desenvolvimento ser iniciado. Já o comando da linha 3 permite rodar a aplicação em um servidor local. O *script* "*start*" executado por intermédio do comando npm irá executar internamente o *script react-scripts start* que é responsável por configurar o ambiente de desenvolvimento e iniciar um servidor, bem como o *hot module reloading* que irá atualizar a aplicação automaticamente a cada alteração no código. Por fim, após iniciar a aplicação, ela estará acessível em http://localhost:3000/. A Figura 1.3 apresenta como estão dispostos os arquivos da estrutura inicial do projeto *ReactJS*:

[15] https://github.com/facebook/create-react-app

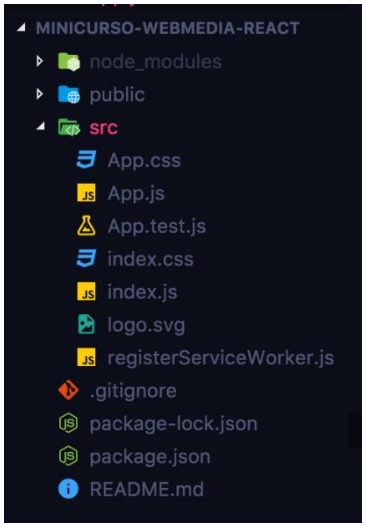


Figure 1.3. Estrutura inicial do projeto em ReactJS

O arquivo *App.css* contém algumas estilizações prontas criadas a partir do *boilerplate* da aplicação. Caso seja necessário é possível criar novas classes CSS para serem utilizadas na aplicação. O arquivo *index.js* é o ponto de entrada para a aplicação e a partir dele é chamado o arquivo *App.js*, que contém a implementação inicial de fato que é visualizada ao abrir a aplicação no servidor local. Portanto, para construir uma aplicação "Olá Mundo!" é necessário alterar o conteúdo do arquivo *App.js* conforme o código apresentado na Listagem 1.6:

Listagem 1.6. Aplicação "Olá Mundo!

```
1.import React, { Component } from 'react';
2.
3.class App extends Component {
4.    render() {
5.        return <h1>Olá mundo!</h1>;
6.    }
7.}
8.
9.export default App;
```

Vale salientar que o código apresentado na Listagem 1.6 está escrito em JSX. Além disso, ele também contém funcionalidades novas do *JavaScript* descritas no ES2015+ (*EcmaScript* 2015) ou popularmente conhecida como ES6+[16]. Por exemplo, na linha 1 é utilizada a palavra-chave *import* ao invés de ser utilizado o *require* para importar uma biblioteca externa. Na linha 2 é utilizado o conceito de *class* e na linha 9 é possível observar a utilização da palavra-chave *export*. Ao salvar este trecho de código, a aplicação irá ser atualizada automaticamente gerando o resultado que pode ser observado na Figura 1.4

[16] http://es6-features.org/

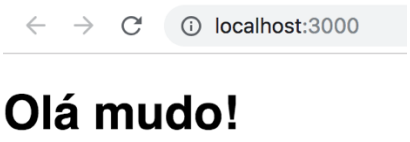


Figure 1.4. Resultado "Olá Mundo!" no navegador

1.3.1.7. Aplicação Lista de Tarefas

Para aprofundar o conhecimento sobre o *React*, esta subseção do presente capítulo demonstra a construção de uma aplicação simples baseada na ideia de listagem de tarefas a partir do mesmo projeto, previamente apresentado.

1.3.1.7.1. *Components* e *props*

Em uma aplicação *React* é necessário pensar em usar a arquitetura baseada em componentes (*Components*), que permite reutilizar o código separando a funcionalidade em partes fracamente acopladas. Por exemplo, na aplicação de lista de tarefas, ao invés de repetir o código de um item de tarefa na lista, é possível criar um componente denominando, neste caso, de "*TaskItem*" e reutilizá-lo sempre que for necessário. Adotando essa estratégia, o código torna-se escalável, legível, reutilizável e simples de mantê-lo. Essa abstração permite a reutilização de interfaces de usuário em aplicativos grandes e complexos, bem como em projetos diferentes.

As *tags* padrões do HTML (*div*, *input*, *p*, *h1*, dentro outras) podem ser utilizadas para compor as classes de componentes *React*, assim como outros componentes. Isso permite uma flexibilidade para criação de componentes robustos e potencialmente reutilizáveis. Teoricamente, os componentes são como funções *JavaScript*. É possível fornecer entradas de dados chamadas de "*props*" (que significa propriedades), e eles retornam elementos *React* que descrevem o que deve ser exibido na tela. Na linha 4 do trecho de código apresentado na Listagem 1.7, "*TaskItem*" é um componente que está recebendo como *props* um objeto *JavaScript*, descrito na linha 1, denominado "*task*". O componente "*TaskItem*" pode acessar internamente o objeto "*task*" por intermédio de suas *props*, definindo assim o que fazer com esse objeto, como por exemplo exibi-lo na tela.

Listagem 1.7. Funcionamento de *Components* e *props*

```
1.task = { title: "Estudar React" };
2.// ...
3.render() {
4.    return <TaskItem task={task} />;
5.}
6.// ...
```

É possível definir um componente de várias formas, sendo a forma mais simples por meio de uma função *Javascript*. Na Listagem 1.8 é possível observar que o objeto *props* é definido como argumento da função.

Listagem 1.8. Funcionamento de *Components* e *props*

```
1.function TaskItem(props) {
2.    render() {
3.        return <li>{props.task.title}</li>;
4.    }
5.}
```

Essa função é um componente *React* válido, pois aceita um único argumento, no caso o objeto "*props*", e retorna um elemento *React*. Componentes que são criados literalmente como funções *JavaScript* são denominados de componentes funcionais.

Dando continuidade ao assunto, outra forma de definir um componente é por meio de classes (classes ES6). Observe o trecho de código apresentado na Listagem 1.8:

Listagem 1.8. Funcionamento de *Components* e *props*

```
1.class TaskItem extends React.Component {
2.    render() {
3.        return <li>{this.props.task.title}</li>;
4.    }
5.}
```

A principal diferença entre as duas abordagens é que a segunda traz métodos adicionais, tais como métodos de ciclo de vida dos componentes provenientes da extensão "*React.Component*" (linha 1). Portanto, a decisão de uma em detrimento de outra depende apenas do que o programador deseja fazer com o seu componente.

1.3.1.7.2. Manipulando *state*

Para iniciar o desenvolvimento, será adicionado um elemento <*input*> que é responsável por receber do usuário o nome das tarefas que ele deseja adicionar em sua lista de tarefas. Para tanto, é necessário controlar o *input* para que seja possível ter acesso ao seu valor e também atualizá-lo sempre que o usuário digitar um novo caractere.

Dessa forma, esse é o momento adequado para introduzir o conceito *state*. O *state* é similar ao *props*, pois também se trata de um objeto *Javascript*, porém ele é privado e totalmente controlado pelo seu componente, ou seja, apenas o próprio componente pode alterá-lo. Suas alterações, assim como as das *props*, geram uma atualização no componente, afetando sua visualização e possíveis lógicas internas. *Props* e *states* podem possuir comportamentos semelhantes, mas eles são utilizados para finalidades diferentes.

Em HTML, elementos de formulário, como <*input*>, normalmente mantêm seu próprio estado e o atualizam com base na entrada do usuário. Em *React*, deve-se tratar o estado mutável no estado (*state*) dos componentes e atualizá-lo apenas com a função "*setState()*". Portanto, para trabalhar com esse tipo de situação deve-se manter uma única fonte e, dessa forma, é necessário combinar os dois no estado do componente. Para isso existe um *controlled component* (componente controlado), ou seja, um elemento de formulário de entrada cujo valor é controlado pelo *React*.

O código apresentado na Listagem 1.9 demonstra o acesso único ao valor contido no *input* (linha 14) que pode ser atualizado sempre que necessário. Para tanto, é utilizado o método provindo da classe *Component* chamado "*setState()*" (linha 5). Sempre que se faz necessário atualizar o estado de um componente este método é acionado, pois só assim o componente consegue saber que houve uma alteração no seu estado e que ele necessita renderizar novamente. Por exemplo, caso ocorra uma tentativa de alteração do estado de forma direta, o componente não saberá que houve atualização e consequentemente essa ela não será renderizada. Na linha 2 o objeto *state* do componente *App* é definido. Em seguida, na linha 4, o método que será responsável por acionar o *setState* é definido. O método *onChangeInputValue* é passado como valor da *props onChange* do elemento <*input*> (linha 17) e o valor do elemento <*input*> será o estado do componente, no caso, *state.inputValue* (linha 16).

Listagem 1.9. Exemplo de Código para manipulação do *state*

```
 1.class App extends Component {
 2.    state = { inputValue: '' };
 3.
 4.   onChangeInputValue = (event) => {
 5.        this.setState({ inputValue: event.target.value });
 6.    }
 7.
 8.    render() {
 9.        return (
10.            <div>
11.                <form>
12.                    <label>
13.                        Tarefa:
14.                        <input
15.                            type="text"
16.                            value={this.state.inputValue}
17.                            onChange={this.onChangeInputValue}
18.                        />
19.                    </label>
20.                </form>
21.            </div>
22.        );
23.    }
24.}
```

Vale ressaltar que O JSX nos permite que seja utilizado *props* (*value* e *onChange*) no elemento <*input*>. Ainda sobre o código da Listagem 1.9, o trecho da linha 14 a 18, que contém o elemento <*input*>, será compilada para o código a seguir da Listagem 1.10:

Listagem 1.10. Código gerado a partir do elemento <*input*>

```
1. React.createElement(
2.     'input',
3.     { value: this.state.inputValue, onChange: this.onChangeInputValue },
4.     null
5. );
```

O próximo passo pode ser observado na Listagem 1.11 em que é criada uma lista para armazenar as tarefas adicionadas pelo usuário. Assim, é adicionado mais um estado no componente denominado "*tasks*". Além disso, um *input* do tipo *submit* é criado (linha 26) para que o usuário adicione uma tarefa ao submeter o formulário.

Listagem 1.11. Código de lista para armazenamento das tarefas adicionadas pelo usuário

```
 1.   // ...
 2.    handleSubmit = (event) => {
 3.        event.preventDefault();
 4.        this.addTask();
 5.    }
 6.
 7.    addTask = () => {
 8.        const tasks = this.state.tasks;
 9.        const task = { title: this.state.inputValue };
10.        tasks.push(task);
11.        this.setState({ tasks: tasks });
12.    };
13.
14.    render() {
15.        return (
16.            <div>
17.                <form onSubmit={this.handleSubmit}>
18.                    <label>
19.                        Tarefa:
20.                        <input
21.                            type="text"
22.                            value={this.state.inputValue}
23.                            onChange={this.onChangeInputValue}
24.                        />
25.                    </label>
26.                    <input type="submit" value="Adicionar" />
27.                </form>
28.            </div>
29.        );
30.    }
```

Analisando o trecho de código da Listagem 1.11, na linha 2 é definido o método que será chamado quando o usuário inserir uma nova tarefa. Esse método é passado via

props para o elemento <*form*>. Na linha 7, o método *addTask* é responsável por adicionar uma tarefa na lista de tarefas. Vale ressaltar que ele acessa o estado do componente (linha 8) e o atualiza por intermédio do método *setState* (linha 11).

1.3.1.7.3. Exibindo lista de tarefas

Por ser possível adicionar as tarefas em uma lista, agora também é permitido exibi-las na tela do usuário. Esta atividade pode ser realizada de várias maneiras, inclusive criando um novo componente, contudo, será adotada a estratégia mais simplista. Na Listagem 1.12 pode ser observado como foi construído o método para renderizar os itens na tela e executar essa função no método "*render()*" (linha 15) do componente *App*.

Listagem 1.12. Código para exibição da lista de tarefas

```
1.    renderTasks = () => {
2.        return (
3.            <ul>
4.                {this.state.tasks.map((task, index) => (
5.                    <TaskItem
6.                        onClick={() => this.removeTask(index)}
7.                        key={index}
8.                        task={task}
9.                    />
10.               ))}
11.           </ul>
12.       );
13.   };
14.
15.   render() {
16.       return (
17.           <div>
18.               <form onSubmit={this.handleSubmit}>
19.                   <label>
20.                       Tarefa:
21.                       <input
22.                           type="text"
23.                           value={this.state.inputValue}
24.                           onChange={this.onChangeInputValue}
25.                       />
26.                   </label>
27.                   <input type="submit" value="Adicionar" />
28.               </form>
29.               {this.renderTasks()}
30.           </div>
31.       );
32.   }
```

Discutindo melhor o trecho de código da Listagem 1.12, o método "*renderTasks()*", descrito na linha 1, acessa a lista de tarefas pertencente ao estado do componente. Em *React*, transformar *arrays* em listas de elementos é similar ao método *Javascript* tradicional. No método para renderizar as tarefas, o *array* é percorrido de *tasks* utilizando a função "*map()*" (linha 4) do *Javascript*. Assim, é retornado um componente *TaskItem* (linha 5), para cada item, com suas respectivas *props* (linhas 6, 7 e 8). O componente *TaskItem*, como definido anteriormente, é um componente que retorna um elemento *<li>*.

1.3.1.7.4. Removendo itens da lista de tarefas

No trecho de código da Listagem 1.13 correspondente à implementação do componente *App*, foi definido o método *removeTask* (linha 15) que acessa o estado do componente (linha 16) e o atualiza com a nova lista de *tasks* (linha 18) após remover a tarefa solicitada pelo usuário. No método *renderTasks* (linha 21), mais especificamente na chamada do componente *TaskItem* (linhas 25 a 29), é passado por meio da *props onClick* o método *removeTask* (linha 15). Com isso o componente *TaskItem (linha 1)* acessa e executa (linha 5), ao usuário clicar no elemento, o método recebido em suas *props* que irá remover o item da lista clicado. Dessa forma a aplicação de lista de tarefas está concluída. Vale ressaltar que por motivos didáticos não foi utilizada a estilização CSS nesta aplicação. Assim, foi possível focar nos conceitos do *React*. O código fonte apresentado está disponível no GitHub[17]. Para executar a aplicação, após clonar o projeto, basta executar o comando *npm install* para baixar todas as dependências do projeto. Em seguida, digite o comando *npm start* para executar a aplicação.

Para finalizar a lista de tarefas, o usuário deve ser capaz de remover uma tarefa ao concluí-la. Existem várias maneiras de excluir um item da lista, sendo aqui desenvolvida uma abordagem simples que exclui um item por meio do clique apresentada na Listagem 1.13. Porém é necessária cautela na forma que isso será implementado na aplicação *React*. O componente *TaskItem* recebe via *props* a *task* a ser exibida, portanto ele não pode alterá-la diretamente, visto que as *tasks* pertencem ao estado do componente *App* e somente ele pode alterá-las. Para resolver este impasse, o componente *TaskItem* recebe uma função por intermédio de suas *props*, que executa (linha 5) por meio do clique do usuário excluindo o item da lista. Essa função fica localizada no componente *App* (linha 15), que tem acesso e pode alterar o seu estado.

[17] https://github.com/rkulesza/webdev

Listagem 1.13. Código para remoção de itens da lista de tarefas

```
class TaskItem extends React.Component {
  render() {
      return (
          <li
              onClick={this.props.onClick}
          >
              {this.props.task.title}
          </li>
      );
  }
}

 class App extends Component {
   // ... App component ...
   removeTask = (index) => {
     const tasks = this.state.tasks;
     tasks.splice(index, 1);
     this.setState({ tasks: tasks });
   };

   renderTasks = () => {
     return (
         <ul>
             {this.state.tasks.map((task, index) => (
                 <TaskItem
                     onClick={() => this.removeTask(index)}
                     key={index}
                     task={task}
                 />
             ))}
         </ul>
     );
   };
  // ...
 }
```

1.4. Tecnologias de servidor para desenvolvimento de Sistemas Web

Esta seção apresenta exemplos do uso de uma tecnologia de desenvolvimento do lado do servidor: Spring

1.4.1. Framework Spring

Spring fornece modelos de programação e configuração para aplicações Java EE modernas em qualquer tipo de plataforma. Seu principal foco é infraestrutura para o nível de aplicação, permitindo as equipes se concentrarem nas regras de negócios. Ele é

dividido em módulos, oferecendo suporte para desenvolvimento *Web*, *Aspect Oriented Programming* (AOP), processamento de dados, transações e integração com outras tecnologias; possui *Dependency Injection* (DI) através do seu container e suporte a testes de funcionalidades implementadas com seus módulos. A Figura 1.5 apresenta uma visão geral dos módulos que compõe o ecossistema Spring que serão detalhados nas próximas seções.

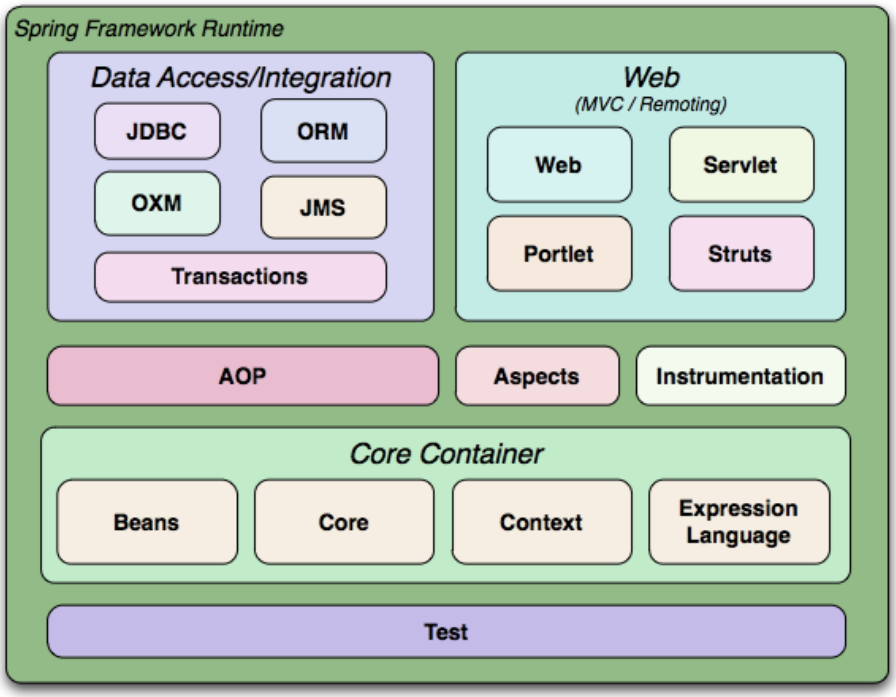


Figure 1.5 Módulos Spring Framework, Fonte: Pivotal [2018a]

1.4.1.1. Spring Core e Dependency Injection

O núcleo do Spring é um *container* que cria e gerencia *beans* automaticamente. *Beans* são instancias de objetos pertencentes ao *container Inversion of Control* (IoC). IoC é um princípio da engenharia de software que também é conhecido como *Dependency Injection* (DI). A instância de um objeto é controlada pelo *container* sendo "injetada" em outro objeto que possui dependência com o primeiro. Isso permite conexão desacoplada, aumenta a modularidade do programa e o torna mais extensível, como demonstra [Hall 2017]. Um cenário de uso real é quando o acesso a instância de um objeto é necessário em diferentes partes do programa ou quando um objeto acessa conexão entre camadas. Para realizar a injeção, ele procura por *beans* no contexto da aplicação por meio de *component scanning* e satisfaz suas dependências com *autowiring*. *Beans* possuem escopo padrão *singleton* por *container*, o que significa que é criada apenas uma instância por *container*, podendo inclusive ser configurado com outras opções de escopo. Essa implementação é um pouco diferente do padrão definido por [Gamma et al. 1996], onde o escopo é feito por classe. Uma *bean* deve ser definida em um contexto de configuração, e seu *Plain Old Java Object* (POJO) permanece o mesmo, sem invasão do *framework*.

Com fins didáticos e sem qualquer lógica de negócios real, a Listagem 1.14 deixa claro como funciona a injeção de dependência. *@Configuration* (linha 13) indica

que a classe *RefeicaoConfig* (linha 14) declara *beans* através de métodos para serem processadas pelo *container* do Spring e disponibilizadas em tempo de execução; ela prepara sua refeição. Para dar ênfase no desacoplamento de tipos, foi utilizada herança para o contexto deste exemplo, mas, para maior desacoplamento, poderia ser interface. Assim, *Refeicao* (linha 5) conhece apenas *Feijao* (linha 1), sem conhecer a especialidade dele que, nesse caso, é *FeijaoPreto* (linha 3). Poderia ser *FeijaoBranco*, outra especialidade de *Feijao* - depende do seu objetivo. E a classe que faz uso recebe a instância injetada de *Refeicao*.

Listagem 1.14. Código exemplo de injeção de dependência

```
1.  class Feijao {}
2.
3.  class FeijaoPreto extends Feijao {}
4.
5.  class Refeicao {
6.
7.      Feijao feijao;
8.
9.      Refeicao(Feijao feijao) {
10.         this.feijao = feijao;
11.     }
12. }
13. @Configuration
14. public class RefeicaoConfig {
15.
16.     @Bean
17.     Refeicao prepararRefeicao() {
18.         return new Refeicao(new FeijaoPreto());
19.     }
20. }
```

A classe *MinhaRefeicao*, apresentada na Listagem 1.15, é anotada com *@Component*, indicando que ela será detectada e inicializada através de *component scanning*, que será executado automaticamente ao iniciar uma aplicação Spring Boot. Isso permite o uso de injeção com *@Autowired*, sendo aqui uma injeção de construtor, onde é obtida uma instância de *Refeicao* criada pelo método *prepararRefeicao* na classe *RefeicaoConfig*. Assim, Spring permite desacoplamento entre objetos no desenvolvimento de aplicação e sua lógica de negócios. Inclusive entre seus módulos e a aplicação desenvolvida, como pôde ser visto na definição de *bean* e componente, e como será apresentado mais à frente na implementação de outros componentes.

Listagem 1.15. Código exemplo de injeção de dependência

```
1.  @Component
2.  public class MinhaRefeicao {
3.
4.      Refeicao refeicao;
5.
6.      @Autowired
7.      public MinhaRefeicao(Refeicao refeicao) {
8.          this.refeicao = refeicao;
9.      }
10. }
```

1.4.1.2. Spring *Aspect Oriented Programming*

Spring também oferece desacoplamento através de AOP, de acordo com [Wall 2015], que permite acesso a funcionalidades já desenvolvidas no seu programa por meio de componentes reutilizáveis. Ele permite definição dos seus próprios aspectos, bem como utilizar seus aspectos existentes, como no seu serviço de segurança *Spring Security*, que protege o sistema que o implementa realizando autorização antes de acessar suas funcionalidades. Pode ser utilizado também para gerenciamento de transações e *logging*.

1.4.1.3 Spring Web

O objeto desta seção é apresentar o desenvolvimento *Web* com o módulo Spring *Web* MVC, sendo *Movel-View-Controller*, de acordo com [Sommerville 2015], uma abordagem reconhecidamente utilizada para construção de sistemas, onde a visão do cliente da aplicação é desacoplada da sua lógica de negócios. Com esse módulo, é possível definir métodos que recebem requisições HTTP e devolvem respostas em JSON através de API *RESTful* ou uma página HTML. Também é utilizado uso também do módulo de acesso a dados Spring Data.

Spring roda na *Java Virtual Machine* (JVM) e possui duas arquiteturas de pilha: a tradicional síncrona e a mais recente nos últimos poucos anos, assíncrona. A arquitetura síncrona funciona com entrada e saída (E/S) bloqueante e atende uma requisição por *thread*. Ela é construída sobre a API *Servlet*, conhecida como *Servlet Stack,* dando origem ao módulo *Spring Web MVC,* que está presente no seu *framework* desde o início. A arquitetura assíncrona foi construída para aproveitar os processadores *multicore,* funcionar com E/S não bloqueante e atender muitas requisições concorrentes. Ela é conhecida por *Reactive Stack* e, em Spring, seu módulo é o *Spring WebFlux*. A arquitetura a ser utilizada depende do caso de uso e a escolha é sua.

1.4.1.4 Spring Data

Persistência de dados tende a gerar código *boilerplate* para criar a conexão ao banco de dados, realizar consulta, processar o resultado e finalizar a conexão, além de *queries* repetidas para cada tipo de consulta e tabela modelada. *Spring Data* abstrai o código *boilerplate* necessário para procedimentos como esses e ainda oferece métodos prontos para persistência quando implementados em conjunto com seus objetos. Isso evita erros lógicos de implementação de *query,* fechamento de recursos e tratamento de erros. Ele tem suporte a *Java Database Connectivity* (JDBC) e *Object Relational Mapping* (ORM)

com Hibernate e *Java Persistence API* (JPA). Funciona com bancos SQL e NoSQL. Ainda, possui suporte às transações com abstração para *Java Message Service* (JMS) para integração assíncrona com outras aplicações através de mensagens e faz uso de AOP para gerenciamento de transações.

1.4.1.5 Spring Test

Um processo fundamental na construção de sistemas é o *Test-Driven Development* (TDD), conforme Langr [2015], onde requisitos de software são tratados de forma específica. Testes podem ser feitos em funcionalidades pequenas e até em funcionalidades integradas, onde são tratadas as partes do sistema em conjunto. Spring permite testes com JUnit ou TestNG e também fornece suporte com seu módulo de teste para testar funcionalidades desenvolvidas com eles, inclusive em conjunto com injeção de dependência. Por exemplo, como requisições HTTP, configuração de segurança e persistência de dados. Ele disponibiliza objetos *mock* configuráveis*,* que são implementações de abstrações de contexto e ambiente de execução.

1.4.1.6. Configuração Inicial Spring Boot

Spring Boot é o ponto de partida para aplicações Spring. Ele foi projetado para facilitar a criação de aplicações prontas para produção onde você pode rapidamente iniciar o desenvolvimento com pouca configuração, incluindo integração com outras tecnologias. Spring faz uso de um modelo de programação baseado em anotações nas classes Java, permitindo substituição da programação baseada em XML, para o qual também possui suporte. Spring Boot levou isso para um outro nível dando suporte à autoconfiguração para códigos *boilerplates* de configuração e desenvolvimento.

A partir deste ponto é apresentado um fluxo de desenvolvimento com Spring Boot para entendê-lo na prática. Uma interface para criação do projeto inicial é disponibilizada como uma página *Web* em [Pivotal 2018b]. Através dela, é possível escolher a ferramenta para *build* e gerenciamento de dependências, a linguagem de desenvolvimento, o tipo de *packaging*, a versão do Spring Boot e as dependências necessárias. Aqui, é utilizado projeto Maven, linguagem Java com *packaging* Jar, que é o mais moderno, Java 8, Spring Boot 2.0.4 e as dependências *Web*, para Spring MVC e arquitetura *Servlet Stack,* JPA (do inglês, *Java Persistence API),* banco de dados H2 e *Thymeleaf*. É utilizado somente *Servlet Stack*. Por padrão, Spring utiliza o *servlet Apache Tomcat* embutido, mas pode ser configurado para outros *containers* compatíveis com a versão *servlet* mínima exigida. Após *download* do projeto, no diretório principal está o arquivo *pom.xml* com toda a configuração do projeto Maven. Em *src/main/java/nome/do/pacote* existe uma classe que inicia a aplicação Spring Boot no famoso método *main* Java com o método estático *run* da classe *SpringApplication*. Ela também possui uma anotação *@SpringBootApplication que* habilita autoconfiguração e busca por todas as *Spring beans* existentes no projeto ao ser executado.

Beans são instancias de objetos pertencentes ao *container Inversion of Control* (IoC) do Spring. IoC é um princípio da engenharia de software que também é conhecido como *Dependency Injection* (DI). A instância de um objeto é controlada pelo *container* sendo "injetada" em outro objeto que possui dependência com o primeiro. Isso aumenta a modularidade do programa e o torna mais extensível, como demonstra [Hall 2017]. *Beans* possuem escopo padrão *singleton* por *container*, o que significa que é criada apenas uma instância por *container*, semelhantemente ao padrão definido por [Gamma

et al 1996], podendo inclusive ser configurado com outras opções de escopo. Com isso, Spring permite um alto nível de desacoplamento entre objetos.

Antes de começar o desenvolvimento, é uma boa prática executar o projeto para garantir que está tudo funcionando. Pode-se fazer isso a partir de um *Integrated Development Environment* (IDE) com suporte ao gerenciador de projeto Apache Maven ou diretamente com o Maven a partir de um terminal. A partir de um IDE é simples construir e executar o projeto com sua interface. No terminal, pode-se utilizar os arquivos *mvnw*, escrito em *shell script*, para sistemas Unix e *mvnw.cmd,* escrito em *batch*, para *Windows*. Basta executar um dos seguintes comandos: *./mvnw spring-boot:run* ou *mvnw.cmd spring-boot:run*. Também é possível gerenciar o projeto diretamente pelo comando *mvn,* independente desses arquivos, com Maven configurado no sistema. Ao inserir o comando de execução, todas as classes são compiladas e os arquivos *.class* resultantes são armazenados no diretório *target* criado automaticamente.

1.4.1.7. RESTful Web Service com Spring MVC

Para o desenvolvimento do projeto é utilizado o módulo Spring MVC (do inglês, *Model View Controller*), que mapeia requisições com anotações. Até o presente momento existe foi obtido um projeto criado pelo inicializador do Spring, que vem pronto para execução. O endereço padrão de execução do servidor é *localhost:8080*, e pode ser acessado diretamente no navegador ou com outro cliente HTTP. Agora é criada uma API *REST* para simplesmente retornar uma mensagem. Ela pode ser visualizada em *localhost:8080/mensagens.*

Na Listagem 1.16 é possível observar um serviço REST que mapeia requisições HTTP do tipo GET no caminho */mensagens* (linha 2). Para entender melhor o funcionamento, é preciso conhecer os estereótipos *@Componente* e *@Controller* fornecidos pelo Spring. Anotações de estereótipos declaram componentes que serão identificados e registrados como *beans* gerenciadas pelo *container* IoC e iniciadas junto com a aplicação. *@Component* é um tipo genérico para componentes. *@Controller* é uma especialização de *Component* utilizada para representar controladores *Web* que recebem requisições. Ela costuma ser usada com *@RequestMapping* (linha 2) para mapear requisições em nível de classe ou de método. Em nível de método, normalmente utiliza-se especializações para identificar métodos HTTP como *@GetMapping* (linha 5), que possui a mesma semântica do código a seguir. E assim por diante para os métodos POST, PUT, DELETE E PATCH. *@RestController* (linha 1) é uma especialização de *Controller* com adição de *@ResponseBody,* que escreve diretamente no corpo da resposta HTTP.

Listagem 1.16. Código de serviço REST que mapeia requisições HTTP

```
1. @RestController
2. @RequestMapping("/mensagens")
3. class ControladorMensagem {
4.
5.     @GetMapping
6.     String encontrarMensagem() {
7.         return "Opa!";
8.     }
9. }
```

Classes com papel de controlador e seus métodos que recebem requisição são normalmente públicos, mas foram omitidos aqui. Agora, um objeto é retornado no formato *Javascript Object Notation* (JSON). Para tanto, um sistema de avaliação é simulado. Além disso, é definada uma classe *Avaliacao* na Listagem 1.17 em que seus métodos *gets* e seu construtor estão omitidos.

Listagem 1.17. Código de sistema de avaliação

```
1. class Avaliacao {
2.     int classificacao;
3.     String comentario;
4. }
```

Na Listagem 1.18 o controlador retorna um objeto *Avaliacao* (linha 7), no formato JSON, requerendo obrigatoriamente métodos *get* para os atributos da classe.

Listagem 1.18. Código da classe *ControladorAvaliacao*

```
1. @RestController
2. @RequestMapping("/avalicoes")
3. class ControladorAvaliacao {
4.
5.     @GetMapping
6.     Avaliacao encontrarAvaliacao() {
7.         return new Avaliacao(5, "Ótimo");
8.     }
9. }
```

Dando continuidade, o código da Listagem 1.19 apresenta como acessar recursos em *Web services* através de identificadores. Para isso, adiciona-se um parâmetro na API. Mas antes, é necessário incluir o novo campo *id* na classe *Avaliacao* e seu método *get*. O recurso pode ser acessado em *http://localhost:8080/avaliacoes/100*, onde 100 é um argumento para o ID do objeto procurado. Variáveis *Uniform Resource Identifier* (URI) podem ser declaradas entre {} e acessadas com *@PathVariable* no parâmetro do método (linha 2). Elas são convertidas automaticamente com suporte padrão a tipos como *long*, *double* e *String*.

Listagem 1.19. Código do método *encontrarAvaliacao*

```
1. @GetMapping("/{id}")
2. Avaliacao encontrarAvaliacao(@PathVariable long 1. id) {
3.     return new Avaliacao(id, 5, "Ótimo");
4. }
```

Até aqui, tem-se uma API que permite consulta, mas receber dados também é necessário e isso é demonstrado Listagem 1.20. Para receber requisições POST, basta utilizar *@PostMapping* e exigir um corpo na requisição com *@RequestBody* (linha 2) seguido do tipo do objeto esperado no parâmetro do método. Além disso, a classe *Avaliacao* precisa de um construtor padrão para instância do objeto. Construtor padrão não possui parâmetros e Java define um de forma implícita para cada classe. Se for definido algum outro, o padrão é sobrescrito e deve ser declarado de forma explícita, se desejado.

Listagem 1.20. Código do método *encontrarAvaliacao*

```
1. @PostMapping
2. void criarAvaliacao(@RequestBody Avaliacao avaliacao) {
3.
4. }
```

Para os métodos PUT ou PATCH, pode-se usar dois parâmetros: um identificador com *PathVariable* e *RequestBody* para o conteúdo que se pretende atualizar. Para o método DELETE, basta receber um identificador com *PathVariable* para procurar o recurso alvo. É exemplificado um JSON, observe Listagem 1.21, para a classe *Avaliacao*, enviado no corpo da requisição de métodos que exigem um, como o nosso método POST anterior. Assim, tem-se uma API *RESTful* que permite comunicação entre cliente e servidor com transmissão de dados no formato JSON.

Listagem 1.21. Exemplo de arquivo JSON utilizado na comunicação cliente/servidor

```
{
    "id": 100,
    "classificacao": 5,
    "comentario": "Ótimo!"
}
```

É possível utilizar o Spring MVC *Test* para testar a API ou um cliente HTTP como a aplicação *Postman*, um *API Development Environment* (ADE) gratuito.

1.4.1.8. Persistência de Dados com Spring Data JPA

Spring Data reduz significativamente a quantidade de código *boilerplate* necessária para implementar a camada de acesso a dados. O acesso da camada de dados será implementado com Spring *Data* JPA. Ele funciona com uma das mais famosas implementações da JPA: *Hibernate*. Spring Boot fornece o arquivo *application.properties* para configuração do projeto em *src/main/resources*. Por conveniência, pode-se usar um arquivo YAML, pois ele possui um formato de

configuração em hierarquia. Assim, o arquivo *application.yml* é criado no mesmo local e o outro é dispensado. O banco de dados utilizado será o H2, um banco relacional em memória que funciona apenas durante a execução da aplicação. Spring Boot fornece integração embutida para ele e não exige instalação, tornando-o conveniente para testes e para este tutorial. Para utilizar outro banco relacional, inclusive para persistência em memória não volátil, como MySQL, seria necessário apenas incluir seu conector em *pom.xml* e a configuração adequada em *application.yml*; o código Java da entidade permaneceria o mesmo.

A Listagem 1.22 mostra como acessar o banco de dados H2 por meio da configuração padrão do Spring *Boot* (linha 4). Neste código também é habilitada a visualização das *queries* (linha 6) realizadas pela JPA. Isso é feito adicionando-se o trecho de código da Listagem 1.22 no arquivo *application.yml.*

Listagem 1.22. Configuração do acesso ao banco de dados H2

```
1. spring:
2.   h2:
3.     console:
4.       enabled: true
5.   jpa:
6.     show-sql: true
```

A aplicação pode ser executada e o painel H2 visualizado em *localhost:8080/h2-console* em um navegador. São utilizados os mesmos dados no painel conforme Figura 1.6. Ao testar a conexão deve ser visualizada uma mensagem de sucesso.

Figure 1.6. Painel de acesso ao banco H2 num navegador com Spring, Fonte: Autores [2018]

Após configuração, segue a atividade de desenvolvimento que pode ser observada no código da listagem 1.23. JPA faz uso de anotações pertencentes ao pacote *javax.persistence* para mapear objetos para tabelas relacionais (ORM). A classe *Avaliacao* é atualizada e o restante dela pode permanecer o mesmo. *@Entity* (linha 1)

indica que *Avaliacao* é uma entidade JPA mapeada para uma tabela relacional com todos seus atributos como colunas. *@Id* (linha 4) especifica a chave primária da tabela e *@GeneratedValue* (linha 5) indica que o ID deve ser gerado automaticamente. Um construtor padrão é necessário porque JPA exige. A tabela e suas colunas podem ter seus nomes personalizados com as anotações *@Table(name = "nometabela")* no topo da classe e *@Column(name = "nomecoluna")* nos atributos. Caso não sejam especificados, são utilizados os mesmos nomes da classe e dos atributos. Existem outras personalizações normalmente utilizadas, como validação de dados dos atributos das classes com anotações contidas no pacote *javax.validation.constraints*.

Listagem 1.23. Código da Classe *Avaliacao*

```
1.  @Entity
2.  class Avaliacao {
3.
4.      @Id
5.      @GeneratedValue
6.      private long id;
7.
8.      public Avaliacao() {}
9.  }
```

Alguns tipos de operações em bancos de dados são comuns em muitos sistemas, como as operações CRUD, por exemplo. A característica mais destacada do Spring *Data* é a capacidade de criar repositórios de forma automática. Repositórios do Spring são interfaces que podem ser definidas para acessar dados. Eles disponibilizam métodos comuns para persistência e podem criar *queries* a partir do nome de métodos personalizados em tempo de execução. Para *queries* mais complexas, pode-se também defini-las utilizando *Java Persistence Query Language* (JPQL) ou SQL nativa com *@Query* acima do método. A Listagem 1.24 apresenta a classe que representa um repositório para a classe *Avaliacao*.

Listagem 1.24. Código do repositório da classe *Avaliação*

```
1. @Repository
2. interface RepositorioAvaliacao extends
3.     CrudRepository<Avaliacao, Long> {
4.
5. }
```

Apenas com o código da Listagem 1.24 já é possível realizar operações no banco. *@Repository* indica que uma classe possui papel *Data Access Object* (DAO). *CrudRepository* (linha 3 do código previamente mencionado), como o nome sugere, fornece funcionalidades CRUD para a entidade especificada no tipo genérico esperado, seguida pelo tipo da sua chave primária. Agora, o repositório é acessado no *ControladorAvaliacao* (Listagem 1.25) para aprimorar a API.

Listagem 1.25. Código da classe *ControladorAvaliacao*

```
1. class ControladorAvaliacao {
2.
3.     RepositorioAvaliacao repositorioAvaliacao;
4.
5.     @Autowired
6.     ControladorAvaliacao(RepositorioAvaliacao r) {
7.         this.repositorioAvaliacao = r;
8.     }
9. }
```

Discorrendo um pouco mais sobre o código da Listagem 1.25, observa-se que nele é definido um atributo do tipo *RepositorioAvaliacao* (linha 3) que é iniciado no construtor da classe. *@Autowired* (linha 5) solicita a instância de uma *bean* criada anteriormente, nesse caso, do *RepositorioAvaliacao*. O restante do código anterior da classe pode permanecer o mesmo. O *Autowired* pode ser removido do construtor e Spring vai reconhecer a solicitação da *bean* do mesmo modo. O principal objetivo de utilizá-la aqui foi mostrar o seu funcionamento. Com acesso à instância do repositório, serão utilizados os métodos de criar e encontrar avaliação (linha 2 e 8 da Listagem 1.26).

Listagem 1.26. Código dos métodos *criarAvaliacao* e *encontrarAvaliacao*

```
1.  @PostMapping
2.  void criarAvaliacao(@RequestBody Avaliacao avaliacao) {
3.      repositorioAvaliacao.save(avaliacao);
4.  }
5.
6.  @GetMapping("/{id}")
7.  Avaliacao encontrarAvaliacao(@PathVariable long id) {
8.      return repositorioAvaliacao.findById(id)
9.      .orElse(null);
10. }
```

A Listagem 1.27 apresenta trechos de código que ajudam a entender como tirar proveito ainda mais do Spring *Data*. Nela, uma *query* é criada a partir do nome de um método personalizado com palavras-chave fornecidas pelo Spring. Em *RepositorioAvaliacao*, é adicionado o método a seguir para encontrar todas as avaliações por valor de classificação. *find* indica o inicio de uma busca; *Classificacao* corresponde ao atributo da classe *Avaliacao*; e a palavra-chave *Equals* é utilizada para comparação. Em seguida, compare com a SQL nativa que tem a mesma semântica. Nela, *classificacao* corresponde à uma coluna da tabela de *Avaliacao* e *valor* corresponde ao parâmetro *int classificacao* do método *findByClassificacaoEquals*.

Listagem 1.27. Recursos do Spring *Data*

```
List<Avaliacao> findByClassificacaoEquals(int classificacao);

"select * from avaliacao where classificacao = valor;"
```

A nova busca é adicionada no *ControladorAvaliacao* com o método a da Listagem 1.28. Ele exige um parâmetro para filtrar avaliações (linha 1) de acordo com o valor de classificação. *@RequestParam* (linha 3) é utilizada para parâmetros de busca ou dados de formulário. Essa funcionalidade pode ser acessada através de uma requisição GET em *localhost:8080/avaliacoes?classificacao=5*.

Listagem 1.28. Método *encontrarPorClassificacao*

```
1. @GetMapping(params = "classificacao")
2. Iterable<Avaliacao> encontrarPorClassificacao(
3. @RequestParam int classificacao) {
5.     return repositorioAvaliacao
6.     .findByClassificacaoEquals(classificacao);
7. }
```

É possível também buscar avaliações por palavras contidas na *string* comentário. O método apresentado na Listagem 1.29 em *RepositorioAvaliacao*. Nele, as palavras-chave utilizadas são: *Containing* para encontrar avaliações que possuam uma *substring* no seu comentário e *IgnoreCase* para não diferenciar letras maiúsculas e minúsculas.

Listagem 1.29. Método *findByComentarioContainingIgnoreCase*

```
List<Avaliacao> findByComentarioContainingIgnoreCase(String
comentario);
```

Com isso, tem-se um RESTful *Web Service* integrado com persistência de dados. As *queries* realizadas podem ser acompanhadas no *console* e seus dados verificados no banco através do painel *Web* do H2 em *localhost:8080/h2-console*.

1.4.1.9. Lógica de Negócios como Serviço

Spring disponibiliza também um estereótipo *@Service* que é especialização de *@Component*. Ele é baseado no *Domain-Driven Design*, proposto por Evans [2003], onde podem ser oferecidas operações isoladas através de interfaces. Semelhantemente, pode também indicar uma fachada para a lógica de negócios. *@Service* é um estereótipo de propósito geral e cabe ao desenvolvedor definir bem a sua semântica. Nesse projeto, podemos abstrair o acesso aos dados como regra de negócios. Nossos controladores não vão conhecer o repositório. E, caso exista algum processamento antes ou depois da consulta no repositório, pode ser realizado no componente de serviço e nossos controladores - ou outras classes clientes - não precisam implementar nem conhecer seus detalhes.

Duas classes de serviço são adicionadas conforme o código da Listagem 1.30, sendo obtida uma instância do repositório na implementação do serviço através de inicialização no construtor, como anteriormente. No *ControladorAvaliacao* (linha 19), é obtida uma instância de *ServicoAvaliacaoImpl* (linha 6) através da interface *ServicoAvaliacao* (linha 1). Spring reconhece automaticamente a classe que implementa a interface pelo tipo de dado, desde que ela tenha anotação de componente, nesse caso, com *@Service* (linha 5). Todos os acessos anteriores ao repositório apresentados aqui também podem ser substituídos por acessos ao serviço.

Listagem 1.30. Classes de serviço

```
1. interface ServicoAvaliacao {
2.
3.     Optional<Avaliacao> encontrarPorId(long id);
4. }
5.  @Service
6.  class ServicoAvaliacaoImpl implements ServicoAvaliacao {
7.
8.      RepositorioAvaliacao repositorioAvaliacao;
9.
10.      ServicoAvaliacaoImpl(RepositorioAvaliacao r) {
11.          this.repositorioAvaliacao = r;
12.      }
13.
14.      @Override
15.      public Optional<Avaliacao> encontrarPorId(long id) {
16.          return repositorioAvaliacao.findById(id);
17.      }
18.  }
19.  class ControladorAvaliacao {
20.
21.      ServicoAvaliacao servicoAvaliacao;
22.
23.      ControladorAvaliacao(ServicoAvaliacao sa) {
24.          this.servicoAvaliacao = sa;
25.      }
26.  }
```

1.4.1.10. Página Web com Spring MVC

Spring permite a criação de páginas *Web* no *server-side* com diferentes tecnologias. Uma delas é *Thymeleaf*, um modelo Java que funciona com HTML5 e possui autoconfiguração com Spring Boot. Será criada uma simples página *Web* para demonstrar basicamente sua funcionalidade. Com configuração padrão, os arquivos são procurados em *src/main/resources/templates*. Lá, é criado um arquivo *inicial.html* e incluído o código a da Listagem 1.31. Nele, é feita a importação do *Thymeleaf* (linha 2), o que permite a utilização da palavra-chave *th* para acessar dados recebidos do *server-side* como tipos primitivos ou não primitivos, como *mensagem*, definida entre ${} (linha 3) para indicar uma variável.

Listagem 1.31. Páginas *Web* com *Thymeleaf*

```
1. <!DOCTYPE html>
2. <html xmlns:th="http://www.thymeleaf.org">
3. <h1 th:text="${mensagem}"></h1>
4. </html>
```

Agora, é preciso tratar requisições para esta página. Como pode ser observado na Listagem 1.32, a classe *ControladorInicial.java* é criada em *src/main/java/nome/do/pacote* com o código a seguir. *@Controller* (linha 1) indica que essa classe é um componente que trata requisições, de forma semelhante ao que vimos antes, mas não implementa *@ResponseBody* como em *@RestController*. O método *inicial* (linha 6) recebe um *Model* e retorna uma *view*, nesse caso, *inicial.html* (linha 8). Através do *model*, é possível mapear dados entre controlador e visão com chave-valor.

Listagem 1.32. Classes *ControladorInicial*

```
1. @Controller
2. @RequestMapping("/inicial")
3. class ControladorInicial {
4.
5.     @GetMapping
6.     String inicial(Model model) {
7.         model.addAttribute("mensagem", "Avaliações");
8.         return "inicial";
9.     }
```

Requisições e seus parâmetros podem ser tratadas com anotações como visto antes em *RestController*. Para exibir dados de um objeto, os trechos de código da Listagem 1.33 são inseridos nos arquivos HTML e Java. Para mais funcionalidades, a documentação *Thymeleaf* está disponível em *https://www.thymeleaf.org/*

Listagem 1.33. Códigos HTML e Java para exibir dados de um objeto

```
1.  <div th:object="${avaliacao}">
2.      <td th:text="${avaliacao.classificacao}"></td>
3.      <td th:text="${avaliacao.comentario}"></td>
4.  </div>
5.  @GetMapping
6.  String inicial(Model model) {
7.
8.      Avaliacao avaliacao = new Avaliacao(5, "Muito bom");
9.      model.addAttribute("avaliacao", avaliacao);
10.  }
```

O código fonte aqui apresentado está disponível no *Github*[18] em. Spring disponibiliza uma rica documentação em sua página oficial *https://spring.io/* onde é possível conhecer melhor os projetos fornecidos.

[18] https://github.com/rkulesza/webdev

1.5. Arquiteturas de Sistemas Web

As arquiteturas de sistemas Web evoluíram drasticamente desde a criação da Internet. No início os sistemas eram feitos utilizando a arquitetura CGI (do inglês, *Common Gateway Interface*). Esse recurso deu um grande poder aos servidores, já que passou a oferecer a capacidade de executar scripts de códigos – geralmente *Perl* – ao processar as requisições HTTP, fazendo com que os sistemas Web pudessem processar requisições de uma forma mais dinâmica [Hunter e Crawford 2001].

Em seguida outro problema no início da Web: era muito difícil separar os códigos de apresentação e lógica das aplicações, dificultando o seu desenvolvimento. Surgiram então os *template systems*, eles permitiam que códigos executáveis de uma linguagem de programação fossem inseridos diretamente nos arquivos responsáveis pela apresentação do sistema. Assim, dividia-se melhor as 2 camadas (apresentação e lógica) [Fields e Marc 2012]. Depois disso, diversas arquiteturas surgiram, entre elas o "modelo 2" da arquitetura MVC, que mais tarde tornou-se um dos principais modelos de sistemas Web [Sommerville 2015] e impulsou tecnologias como os frameworks Struts, Tapestry e JSF (do inglês, *Java Server Pages*). Também nessa época foram desenvolvidos frameworks para facilitar o mapeamento entre modelos orientado à objetos e relacionais (por exemplo, o *Hibernate*), dando origem as arquiteturas em 3 camadas (apresentação, lógica de negócio e dados) [Fields e Marc 2012].

Com o crescimento do uso dos sistemas em ambiente corporativo e de acesso global, foi necessário a divisão do processamento de 3 para n camadas [Fowler et al. 2002], dando origem às plataformas de execução distribuída como o JEE (do inglês, *Java Platform, Enterprise Edition*), Net. e Spring. Surgiram também protocolos de comunicação (SOAP, REST, etc.) que permitiam que os sistemas se comunicassem independentemente da linguagem de programação utilizada e facilitando a integração de sistemas heterogêneos e legados. Com isso, os desenvolvedores não estavam mais somente construindo aplicações que serviam conteúdos para os navegadores; mas sim, sistemas complexos que envolviam várias camadas de comunicação interna e externa (com outros sistemas) [Holdner 2018]. A partir daí os sistemas cresceram bastante, e a quantidade de usuários aumentou consideravelmente, fazendo com que esses sistemas se tornassem grandes demais, transformando-os em gigantes sistemas monolíticos [Newman 2015]. Esses sistemas possuem diversos problemas de escalabilidade e desempenho quando muitos usuários o utilizam. A solução foi encontrada em arquiteturas menos monolíticas e mais distribuídas – como as de SOA (do inglês, *service-oriented architecture*) utilizando o conceito de Microserviços e persistência poliglota [Sadalage e Fowler 2012]. Esses modelos possuem uma melhor distribuição de cada serviço do sistema, fazendo com que a carga de requisições em cada um seja melhor distribuída, melhorando consideravelmente requisitos de escalabilidade (por exemplo, balanceamento de carga e alta confiabilidade) [Newman 2015].

Como já citado anteriormente, grandes avanços também foram feitos na camada de apresentação no lado do cliente, diversos frameworks foram lançados e permitem que os sistemas Web tenham uma desempenho e usabilidade comparáveis com sistemas desktop tradicionais [Scott 2016]. Esses frameworks utilizam arquiteturas SPA [Scott 2016], atualizando somente o necessário através do uso de versões mais recentes do *Javascript* (por exemplo, ECMAScript versão 5 e 6) e comunicações AJAX com o

servidor [Scott 2016]. Esse modelo remove a responsabilidade de gerar a camada de visão dos servidores, deixando os sistemas mais leves, rápidos e fluídos [Scott 2016].

Nesse contexto e reforçado pelo mapa de opções que podemos observar na Figura 1.7, existem atualmente inúmeras opções de plataformas de desenvolvimento (linguagens, APIS, bibliotecas, frameworks etc.) para sistemas, Na próxima seção é descrito um estudo de caso que demonstra um conjunto de opções do estado da técnica em relação ao uso de tecnologias (cliente e servidor) e a aplicação de conceitos de arquiteturas modernas para sistemas Web.

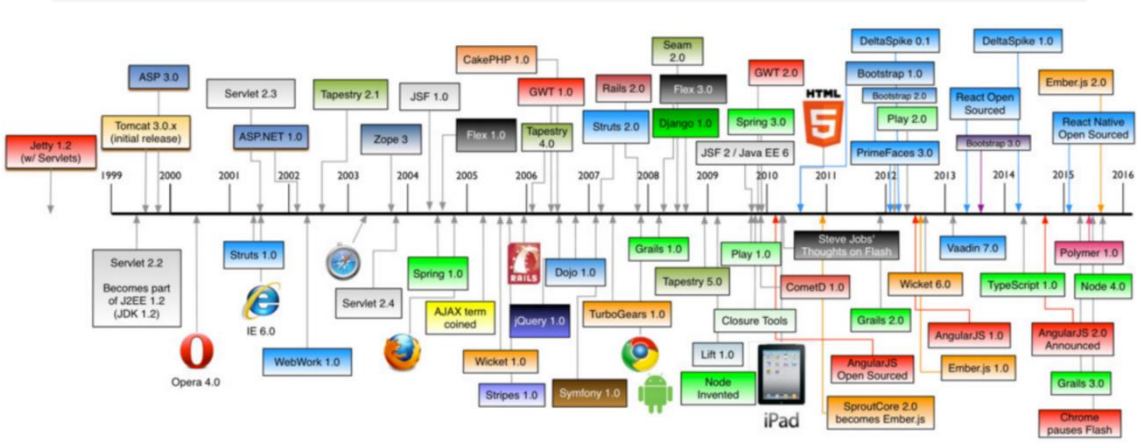


Figura 1.7 – Plataformas para desenvolvimento de sistemas Web. Fonte: Raible [2018].

1.5.1. Estudo de Caso: Você Digital

Você Digital compreende um projeto de pesquisa e desenvolvimento entre o LAVID/UFPB e o TCE-PB para modelagem e desenvolvimento de uma plataforma computacional colaborativa de governo eletrônico. O principal objetivo é permitir a automação de diagnósticos e escutas populares, que possibilite uma melhor interação e comunicação entre a sociedade e os gestores públicos.

Adicionalmente, a solução pretende explorar a inteligência coletiva presente nas redes, construindo uma participação cidadã, o que pode ajudar tanto na redução de custos no processo de geração destas diretrizes, bem como na promoção da transparência e confiabilidade com um real e efetivo ganho de tempo diante da visualização virtual e democrática de demandas e necessidades da sociedade. Como ferramenta de gestão pública digital a ideia é avaliar parte dos serviços públicos como também fomentar a participação popular no processo de tomada de decisões de auditores e gestores públicos. Como forma de avaliar a plataforma, no projeto está em desenvolvimento um aplicativo que utilizará novos métodos capazes de aumentar o envolvimento de cidadãos no contexto de diagnóstico de problemas em diversas áreas de gestão pública (por exemplo, educação, saúde e segurança).

1.5.1. 1. Projeto arquitetural de alto nível

Na Figura 1.8 é possível visualizar o projeto arquitetural de alto nível do sistema “Você Digital” com seus subsistemas destacados na cor azul. Os possíveis sistemas internos do TCE-PB estão destacados em laranja e os sistemas externos são visualizados no canto

superior da ilustração (ver na Figura 1.8 “Sistemas com *OpenID*” e “*Google Maps* e *Google Places*”). Tanto uma parte do sistema “Você Digital”, como os sistemas internos do TCE-PB utilizarão a infraestrutura de *datacenter* e virtualização disponível atualmente no TCE-PB.

O sistema "Você Digital" é composto por dois grandes subsistemas: 1) 2 (dois) sistemas de software cliente (*front-end*) e 2) 1 (um) sistema de retaguarda (*backend*). O primeiro (1) conjunto possui um aplicativo para dispositivos móveis compatíveis com a tecnologia *React Native* (ver figura 1.8 “App Você Digital”) e estará disponível para *download* nas lojas virtuais da *Apple* e *Google*. Além disso, também há um sistema de software cliente (ver na Figura 1.8, “Administração Você Digital”) que permitirá a administração do sistema “Você Digital” por meio de tarefas como gerenciamento de cadastros e dados, permissões de usuários, geração de relatórios estatísticos etc. Tal aplicativo é baseado na abordagem SPA (do inglês, *Single Page Application*) e na tecnologia *React*, de modo a permitir sua execução em qualquer navegador *Web* (*desktop* ou dispositivo móvel). O segundo conjunto (2) tem como papel o processamento dos cadastros de dados disponível no sistema, bem como processamento desses dados para gerar informações para os usuários. Este subsistema é dividido em três partes: I) Controlador: responsável pela distribuição de carga, alta disponibilidade e acesso seguro de dados e informações disponíveis para os aplicativos citados por meio de uma API *RESTful*; II) Servidores de aplicação (*containers* baseados na plataforma *Docker* e em tecnologias para desenvolvimento de arquiteturas de microserviços) responsável pelo tratamento da integração e processamento de dados interno e externos e particionamento das funcionalidades disponíveis para os softwares clientes e; III) Sistemas de Banco de Dados: responsável pelo armazenando dos dados utilizando persistência poliglota (neste módulo são utilizados tecnologias SQL e/ou NoSQL). A comunicação entre os aplicativos (1) e os servidores (2) é realizada por meio do protocolo HTTPS e APIs *RESTFul*.

Em relação a requisitos de integração do sistema “Você Digital” com os sistemas externos, foram utilizadas APIs disponíveis em sistemas compatíveis com *OpenID* para permitir autenticação externa (por exemplo, redes sociais) de modo a não ser necessário um cadastro no sistema “Você Digital” para ter acesso aos serviços do aplicativos. Do mesmo modo, foram utilizadas as APIs da plataforma *Google Maps* de modo a obter informações de geolocalização (*Google Maps*) e pontos de interesses geolocalizados cadastrados por pessoas físicas e jurídicas (*Google Places*).

Já para a integração com os sistemas internos, foi realizado um mapeamento de que dados (que podem estar um banco de dados SQL ou servidores) e/ou informações que seriam necessárias. A partir dessa identificação, a equipe do TCE-PB ofereceu uma API *RESTFul* para comunicação entre os sistemas. De forma análoga, o sistema “Você Digital”, oferece APIS *RESTFul* para o TCE-PB acessar dados.

De acordo com a Figura 1.8, o subsistema servidor de aplicação foi organizando nos seguintes módulos: AAA (do inglês, *Authentication, Authorization and Accounting*): trata-se dos procedimentos relacionados à autenticação, autorização e auditoria. Como se sabe, a autenticação verifica a identidade dos usuários, a autorização lida com as permissões, ou seja, garante que um usuário autenticado somente tenha acesso aos recursos autorizados para seu perfil e, por fim, a auditoria está relacionada à ação de coleta de dados sobre o comportamento dos usuários em relação ao sistema.

Vale salientar que este módulo se comunica com serviços externos de autenticação e é o responsável por gerenciar as seções de modo assíncrono e utilizando um banco de dados que não utilize sistema de arquivos baseado em disco; Administração (CRUD): gerencia todas as entidades do modelo de classes, ou seja, é responsável por realizar as quatro operações básicas de criação, consulta, atualização e destruição em banco de dados; Publicação (*inputs*): este módulo é a principal fonte de entrada de dados do sistema "Você Digital", sendo responsável por receber todas as informações geradas pelos usuários como avaliações, comentários, gravações de vídeos e fotos. Além desta função, este módulo também acumula a responsabilidade de realizar o tratamento de segurança dos dados. Em virtude da natureza do sistema, faz-se necessária a aplicação de filtros de textos nos comentários a fim de identificar colocações inadequadas, bem como também é adequado "sanitizar" os dados para evitar ataques de injeção. Outra funcionalidade é prover autenticação do aplicativo a fim de evitar fraudes por meio de programas de inteligência artificial (robôs) que podem ser usados para manipulação de informação; Busca: lida com pesquisas de baixa granularidade, como consultas diversas ao banco de dados; e por fim, Consumo (*outputs*): este módulo utiliza o módulo de Busca para realizar *Data Analytics*, ou seja, gerar dados estatísticos, transformação de dados, gráficos, relatórios, dentre outras análises.

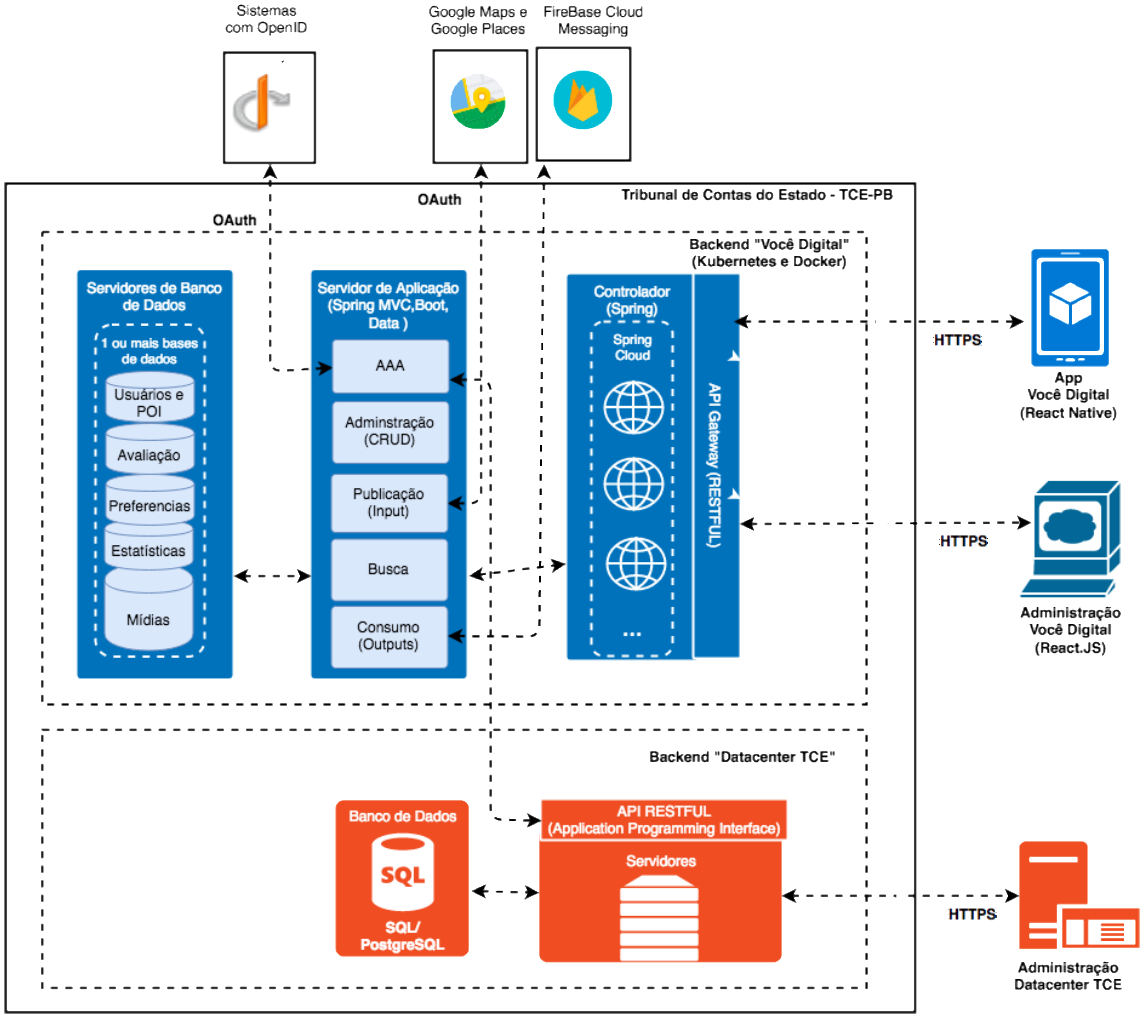


Figura 1.8. Projeto arquitetural de alto nível do sistema "Você Digital", Fonte: Autor [2018]

Dando continuidade ao detalhamento da arquitetura, o subsistema servidor de Banco de Dados foi organizando nas seguintes bases: Usuários e POI: essa base guarda as informações de cadastro de usuários e os seus pontos de interesse; Avaliação: essa base guarda as informações relacionadas ao histórico das avaliações realizadas pelos usuários; Preferências (perfil): essa base guarda informações dinâmicas relacionadas aos usuários, como: IP, Latitude e Longitude, dentre outras; Estatísticas: essa base guarda estatísticas persistentes quem podem ser utilizadas pelo módulo Consumo do subsistema servidor de aplicação; e por último, Mídias: que é uma base que guarda todas as mídias geradas pelos usuários, como textos, imagens, vídeos e áudios.

Do ponto de vista de tecnologia, após estudos realizados, foi definida a adoção das soluções do ecossistema *Spring* para a implementação do subsistema Servidor de Aplicações. Em resumo, o *framework Spring* é ferramenta utilizada para aumentar a produtividade na escrita de aplicações corporativas explorando conceitos como injeção de dependência e inversão de controle. Além da tecnologia *Spring* no desenvolvimento da lógica de negócio e acesso aos dados no servidor, também será adotado no subsistema Controlador soluções da suíte *Spring Cloud*[19], que traz funcionalidades para realizar a configuração, roteamento, distribuição de carga e alta disponibilidade para os serviços implementados.

1.6. Conclusão

Este capítulo apresentou um breve histórico e tecnologias atuais de plataformas de desenvolvimento de software *Web* do lado do cliente e do servidor. Foi descrito o histórico da evolução de modelos arquiteturais de Sistemas *Web* e também apresentado um estudo de caso por meio das soluções que foram desenvolvidas no Tribunal de Contas do Estado da Paraíba (TCE-PB) em parceira com o Laboratório de Aplicações de Vídeo Digital (LAVID) da Universidade Federal da Paraíba (UFPB). Tal solução adotou as tecnologias e modelos arquiteturais discutidos em um projeto real. A principal contribuição deste trabalho foi disseminar o histórico dos sistemas *Web* e elucidar as tecnologias e arquiteturas utilizadas hoje e tendências para o futuro.

References

[Benioff 2018] Benioff, M., “Welcome to Web 3.0: Now Your Other Computer is a Data Center | TechCrunch”, Disponível em: <http://techcrunch.com/2008/08/01/welcome-to-web-30- now-your-other-computer-is-a-data-center-2/>. Acesso em: 26/09/2018.

[Berners-lee 1996] Berners-lee, T. (1996) “WWW: past, present, and future”. Computer, v. 29, n. 10, p. 69–77.

[Bonér 2016] Bonér, J., (2016) “Reactive Microservices Architecture”, Sebastopol: Pearson Education, Inc,

[Burégio 2014] Burégio, V. A. A. (2014) “Social machines: a unified paradigm to describe, design and implement emerging social systems”, Doctor of Computer Science (PhD): Computer Science, Federal University of Pernambuco, Recife, Brasil.

[19] Projeto Spring Cloud. Disponível em: https://cloud.spring.io

[Burns 2018], Burns B., “Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services”, O'Reilly Media, Inc., 2018.

[Burns 2013] Chedeau, C. (2013) “React’s diff algorithm”. Disponível em: https://calendar.perfplanet.com/2013/diff, Acesso em: 01/8/2018.

[Deitel et al. 2010] Deitel, P. J. et al., “Internet & World Wide Web”, Boston: Pearson Education, Inc, 2012.

[Facebook 2018] Facebook (2018) “React - a javascript library for building user interfaces”. Disponível em: https://reactjs.org, Acesso em: 01/8/2018.

[Fields e Marc 2012] Fields, D. K., Mark A. (2002) “Web development with JSP. Greenwich”, Manning.

[Fox e Hao 2018] Fox, R. e Hao, W., “Internet Infrastructure”, New York: Taylor & Francis Group, LLC, 2018.

[Fowler et al. 2002] FOWLER, M. et al. (2002) “Patterns of Enterprise Application Architecture”, Addison Wesley.

[Gamma et al. 1996] Gamma, E., Helm, R., Johnson, R., Vlissides, J., (1996) “Design Patterns”, Boston: Pearson Education Corporate Sales Division.

[Garrett 2005] Garrett, J. J. (2005), “AJAX: A new approach to web applications | adaptive path”, Disponível em: http://adaptivepath.org/ideas/ajax-new-approach-web-applications/. Acesso em: 08/9/2018.

[Groef 2016] Groef, W. (2016) “Client- and Server-Side Security Technologies for JavaScript Web Applications”, Doctor of Engineering Science (PhD): Computer Science, Faculty of Engineering Science, Ku Leuven, Leuven.

[Hall 2017] Hall, G., “Adaptive Code”, Redmond: Online Training Solutions, Inc, 2017.

[Holdener 2008] Holdener T. (2008), “AJAX The definitive guide”. 1. ed.

[Hunter e Crawford 2001] Hunter, J. Crawford (2001), W. “Java Servlet Programming”. 2. ed.

[Java 2008] Java Servlet Specification (2018), Disponível em: https://javaee.github.io/servlet-spec/. Acesso em: 18/09/2018.

[Langr 2015] Langr, J., “Pragmatic Unit Testing in Java 8 with JUnit”, Raleigh: The Pragmatic Bookshelf, 2015.

[Mardan 2017] Mardan, A. “React Quickly: Painless web apps with React, JSX, Redux, and GraphQL.”, 2017

[Maximilien, Ranabahu e Gomadam 2008] Maximilien, E. M.; Ranabahu, A.; Gomadam, K., (2008) “An Online Platform for Web APIs and Service Mashups”, IEEE Internet Computing, v. 12, n. 5, p. 32–43.

[Mikowski e Powell 2013] Mikowski, M, e Powell, J, (2013) “Single page web applications: JavaScript end-to-end”. Manning Publications Co.

[Newman 2015] (2015) Newman, S., “Building Microservices”, Sebastopol: Pearson Education, Inc.

[Patterson e Fox 2012] Patterson, D., Fox, A. (2012) “Engineering Long-Lasting Software: An Agile Approach Using SaaS and Cloud Computing”, Strawberry Canyon LLC.

[Pivotal 2018a] Pivotal (2018), “Spring Framework”, Disponível em: https://spring.io/. Acesso em: 18/8/2018.

[Pivotal 2018b] Pivotal (2018), “Spring Initializr”, Disponível em: https://start.spring.io/. Acesso em: 18/8/2018.

[Postman 2018] Postman (2018), “Postdot Technologies”, Inc. Disponível em: https://www.getpostman.com/. Acesso em: 18/8/2018.

[Raible 2015] Raible, M. (2015) “Comparing Hot JavaScript Frameworks: AngularJS, Ember.js and React.js”, Available at: http://raibledesigns.com, 2015. Acesso em: 10/07/2018

[Raible 2018] Raible, M. (2018) “Front End Development for Back End Developers”, Available at: http://raibledesigns.com.

[RFC 7231 2018] RFC 7231 (2018) “Hypertext transfer protocol (http/1.1): Semantics and content”, Disponível em: https://tools.ietf.org/html/rfc7231, Acesso em: 26/09/2018.

[Sadalage e Fowler 2012] Sadalage, P. J.; Fowler, M. Nosql Distilled: A Brief Guide To The Emerging World Of Polyglot Persistence, 1. Ed.,Addison Wesley, 2013.

[Scott 2016] Scott JR, E. A. “SPA Design and Architecture Understanding single-page web applications. Manning Publications”. 1. Ed. 2016.

[Sommerville 2015] Sommerville, I., “Software Engineering”, 10. ed. London: Pearson Education, Inc, 2015.

[Thymeleaf 2018] Thymeleaf Template (2018), “The Thymeleaf Team”, Disponível em: https://www.thymeleaf.org/. Acesso em: 18/8/2018.

[Wall 2015] Wall, C. “Spring In Action”, 2. ed. Shelter Island: Manning Publication Co, 2015.

[Webber, Parastatidis e Robinson 2010] Webber, J., Parastatidis S., Robinson I. (2010) “REST in Practice: Hypermedia and Systems Architecture”, O'Reilly Media, Inc.

[Yu et al. 2008] Yu, J. et al., (2008) “Understanding Mashup Development”, IEEE Internet Computing, v. 12, n. 5, p. 44–52, set.

Chapter 2

Extração e Classificação de Dados Semânticos do Twitter

Clarissa Castellã Xavier[1] e Marlo Souza[2]

[1]Universidade Federal do Rio Grande do Sul (UFRGS)

[2]Universidade Federal da Bahia (UFBA)

clarissacastella@gmail.com, marlo@dcc.ufba.br

Abstract

Twitter is a social network and microblogging service in which registered users read and post messages called tweets. Tweets have a maximum of 280 characters and cover every conceivable subject, from simple activity updates and news coverage to opinions about arbitrary topics. In this way, Twitter emerges as a valuable data source to get information about what people think and feel about the most different subjects. In this context, this course presents different approaches for extracting and processing information from Twitter using Natural Language Processing (NLP) and Machine Learning techniques, examining tools and methods to collect and analyze semantic information from tweets.

Resumo

O Twitter é uma rede social e serviço de microblog onde usuários registrados leem e postam mensagens chamadas tweets. Os tweets podem ter no máximo 280 caracteres e abrangem todos os assuntos concebíveis, desde simples atualizações sobre atividades e notícias jornalísticas até opiniões sobre tópicos arbitrários. Desta forma, o Twitter surge como uma valiosa fonte de dados para obtenção de informações sobre o que as pessoas pensam e sentem sobre os mais diversos assuntos. Neste contexto, este curso apresenta diferentes abordagens para extração e processamento de informações do Twitter usando técnicas de Processamento de Linguagem Natural (PLN) e Aprendizado de Máquina, examinando ferramentas e métodos para coleta e análise de informações semânticas dos tweets.

2.1. Introdução

É clara a posição privilegiada que as mídias sociais têm na vida das pessoas, particularmente no contexto brasileiro. Portanto, entender o comportamento dos seus usuários

é um passo importante para qualquer organização. Toda esta informação é, no entanto, difícil de gerir, sendo que esta dificuldade deriva principalmente do fato de que estes dados se encontram em formato não-estruturado ou semiestruturado, como texto, imagens e páginas da web.

O Twitter [1] é um serviço de microblog e rede social lançado no final de 2006, no qual usuários postam mensagens de até 280 caracteres. No primeiro trimestre de 2018 o serviço teve uma média de 336 milhões usuários mensais ativos [2]. O Twitter é caracterizado como uma mídia informativa para o público brasileiro. De acordo com Recuero [da Cunha Recuero 2003], no contexto brasileiro, os usuários se envolvem com a plataforma mais como um meio de coletar e transmitir informações do que de se envolver em interações sociais, como conversas. De acordo com esse estudo, cerca de 62% dos *tweets* têm conteúdo informativo, enquanto cerca de 48% são de natureza conversacional, com 10% de textos com ambas as características. Em relação aos textos com perfil informativo, cerca de 25% possuem conteúdo opinativo, ou seja, o usuário expressa opiniões ou sentimentos. De fato, de acordo com um estudo recente sobre o perfil de acesso a notícias no mundo [Newman et al. 2016], 72% dos brasileiros residentes em áreas urbanas usam mídias sociais como fonte de notícias, dos quais 13% utilizam o Twitter como principal rede social para esse fim (15% na população abaixo de 35 anos).

É evidente que a quantidade de informações disponíveis cresceu significativamente devido aos novos meios de indexação de informações e novos recursos de distribuição. Devido à esta grande quantidade de dados, no entanto, não é fácil, nem mesmo gerenciável, encontrar e explorar informações que sejam estratégicas ou mesmo relevantes para um determinado indivíduo ou organização. A principal razão para essa dificuldade está na natureza não estruturada dos dados, já que seu tratamento computacional não é trivial. Neste contexto as tecnologias de informação e, em particular, a Análise de Textos têm um papel crucial em facilitar a obtenção e o processamento de dados relevantes na Web.

A Análise de Textos estuda como aplicar métodos de Inteligência Computacional para extrair automaticamente informações estruturadas de documentos não estruturados [Dale 2008]. Para Dale [Dale 2008], esta área visa desenvolver soluções para oito grandes problemas: Recuperação de Informação, Categorização e Agrupamento de Textos, Reconhecimento de Entidades, Co-Referência Nominal, Sumarização de Textos, Extração de Informação, Análise de Sentimentos (Polaridade) e Sistemas de Perguntas e respostas.

Neste trabalho vamos nos concentrar em dois problemas que acreditamos serem relevantes para lidar com os microtextos do Twitter: Classificação de Polaridade e Reconhecimento de Entidades.

A classificação de polaridade é uma tarefa bem conhecida de PLN. Dado um texto, seu objetivo é identificar se existe um conteúdo subjetivo no mesmo e obter a polaridade do sentimento transmitido pela informação, por exemplo, se o texto expressa um sentimento positivo, negativo ou neutro. Dadas as limitações de tamanho dos *tweets*, a classificação de sentimentos das mensagens do Twitter é normalmente executada no nível da sentença; no entanto, a linguagem informal e especializada faz com que esta seja uma

[1] https://twitter.com/

[2] https://www.statista.com/statistics/282087/number-ofmonthly-active-twitter-users/

tarefa singular.

A extração de entidades (EE) (também conhecida como Reconhecimento de Entidades Nomeadas (REN)) é uma sub-tarefa da área de extração de informações. Seu objetivo é localizar e classificar entidades nomeadas no texto. A EE costuma ser uma das primeiras etapas do *pipeline* de extração de informações. O estilo breve, informal e ruidoso do Twitter apresenta desafios. Estudaremos diferentes abordagens para detectar as entidades citadas em um *Tweet*.

Neste curso apresentamos diferentes técnicas de extração e classificação de dados semânticos para obtenção de informações do Twitter. Também falamos sobre as ferramentas que implementam essas técnicas aprendendo a usar esses recursos na prática. Aplicaremos estas habilidades em um estudos de caso, a fim de praticar importantes habilidades de manipulação de dados, aprendendo como diferentes formas de análise de dados podem ser usadas.

2.2. Extraindo dados do Twitter

O Twitter é uma ferramenta de *microblogging* lançada no final de 2006 na qual os usuários postam mensagens de até 280 caracteres chamadas *tweets*. Apenas usuários registrados podem postar *tweets*, mas aqueles que não estão registrados podem lê-los. O Twitter pode ser acessado através de sua interface no *website*, por meio do Serviço de Mensagens Curtas (SMS) ou através do aplicativo para dispositivos móveis ("app").

É possível coletar informações do Twitter utilizando a API pública [3], bem como utilizando aplicativos e bibliotecas alternativas. A seguir veremos o funcionamento da API do Twitter e da ferramenta Twint [4] e como utilizá-las para extrair dados.

2.2.1. API do Twitter

Os *tweets* estão gratuitamente disponíveis para desenvolvedores de *software* através de APIs públicas[5]. Essas APIs podem ser classificadas em dois tipos [Kumar et al. 2013]:

1. REST APIs: usam a estratégia *pull* para recuperação de dados, ou seja, para coletar informações o usuário deve explicitamente fazer uma solicitação.
2. *Streaming* APIs: usam a estratégia *push* para recuperação de dados, ou seja, depois que uma solicitação de informações é feita, a API fornece um fluxo contínuo de atualizações sem necessitar nenhuma outra solicitação.

As APIs do Twitter incluem uma ampla variedade de terminais que se dividem em cinco grupos principais: Contas e usuários, Mensagens diretas, Anúncios, Ferramentas de *publisher* e SDKs, *Tweets* e respostas [Twitter 2018]. Neste curso iremos nos concentrar no último grupo.

O grupo de APIs *Tweets* e Respostas torna os *tweets* e as respostas públicas disponíveis para os desenvolvedores e permite que estes também postem *tweets*. Os de-

[3]https://developer.twitter.com/
[4]https://github.com/twintproject/twint
[5]https://developer.twitter.com/

senvolvedores podem acessar os *tweets* pesquisando por palavras-chave específicas ou solicitando conteúdo de contas específicas [Twitter 2018].

2.2.1.1. Acessar a API

Para acessar a API do Twitter é necessário registrar uma aplicação. Por padrão os aplicativos podem acessar apenas informações públicas. Algumas aplicações, como por exemplo aquelas responsáveis pelo envio ou recebimento de mensagens diretas, exigem permissões adicionais [Twitter 2018].

Para obter este acesso é necessário obter uma chave e um *token* de acesso da API, um *token* e uma chave consumidora. Esse procedimento é feito em `http://dev.twitter.com/apps`.

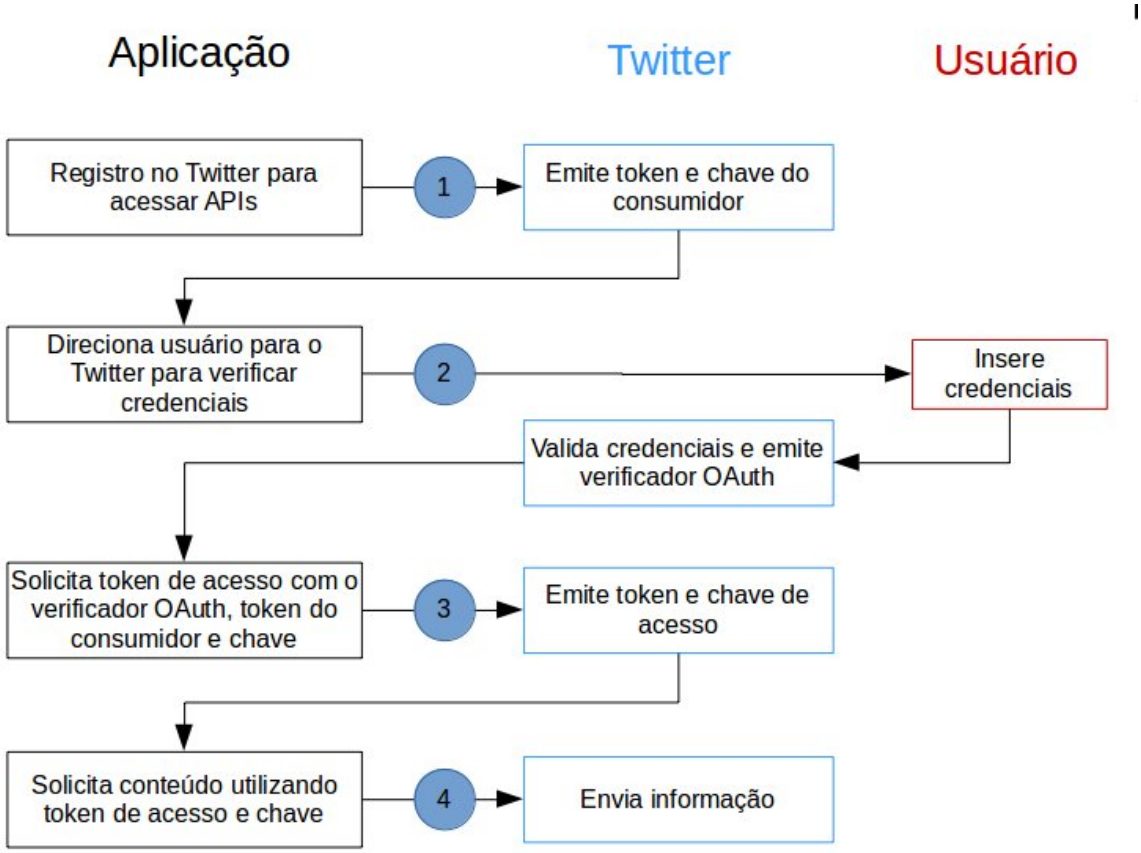


Figura. 2.1. Fluxo OAuth para obtenção do token de acesso à API do Twitter - adaptado de [Kumar et al. 2013]

A autenticação das requisições à API são realizadas utilizando autenticação aberta (OAuth). A Figura 1.1 sumariza os passos involvidos na autenticação. A API somente pode ser acessada por aplicações seguindo os passos abaixo [Kumar et al. 2013]:

1. As aplicações (consumidores) precisam se registrar no Twitter (em `http://dev.`

`twitter.com`). Neste processo a aplicação recebe uma chave e um *token* que o aplicativo deve usar para se autenticar.

2. A aplicação usa a chave e o *token* para criar um link exclusivo do Twitter para o qual o usuário é direcionado para autenticação. O usuário autoriza a aplicação autenticando-se no Twitter. O Twitter verifica a identidade do usuário e fornece um verificador OAuth (PIN).

3. O usuário informa o PIN para o aplicativo. O aplicativo usa o PIN para solicitar um *token* e uma chave de acesso exclusivos para o usuário.

4. Utilizando o *token* e a chave de acesso o aplicativo autentica o usuário no Twitter e chama a API em nome do usuário.

O *token* e a chave de acesso do usuário não podem ser alterados e podem ser armazenados em cache pelo aplicativo para futuras solicitações. Desta forma, este processo só precisa ser executado uma vez.

O código Python reproduzido na Listagem 1.1 implementa o fluxo OAuth para acesso à API do Twitter ilustrado na Figura 1.1. Como podemos ver, foi utilizada a biblioteca *oauth2*[6]. Ela lida com todas as etapas do protocolo OAuth 2.0 necessárias para fazer chamadas de API.

Listagem 2.1. Código em Python para acessar API do Twitter

```
import oauth2
def oauth_req(url, CHAVE_ACESSO, TOKEN_ACESSO, http_method=
   "GET", post_body="", http_headers=None):
   token = oauth2.Token(key=CHAVE_ACESSO, secret=
      TOKEN_ACESSO)
   consumo = oauth2.Consumer(key=CHAVE_CONSUMO, secret=
      TOKEN_CONSUMO)
   cliente = oauth2.Client(consumo, token)
   resp, conteudo = cliente.request( url, method=
      http_method, body=post_body, headers=http_headers )
   return conteudo
```

2.2.2. Coletar dados sem acessar a API

Existem outras abordagens para coletar dados do Twitter além da API. Uma opção é a ferramenta Twint implementada em Python. Ele utiliza os operadores de pesquisa do Twitter para capturar *tweets* de usuários específicos, relacionados a determinados tópicos, *hashtags* e tendências. Também faz consultas especiais ao Twitter, permitindo obter os seguidores de um usuário, *tweets* curtidos e seguidores.

A seguir veremos exemplos do uso da API e do Twint para extrair os seguintes tipos de informação:

[6] `https://pypi.org/project/python-oauth2/`

- Informações sobre um usuário
- Seguidores de um usuário
- Quem o usuário segue
- *Tweets* publicados
- Resultados de uma pesquisa

2.2.3. Exemplos Práticos

A seguir veremos exemplos de como coletar informações do Twitter utilizando a API pública e a biblioteca Python Twint. Abordaremos as seguintes tarefas: obter informações sobre um usuário, obter seguidores de um usuário, obter quem o usuário segue, obter *tweets* e obter resultados de uma pesquisa

2.2.3.1. Obter informações sobre um usuário

Via API

A API principal do Twitter é responsável pela manipulação e consulta dos dados. Qualquer método dessa API é precedido da URI *http://api.twitter.com/version/*, sendo que *version* é a versão da API (atualmente 1).

Cada usuário do Twitter está associado a um identificador, também chamado de nome de tela (*screen_name*), e um ID (*user_id*).

O método *users/show*[7] retorna as informações do perfil do usuário. Ela aceita um nome de usuário válido como parâmetro e retorna o perfil deste usuário no Twitter.

A Listagem 1.2 mostra o código Python para obter os dados de um perfil.

Listagem 2.2. Código em Python para chamada API *users/show*.

```
def info_usr(usuario):
   GET_USR_URL = "https://api.twitter.com/1.1/users/show.
      json?screen_name="+usuario
   req = oauth_req(GET_USR_URL,TOKEN_ACESSO,CHAVE_ACESSO)
   return req
```

Um exemplo de chamada do método criado no código 1.2 seria: `info_usr('twitterbrasil')` . Um objeto de usuário típico é formatado como na Listagem 1.3.

Listagem 2.3. Parte do objeto JSON retornado pelo método *info_usr('twitterbrasil')*.

```
{...
   "created_at": "Thu␣Mar␣10␣22:54:23␣+0000␣2011",
```

[7] https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-users-show.html

```
   "description": "Bem-vindos␣\u00e0␣conta␣oficial␣do␣
      Twitter␣Brasil!␣Precisa␣de␣ajuda?␣Acesse␣https://t.co
      /Nu5ZS0w4UD",
   "favourites_count": 1537,
   "followers_count": 2181337,
   "id": 263884490,
   "lang": "pt",
   "location": "Brasil",
   "name": "Twitter␣Brasil",
   "screen_name": "TwitterBrasil",
   "statuses_count": 7434,
   "time_zone": null,
   "translator_type": "regular",
   "url": "http://t.co/GuzHOnaY84",
   "verified": true
...}
```


A ferramenta Twint não disponibiliza esta funcionalidade.

2.2.3.2. Obter seguidores de um usuário

Via API

O método *followers/list*[8] retorna uma coleção de objetos de usuário contendo os usuários que seguem o perfil informado como parâmetro. Os resultados são fornecidos em grupos de 20 usuários e várias páginas de resultados podem ser navegadas usando o valor *next_cursor* em solicitações subsequentes.

A Listagem 1.4 mostra o código Python para obter seguidores de um perfil.

Listagem 2.4. Código em Python para chamada API *followers/list*.

```
def seguidores_usr(usuario):
   GET_USR_URL = "https://api.twitter.com/1.1/followers/
      list.json?cursor=-1&skip_status=true&
      include_user_entities=false&screen_name="+usuario
   req = oauth_req(GET_USR_URL,TOKEN_ACESSO,CHAVE_ACESSO)
   return req
```

Via Twint

O comando a seguir lista os usuários que seguem uma determinada conta:
`python3 Twint.py -u [usuário] -followers`.

A Figura 1.2 mostra a chamada e o retorno do comando
`python3 Twint.py -u twitterbrasil -followers`.

[8] https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-followers-list.html

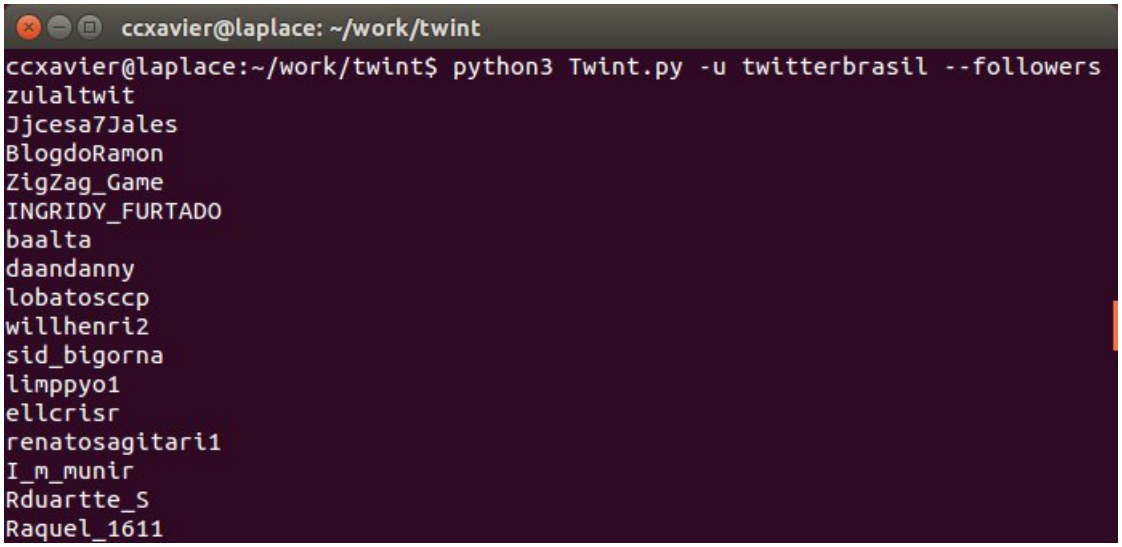


Figura. 2.2. Chamada e retorno do comando `$ python3 Twint.py -u twitterbrasil -followers`

2.2.3.3. Obter quem o usuário segue

Via API

O método *friends/list*[9] retorna uma coleção de objetos de usuário contendo os usuários seguidos pelo perfil informado como parâmetro. Os resultados são fornecidos em grupos de 20 usuários e várias páginas de resultados podem ser navegadas usando o valor *next_cursor* em solicitações subsequentes.

A Listagem 1.5 mostra o código Python para obter os amigos de um perfil.

Listagem 2.5. Código em Python para chamada API *friends/list*.

```
def seguidos_usr(usuario):
   GET_USR_URL = "https://api.twitter.com/1.1/friends/list.
      json?cursor=-1&skip_status=true&include_user_entities
      =false&screen_name="+usuario
   req = oauth_req(GET_USR_URL,TOKEN_ACESSO,CHAVE_ACESSO)
   return req
```

Via Twint

O comando `python3 Twint.py -u [usuário] -following` retorna a lista os usuários que seguem a conta do usuário informado como parâmetro. A Figura 1.3 mostra a chamada e o retorno do comando
`python3 Twint.py -u twitterbrasil -following`.

[9] `https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-friends-list`

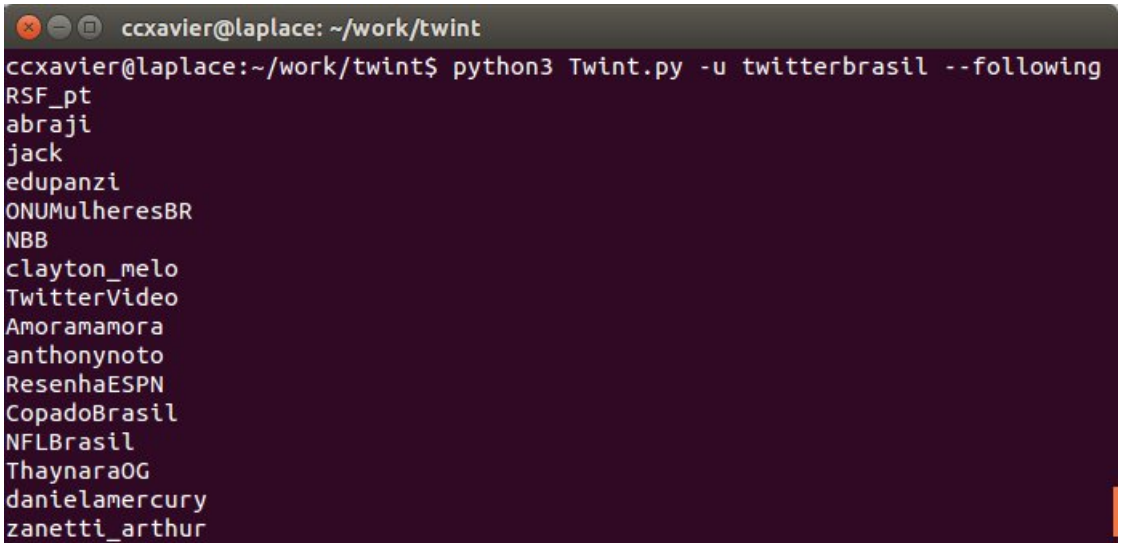


Figura. 2.3. Chamada e retorno do comando `$ python3 Twint.py -u twitterbrasil -following`

2.2.3.4. Obter *Tweets*

Via API

Um *Tweet* pode ser de autoria do usuário ou um tipo especial de *Tweet* chamado *Retweet*, criado quando um usuário reposta o *Tweet* elaborado por outro usuário. Os *tweets* de um usuário podem ser recuperados usando o REST e a API de *streaming*.

O método *statuses/usertimeline*[10] retorna uma coleção dos *tweets* mais recentes postados por um usuário utilizando REST API. As linhas de tempo pertencentes a usuários protegidos só podem ser solicitadas quando o usuário autenticado "possui" a linha do tempo ou é um seguidor aprovado.

Esse método retorna até 3.200 dos *tweets* mais recentes de um usuário. Os resultados são fornecidos em grupos e várias páginas de resultados podem ser navegadas usando o valor *next_cursor* em solicitações subsequentes. Cada página retorna até 200 *tweets*. O parâmetro *max id* é usado para paginar. Para recuperar a próxima página, usamos o ID do *Tweet* mais antigo na lista como o valor desse parâmetro na solicitação subsequente. Em seguida, a API recuperará apenas os *tweets* cujos IDs estão abaixo do valor fornecido.

A Listagem 1.6 mostra o código Python para obter o último *Tweet* de um perfil.

Listagem 2.6. Código em Python para chamada API *statuses/usertimeline*.

```
def tweets_usr(usuario):
   GET_USR_URL = "https://api.twitter.com/1.1/statuses/
      user_timeline.json?count=1&screen_name="+usuario
   req = oauth_req(GET_USR_URL,TOKEN_ACESSO,CHAVE_ACESSO)
   return req
```

[10] https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-statuses-user_timeline.html

O método *statuses/filter*[11] retorna uma coleção dos *tweets* que correspondam a um ou mais parâmetros de filtro usando *Streaming*. Ele permite que o cliente use uma única conexão, persistindo a conexão com a API.

A Listagem 1.7 mostra o código Python para criar uma solicitação POST para a API e buscar os resultados da pesquisa. O parâmetro é o termo que será sendo seguido. Por exemplo, se o parâmetro *termo* for *twitter*, o método irá imprimir os *tweets* contendo este termo sendo criados publicamente na plataforma. O código chama as bibliotecas auxiliares oauth2, json[12] e urllib2[13].

Listagem 2.7. Código em Python para chamada API *statuses/filter*.

```
def segue_tweets(termo):
   url = "https://stream.twitter.com/1.1/statuses/filter.
      json?track="+termo
   http_method="POST"
   post_body=""
   http_headers=None
   token = oauth2.Token(key=CHAVE_ACESSO, secret=
      TOKEN_ACESSO)
   consumo = oauth2.Consumer(key=CHAVE_CONSUMO, secret=
      TOKEN_CONSUMO)
   cliente = oauth2.Client(consumo, token)
   headers = {}
   req = oauth2.Request.from_consumer_and_token(
      cliente.consumer, token=cliente.token,
      http_method="POST", http_url=url)
   req.sign_request(cliente.method, cliente.consumer,
      cliente.token)
   headers.update(req.to_header())
   body = req.to_postdata()
   headers['Content-Type'] = 'application/x-www-form-
      urlencoded'
   req = urllib2.Request(url, body, headers=headers)
   try:
      f = urllib2.urlopen(req)
   except urllib2.HTTPError, e:
      data = e.fp.read(1024)
      raise Exception(e, data)
   for line in f:
      d = json.loads(line)
      try:
         print d["user"]["name"], d["text"]
      except:
```

[11] https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-statuses-user_timeline.html

[12] https://docs.python.org/2/library/json.html

[13] https://docs.python.org/2/library/urllib2.html

```
        print d.get("id")
```

Via Twint

O comando a seguir retorna todos os *Tweets* da *timeline* do usuário informado como parâmetro: `python3 Twint.py -u [usuário]` . A Figura 1.4 mostra a chamada e o retorno do comando `python3 Twint.py -u twitterbrasil`.

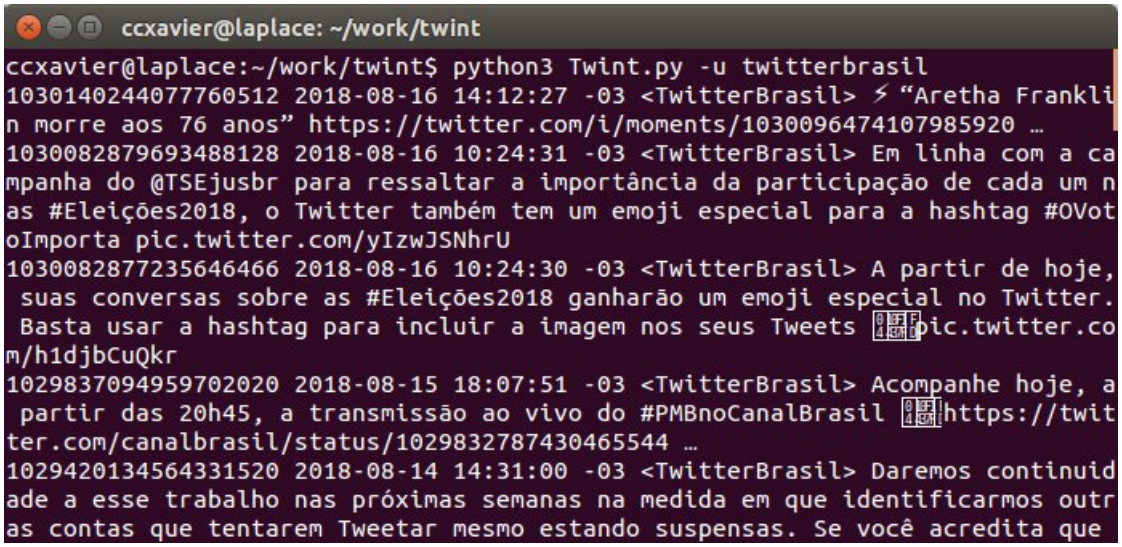


Figura. 2.4. Chamada e retorno do comando `$ python3 Twint.py -u twitterbrasil`

2.2.3.5. Obter os resultados de uma pesquisa

Via API

O método *search/tweets*[14] retorna os *tweets* que correspondem aos parâmetros da consulta. Estes parâmetros podem incluir palavras-chave, *hashtags*, frases, regiões, nomes de usuários ou *ids*.

A Listagem 1.8 mostra o código Python para obter *tweets* contendo a palavra chave enviada pelo parâmetro *busca*.

Listagem 2.8. Código em Python para chamada API *search/tweets*.

```
def busca_tweets(busca):
   GET_USR_URL = "https://api.twitter.com/1.1/search/tweets
      .json?q="+busca
   req = oauth_req(GET_USR_URL,TOKEN_ACESSO,CHAVE_ACESSO)
   return req
```

Via Twint

[14] `https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets.html`

O comando a seguir retorna todos os *tweets* contendo a palavra-chave informada como parâmetro. `python3 Twint.py -s [palavra-chave]` . A Figura 1.5 mostra o comando e o retorno do Twint para a busca da palavra-chave "eniac".

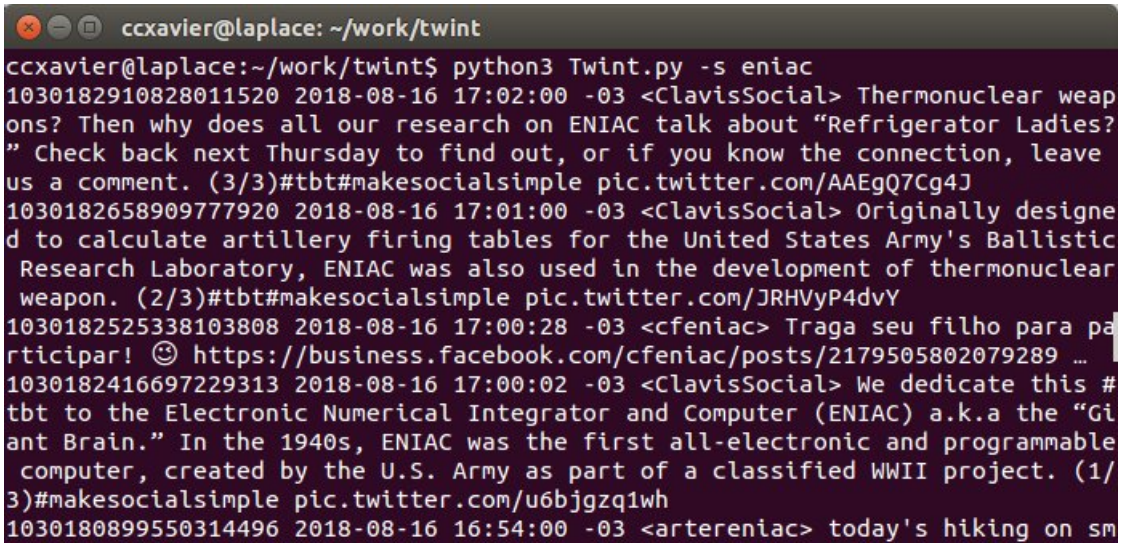


Figura. 2.5. Chamada e retorno do comando `$ python3 Twint.py -s eniac`

2.3. Análise de Polaridade

De acordo com Devika *et al.* [Devika et al. 2016] a Análise de Polaridade (incluindo a Análise de Sentimentos) é um tipo de classificação de texto que se baseia na orientação do sentimento expresso na opinião que ele denota. Em nossa opinião, esta tarefa é uma das aplicações mas interessantes de Análise de Texto.

A principal tarefa da Análise de Sentimento no nível da sentença é encontrar sua polaridade. Por exemplo, um sentimento positivo é atribuído quando o *Tweet* analisado indica felicidade, excitação ou simpatia. Um sentimento negativo está relacionado a raiva, tristeza, situações tristes ou difíceis. Quando nenhuma emoção é sugerida, o texto pode ser classificado como neutro [Lalrempuii and Mittal 2016].

Por causa da linguagem informal e específica, a análise da polaridade das mensagens do Twitter é uma tarefa singular. A classificação das postagens de acordo com o sentimento expresso tem várias aplicações na ciência política, ciências sociais, pesquisa de mercado, etc [Martínez-Cámara et al. 2014, Mejova et al. 2015, Nakov et al. 2016]. Pak e Paroubek [Pak and Paroubek 2010] apresentam razões para o uso do Twitter como um corpus para análise de sentimentos e mineração de opinião, apresentando as seguintes razões para isso:

- As plataformas de *microblogging* são utilizadas por muitas pessoas para expressar seu ponto de vista sobre diferentes tópicos, sendo assim, uma fonte valiosa de opiniões.
- O Twitter contém uma quantidade imensa de postagens de texto que cresce a cada dia. Desta forma, o corpus coletado pode ser bastante grande.

Table 2.1. Exemplo de classificação de polaridade de *tweets*

Usuário	@ivetesangalo
Tweet	Estou muito feliz e muito agradecida por todo esse amor ♥♥♥
Polaridade	Positiva
Usuário	@EPTC_POA
Tweet	Neste momento, bem complicado o acesso a Rodoviária no Largo Vespasiano Julio Veppo, pelo Túnel da Conceição.
Polaridade	Negativa

- A audiência do Twitter varia de usuários comuns à celebridades, representantes de empresas, políticos e até mesmo presidentes de países. Portanto, é possível coletar mensagens de texto de usuários de diferentes grupos sociais e de interesses.
- A audiência do Twitter é representada por usuários de vários países. É possível coletar dados em diferentes idiomas.

Para exemplificar como funciona a classificação de polaridade de *tweets*, a Tabela 1.1 apresenta exemplos de classificação positiva e negativa.

2.3.1. Abordagens

Existem diferentes abordagens para a realização desta tarefa. As principais são:

2.3.1.1. Abordagem baseada em regras

Nesta abordagem são definidas regras ou padrões para compreender as opiniões sobre o texto. Funcionamento [Devika et al. 2016]:

- Para cada sentença:
  Faz a tokenização.
  Inicia com pontuação neutra (0).
  Para cada padrão encontrado, aplica uma classificação (pontuação).
- A sentença é considerada positiva se a pontuação de polaridade final for maior que zero ou negativa se a pontuação geral for menor que zero.

2.3.1.2. Abordagem baseada em léxico

Técnicas baseadas em léxico partem do pressuposto de que a polaridade ou o sentimento expresso por uma frase ou documento pode ser identificada pelas polaridades

das unidades lexicais que a compõem. Segundo Gómez Molina [Gómez Molina 2004], a unidade lexical (item léxico) é a unidade de significado no léxico mental, que serve como um veículo para a cultura do idioma e pode ser composta de uma ou mais palavras (cabeça, guarda-chuva, dinheiro sujo, etc.).

Para realizar a análise de sentimentos via métodos léxicos, é necessário, inicialmente, efetuar o pré-processamento dos textos utilizados. Este pré-processamento visa reduzir o volume dos dados, antes de iniciar a execução das etapas de análise. Após o pré-processamento, o método de análise de sentimentos precisa representar cada texto e suas palavras. Nessas representações, cada palavra é representada por meio de um peso. Esse peso pode ser simplesmente sua frequência, ou, por exemplo, o valor TF-IDF. Em vista da métrica utiliza para definição do peso das palavras analisadas, o peso da palavra positiva ou negativa, aumenta proporcionalmente à medida que aumenta o número de ocorrências dela no documento. Esse valor pode ser equilibrado pelo inverso da frequência definida para cada termo. Com isto, é possível distinguir o fato de alguns termos serem geralmente mais comuns que outros no âmbito positivo ou negativo [de Souza et al. 2017].

2.3.1.3. Abordagem baseada em aprendizado de máquina

As estratégias utilizadas nesta abordagem funcionam através da aplicação de um algoritmo de treino. Este algoritmo primeiro treina a si mesmo utilizando um conjunto de dados de treinamento e somente depois realiza o aprendizado em si. Sendo assim, as técnicas de aprendizado de máquina primeiro treinam o algoritmo com algumas entradas específicas, para depois trabalhar com novos dados desconhecidos [Devika et al. 2016].

Nós iremos focar nesta abordagem para realizar a classificação da polaridade de *tweets*. Desta forma, avaliaremos os seguintes classificadores:

- Máxima Entropia: de acordo com [Ratnaparkhi 1997] "*segundo o princípio da máxima entropia, a distribuição correta é aquela que maximiza a entropia ou sujeito incerto às restrições que representam a "evidência", isto é, os fatos conhecidos pelo experimentador*". Este princípio é frequentemente invocado na criação de modelos, assumindo que os dados observados em si são a informação testável.

- Naive Bayes: é um modelo de probabilidade que assume a independência entre os recursos de entrada, baseado na aplicação do teorema de Bayes. Ele assume que a presença de uma *feature* específica não se relaciona com a existência de qualquer outra *feature* [John and Langley 1995].

- *Support Vector Machine*[15] (SVM): um algoritmo de classificação supervisionado que foi usado extensivamente e com sucesso para a tarefa de classificação de texto [Pawar and Gawande 2012]. Dado um conjunto de exemplos de treino, cada um marcado como pertencente a uma categoria, o algoritmo de treinamento SVM cria um modelo que atribui novos exemplos a uma categoria. Um modelo de SVM é uma representação dos exemplos como pontos em um hiperplano. O que o algoritmo faz

[15]Máquina de vetores de suporte

é encontrar uma linha de separação (hiperplano) entre dados de duas classes (Representado na Figura 1.6. Essa linha busca maximizar a distância entre os pontos mais próximos em relação a cada uma das classes [D'Andrea et al. 2015].

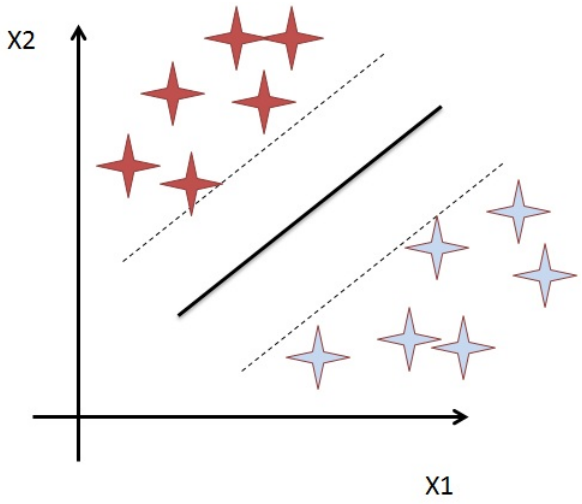


Figura. 2.6. Representação gráfica de um hiperplano (linha central).

2.3.2. Exemplo Prático

Buscando aplicar os conceitos vistos anteriormente na prática, vamos analisar e editar um classificador de polaridade implementado em Python usando a biblioteca NLTK [16].

A biblioteca Python NLTK implementa ferramentas para processamento de linguagem natural. Ela fornece interfaces para corpora e recursos lexicais como a WordNet, juntamente com um conjunto de bibliotecas de processamento de texto para classificação, tokenização, *stemming*, *tagging*, *parsing* e análise semântica [Bird et al. 2009].

Classificação é a tarefa de aplicar o rótulo correto para uma determinada entrada. Em tarefas de classificação básica, cada entrada é considerada isoladamente de todas as outras entradas e o conjunto de rótulos é definido antecipadamente. Neste exemplo iremos considerar a análise de polaridade como uma tarefa de classificação onde os *tweets* podem ser classificados como Positivos, Negativos ou Neutros. Mais especificamente, trabalharemos com "classificação supervisionada". Um classificador é denominado supervisionado quando utiliza um corpora de treinamento. A Figura 1.7 representa o seu *framework* que é dividido em duas etapas definidas: treino e aprendizado, conforme detalhamento abaixo.

A Treinamento: um extrator de *features* converte cada valor de entrada em um *feature set*. Pares de *feature sets* e rótulos alimentam o algoritmo de aprendizado de máquina para gerar um modelo.

B Previsão: o mesmo extrator de *features* é usado para converter novas entradas (não presentes nos *feature sets*. Os *feature sets* são alimentados no modelo, o que gera os novos rótulos (classificação).

[16] `http://www.nltk.org/`

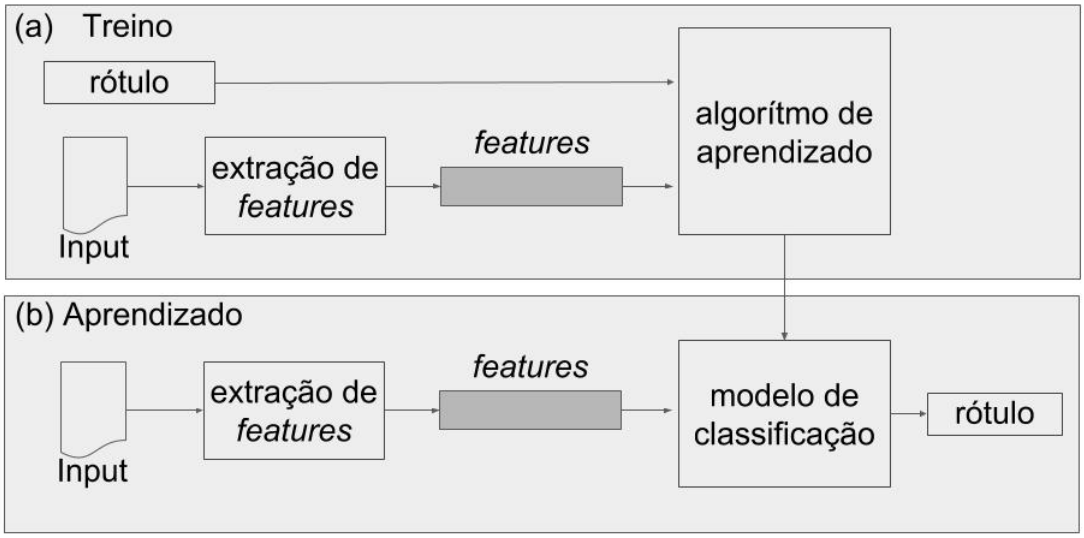


Figura. 2.7. *Framework* de um classificador supervisionado - baseado em [Bird et al. 2009].

Os classificadores de aprendizado de máquina geralmente exigem que a entrada de texto seja representada como um vetor de comprimento fixo. Provavelmente o mais comum destes vetores seja o *bag-of-words* devido à sua simplicidade, eficiência e, muitas vezes, precisão [Le and Mikolov 2014]. Neste modelo, o texto é representado como um saco (*bag*) ou conjunto de suas palavras, desconsiderando a gramática e até a ordem das palavras, mas mantendo a multiplicidade. O exemplo de um vetor *bag-of-words* para a frase `Ana gosta de Zeca, Zeca gosta de Lia e Lia não gosta de ninguém` seria `["Ana":1, "gosta":3, "de":3, "Zeca":2, "Lia":2, "e":1, "não":1, "ninguém":1]`

Nossos experimentos usam este modelo como *feature set*.

2.3.2.1. Extração de *Features*

Na listagem 1.9 podemos ver o código Python que implementa a extração das *features*, ou seja, criação do vetor de entrada no modelo *bag-of-words* que será utilizado tanto pelo algoritmo de aprendizado, como pelo modelo de classificação, conforme ilustrado na Figura 1.7

A função `divide()` recebe como entrada um vetor de frases e retorna um vetor de palavras para cada frase no vetor de entrada. Por exemplo, considerando um vetor de entrada `["Trânsito acentuado nos dois sentidos da Av. Carlos Gomes x Campos Sales."]` a função retorna o vetor `['trânsito', 'acentuado', 'nos', 'dois', 'sentidos', 'da', 'av.', 'carlos','gomes', 'x', 'campos', 'sales.']`

A função descrita na Listagem 1.9 cria o vetor *bag-of-words*

Listagem 2.9. Código em Python implementando extração de features

```
def divide(dados):
   dados_new = []
   for palavra in dados:
      palavra_filter = [i.lower() for i in palavra.split()]
      dados_new.append(palavra_filter)
   return dados_new[0]

def bag_of_words(palavras):
   return dict([(palavra, palavras.count(palavra)) for
      palavra in palavras])
```

2.3.2.2. Treino

Na listagem 1.10 podemos ver o código Python que implementa o treinamento de três tipos de classificadores supervisionados utilizando a biblioteca NLTK: SVM, Naive Bayes e Máxima Entropia.

Os dados de treino são *tweets* da conta @EPTC_POA[17] rotulados manualmente como 'pos' (positivo), 'neg' (negativo) e 'neu' (neutro). Estes dados estão disponíveis para *download* em `https://github.com/clarissacastella/twittercourse/`.

A função utiliza as *features bag-of-words* geradas pelas funções da listagem 1.9 e retorna um modelo treinado para cada um dos três algoritmos de classificação.

Listagem 2.10. Código em Python implementando treinamento dos classificadores.

```
def treina_classificadores():
   posdados = []
   with open('./dadostreino/train_EPTC_POA_v3nbal_1.data',
      'rb') as myfile:
      reader = csv.reader(myfile, delimiter=',')
      for val in reader:
         posdados.append(val[0])
   negdados = []
   with open('./dadostreino/train_EPTC_POA_v3nbal_0.data',
      'rb') as myfile:
      reader = csv.reader(myfile, delimiter=',')
      for val in reader:
         negdados.append(val[0])
   neudados = []
   with open('./dadostreino/train_EPTC_POA_v3nbal_2.data',
      'rb') as myfile:
      reader = csv.reader(myfile, delimiter=',')
      for val in reader:
         neudados.append(val[0])
```

[17] `https://twitter.com/EPTC_POA`

```
   negfeats = [(bag_of_words(f), 'neg') for f in divide(
      negdados)]
   posfeats = [(bag_of_words(f), 'pos') for f in divide(
      posdados)]
   neufeats = [(bag_of_words(f), 'neu') for f in divide(
      neudados)]
   treino = negfeats + posfeats+ neufeats
   #'Maximum Entropy'
   classificadorME = MaxentClassifier.train(treino, 'GIS',
      trace=0, encoding=None, labels=None,
      gaussian_prior_sigma=0, max_iter = 1)
   #SVM
   classificadorSVM = SklearnClassifier(LinearSVC(), sparse
      =False)
   classificadorSVM.train(treino)
   # Naive Bayes
   classificadorNB = NaiveBayesClassifier.train(treino)
   return ([classificadorME,classificadorSVM,
      classificadorNB])
```

2.3.2.3. Classificação

Os três modelos criados pela função `treina_classificadores` são utilizados no código da Listagem 1.11 que apresenta a função que classifica uma sentença de entrada (no caso um *Tweet*), atuando conforme os passos apresentados na figura 1.7.

Listagem 2.11. Código em Python implementando um classificador de polaridade.

```
def classifica(sentencas, classificadores):
   ret = []
   for s in sentencas:
      c = divide([s])
      feats= bag_of_words(c[0])
      classificacao = []
      classificacao.append(classificadores[1].classify(
         feats))
      classificacao.append(classificadores[2].classify(
         feats))
      classificacao.append(classificadores[0].classify(
         feats))
      ret.append(classificacao)
   return ret
```

Por exemplo, quando a função `classifica` recebe como entrada um vetor `['Fluxo muito congestionado na Osvaldo Aranha no acesso para o Túnel. Agora, tá chovendo também. Então, atenção!',`

`'Não use o celular ao volante, 80% da sua atenção é desviada']` com duas sentenças, ela retorna o vetor de resultados `[['neg','pos', 'neg'], ['neu', 'neu', 'neu']]` . Desta forma, o primeiro *Tweet* foi classificado com polaridade negativa pelo classificador Máxima Entropia, positiva pelo SVM e negativa pelo Naive Bayes. Ou seja, os classificadores Máxima Entropia e Naive Bayes classificaram corretamente a sentença e o classificador SVM não. Já o segundo *Tweet* foi classificado corretamente como neutro pelos três classificadores.

Para maiores detalhes, a implementação completa do classificador de polaridade encontra-se em `https://github.com/clarissacastella/twittercourse`, arquivo `polaridade.py`.

2.4. Extração de Entidades

A Extração de Entidades (EE), também conhecida como Reconhecimento de Entidades Nomeadas(REN), é uma tarefa de Extração de Informações que consiste na identificação de referências feitas a determinadas entidades e sua classificação. Essa tarefa é frequentemente uma das primeiras etapas na análise semântica de um texto, posto que as entidades mencionadas transmitem bastante informação sobre o conteúdo do texto em si.

O formato reduzido das mensagens do Twitter impõe algumas dificuldades em seu processamento. Por esse motivo, os métodos de EE que são aplicados em outros tipos de texto podem não funcionar muito bem nos *tweets*. Por exemplo, Locke [Locke 2009] mostra que um anotador de última geração como o *Stanford NER*[18], tem seu desempenho bastante reduzido quando aplicado a textos do Twitter.

O tamanho das mensagens limitado à 280 caracteres apresenta uma dificuldade para a tarefa de EE. Textos curtos oferecem pouca informação contextual para a identificação de entidades. Além disso, recursos importantes para identificar nomes próprios, como capitalização, não são utilizados com rigor nos textos do Twitter. É muito comum a sub-capitalização das palavras, ou seja, nenhuma palavra capitalizada no texto, bem como a supercapitalização, onde várias ou todas as palavras estão capitalizadas, geralmente com a intenção de denotar intensidade ao conteúdo do texto. Assim, geralmente, os métodos de EE devem ser adaptados a esse contexto.

Para demonstrar como a EE de *tweets* funciona, o exemplo abaixo mostra quais entidades podem ser extraídas do Tweet da @EPTC_POA listado na Tabela 1.1 "*Neste momento, bem complicado o acesso a Rodoviária no Largo Vespasiano Julio Veppo, pelo Túnel da Conceição*" usando o formato `[entidade (classificação)]`.

```
[Rodoviária (LOC)] [Largo Vespasiano Julio Veppo (LOC)] [Tú
   nel da Conceição (LOC)]
```

Os sistemas de EE mais recentes têm utilizado técnicas de aprendizado de máquina, de modo que esta se tornou a principal técnica de abordagem, em contraste com os sistemas mais antigos que utilizavam regras manualmente codificadas e heurísticas para efetuar esta tarefa. Desta forma, EE vem sendo tratada como uma tarefa de rotulagem sequencial [Souza 2012].

[18] `https://nlp.stanford.edu/software/CRF-NER.shtml`

As ferramentas de EE a partir do Twitter, em geral, usam técnicas do estado-da-arte associadas a estratégias de otimização, como heurísticas de pré-processamento [Ritter et al. 2011] ou métodos semi-supervisionados de aprendizado [Liu 2010] que exploram a grande quantidade de dados não anotados disponíveis.

2.4.1. Exemplo Prático

Buscando aplicar os conceitos vistos na prática, vamos explorar uma solução de aprendizado de máquina que extrai entidades de implementada em Python. Esta ferramenta utiliza a biblioteca SpaCy [19], uma biblioteca de código aberto para o processamento avançado de linguagem natural. A SpaCy é uma solução baseada em *deep learning* que interage com as bibliotecas do ecossistema de Inteligência Artificial do Python. A biblioteca usa um sistema de EE estatístico, que atribui rótulos a extensões contíguas de *tokens*[20].

2.4.1.1. Treino

Os modelos gerados pela biblioteca são estatísticos e cada decisão, como por exemplo se uma palavra é uma entidade, é uma previsão. Essa previsão é baseada nos exemplos que o modelo viu durante o treinamento. Para treinar um modelo é necessário um conjunto de dados de treinamento que consistem de exemplos de texto e de rótulos que o modelo deve prever. Como treinar um modelo não é simplesmente memorizar exemplos, mas gerar um padrão que possa ser generalizado em outros exemplos, os dados de treinamento devem ser representativos dos dados que serão processados.

Na listagem 1.12 podemos ver o código Python que implementa transforma os dados de treino contidos no arquivo `./data/dadosTreinoLoc.txt` para o formato de entrada do modelo. Desta forma, a função `cria_dados_treino` transforma o conteúdo de um arquivo texto contendo uma lista de *tweets* e os locais citados neles em um vetor contendo uma tupla. A Tabela 1.2 apresenta um exemplo do funcionamento da função. A linha Entrada consiste em um trecho do arquivo de entrada e a linha Saída representa a tupla correspondente ao trecho dentro do vetor de saída.

Listagem 2.12. Código em Python formatando os dados de treino.

```
def cria_dados_treino(arq='./data/dadosTreinoLoc.txt'):
      dados_treino = []
      fin = open(arq, 'rb')
      n=0
      post = u''
      for val in fin:
            d = {}
            n = n + 1
            if (n % 2 == 1) :
```

[19] https://spacy.io/

[20] Os *tokens* usualmente correspondem as unidades lexicais de um texto, i.e. aproximadamente às palavras do texto. Por exemplo os tokens da frase "*Complicado o trânsito*" são `[complicado] [o] [trânsito]`

```
                post = val.replace('\n','')
          else :
                d['entidades']=ast.literal_eval(val.
                   replace('\n',''))
                dados_treino.append((post, d))
    fin.close()
    return dados_treino
```

Table 2.2. Exemplo do funcionamento da função `cria_dados_treino`.

Entrada	... 17h53 - ATENÇÃO! Acidente entre carro e moto na R. Dom Pedro II, sentido sul/norte, próximo a R. Barão do Cotegipe. [(48,63,'LOC'), (94,114,'LOC')]
Saída	{ [...('17h53 - ATENÇÃO! Acidente entre carro e moto na R. Dom Pedro II, sentido sul/norte, próximo a R. Barão do Cotegipe.', ({'entidades': { [(48,63,'LOC'), (94,114,'LOC')] } }] }

Na listagem 1.13 podemos ver o código Python que implementa a criação de um novo modelo SpaCy para realizar EE e o treina com os dados importados pela função `cria_dados_treino` (listagem 1.12) seguindo os seguintes passos:

- Cria modelo em branco.
- Adiciona ao modelo o *pipeline* NER, responsável por executar a EE.
- Lê os dados de treino.
- Treina o modelo.

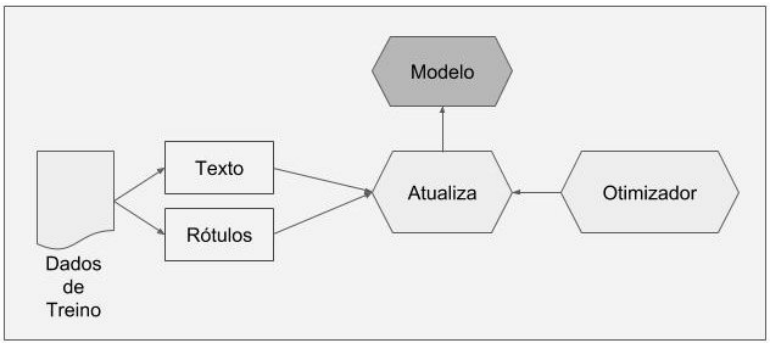


Figura. 2.8. Fluxo de treino do EE - adaptado de [SpaCy 2018]

O treino é feito em várias iterações, como pode ser visto na Imagem 1.8. Em cada iteração os dados de treinamento são embaralhados para que o modelo não faça generalizações com base na ordem dos exemplos. Também é definida uma taxa de desistência (*loss*) para "descartar" recursos e representações individuais aleatoriamente. Por exemplo, um abandono de 0,25 significa que cada recurso ou representação interna tem uma probabilidade de 1/4 de ser descartado.

Listagem 2.13. Código que treina um novo modelo de EE.

```
dir = "./modeloEE" # cria modelo em branco
dir = Path(dir)
nlp = spacy.blank('pt')
dir.mkdir()
ner = nlp.create_pipe('ner')
nlp.add_pipe(ner, last=True)

import dados_treino as tr
dados_treino = tr.cria_dados_treino('./data/testP.txt') #
   dados de treino

for _, annotations in dados_treino:
      for ent in annotations.get('entities'):
            ner.add_label(ent[2])

# treina EE
optimizer = nlp.begin_training()
n_iter = 2 #nro iteracoes
for itn in range(n_iter):
   c = 0
   losses = {}
   random.shuffle(dados_treino) #embaralha para nao viciar
   for text, annotations in dados_treino:
      c = c + 1
      nlp.update(
         [unicode(text)], # batch de textos
         [annotations], # batch de anotacoes
            correspondentes aos textos
         drop=0.25,
         sgd=optimizer,
         losses=losses)
   print(losses)

nlp.to_disk(dir) #salva modelo
```

2.4.1.2. Aprendizado

Na listagem 1.14 podemos ver o código Python que implementa a EE a partir do modelo SpaCy gerado anteriormente seguindo os seguintes passos:

- Carrega modelo.
- Lê arquivo com posts.
- Para cada post: aplica o modelo de EE.

Listagem 2.14. Código que implementa o EE usando o modelo criado.

```
modelo_dir = Path ("./modeloEE" )
modelo = spacy.load(modelo_dir) # roda modelo
fin = open('./data/postsP.txt', 'rb')
for text in fin:
      doc_model = modelo(unicode(text))
      locs = []
      for ent in doc_model.ents:
         aux = (ent.start_char, ent.end_char, str(ent.
            label_))
         locs.append(text[ent.start_char:ent.end_char])
      print (text,locs)
```

A Figura 1.9 mostra a chamada e o retorno do programa correspondente ao código listado em 1.14.

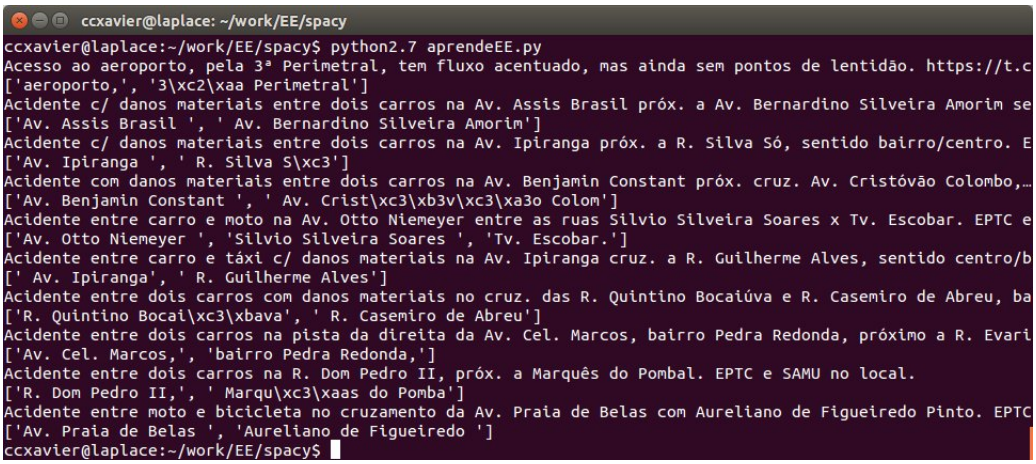


Figura. 2.9. Chamada e retorno do comando `$ python2.7 aprendeEE.py`

Para maiores detalhes, a implementação do Extrator de Entidades encontra-se em `https://github.com/clarissacastella/twittercourse`, arquivos `treinaEE.py` e `aprendeEE.py`.

2.5. Dificuldades no processamento de textos do Twitter

Mensagens do Twitter possuem tamanho limitado, contendo até 280 caracteres. Por tal motivo, podemos caracterizar os *tweets* como uma forma de texto curto, similar a textos de outras mídias como mensagens de texto curta (SMS) ou mensagens de telegrama.

Textos curtos possuem características específicas como uma maneira informal de escrita, utilizando-se de abreviações, gírias, etc. Tais características dificultam o seu processamento ou até mesmo seu entendimento por leitores humanos. Além disso, textos curtos não oferecem informações contextuais importantes que são comumente exploradas por diversas ferramentas de Análise de Texto.

Além disso, por ser um serviço na Internet, funcionando em tempo real ou seja *on the fly*, dados provenientes do Twitter apresentam algumas características próprias, como grande volume, contextos incertos e distribuídos em mensagens diferentes (estrutura de fluxo ou *stream*).

Diferentes estratégias foram experimentadas para lidar com as particularidades de textos do Twitter, como:

- Normalização de texto: utilização técnicas de análise automática para converter o texto ruidoso do Twitter em uma variante mais formal da língua, como corrigir erros de grafia;

- Agrupamento de *tweets*: agrupamento automático de *tweets* tratando de um mesmo assunto, ou que estejam contextualmente ligados, de forma a criar textos capazes de fornecer informação contextual mais relevante às ferramentas de Análise de Texto;

- Séries temporais: como os *tweets* possuem uma estrutura de fluxo, i.e. estão deslocados no tempo, alguns métodos para processamento de *tweets* utilizam modelagens baseadas em séries temporais tentando capturar organicamente o contexto dos *tweets* pela sua informação temporal;

- Ferramentas específicas: alguns métodos e ferramentas parecem precisar de estratégias específicas para textos provenientes de *tweets*, que levam em consideração a pobreza contextual e as variações lexicais próprias desses textos.

Cada uma de tais estratégias possui suas vantagens e desvantagens e não existe uma solução estabelecida para tal problema. Por exemplo, a normalização textual permite a utilização de técnicas e ferramentas já bem desenvolvidas que são especializadas no processamento de texto mais formal, como texto jornalístico, entretanto métodos de normalização são ainda bastante imprecisos e não superam as dificuldades de falta de contexto de *tweets*.

2.6. Conclusão

O Twitter é uma fonte valiosa de informações sobre o que as pessoas pensam e sentem sobre os mais variados assuntos. Por esse motivo academia, empresas e organizações de mídia estão procurando maneiras de extrair conhecimento desta ferramenta.

Este minicurso analisou diferentes abordagens para coletar informações dos *tweets*. Primeiro vimos como coletar informações do Twitter utilizando a API pública, bem como a biblioteca Python Twint. Para realizar a extração e classificação semântica desses dados, usamos técnicas de Processamento da Linguagem Natural e Aprendizado de Máquina. Focamos nosso estudo em duas tarefas de análise de texto: Análise de Polaridade e Extração de Entidades.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Referências

[Bird et al. 2009] Bird, S., Klein, E., and Loper, E. (2009). *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

[da Cunha Recuero 2003] da Cunha Recuero, R. (2003). Weblogs, webrings e comunidades virtuais. *Revista 404notfound-Revista Eletrônica do Grupo Ciberpesquisa*, 31.

[Dale 2008] Dale, R. (2008). Where is text analytics today. Presentation to Sydney Data Miners Meeting.

[D'Andrea et al. 2015] D'Andrea, E., Ducange, P., Lazzerini, B., and Marcelloni, F. (2015). Real-time detection of traffic from twitter stream analysis. *IEEE transactions on intelligent transportation systems*, 16(4):2269–2283.

[de Souza et al. 2017] de Souza, K. F., Pereira, M. H. R., and Dalip, D. H. (2017). Unilex: Método léxico para análise de sentimentos textuais sobre conteúdo de tweets em português brasileiro. *Abakós*, 5(2):79–96.

[Devika et al. 2016] Devika, M., Sunitha, C., and Ganesh, A. (2016). Sentiment analysis: A comparative study on different approaches. *Procedia Computer Science*, 87:44–49.

[Gómez Molina 2004] Gómez Molina, J. R. (2004). La subcompetencia léxico-semántica. *Vademécum para la formación de profesores. Enseñar español como segunda lengua (L2)/lengua extranjera (LE), Madrid, SGEL*, pages 491–510.

[John and Langley 1995] John, G. H. and Langley, P. (1995). Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 338–345. Morgan Kaufmann Publishers Inc.

[Kumar et al. 2013] Kumar, S., Morstatter, F., and Liu, H. (2013). *Twitter Data Analytics*. Springer Publishing Company, Incorporated.

[Lalrempuii and Mittal 2016] Lalrempuii, C. and Mittal, N. (2016). Sentiment classification of crisis related tweets using segmentation. In *Proceedings of the International Conference on Informatics and Analytics*, page 89. ACM.

[Le and Mikolov 2014] Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196.

[Liu 2010] Liu, B. (2010). Sentiment analysis and subjectivity. *Handbook of natural language processing*, 2:627–666.

[Locke 2009] Locke, B. W. (2009). Named entity recognition: Adapting to microblogging. Technical report, University of Colorado.

[Martínez-Cámara et al. 2014] Martínez-Cámara, E., Martín-Valdivia, M. T., Urena-López, L. A., and Montejo-Ráez, A. R. (2014). Sentiment analysis in twitter. *Natural Language Engineering*, 20(1):1–28.

[Mejova et al. 2015] Mejova, Y., Weber, I., and Macy, M. W. (2015). *Twitter: a digital socioscope*. Cambridge University Press.

[Nakov et al. 2016] Nakov, P., Ritter, A., Rosenthal, S., Sebastiani, F., and Stoyanov, V. (2016). Semeval-2016 task 4: Sentiment analysis in twitter. In *Proceedings of the 10th international workshop on semantic evaluation (semeval-2016)*, pages 1–18.

[Newman et al. 2016] Newman, N., Fletcher, R., Levy, D. A. L., and Nielsen, R. K. (2016). Reuters institute digital news report 2016. Technical report, Reuters Institute for the Study of Journalism, University of Oxford.

[Pak and Paroubek 2010] Pak, A. and Paroubek, P. (2010). Twitter as a corpus for sentiment analysis and opinion mining. In *LREc*, volume 10, pages 1320–1326.

[Pawar and Gawande 2012] Pawar, P. Y. and Gawande, S. (2012). A comparative study on different types of approaches to text categorization. *International Journal of Machine Learning and Computing*, 2(4):423.

[Ratnaparkhi 1997] Ratnaparkhi, A. (1997). A simple introduction to maximum entropy models for natural language processing. *IRCS Technical Reports Series*, page 81.

[Ritter et al. 2011] Ritter, A., Clark, S., Etzioni, O., et al. (2011). Named entity recognition in tweets: an experimental study. In *Proceedings of the conference on empirical methods in natural language processing*, pages 1524–1534. Association for Computational Linguistics.

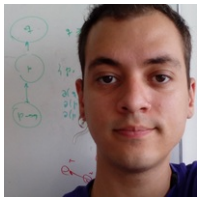
[Souza 2012] Souza, M. V. d. S. (2012). Mineração de opiniões aplicada a mídias sociais. Master's thesis, Pontifícia Universidade Católica do Rio Grande do Sul.

[SpaCy 2018] SpaCy (2018). Training spacy's statistical models. `https://spacy.io/usage/training`. Accessed: 2018-09-10.

[Twitter 2018] Twitter (2018). Sobre as apis do twitter. `https://help.twitter.com/pt/rules-and-policies/twitter-api`. Accessed: 2018-09-10.

Biografia resumida dos autores

Clarissa Castellã Xavier é pesquisadora de pós-doutorado na Universidade Federal do Rio Grande do Sul (UFRGS), mestre e doutora em Ciências da Computação pela Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS). Começou a pesquisar em Processamento da Linguagem Natural (PLN) em 1999 no Grupo de Pesquisa em PLN da PUCRS. Desde então, trabalhou para empresas multi-culturais em todo o mundo, desenvolvendo ferramentas de processamento de linguagem com foco em mídias sociais e redes. Seu trabalho atual tem como foco a extração semântica de dados na área do transporte urbano a partir de redes sociais. Também é pesquisadora convidada do grupo FORMAS da Universidade Federal da Bahia (UFBA).



Marlo Souza é professor adjunto na Universidade Federal da Bahia (UFBA), mestre em Ciências da Computação pela Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS) e doutor em Ciências da Computação pela Universidade Federal do Rio Grande do Sul (UFRGS). Suas atividades de ensino e pesquisa concentram-se nas áreas de computação teórica, representação do conhecimento, lógica aplicada e PLN. Iniciou suas pesquisas em PLN no ano de 2007 no contexto do projeto CoGROO - Corretor Gramatical livre para o OpenOffice, passando posteriormente pelo Grupo de Pesquisa em PLN da Pontifícia Universidade Católica do Rio Grande do Sul, em que estudou métodos de identificação de entidades e mineração de opiniões em textos do Twitter para monitoramento de marcas. Na UFBA, integra o grupo FORMAS trabalhando com métodos de extração de informações semânticas em texto.

Capítulo 3

Desenvolvendo Modelos de Deep Learning para Aplicações Multimídia no Tensorflow

Antonio José G. Busson[1], Lucas C. Figueiredo[1], Gabriel P. dos Santos[2], André Luiz de B. Damasceno[1], Sérgio Colcher[1] e Ruy L. Milidiú[1]

[1]Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (INF/PUC-Rio)

[2]Faculdade ISL | Wyden

{abusson,lfigueiredo,adamasceno,colcher,milidiu}@inf.puc-rio.br

gabrieainha@gmail.com

Abstract

The availability of massive quantities of data, combined with increasing computational capabilities, makes it possible to develop more precise Machine Learning algorithms. These new tools provide advances in areas such as Natural Language Processing and Computer Vision, allowing efficient processing of images, text and audio. Now, cognitive functionalities, such as learning, recognition and detection, can be used in multimedia applications to create mechanisms beyond traditional capture, streaming and presentation uses. Methods based on Deep Learning became state-of-the-art in several Multimedia challenges. This short course presents the grounds and ways to develop models using Deep Learning. It prepares the participant to: (1) understand and develop models based on Deep Neural Networks, Convolutional Neural Networks (CNN), Recurrent Neural Networks, including LSTM and GRU; (2) apply the Deep Learning models to solve problems within the multimedia domain like Image Classification, Facial Recognition, Object Detection, Video Scenes Classification. The Python programming language is shown alongside TensorFlow, a package for developing Deep Learning models.

Resumo

A disponibilidade de massivos volumes de dados somado ao aumento do poder computacional tornou possível a criação de métodos de Machine Learning mais precisos, provocando significativos avanços nas áreas de processamento de linguagem natural, visão e audição computacional. Tais avanços refletem em novas funcionalidades cognitivas

(e.g. aprendizagem, reconhecimento, detecção) que podem ser incorporadas em aplicações multimídia, dessa forma, permitindo a criação de novos mecanismos para uso das mídias, além do uso tradicional (e.g. captura, transmissão e apresentação). Métodos baseados em Deep Learning se tornaram o estado-da-arte em muitos problemas do domínio da Multimídia. Este minicurso tem como foco apresentar os fundamentos e tecnologias para desenvolver tais modelos de Deep Learning. Em especial, o minicurso prepara o participante para: (1) entender e desenvolver redes neurais profundas, redes neurais convolucionais (CNN), e redes neurais recorrentes (LSTM/GRU); (2) aplicar os modelos de Deep Learning para resolver problemas do domínio da multimídia: classificação de imagens, reconhecimento facial, detecção de objetos e classificação de cenas de vídeo. A linguagem de programação Python é apresentada em conjunto com a biblioteca TensorFlow para implementação dos modelos de Deep Learning ao decorrer do minicurso.

3.1. Introdução

Estamos vivendo em um período de produção de massivos volumes de conteúdo multimídia. Tomando como exemplo o YouTube[1] que, conforme mostra a Figura 3.1, em 2014 registrou uma taxa de *upload* de 72 horas de vídeo por minuto[2]. Já em 2018, esse número subiu para 400 horas[3].

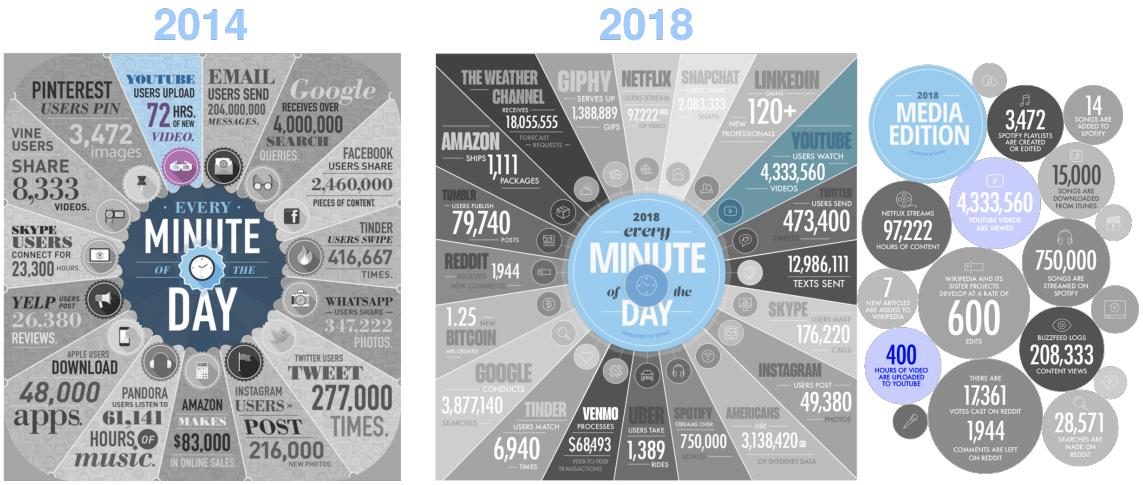


Figura 3.1. *Upload* de vídeos por minuto no YouTube (2014 e 2018).

Uma das razões para esse crescimento está relacionado: (i) a popularização e barateamento de equipamentos de produção de conteúdo multimídia, como *smartphones* e *tablets*; (ii) advento de serviços/plataformas para armazenamento e distribuição de conteúdo multimídia. Como consequência, o grande desafio do cenário atual consiste em extrair e organizar informações desse conteúdo multimídia de forma eficiente e prática. Isso implica na necessidade de estratégias e desenvolvimento de processos automáticos

[1] https://www.youtube.com
[2] https://www.domo.com/learn/data-never-sleeps-2
[3] https://www.domo.com/learn/data-never-sleeps-6

para análise do conteúdo multimídia, contemplando não somente a identificação de objetos mas também a compreensão de eventos mais complexos.

Nos últimos anos, o *Deep Learning* (DL) permitiu significativo avanço de vários segmentos da multimídia, principalmente em tarefas relacionadas a processamento de fala, audição e visão computacional [Ota et al. 2017]. Plataformas como IBM Watson[4] e Microsoft Azure ML[5] já oferecem DLaS (Deep Learning as a Service), permitindo que sistemas multimídia (e.g. Helpicto [Equadex 2018], PersonalizedTV [Thomas and Sadagopan 2016], ShrewsburyMuseum [Kearn and Beeby 2017]) possam incorporar novas funcionalidades baseadas em aprendizagem e reconhecimento de padrões, o que os leva para a categoria de IIMS (Intelligent Interactive Multimedia Systems).

Este capítulo apresenta os fundamentos e tecnologias para desenvolver modelos de *Deep Learning*. Em especial, o minicurso prepara o participante para: (1) entender e desenvolver redes neurais profundas, especialmente os modelos baseados em redes neurais convolucionais (CNN), e redes neurais recorrentes (LSTM/GRU); (2) aplicar os modelos de *Deep Learning* para resolver problemas do domínio da multimídia: classificação de imagens, reconhecimento facial, detecção de objetos e classificação de cenas de vídeo. A linguagem de programação Python é apresentada em conjunto com o *framework TensorFlow* para implementação dos modelos de *Deep Learning* ao decorrer do minicurso. Vale ressaltar que todos os projetos implementados neste minicurso estão disponíveis no Github[6].

O restante desse trabalho está organizado como a seguir. A Seção 3.2 apresenta o *framework TensorFlow*. A Seção 3.3 introduz os conceitos básicos de aprendizado de máquina. Em seguida, as Seções 3.4, 3.5 e 3.6 apresentam os fundamentos de Redes Neurais, Redes Neurais Convolucionais e Redes Neurais Recorrentes, respectivamente. As implementações para resolução de problemas de reconhecimento facial e detecção de objetos são descritos nas Seções 3.7 e 3.8, respectivamente. Por fim, a Seção 3.9 apresenta as considerações finais.

3.2. Framework TensorFlow

O *Tensorflow*[7] é um framework *open-source* para Python, Javascript, C/C++ e Go e tem o objetivo de auxiliar o processamento de dados em *machine learning* por meio de um modelo baseado em fluxo de dados. Este foi criado em 2015 pelo time da Google responsável por pesquisas na área de Inteligência Artificial e *Deep Learning* (Google Brain). O Tensorflow começou como uma refatoração do antigo sistema DistBelief[8] (criado em 2011), com o intuito de melhorar seu desempenho.

Nesta Seção é apresentada uma visão geral do *framework Tensorflow*. A Subseção 3.2.3 descreve o processo de instalação. Em seguida, a Subção 3.2.2 apresenta os fundamentos e estruturas básicas do *framework*. Por fim, a Subseção 3.2.3 descreve como utilizar o *framework*.

[4]https://www.ibm.com/watson/

[5]https://azure.microsoft.com/pt-br/services/machine-learning-studio/

[6]https://github.com/Busson/curso_deep_learning_para_multimidia

[7]https://www.tensorflow.org/?hl=pt-br

[8]https://ai.google/research/pubs/pub40565

3.2.1. Instalação

No decorrer desta subseção é descrito como instalar o *Tensorflow* em sistemas Ubuntu e outras distribuições derivadas do Linux. Os projetos que são apresentados no decorrer do minicurso são implementados na linguagem Python.

O Python e Pip (sistema de gerenciamento de pacotes do Python) já estão presentes no Ubuntu e na maioria das outras distribuições Linux. Porém, com Python2.7 e não o mais recente, Python3.

Para instalação do Python2.7, execute:

```
sudo apt-get install python-pip
python-dev python-virtualenv
```

Para instalação do Python3.x, execute:

```
sudo apt-get install python3-pip
python3-dev python-virtualenv
```

Tensorflow necessita que a versão do pip seja a 8.1 ou mais recente. Para verificar a versão atual do pip no Python2 e Python3 execute respectivamente:

```
pip –V
```

ou

```
pip3 –V
```

Para atualizar o pip para a versão mais recente no Ubuntu:

```
sudo pip install –U pip
```

Para atualização do pip em distribuições diferentes da Ubuntu:

```
easy_install –U pip
```

A seguir, deve-se escolher a instalação com ou sem suporte para GPU. Para instalar o *Tensorflow* sem e com suporte para GPU execute respectivamente:

```
pip install –U tensorflow
```

ou

```
pip install –U tensorflow-gpu
```

Para testar a instalação inicie o terminal Python e execute:

```
import tensorflow as tf
tf.__version__
exit()
```

3.2.2. Fundamentos e estruturas básicas do Tensorflow

O *Tensorflow* oferece facilidade para desenvolver modelos de redes neurais para uma multiplicidade de diferentes hardwares, bem como possibilita que o sistema seja executado sem ou com **GPU**s. Para utilizar o *framework*, primeiro é necessário entender os três conceitos que são descritos a seguir.

1. **Tensor**: consiste de um vetor de n dimensões. Tensores são as estruturas de dados básicas utilizadas no *TensorFlow*.

2. **Grafo de computação**: são malhas que consistem em nós conectados entre si por arestas. Cada nó possui seus *inputs* e outputs, assim como a operação que deve ser feita com os *inputs* para que o *output* seja criado. Tais arestas consistem nos valores que são passados de um nó para outro. Cada nó realiza a determinada operação assim que recebe todos os *inputs* necessários. Ao gerar seu resultado e passar para um próximo nó (ou vários) ligado a este.

3. **Sessões**: os nós do grafo de computação podem ser agrupados em sessões. Cada sessão pode ser executada separadamente em *threads* ou até mesmo em forma de computação distribuída.

3.2.3. Utilizando o Tensorflow

Após entender as estruturas básicas do *framework*, para usar o *Tensorflow*, como mostra a Listagem 3.1, basta importar o pacote para o projeto (linha 1). As linhas 3-5 mostram como criar o grafo de computação, para isso, são criados três tensores, onde o tesor **c**, consiste na multiplicação dos tensores **a** e **b**. Em seguida, as linhas 7 e 8 mostram como criar uma sessão e executar o grafo de computação. Por fim, a linha 10 mostra a saída do programa.

Listagem 3.1. Executando um grafo de computação no Tensorflow.

```
1 import tensorflow as tf
2
3 a = tf.constant([ [1.0, 2.0], [3.0, 4.0] ])
4 b = tf.constant([ [5.0, 6.0], [7.0, 8.0] ])
5 c = tf.matmul(a,b)
6
7 sess = tf.Session()
8 print(sess.run(c))
9 ------------------------------------ OUTPUT -------------------------------------
10 > [[19. 22.][43. 50.]]
```

Placeholders são tensores indefinidos, os quais receberão um valor posteriormente. Eles são úteis para receber as amostras de entrada e a saída que serão utilizadas no grafo de computação da rede neural. A Listagem 3.2 mostra como usar um *placeholder* no *Tensorflow*. Na linha 4 um *placeholder* do tipo *float* é criado com as dimensões 3x3. Em seguida a biblioteca Numpy é usada para gerar um tensor 3x3 com valores aleatórios. Nas linhas 9 e 10, a sessão é criada e o grafo de computação é executado. Note que desta

vez o parâmetro **feed_dict** é explicitamente definido. Esse parâmetro recebe como valor um *dict* que possui como chave o *placeholder* criado, e como valor o *array* de valores aleatório chamado *rand_array*). Por fim, na linha 13 é mostrada a saída do programa.

Listagem 3.2. Usando placeholders no Tensorflow.

```
1 import tensorflow as tf
2 import numpy as np
3
4 a = tf.placeholder(tf.float32, shape=(3,3))
5 b = tf.matmul(a,a)
6
7 rand_array = np.random.rand(3,3)
8
9 sess = tf.Session()
10 result = sess.run(b, feed_dict={a: rand_array})
11 print(result)
12 ---------------------------------- OUTPUT ----------------------------------
13 > [[1.9946331 1.5735985 2.126033 ][1.9040278 1.517215  2.0398355]
14 [1.7058356 1.3416395 1.8197424]]
```

3.3. Fundamentos de Aprendizado de Máquina

Aprendizagem de máquina, ou aprendizado automático é um subcampo da área de Inteligencia Artificial que automatiza a construção de modelos analíticos a partir dos dados. Em 1959, o cientista da computação Artur Samuel definiu o conceito de aprendizado de máquina como "campo de estudo que dá aos computadores a habilidade de aprender sem serem explicitamente programados" [Samuel 1959]. Em outras palavras, tais algoritmos operam construindo de forma automática um modelo interno a partir das amostras de entrada e fazem previsões guiadas pelos dados ao invez de seguir instruções programadas.

O aprendizado de máquina é usado em um vasto domínio onde algoritmos tradicionais são impraticáveis. Este minicurso é focado especialmente em problemas da área de sistemas multimídia, mas existem aplicações de que vão desde problemas relacionados a robótica até problemas de sequenciamento de DNA. As tarefas de aprendizado de máquina geralmente são categorizados em três tipos: supervisionado, não-supervisionado e por reforço.

Aprendizado supervisionado: O humano fornece amostras pré-classificadas a máquina. O objetivo é aprender uma função que mapeia a entrada para um tipo de saída. Exemplos de tarefas supervisionadas são: classificação e regressão.

Aprendizado não-supervisionado: O humano fornece apenas os dados, sem classificação. O objetivo é encontrar alguma estrutura nos dados. Técnicas de agrupamento (*clustering*) são tipicamente tarefas não-supervisionadas, pois os grupos descobertos não são conhecidos previamente.

Aprendizado por reforço: A máquina recebe sinais (premiações ou punições) de um ambiente dinâmico em que se deve desempenhar um determinado objetivo.

Este minicurso é especializado em aprendizado do tipo supervisionado e aborda principalmente os algoritmos baseados em redes neurais. Por consequência, são utilizados *datasets* anotados, os quais são divididos em conjuntos distintos e são usados para avaliar as capacidades de generalização dos modelos. No método de validação cruzada geralmente os *datasets* são divididos em três conjuntos: treino, validação e teste. O conjunto de treino é usado para realizar o treinamento do algoritmo, o conjunto de validação é usado para verificar a capacidade de generalização o algoritmo ainda em tempo de treinamento. Após o treinamento o cojunto de teste é usado avaliar o modelo. No entanto, devido ao tamanho pequeno da maioria dos *dataset* utilizados neste minicurso, eles são divididos em apenas dois conjuntos, treino e teste.

3.4. Redes Neurais

Métodos modernos de redes neurais são considerados os mais importantes modelos da área aprendizado de máquina. No entanto, até antes de 2006, não era possível treinar redes neurais para superar técnicas de aprendizado de máquina mais tradicionais (e.g. SVM, árvore de decisão), exceto em alguns domínios de problemas especializados. O que mudou em 2006 foi a descoberta de métodos que possibilitaram o advento dos modelos baseados em redes neurais profundas, também conhecido como aprendizado profundo (em inglês *Deep Learning*). A evolução das redes neurais profundas permitiu que esse tipo de arquitetura se tornasse o principal modelo para tarefas de classificação que estão relacionadas as áreas de visão computacional, reconhecimento de fala e processamento de linguagem natural.

Nesta seção são apresentados os conceitos básicos de redes neurais. Primeiro, na Subseção 3.4.1 são apresentados os modelos Perceptron e Perceptron de Múltiplas Camadas. Em seguida, na Subseção 3.4.2 é descrito o algoritmo de Retropropagação. Na Subseção 3.4.3 é apresentada um tipo de camada chamada *Softmax*. Por fim, na Subseção é descrita uma implementação que utiliza um Perceptron de Múltiplas Camadas para classificar pontos de um *dataset* artificial.

3.4.1. Perceptron

O Perceptron [Rosenblatt 1957], inventado em 1957 por Frank Rosenblatt, é a estrutura mais básica de uma Rede Neural. A Figura 3.2 ilustra a estrutura do Perceptron, cada entrada x possui um peso w associado. Em seguida é calculado o produto escalar entre os dados de entrada e seus pesos ($z = w_1x_1 + w_2x_2 + ... + w_nx_n = w^t.x$). Então uma função de ativação é aplicada sobre o produto escalar, resultando na saída do perceptron: $a_w(x) = ativ(z) = ativ(w^t.x)$. Algumas fontes da literatura utilizam uma entrada com valor constante 1 para representar o viés (em inglês, *bias*) do neurônio. Em fontes mais recentes, o *bias* é considerado, por padrão, um dado interno do neurônio, resultando na equação: $z = w^t.x + b$.

Múltiplos Perceptrons podem ser usados para realizar tarefas de classificação múltipla. Nesse caso, como ilustra a rede A da Figura 3.3, os neurônios são organizados em paralelo e cada um fica responsável por aprender a ativar para uma classe específica. A classe predita é selecionada ao usar a função *argmax* para obter a maior ativação dentre todas as saídas dos neurônios. Esse modelo é conhecido como Perceptron Multiclasse. Adicionalmente, como ilustra a rede B da Figura 3.3, os neurônios também podem ser

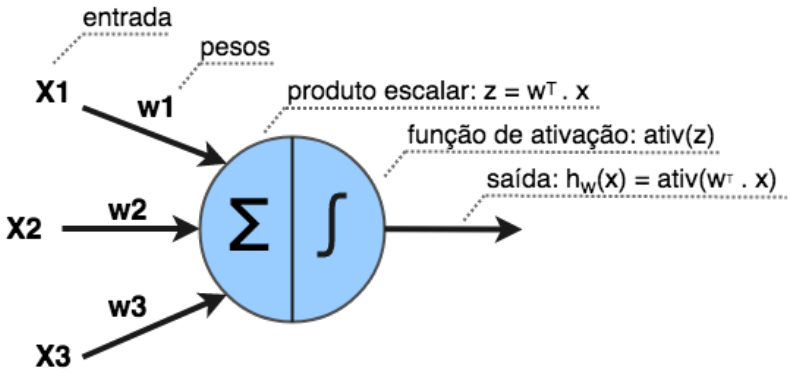


Figura 3.2. Estrutura de um Perceptron

estruturados em múltiplas camadas, onde cada neurônio das camadas intermediarias (ou escondida) é conectado com todos os neurônios da camada anterior. Os dados da amostra de entrada são considerados os neurônios da camada de entrada, enquanto a última camada da rede é chamada de camada de saída. Nesse caso a rede aprende a aplicar uma hierarquia de transformações lineares ou não lineares (através das ativações) para gerar novas representações do dado de entrada, para que seja possível, por exemplo, realizar classificações. Esse modelo é conhecido como Perceptron de Múltiplas Camadas, ou MLP (do inglês, *Multilayer Perceptron*). Uma rede neural é considerada profunda quando ela possui mais de duas camadas escondidas.

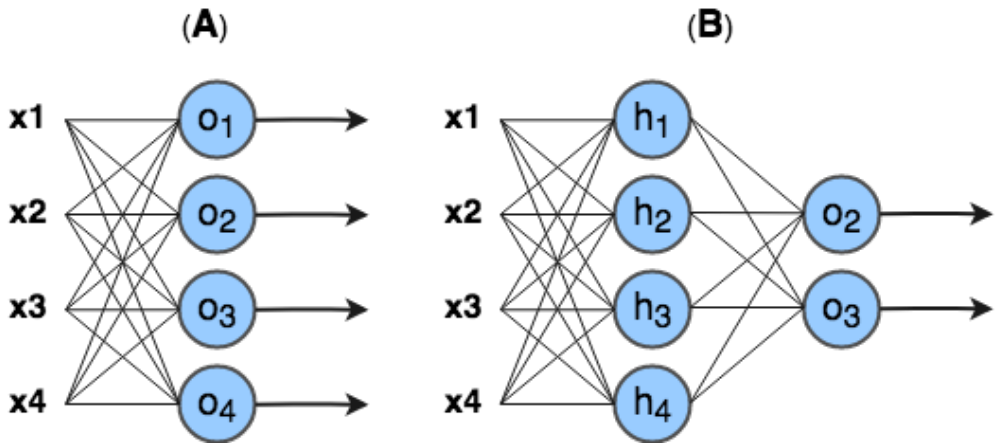


Figura 3.3. (A) Perceptron de múltiplas classes. (B) Perceptron múltiplas camadas.

A Figura 3.4[9] mostra as funções de ativação mais conhecidas. A função de ativação passo (*step*) foi utilizada na primeira versão do Perceptron. No entanto ela não oferece uma derivada útil para que possa ser usada para treinar modelos de múltiplas camadas. Funções de ativação logística (*sigmoid*) e tangente hiperbólica (*tanh*) tornaram-se populares nos anos 80 como aproximações mais suaves da função passo e permitiram a aplicação do algoritmo de retropropagação. Funções de ativação consideradas modernas como linear retificada (*ReLU*) e *maxout* são lineares por partes, computacionalmente baratas e funcionam bem na prática.

[9]https://denizyuret.github.io/Knet.jl/latest/mlp.html

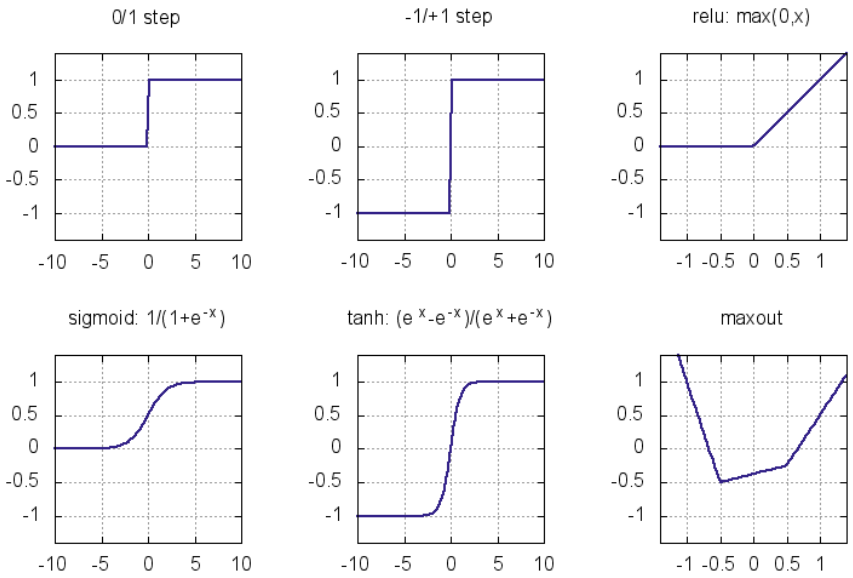


Figura 3.4. Funções de ativação.

O algoritmo que permite o aprendizado da rede neural é chamado de retropropagação (em inglês, *backpropagation*). Basicamente o que se chama de "aprendizado" em redes neurais é o ajuste nos pesos (w) e *biases* (b) dos neurônios para aproximar a saída da rede de uma função $y(x)$ para toda entrada de treinamento x. Para quantificar o quão próximo a rede está do objetivo são utilizadas funções de custo (também conhecidas como funções de perda). Por exemplo, a Equação 1 descreve a função de custo quadrático, comumente utilizada em problemas de regressão. Já a equação 2, descreve a função de custo entropia cruzada, geralmente utilizada em problemas de classificação. No decorrer deste minicurso ambas as funções são utilizadas nas implementações dos projetos práticos.

$$\mathscr{J} = \frac{1}{2n}\sum_{x} || \, y(x) - a \, ||^2 \tag{1}$$

$$\mathscr{J} = -\frac{1}{n}\sum_{x}[y \log a + (1-y)\log(1-a)] \tag{2}$$

3.4.2. Algoritmo de Retropropagação

O *Tensorflow* encapsula o algoritmo de retropropagação, portanto não é necessário implementa-lo. No entanto, para entender redes neurais é inecessário entender com a retropapagação funciona. Basicamente, o algorítimo de retropropagação busca alterar os valores dos pesos e bias da rede para otimizar a função de custo. Este algoritmo realiza um procedimento para computar o δ^l_j (erro do j-ésimo neurônio da l-ésima camada) e então o relaciona com as derivadas parciais dos pesos e bias em relação a função de custo ($\frac{\partial C}{\partial w^l_{jk}}$ e $\frac{\partial C}{\partial b^l_j}$).

A equação do erro δ^L na camada de saída é dada por:

$$\delta_j^L = \frac{\partial C}{\delta a_j^l}\sigma'(z_j^L) \tag{3}$$

O primeiro termo $\frac{\partial C}{\delta a_j^l}$ mede quão importante é a saída de ativação do j-ésimo neurônio para a função de custo C. Se por exemplo, a saída de um neurônio não contribui para a função de custo, então δ_j^L será pequeno. De forma similar, o segundo termo, $\sigma'(z_j^L)$, mede quão importante é o produto escalar do j-ésimo neurônio para sua saída de ativação.

Ao reescrever a fórmula anterior para uma versão baseada em matriz, obtém-se:

$$\delta^L = \bigtriangledown_a C \odot \sigma'(z^L) \tag{4}$$

Onde, $\bigtriangledown_a C$ é um vetor das derivadas parciais $\frac{\partial C}{\delta a_j^l}$ e o símbolo $\odot$ denota multiplicação elementar entre vetores.

A equação do erro δ^l em relação ao erro na próxima camada (δ^{l+1}) é dada por:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{5}$$

Onde, $((w^{l+1})^T$ é a transposta da matriz de pesos e o δ^{l+1} é o erro da (l+1)-ésima camada. Essa equação permite que o erro seja retropropagado através da rede. Ao usar a equação (1) para computar o δ^L, e então usar a equação (2) em sequência para computar $\delta^{L-1}, \delta^{L-2}, \delta^{L-3}, ..., \delta^1$.

A equação para mudar o custo em relação a qualquer bias e peso são dadas respectivamente por:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{6}$$

Isto é, o erro δ_j^l é igual a mudança $\frac{\partial C}{\partial w_{jk}^l}$.

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \tag{7}$$

O termo $\frac{\partial C}{\partial w_{jk}^l}$ é calculado em relação ao erro δ_j^l e ativação da camada anterior a_k^{l-1}. Dessa forma, quando a ativação da camada anterior é pequena, espera-se que o termo do gradiente $\frac{\partial C}{\partial w_{jk}^l}$ tenda a ser pequeno.

Com base nas 4 equações descritas, o algoritmo de retropropagação é resumido nos cinco passos seguintes:

1. **Entrada**: Inserção do X (entrada) na rede e cálculo da ativação da camada de entrada.

2. **Propagação**: Para cada camada l = 2, 3, ..., L, calcular a ativação dos neurônios recebendo a ativação dos neurônio da camada anterior como entrada ($z^l = w^l a^{l-1}$ e $a^l = \sigma(z^l)$).

3. **Erro da saída**: Calcular o vetor de erro $\delta^L = \bigtriangledown_a C \odot \sigma'(z^L)$.

4. **Retropropagação do erro**: Para cada camada l = L-1, L-2, ..., 2, calcular o erro da camada $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.

5. **Atualização dos pesos e bias**: Atualizar os pesos e bias com os respectivos gradientes: $\frac{\partial C}{\partial b^l_j} = \delta^l_j$ e $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$.

3.4.3. Camada *Softmax*

Nesta subseção é apresentada a camada *softmax*, um importante recurso usado em redes que realizam classificação. A ideia do *softmax* é definir uma nova camada saída para a rede com neurônios que usam a função de ativação softmax, a qual é descrita como:

$$a^L_j = \frac{e^{z^L_j}}{\sum_k e^{z^L_k}} \tag{8}$$

Onde o denominador da equação é a soma do produto escalar de todos os neurônios da camada *softmax*. Como resultado, o vetor de saída da camada *softmax* pode ser interpretadas como uma distribuição probabilística. Por exemplo, considerando a camada *softmax* da rede ilustrada na Figura 3.5, se o vetor do produto escalar $(z^L_1, z^L_2, z^L_3, z^L_4)$ = $(0.1, 0.9, 0.4, 2.3)$, então o vetor de ativação $(a^L_1, a^L_2, a^L_3, a^L_4)$ = $(0.07, 0.16, 0.09, 0.66)$. Isso significa que dada entrada X tem 66% de chance de ser da classe 4.

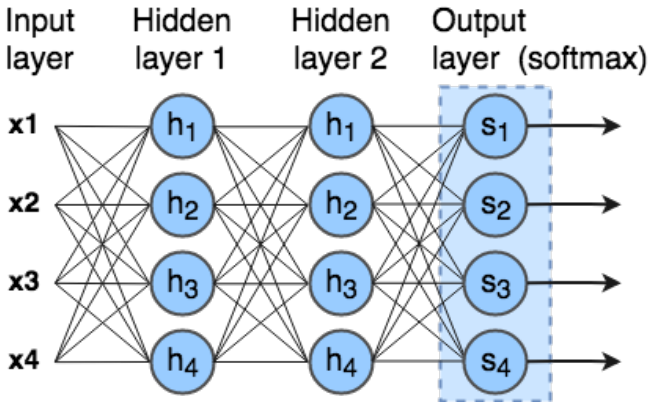


Figura 3.5. MLP Moderno com uma camada *softmax* para classificação.

3.4.4. Implementando um MLP

Nesta subseção é exemplificado o uso de um MLP para resolver um problema não linearmente separável. A Listagem 3.3 mostra o código que cria um pequeno dataset com

pontos distribuídos de forma circular usando a biblioteca scikitlearn[10], que pode ser visualizado na Figura 3.6. O dataset é composto por pontos de duas dimensões (duas *features*) que pertencem a dois conjuntos de classe, vermelho (0) e azul (1).

Listagem 3.3. Criando um dataset não linearmente separável.

```
1 import tensorflow as tf
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import sklearn.datasets
5
6 from aux import draw_separator
7
8 #Setando o seed para gerar uma sequencia  conhecida
9 tf.set_random_seed(0)
10 np.random.seed(0)
11
12 #numero de classes do nosso problema
13 num_classes = 2
14
15 #gerando o dataset
16 dataset_X, dataset_Y = sklearn.datasets.make_circles(200, noise=0.05)
17 #plotando o dataset
18 plt.scatter(dataset_X[:,0], dataset_X[:,1], s=40, c=dataset_Y,
19                                                     cmap=plt.cm.Spectral)
```

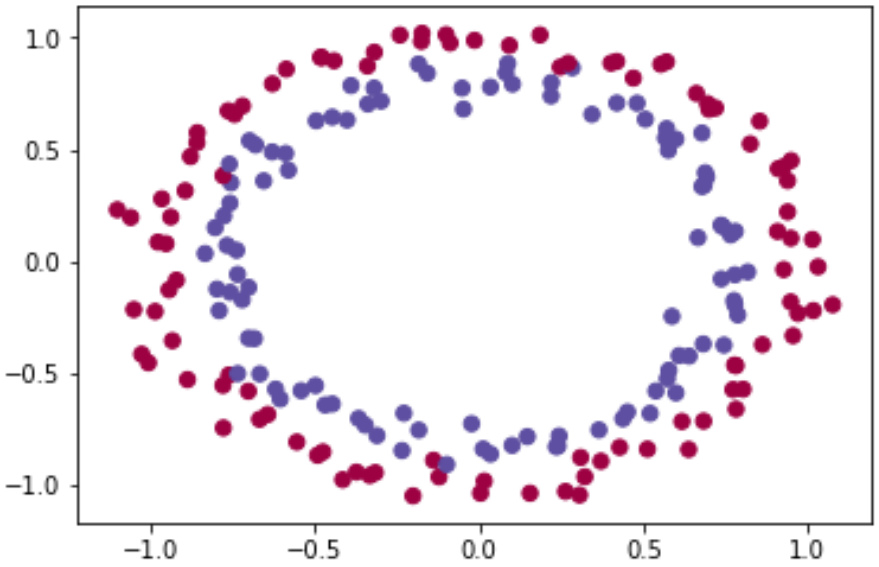


Figura 3.6. Visualização do dataset criado na Listagem 3.3

A Listagem 3.4 descreve a implementação de uma rede neural do tipo MLP.

[10]http://scikit-learn.org/

A função "build_net" recebe como parâmetro a quantidade de *features* e classes usadas no problema, neste caso, tanto a quantidade de *features* quanto a de classes são iguais a 2. Nas linhas 4 e 5 são definidos os *placeholders* do grafo de computação. O "X_placeholder" corresponde a camada de entrada da rede, enquanto o "Y_placeholder" é o vetor que contém os *labels* do dataset, e serão utilizados para comparação com a saída da rede. Na linha 8 é definida a camada escondida da rede, a qual possui cem unidades e usa a função de ativação ReLU. Na linha 11 é definida a camada de saída da rede, que neste exemplo possui apenas duas unidades. Na linha 15 o vetor de labels é convertido para o formato *OneHot* (por exemplo, a label 0 se torna o vetor [1,0] e o label 1 se torna o vetor [0,1]), dessa forma é possível a comparação com a saída da rede. Em seguida, na linha 17 é definida a função de perda, a função Entropia Cruzada é usada em conjunto com uma camada *softmax*. Na linha 20 é definido o otimizador da rede. Nas linhas 23 e 24 são definidos os tensores que classificam um exemplo de entrada da rede. Por fim, nas linhas 27 e 28, são definidos os tensores que calculam a acurácia da rede.

Listagem 3.4. Construindo um MLP.

```
1  def build_net(n_features, n_classes):
2
3      # Placeholders
4      X_placeholder = tf.placeholder(dtype=tf.float32, shape=[None, n_features])
5      Y_placeholder = tf.placeholder(dtype=tf.int64, shape=[None])
6
7      #camada oculta
8      layer1 = tf.layers.dense(X_placeholder, 100, activation=tf.nn.relu)
9
10     #camada de saida
11     out = tf.layers.dense(layer1, n_classes, name="output")
12
13     #adaptando o vetor Y para o modelo One-Hot Label
14     one_hot = tf.one_hot(Y_placeholder, depth=n_classes)
15
16     #funcao de perda/custo/erro
17     loss = tf.losses.softmax_cross_entropy(onehot_labels=one_hot, logits=out)
18
19     #otimizador
20     opt = tf.train.GradientDescentOptimizer(learning_rate=0.07).minimize(loss)
21
22     #classe do exemplo
23     softmax = tf.nn.softmax(out)
24     class_ = tf.argmax(softmax,1)
25
26     #acuracia
27     compare_prediction = tf.equal(class_, Y_placeholder)
28     accuracy = tf.reduce_mean(tf.cast(compare_prediction, tf.float32))
29
30     return X_placeholder, Y_placeholder, loss, opt, class_, accuracy
```

A Listagem 3.5 mostra como iniciar o TensorFlow e carregar o modelo de MLP criado. Na linha 2 é iniciada uma sessão interativa do TensorFlow. Em seguida, na linha 5 é obtida a quantidade de *features* do dataset. Na linha 8 o modelo de MLP criado na listagem anterior é carregado. Por fim, na linha 11, as variáveis do TensorFlow são inicializadas.

Listagem 3.5. Iniciando o TensorFlow e carregando o MLP.

```
1 #iniciando a sessao
2 sess = tf.InteractiveSession()
3
4 #obtendo o numero de features
5 n_features = dataset_X.shape[1]
6
7 #carregando o modelo
8 X_placeholder, Y_placeholder, loss, opt, class_,
9                                   accuracy = build_net(n_features,num_classes)
10 #inicializando as variaveis
11 sess.run(tf.global_variables_initializer())
```

A Listagem 3.6 mostra o código que realiza treinamento da rede. Um laço executa o treinamento da rede mil vezes (mil épocas) usando todo o *dataset*. Vale ressaltar que devido ao tamanho pequeno do *dataset*, neste exemplo, o método de dividir o *dataset* em lotes não é usado. A cada 100 épocas o erro da rede é calculado e impresso. Ao fim do treinamento a acurácia da rede é calculada e impressa. A linha 21 exemplifica como utilizar o modelo para realizar uma classificação. Em seguida, na linha 24, a função "draw_separator" desenha o dataset separado pelo modelo, o qual pode ser visto na Figura 3.7. Por fim, as linhas 27-37 mostram a saída do programa.

Listagem 3.6. Treinamento do MLP.

```
1 #definindo o numero de epocas
2 epochs = 1000
3 for i in range(epochs):
4
5     #treinamento (OBS: mini-batch nao usado por causa do tamanho pequeno do dataset)
6     sess.run(opt, feed_dict={X_placeholder: dataset_X,
7                                                 Y_placeholder: dataset_Y})
8
9     #a cada 100 epocas o erro e impresso
10    if  i % 100 == 0:
11        erro_train = sess.run(loss, feed_dict={X_placeholder: dataset_X,
12                                                 Y_placeholder: dataset_Y})
13        print("O erro na epoca", i,":", erro_train)
14
15 #calculando a acuracia
16 acc = sess.run(accuracy, feed_dict={X_placeholder: dataset_X,
17                                                 Y_placeholder: dataset_Y})
18 print("accuracia do modelo:", acc)
```

```
19
20 cla = sess.run(class_, feed_dict={X_placeholder: dataset_X[:1]})
21 print("a classe do ponto", dataset_X[:1], "e:", cla)
22
23 #desenhando o separador
24 draw_separator(dataset_X, dataset_Y, sess, X_placeholder, class_)
25
26 ---------------------------------- OUTPUT ------------------------------------
27 > 0 erro na epoca 0 : 0.69493294
28 > 0 erro na epoca 100 : 0.67670804
29 > 0 erro na epoca 200 : 0.6603513
30 > 0 erro na epoca 300 : 0.643817
31 > 0 erro na epoca 400 : 0.6265911
32 > 0 erro na epoca 500 : 0.60751265
33 > 0 erro na epoca 600 : 0.58588564
34 > 0 erro na epoca 700 : 0.56282145
35 > 0 erro na epoca 800 : 0.5376183
36 > 0 erro na epoca 900 : 0.510537
37 > accuracia do modelo: 0.97
38 > a classe do ponto [[0.4013312  0.88583093]] e: [0]
```

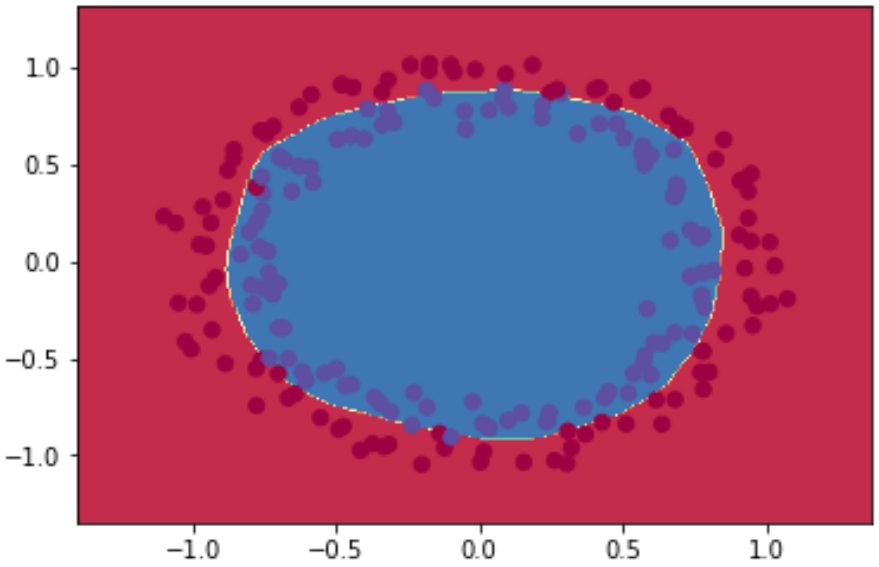


Figura 3.7. dataset separado pelo MLP.

3.5. Redes Neurais Convolucionais

Redes Neural Convolucionais (CNNs ou ConvNets) são redes especializadas no processamento de dados que são comumente organizados em topologia de grade (no caso mais comum, imagens). Esse modelo recebe este nome porque faz uso de uma operação matemática chamada convolução. As camadas de convolução possibilitam que uma

CNN e encontre *features* de baixo nível nas primeiras camadas e então as compõe em *features* de mais alto nível ao decorrer da rede. A habilidade de encontrar uma estrutura hierárquica de *features* é o principal motivo pelo qual CNNs funcionam tão bem para reconhecimento de padrões em imagens.

Nesta Seção são apresentados os fundamentos básicos e técnicas para implementação de CNNs. Na Subseção 3.5.1 a operação de convolução e *pooling* é apresentada. Em seguida, na Subseção 3.5.2 descreve a implementação de uma CNN para classificação de imagens de sinais de mão. Por fim, na Subseção 3.5.2.1 é apresentado o funcionamento e histórico de evolução da rede InceptionNet.

3.5.1. Camadas de Convolução e Pooling

A convolução consiste de um operador linear que, a partir de duas funções, resulta numa terceira que é a soma do produto dessas funções ao longo da região subentendida pela superposição delas em função do deslocamento existente entre elas. Para funções continuas, a convolução é definida como a integral do produto de uma das funções por uma cópia deslocada e invertida da outra:

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da$$

Para funções de domínio discreto, a convolução é dada por:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

Em CNNs a operação de convolução é feita em mais de uma dimensão por vez. Os pesos dos neurônios são representadas por um tensor chamado *kernel* (ou *filter*). O processo de convolução entre os neurônios e os *kernels* produzem saídas chamadas de mapas de *features*. Especificamente, baseado na equação discreta da convolução, a saída de um neurônio localizado na linha i, coluna j do mapa de *features* k em dada camada de convolução l é dada pela equação:

$$z_{i,j,k} = b_k + \sum_{u=1}^{f_h} \sum_{v=1}^{f_w} \sum_{k'=1}^{f'_n} x_{i',j',k'} . w_{u,v,k',k}$$

onde:

- $z_{i,j,k}$ é a saída do neurônio localizado na linha i, coluna j e no mapa de *features* k da camada convolucional l;
- $x_{i',j',k'}$ é a saída do neurônio localizado na linha i', coluna j' e no mapa de *features* k' da camada anterior (l -1);
- $w_{u,v,k',k}$ é o peso de conexão entre qualquer neurônio do mapa de *features* k da camada l e sua entrada localizada na linha u, coluna v e mapa de *features* k'.
- b_k é o bias para o mapa de *features* k na camada l

- Os parâmetros s_h e s_w representam os *strides* (deslocamentos) verticais e horizontais, f_h f_w são a altura e largura do *kernel*, e $f_{n'}$ é o número de mapa de *features* na camada anterior.

A Figura 3.8 mostra um exemplo de convolução entre dois tensores 2D. O *kernel* (em azul) tem dimensões (2,2), o tensor de entrada (i) tem dimensões (3,3) e o *stride* é igual a 1. Na primeira iteração, a saída (o) descrita pelo cálculo: $(1 \times 3) + (-1 \times 7) + (-1 \times 10) + (1 \times 8)$ = -6; Na segunda iteração, $(1 \times 7) + (-1 \times 4) + (-1 \times 8) + (1 \times 11)$ = 6; Na terceira iteração, $(1 \times 10) + (-1 \times 8) + (-1 \times 12) + (1 \times 1)$ = -9. E por fim, na quarta iteração, $(1 \times 8) + (-1 \times 11) + (-1 \times 1) + (1 \times 2)$ = -2. Note que o tensor de saída possui dimensões diferentes do tensor de entrada, isso ocorre porque ...

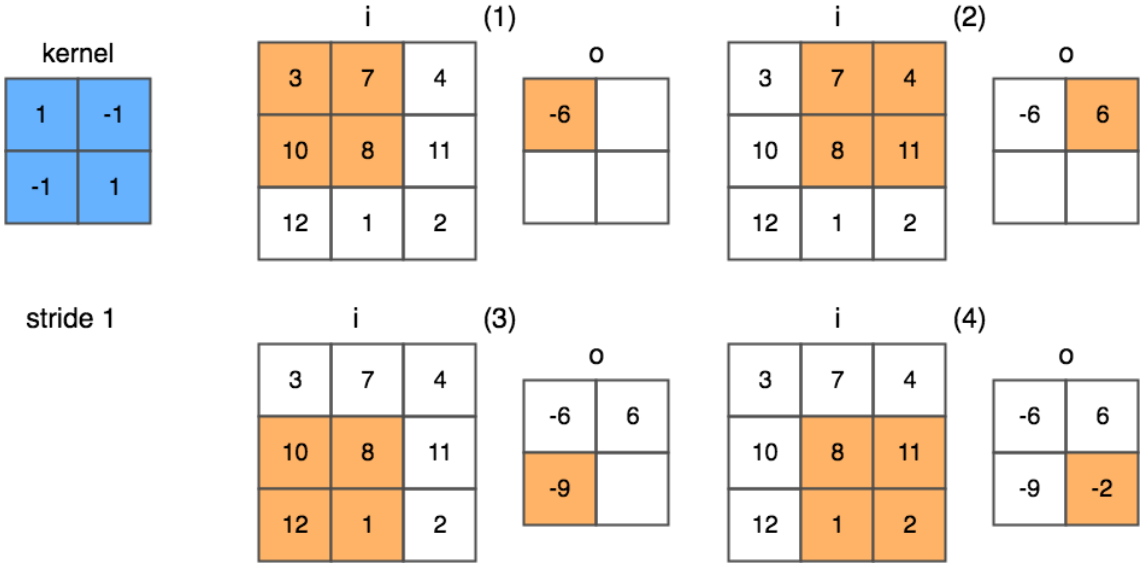


Figura 3.8. Processo de convolução.

Em CNNs a camadas de *pooling* tem a função de reduzir a dimensionalidade dos mapas de features para diminuir a carga computacional, uso de memória e número de parâmetros (dessa forma, reduzindo o risco de *overfitting*). Além disso, a redução de dimensionalidade permite que a rede tolere pequenas mudanças nos mapas de *features* (invariância de localização).

As camadas de *pooling* operam de forma semelhante as camadas de convolução, com a diferença que os *pooling kernels* não possuem pesos. Os *pooling kernels* agregam a entrada através de funções de agregação, como *max* ou *mean*. A função *max pooling*, por exemplo, retorna o maior valor dentro de uma área do tensor. Outras funções de *pooling* incluem, por exemplo, a média ou a distancia L^2 entre os elementos de uma área do tensor. A Figura 3.9 ilustra um exemplo do processo de *max pooling*. Cada área colorida representa uma etapa da operação que usa um *pooling kernel* com dimensões 2x2 e stride 2. Na área de cor laranja o maior valor é 28; Em seguida, na área de cor verde, 21; Na área azul, 27; E por fim, na área lilás, 17.

A Figuta 3.10 ilustra a arquitetura de uma CNN que usa camadas de convolução e *pooling*. A entrada da rede consiste de um tensor 16x16x3, correspondente a uma imagem

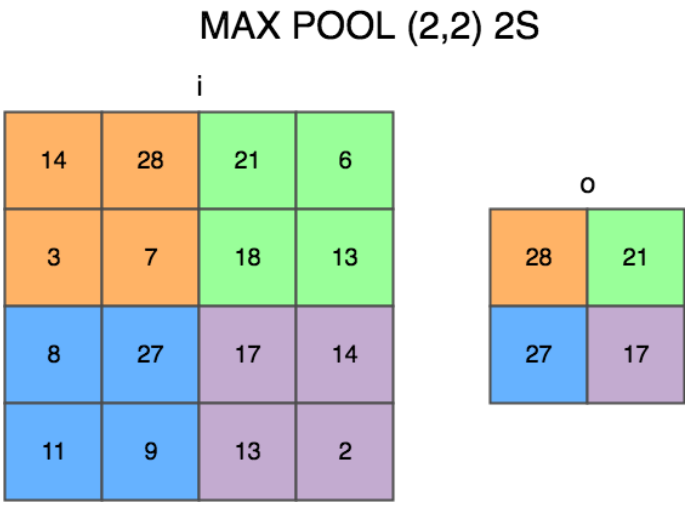


Figura 3.9. Exemplo de um processo de max *pooling* com *kernel* 2x2 e *stride* 2.

com 16 de altura, 16 de largura e 3 canais (RGB). A primeira convolução usa 8 kernels com dimensões (4,4) e *stride* 1 seguido de uma função de ativação ReLu, resultando em 8 mapas de features com dimensões 16x16. Em seguida, é aplicada uma camada de *pooling* que usa um kernel com dimensões (4,4) e *stride* 4, resultando em mapas de *features* com dimensões reduzidas (4,4). A póxima camada de convolução usa 4 *kernels* (2,2) seguido de um ReLU, resultando em 4 mapas de features com dimensões 4x4. Por fim, uma última camada de *pooling* usa um kernel(4,4) e *stride* 1, resultando em 4 mapas de *features* 1x1. A última camada é então conectada a uma camada de saída *Fully Connected* (FC).

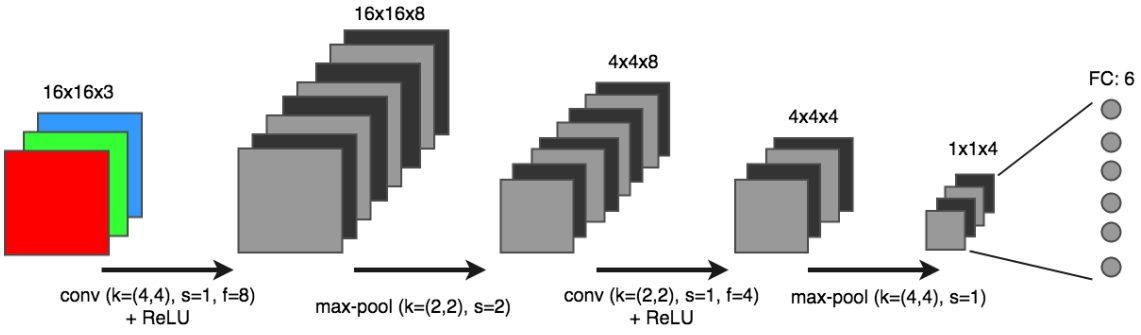


Figura 3.10. Exemplo de uma arquitetura de CNN.

3.5.2. Implementando uma Rede Neural Convolucional

Nesta subseção é exemplificada a implementação de uma CNN para reconhecer imagens de sinais de mãos. O *dataset* usado neste exemplo foi obtido no curso de especialização em Deep Learning do professor Andrew Ng. (deeplearning.ai) [11]. O *dataset* é composto por 1200 fotos de sinais de mão no formato RGB com dimensões 64x64. A Figura 3.12 ilustra os seis tipos de sinais de mãos encontrados no *dataset*, bem como os respectivos vetores de *labels* codificados no formato *OneHot*.

[11] https://www.deeplearning.ai/

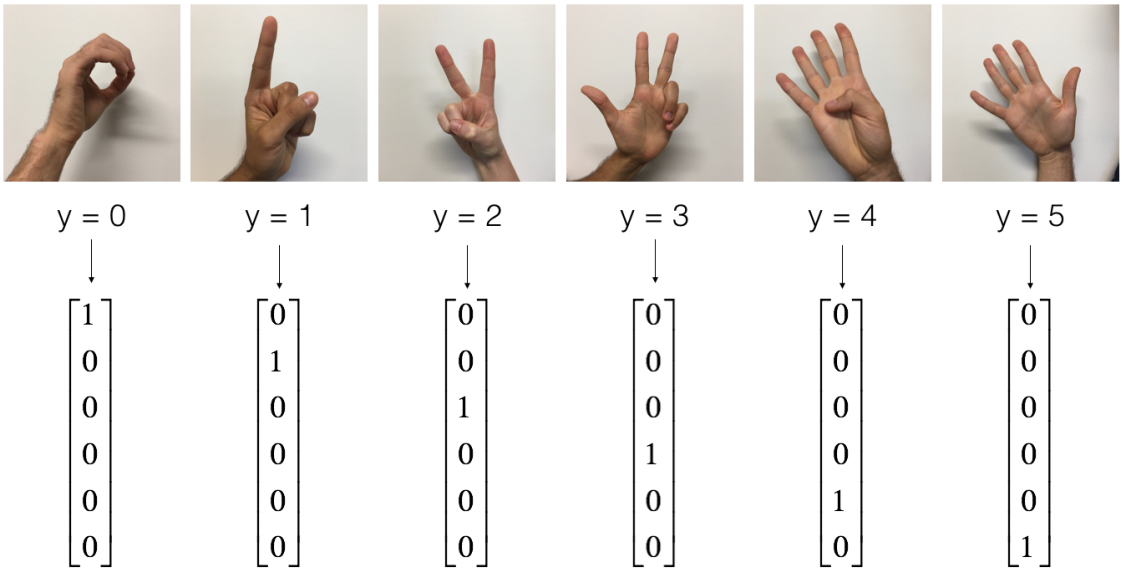


Figura 3.11. exemplos de cada classe do *dataset* de sinais de mão e suas respectivas codificações OneHot (by Prof. Andrew Ng., deeplearning.ai).

A Listagem 3.7 mostra o código que carrega o dataset de sinais de mão. Nas linhas 1-10 são definidas as bibliotecas necessárias para implementação do exemplo. Na linha 17 o *dataset* é carregado. Em seguida, nas linhas 20-23 as dimensões dos conjuntos de treino e teste são impressas. Nas linhas 26-28 é chamada uma função que mostra uma imagem do conjunto de treino, bem como sua classe. Por fim, as linhas 30-34 mostram a saída do programa.

Listagem 3.7. Carregando o *dataset* de sinais de mão.

```
1 import math
2 import numpy as np
3 import h5py
4 import matplotlib.pyplot as plt
5 import scipy
6 from PIL import Image
7 from scipy import ndimage
8 import tensorflow as tf
9 from tensorflow.python.framework import ops
10 from cnn_utils import *
11
12 #setando o seed para gerar uma sequencia conhecida
13 tf.set_random_seed(0)
14 np.random.seed(0)
15
16 #carregando o dataset (dividido em treino e teste)
17 X_train, Y_train, X_test, Y_test, classes = load_dataset()
18
```

```
19 #imprimindo as dimensoes dos conjuntos de treino e teste do dataset
20 print ("X_train shape: " + str(X_train.shape))
21 print ("Y_train shape: " + str(Y_train.shape))
22 print ("X_test shape: " + str(X_test.shape))
23 print ("Y_test shape: " + str(Y_test.shape))
24
25 #exibindo um exemplo
26 index = 6
27 plt.imshow(X_train[index])
28 print ("y =", Y_train[index])
29 ---------------------------------- OUTPUT ------------------------------------
30 > X_train shape: (1080, 64, 64, 3)
31 > Y_train shape: (1080,)
32 > X_test shape: (120, 64, 64, 3)
33 > Y_test shape: (120,)
34 > y = 2
```

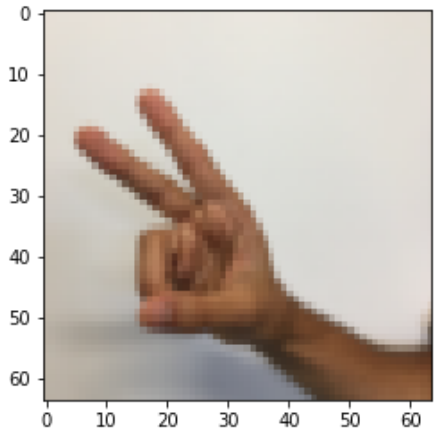


Figura 3.12. Exemplo de imagem renderizada na Listagem 3.7.

A Listagem 3.8 mostra a construção de uma simples arquitetura de CNN. A função "build_cnn" recebe como parâmetro a largura, altura e número de canais da imagem de entrada, bem como o número de classes do problema. Nas linhas 3 e 4 são definidos os *placeholders* da CNN. O "X" é o tensor que armazena as imagens de entrada da rede. Enquanto o "Y" é o vetor que contém as *labels* das imagens de entrada. Entre as linhas 11-26 estão definidas as camadas de convolução e *pooling* da rede. Todas as camadas usam *padding* SAME, ativação ReLU e são inicializadas com o método Xavier. A primeira e a segunda camadas de convolução (linha 11 e 15) tem um *kernel* de dimensões (4,4) e 32 *filters*. Em seguida, na linha 19 é definida a primeira camada de *pooling*, que possui um *kernel* de dimensões (8,8) e usa *stride* 4. A terceira camada de convolução (linha 22) tem um kernel de dimensões (2,2) e 16 *filters*. Por fim, a última camada de *pooling* possui um *kernel* de dimensões (8,8) e também usa *stride* 8. O restante do código possui as definições da função de custo, otimizador e demais tensores já explicados nos exemplos anteriores deste capítulo.

Listagem 3.8. Construindo uma CNN.

```
def build_cnn(input_width, input_height, input_channels, n_classes):

    #placeholders
    X = tf.placeholder(tf.float32, shape=(None, input_width,
                                          input_height, input_channels))
    Y = tf.placeholder(tf.int64, shape=(None))

    initializer = tf.contrib.layers.xavier_initializer(seed = 0)

        #camada convolucao 1
    conv2d_1 = tf.layers.conv2d(inputs=X, filters=32, kernel_size=[4,4],
    strides=1, activation=tf.nn.relu, padding = 'SAME',
                                          kernel_initializer=initializer)
    #camada convolucao 2
    conv2d_2 = tf.layers.conv2d(inputs=conv2d_1, filters=32, kernel_size=[4,4],
    strides=2, activation=tf.nn.relu, padding = 'SAME',
                                          kernel_initializer=initializer)
    #camada pooling 1
    maxpool_1 = tf.layers.max_pooling2d(inputs=conv2d_2, pool_size=[8, 8],
                                          strides=4, padding = 'SAME')
        #camada convolucao 3
    conv2d_3 = tf.layers.conv2d(inputs=maxpool_1, filters=16, kernel_size=[2,2]
    ,strides=1, activation=tf.nn.relu, padding = 'SAME',
                                          kernel_initializer=initializer)
    #camada pooling 1
    maxpool_2 = tf.layers.max_pooling2d(inputs=conv2d_3, pool_size=[8, 8],
                                          strides=8, padding = 'SAME')

    #flatten
    flatten = tf.contrib.layers.flatten(maxpool_2)

    #output (fully_connected)
    out = tf.contrib.layers.fully_connected(flatten, num_outputs=n_classes,
                                                    activation_fn=None)

    #adaptando o Label Y para o modelo One-Hot Label
    one_hot = tf.one_hot(Y, depth=n_classes)

    #funco de perda/custo/erro
    loss = tf.losses.softmax_cross_entropy(onehot_labels=one_hot, logits=out)

    #Otimizador
    opt = tf.train.AdamOptimizer(learning_rate=0.001).minimize(loss)

    #Softmax
```

```
46     softmax = tf.nn.softmax(out)
47
48     #Classe
49     class_ = tf.argmax(softmax,1)
50
51     #áAcurcia
52     compare_prediction = tf.equal(class_, Y)
53     accuracy = tf.reduce_mean(tf.cast(compare_prediction, tf.float32))
54
55     return X, Y, loss, opt, softmax, class_, accuracy
```

A Listagem 3.9 mostra o código que inicializa o TensorFlow e carrega a CNN definida na listagem anterior. Nas linhas 2 e 3 os valores dos pixels das imagens são normalizados para valores entre 0 e 1. Na linha 6 é iniciada uma sessão interativa do TensorFlow. Na linha 9 o modelo de CNN é carregado. E por fim, na linha 13 as variáveis do TensorFlow são inicializadas.

Listagem 3.9. Iniciando o TensorFlow e carregando a CNN.

```
1 #normalizando os dados de entrada
2 X_train = X_train/255.
3 X_test = X_test/255.
4
5 #Iniciando
6 sess = tf.InteractiveSession()
7
8 #carregando o modelo de CNN
9 X, Y, loss, opt, softmax, class_, accuracy =
10                                                build_cnn(64,64,3,6)
11
12 # inicializando as variveis do tensorflow
13 sess.run(tf.global_variables_initializer())
```

A Listagem 3.10 mostra o código que realiza o treinamento da CNN com o *dataset*. Neste exemplo de implementação o modelo é treinado em 100 épocas. Em cada época, como definido nas linhas 7 e 8, uma lista de *mini-batch* é gerada. Em seguida, a rede é treinada com cada *mini-batch*. O erro do treinamento é impresso a cada 10 épocas. Ao fim do treinamento, a acurácia da CNN treinada é impressa.

Listagem 3.10. Treinando a CNN.

```
1 #definindo o numero de epocas
2 epochs = 100
3
4 seed=0
5 for i in range(epochs):
6     #gerando um mini—batch aleatorio
7     seed = seed + 1
8     minibatches = random_mini_batches(X_train, Y_train, 64, seed)
```

```
9     #treinando a rede com cada minibatch
10    for minibatch in minibatches:
11        (minibatch_X, minibatch_Y) = minibatch
12        sess.run(opt, feed_dict={X_placeholder: minibatch_X,
13                                                 Y_placeholder: minibatch_Y})
14
15    #imprimindo o erro a cada 100 epocas
16    if  i % 10 == 0:
17        erro_train = sess.run(loss, feed_dict={X: X_train, Y: Y_train})
18        print("erro na epoca", i,":", erro_train)
19
20
21 #calculando a acuracia da rede
22 acc = sess.run(accuracy, feed_dict={X: X_test, Y: Y_test})
23 print("accurcia do modelo:", acc)
24 ---------------------------------- OUTPUT ------------------------------------
25 > erro na epoca 0 : 1.79012
26 > erro na epoca 10 : 1.53382
27 > erro na epoca 20 : 0.809482
28 > erro na epoca 30 : 0.586155
29 > erro na epoca 40 : 0.505508
30 > erro na epoca 50 : 0.366477
31 > erro na epoca 60 : 0.29243
32 > erro na epoca 70 : 0.240354
33 > erro na epoca 80 : 0.21093
34 > erro na epoca 90 : 0.143943
35 > accurcia do modelo: 0.908333
```

A Listagem 3.11 mostra como utilizar a CNN recém treinada para realizar classificações de sinais de mão. Na linha 4 é feita a predição da classe de uma imagem de entrada. Em seguida a imagem que foi utilizada é desenhada na tela (Figura 3.13). Por fim, a linha 9 mostra a saída do programa.

Listagem 3.11. Usando a CNN treinada para classificar imagens.

```
1 index = 10
2 example = X_test[index:(index+1)]
3
4 cla = sess.run(class_, feed_dict={X: example})
5 print("classe:", cla)
6
7 plt.imshow(X_test[index])
8 ---------------------------------- OUTPUT ------------------------------------
9 > classe: [5]
```

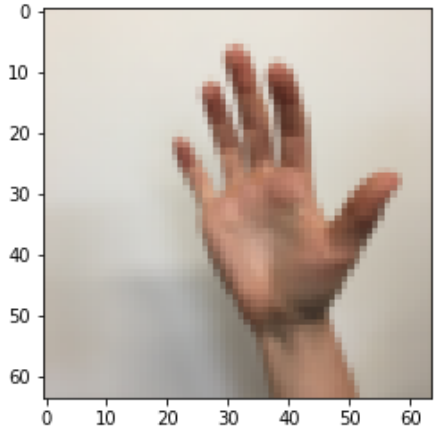


Figura 3.13. Imagem classificada como sinal 5 na Listagem 3.11.

3.5.2.1. Evolução da rede Inception

A primeira versão da rede InceptionNet (ou GoogleNet) [Szegedy et al. 2015] foi a campeã do desafio ImageNet 2014. Essa arquitetura é considerada importante porque ataca o problema de localização da informação, pois elementos na imagem podem ter grande variedades de tamanhos. Como pode ser visto na Figura 3.14, por exemplo, a área ocupada por um cão é diferente em cada imagem. Por causa dessa variedade, escolher um tamanho de *kernel* apropriado se torna difícil. Um *kernel* largo é adequado quando a informação esta distribuída globalmente, e um *kernel* curto quando a informação esta distribuída mais localmente.

Figura 3.14. Esquerda: cão ocupando quase toda a imagem; Centro: cão ocupando uma parte da imagem; Direita: cão ocupando uma pequena parte da imagem.

A solução proposta pela rede InceptionNet é de usar *kernels* de diferentes tamanhos no mesmo nível, deixando a rede um pouco mais larga que profunda. Para isso, os autores da rede projetaram o modulo Inception, o qual é ilustrado na Figura 3.15. O módulo Inception realiza convoluções com três diferentes tamanhos de *kernel*, 1x1, 3x3 e 5x5. Adicionalmente é feito um *maxpooling* com um *kernel* 3x3. Para não deixar o processamento tão pesado, ele limita o número de camadas da entrada usando uma convolução 1x1 antes das convoluções 3x3 e 5x5. Apenas no *maxpooling* a convolução 1x1 é realizada posteriormente.

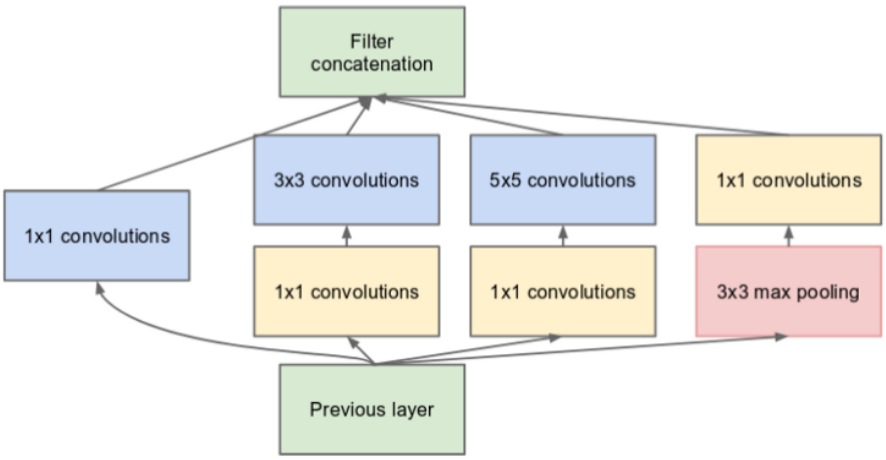


Figura 3.15. Módulo Inception da rede InceptionNet.

Como pode ser visto na Figura 3.16, a rede InceptionNet usa 9 módulos Inception em sequência, resultando em 27 camadas. Por ser uma rede profunda ela esta sujeita ao problema de desaparecimento do gradiente. Para resolver isso, os autores adicionaram duas saídas com classificadores auxiliares na saída de dois módulos Inception. O custo total da rede é dado pela soma ponderada entre o custo dado pelas três saídas da rede.

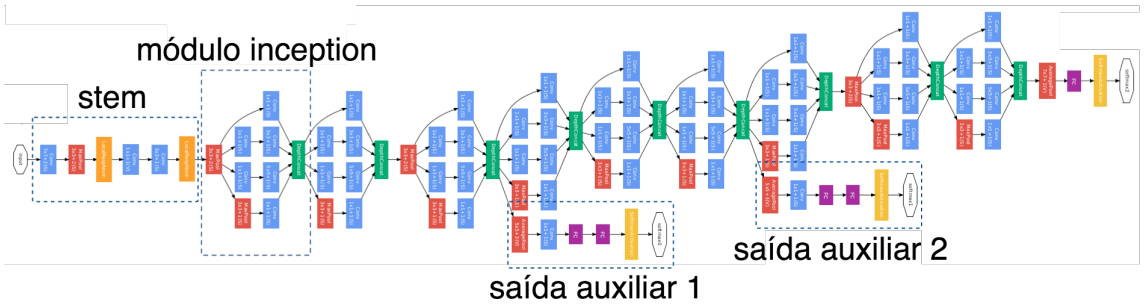


Figura 3.16. Arquitetura da rede InceptionNet (GoogleNet).

A arquitetura InceptionNet v2 [Szegedy et al. 2016] tenta reduzir o impacto de um problema conhecido como "gargalo representacional". CNNs funcionam melhor quando as convoluções não alteram a dimensão da entrada de forma drástica, pois essa redução pode causar perda de informação. Para isso, os autores criaram três novas versões do módulo Inception, que refatoram as convoluções 5x5 em duas convoluções menores. Como pode ser visto na Figura 3.17, no módulo A a convoluções 5x5 foi substituída por uma sequencia de duas convoluções 3x3, o que implica em uma melhora de performance, já que uma convolução 5x5 é 2.78 vezes computacionalmente mais cara que uma convolução 3x3. Já no módulo B, os autores substituíram cada convolução 3x3 por uma sequência de 1xn seguida de uma nx1. Por fim, no módulo C a posição das camadas de convolução foram alteradas, de forma que ficaram mais esparsas do que profunda. Essa decisão de projeto tenta suavizar o gargalo representacional, pois se o módulo fosse mais profundo, haveria redução excessiva nas dimensões e, consequentemente, perda de informação.

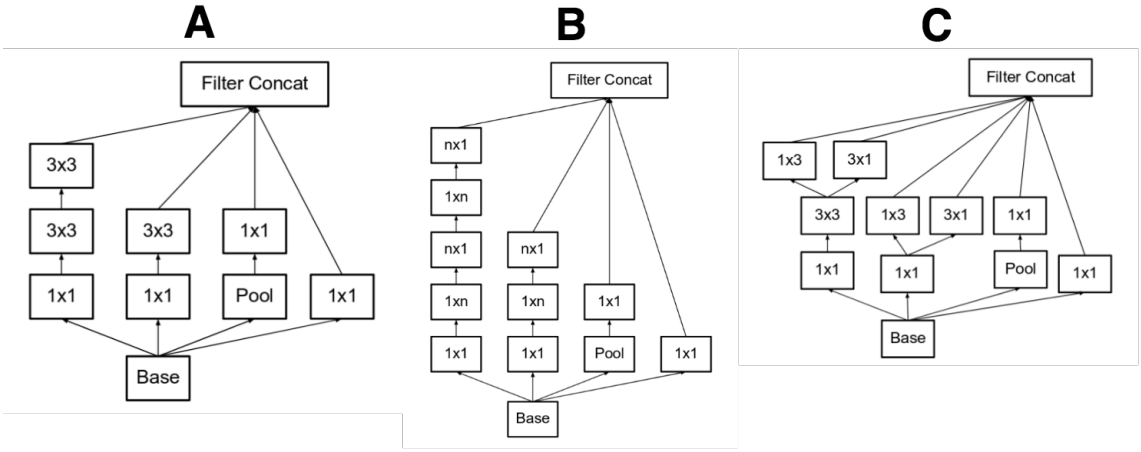


Figura 3.17. Três tipos de módulo inception da arquitetura InceptionNet v2.

A terceira versão, chamada de InceptionNet v3 [Szegedy et al. 2016] adatou o otimizador RMSProp, refatorou as convoluções 7x7 e aplicou a técnica de *Batch Normalization* (*BatchNorm*) as saídas auxiliares. Os autores da rede constataram que as saídas auxiliares não contribuíram muito até o final do processo de treinamento, quando as acurácias se aproximam da saturação. Eles argumentam que elas podem funcionar como regularizes, especialmente se eles tiverem operações *BatchNorm* ou *Dropout*.

A quarta versão, chamada InceptionNet v4 [Szegedy et al. 2017] reformulou algum módulos da arquitetura. A Figura 3.18 ilustra a arquitetura geral da rede. A Figura 3.19 detalha cada bloco da rede. A InceptionNet v4 apresenta um bloco *Stem* modificado. Os módulo Inception A, B e C são similares aos das versões anteriores. Uma novidade proposta pelo InceptionNet v4 é a definição dos módulos de redução, que são usados para diminuir a dimensionalidade dos mapas de *features*. As versões anteriores já tinham essa funcionalidade, mas ela não estava explicitamente formalizada como um módulo da rede.

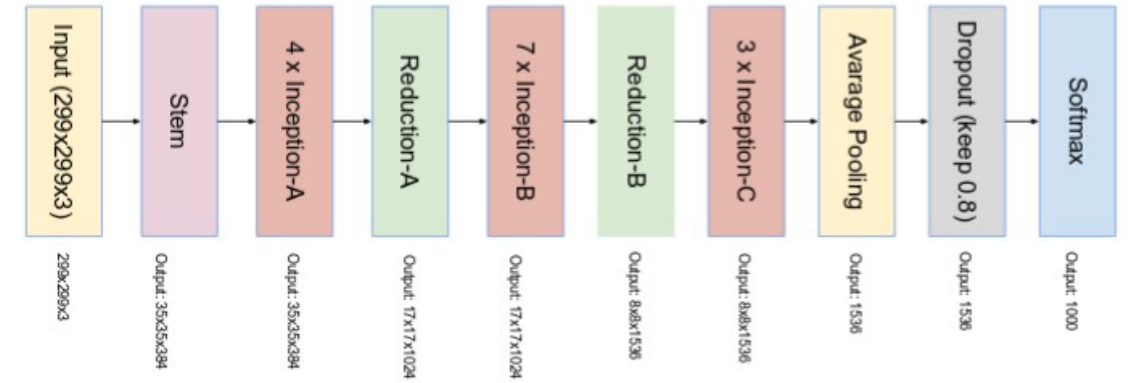


Figura 3.18. Arquitetura da rede InceptionNet v4.

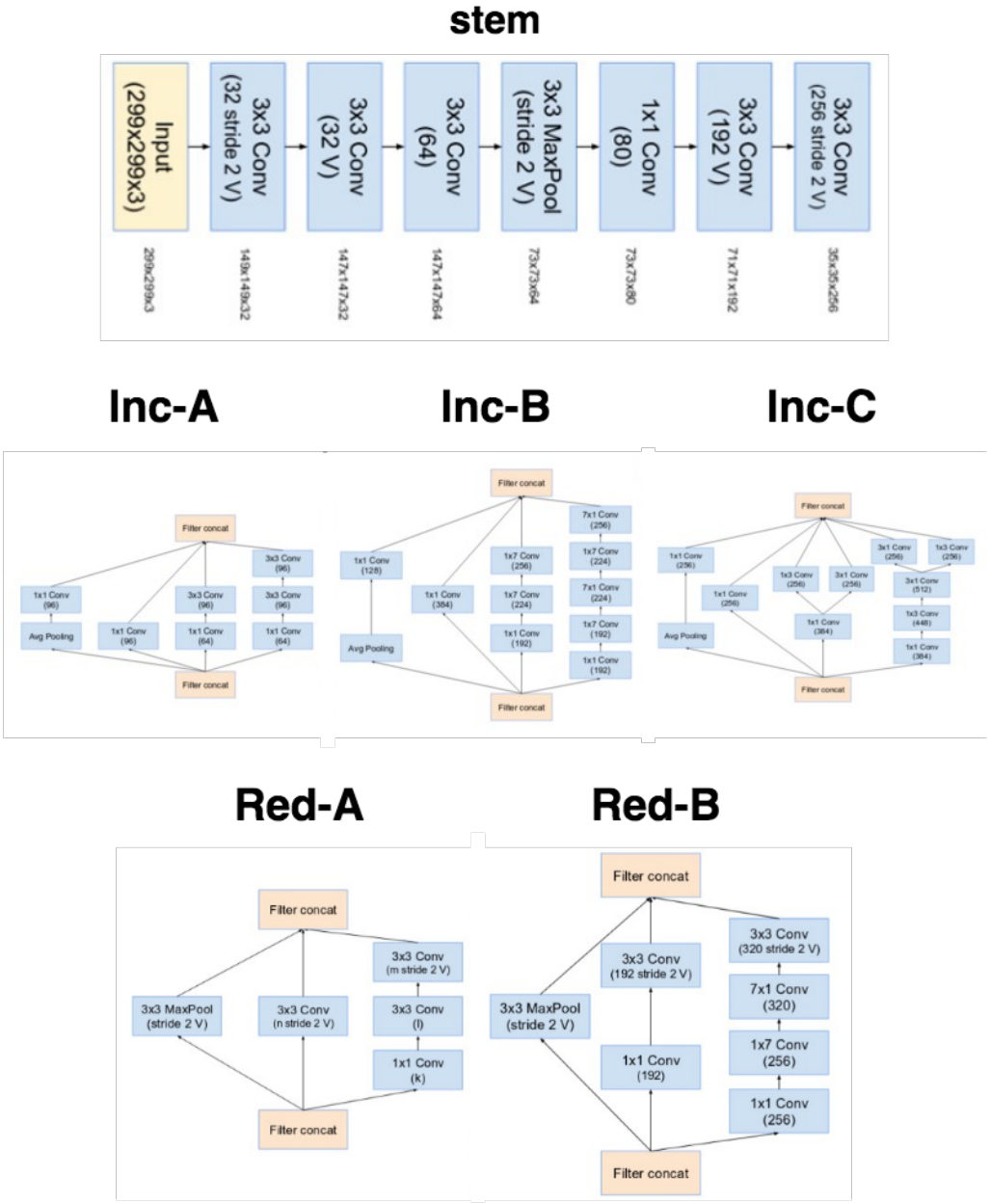


Figura 3.19. (1) Bloco *Stem* da rede InceptionNet v4. (2) IR-A, IR-B e IR-C: Três tipos de módulo Inception da rede InceptionNet v4. (3) Red-A e Red-B - Bloco de redução de 35x35 para 17x17, e 17x17 para 8x8, respectivamente.

Inspirados na performance da rede ResNet [He et al. 2016] (vencedora do desafio ImageNet 2015), os autores do InceptionNet criaram duas redes hibridas chamadas Inception-Resnet v1 e v2 [Szegedy et al. 2017]. A rede Inception-Resnet v1 tem custo computacional semelhante a rede Inception v3, enquanto a rede Inception-Resnet v2 tem custo computacional semelhante a rede Inception v4. A Figura 3.20 ilustra a arquitetura das redes, ambas possuem a mesma estrutura para os módulos Inception-Resnet A, B e C e blocos de redução. As diferenças são os seus blocos *Stem* e hiper-parâmetros. O Inception-Resnet v1 usa o mesmo *Stem* da versão Inception v4, enquanto o Inception-Resnet v2 propõe o novo *Stem* que pode ser visto na Figura 3.21.

A principal ideia da arquitetura Inception-ResNet é a adição das conexões residuais propostas pela rede ResNet. A Figura 3.21 detalha os módulos da arquitetura Inception-Resnet. Para que a incorporação da conexão residual funcione é necessário que a entrada e a saída sejam concatenadAs, e portanto que tenham a mesma dimensão. Para isso, foi adicionada uma convolução 1x1 após as convoluções tradicionais do módulo Inception para padronizar os tamanhos do mapas de *features*. A operação de *pooling* do Inception foi removido em favor da conexão residual. No entanto, essa operação ainda é presente nos blocos de redução A e B. Os autores contataram que a rede tende a instabilidade quando o a rede excede mil filtros, para estabilizar a rede eles escalaram as ativações residuais por valores entre 0.1 e 0.3.

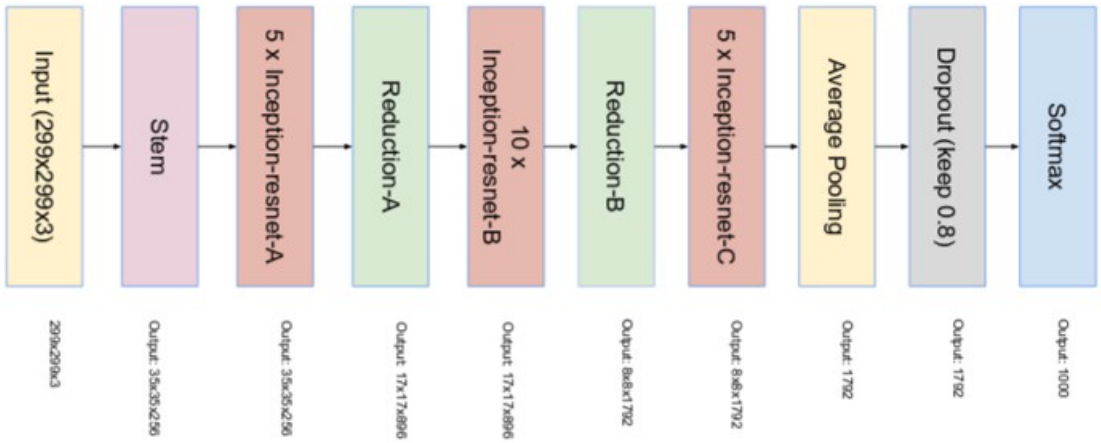


Figura 3.20. Arquitetura das redes Inception-ResNet v1 e v2.

3.6. Redes Neurais Recorrentes

As redes neurais recorrentes, ou RNNs, do inglês *Recurrent Neural Networks* [Rumelhart et al. 1986], são um conjunto de redes neurais especializadas em processamento de dados sequenciais de diferentes tamanhos. Esse tipo de rede são escaláveis para grande sequencias de dados, diferentemente das redes que não são especializadas neste tipo de tarefa. Neste contexto, dados sequenciais são dados ordenados nos quais os valores estão relacionados, como dados temporais, textuais, uma sequência de DNA, dentre outros.

Esta família de redes neurais, diferente do que ocorre nas redes do tipo *feed-forward*, na qual o fluxo de informação tem apenas um sentido, passando de uma camada para a outra, se baseiam na passagem da informação do instante t para o instante $t+1$, assim, as redes recorrentes possuem uma memória do seu estado anterior que é utilizada no processamento do estado atual. Este conceito pode ser observado na Figura 3.22 (a). Esta característica é o que torna estes tipos de redes especiais, uma vez que a sequencia de dados contém informações essenciais sobre o que pode ser observado no próximo instante.

Uma outra maneira de enxergar esta passagem de informação é através do desdobramento da recursão. Considerando que o estado de um nó na rede pode ser dado

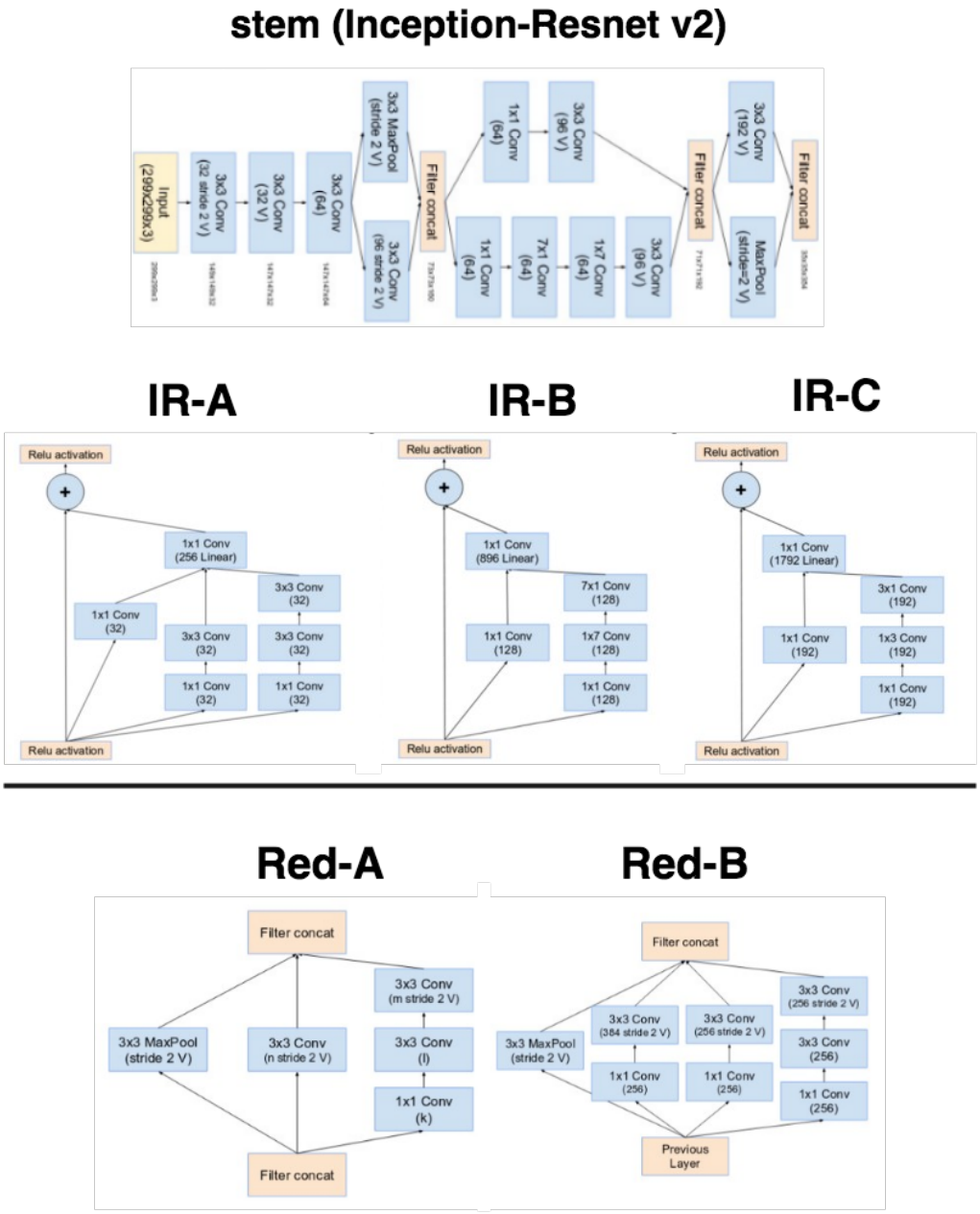


Figura 3.21. (1) Bloco stem da rede Inception-Resnet v2. (2) IR-A, IR-B e iR-C: Três tipos de módulo Inception-Resnet da arquitetura Inception-Resnet v1 e v2. (3) Red-A e Red-B - Bloco de redução de 35x35 para 17x17, e 17x17 para 8x8, respectivamente.

por:

$$s^t = f(s^{t-1}, x^t; \theta),$$

ou seja, o estado atual, s^t, é obtido através de uma função que depende do estado anterior, s^{t-1}, e um sinal externo x^t parametrizados por θ. Para obter o estado no instante T, é possível desdobrar a rede aplicando a operação $T-1$ vezes [Goodfellow et al. 2016]. Este processo de desdobramento é ilustrado na Figura 3.22 (b).

Utilizando os conceitos de passagem de informação entre os instantes e a repre-

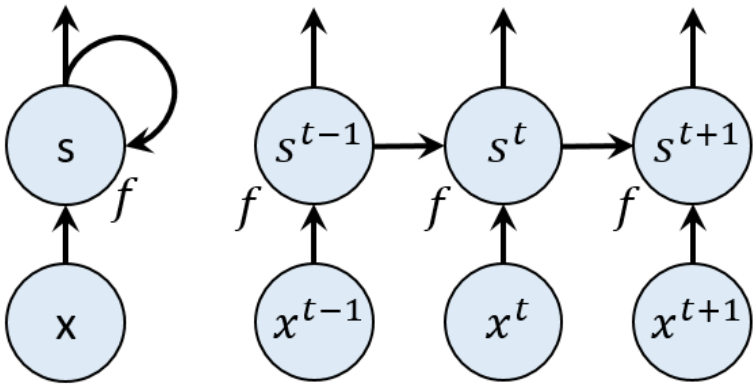


Figura 3.22. Um exemplo de RNN que utiliza a entrada x no instante t juntamente com o estado no instante $t-1$ para processar a saída no instante t. Em (a) a rede é representada com ciclo ou recursão e em (b) a mesma é visualizada com a recursão desdobrada.

sentação desdobrada é possível projetar diversos tipos de redes recorrentes, por exemplo uma rede que possua recorrência nas camadas escondidas e produza uma saída em cada instante (Figura 3.23 (a)) ou que processe todos os dados e gere uma única saída no final (Figura 3.23 (b)).

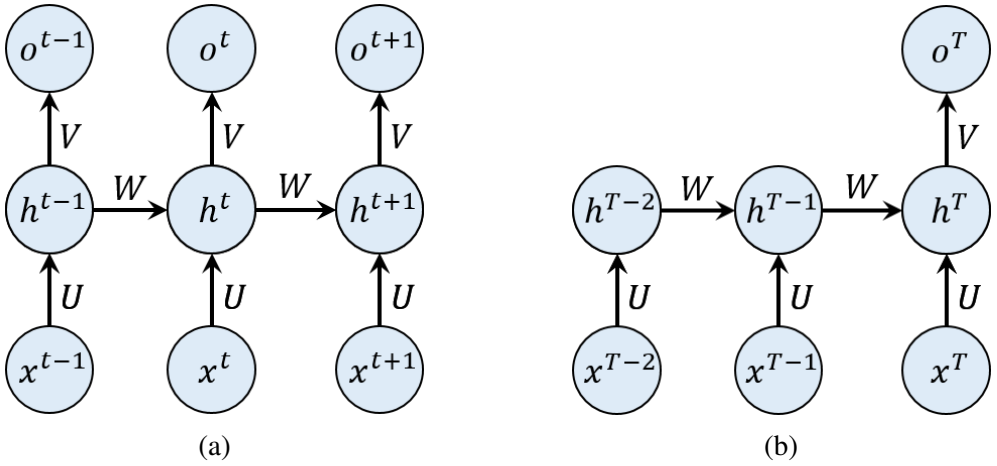


Figura 3.23. Exemplos de RNNs, onde uma sequencia de dados x são processados pela camada escondida h para obter a(s) saída(s) o. Nestas redes, U, W e V são as matrizes de pesos para a entrada, recursão e saída respectivamente. Em (a), é apresentada uma rede com uma saída em cada instante. Já em (b), é produzida uma única saída ao final do processamento.

O cálculo do gradiente neste tipo de rede se da através da aplicação do algoritmo geral de *backpropagation* na representação desdobrada da rede, sendo chamado assim de *Backpropagation Through Time*, ou BPTT. O gradiente então pode ser utilizado por qualquer técnica baseada em gradientes para treinar a RNN.

Os problemas de explosão e diluição de gradiente ao treinar a rede para aprender *long-term dependencies*, apesar de serem evitados por redes do tipo *feed-forward* pois utilizam matrizes de peso diferentes entre as suas camadas [Sussillo 2014], aparecem nas

redes recorrentes uma vez que a mesma matriz W é utilizada em cada passo [Goodfellow et al. 2016]. Diversas estratégias foram desenvolvidas para amenizar o problema de aprendizado de *long-term dependencies*, dentre elas as redes com sistema de portões, que até o presente momento se mostraram as mais eficazes na prática [Goodfellow et al. 2016], incluindo o modelo **LSTM** e redes baseadas em ***Gated Recurrent Unit*** (**GRU**).

As redes com sistema de portões se baseiam na criação de caminhos através dos instantes, nos quais os gradientes não explodem nem diluem, utilizando conexões com pesos que podem variar de acordo com o tempo. Essas conexões permitem que as redes agreguem informação ao longo do tempo e, após utilizarem essa informação, ela pode ser descartada, dessa forma, os portões são responsáveis pelo fluxo de informação dentro da rede. Esses detalhes podem ser observados em uma célula do tipo LSTM, por exemplo, na qual existem 3 portões: *Forget Gate*, *Input Gate* e *Output Gate*, com seu funcionamento esquematizado na Figura 3.24.

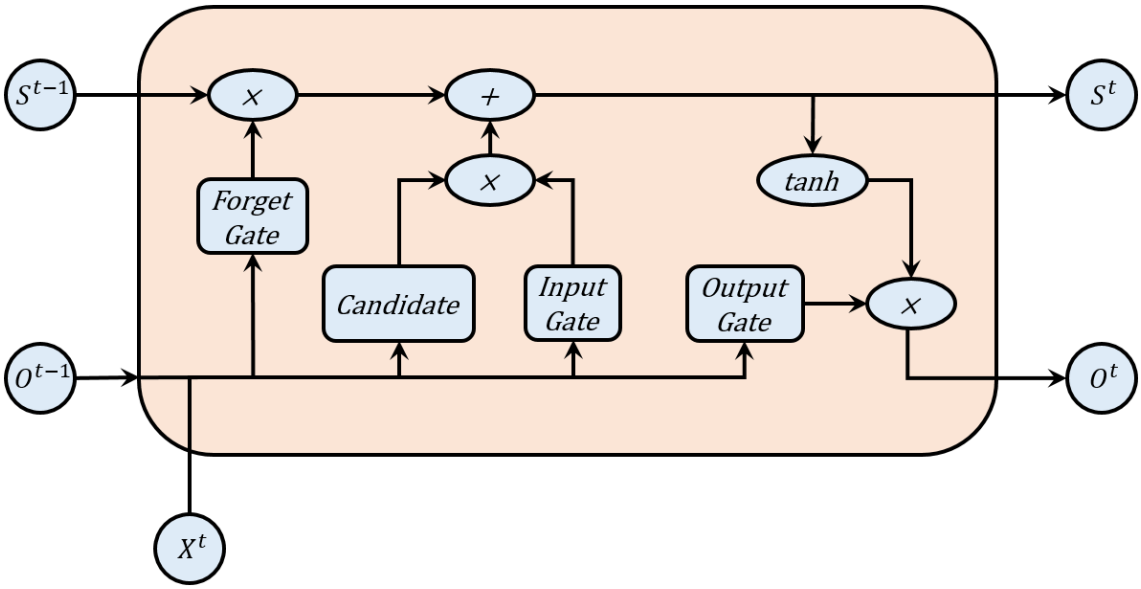


Figura 3.24. Representação de uma célula LSTM, onde S^t representa o estado da célula no instante t, X^t o vetor de entrada no instante t e O^t o vetor com a saída produzida pela célula no instante t.

Dentro de uma célula LSTM, o *forget gate* (f^t) é responsável por decidir o que deve ser removido do estado anterior $t-1$, mantendo apenas o que é relevante para o próximo passo, aplicando uma função *sigmoid* para estabelecer os pesos finais entre 0 e 1. Este portão pode ser dado por:

$$f^t = \sigma(\sum_j W_j^f O_j^{t-1} + \sum_j U_j^f X_j^t + b^f),$$

onde W^f é o vetor de pesos recorrente, U^f é o vetor de pesos para a entrada e b^f é um vetor de *bias*.

No esquema apresentado na Figura 3.24, é possível observar a presença de uma camada chamada *candidate*. Esta camada é responsável por produzir os novos possíveis

valores que serão adicionados ao estado da célula. De maneira similar ao *forget gate*, esta camada utiliza a entrada e a saída do instante anterior parametrizados por vetores de peso W^C e U^C, porém utiliza a função tanh para obter valores entre -1 e 1. Assim, o *candidate* (C^t) é obtido como:

$$C^t = \tanh(\sum_j W_j^C O_j^{t-1} + \sum_j U_j^C X_j^t + b^C)$$

Por sua vez, o *input gate* é responsável por decidir quais dessas novas informações que serão adicionadas no estado da célula, e, de maneira análoga ao *forget gate*, o *input gate* é dado por:

$$i^t = \sigma(\sum_j W_j^i O_j^{t-1} + \sum_j U_j^i X_j^t + b^i)$$

Dessa forma, o estado da célula é atualizado através da combinação dos valores produzidos pela camada *candidate*, parametrizados pelo *input gate*, com os valores do estado no instante anterior, parametrizados pelo *forget gate*, ou seja:

$$S^t = f^t S^{t-1} + C^t i^t$$

Por fim, para obter a saída da célula, a função tanh é aplicada no estado da célula (S^t), obtendo valores entre -1 e 1. Em seguida, o valor obtido é submetido ao *output gate*, que é responsável por decidir quais valores serão colocados na saída. De maneira similar aos outros portões, o *output gate* (q^t) é dado por:

$$q^t = \sigma(\sum_j W_j^q O_j^{t-1} + \sum_j U_j^q X_j^t + b^q),$$

assim, a saída da célula (O^t) é definida por:

$$O^t = \tanh(S^t) q^t$$

Em resumo, no modelo LSTM os portões *forget gate*, *input gate* e *output gate*, utilizam a saída do instante anterior juntamente com a entrada do instante atual para decidir quais valores serão, respectivamente, esquecidos, adicionados ou produzidos como saída. Vale ressaltar que os valores de W e U são independentes entre os portões e podem variar de acordo com tempo, o que diminui as chances do gradiente explodir ou diluir durante o BPTT.

Uma forma simplificada da célula LSTM é a GRU ([Cho et al. 2014, Chung et al. 2014]). A principal diferença com o modelo LSTM é que um único portão passa a controlar simultaneamente a decisão de esquecer e atualizar o estado da célula ([Goodfellow et al. 2016]), assim, a célula passa a ter dois portões, o *reset gate* e o *update gate*. Além disso, as células do tipo GRU não possuem um estado S associado, como nas células LSTM. O fluxo de informação dentro de uma célula do tipo GRU é esquematizado na Figura 3.25.

Dentro de uma célula do tipo GRU, o *reset gate* é responsável por decidir o quanto da informação do instante anterior deve persistir para a geração dos valores candidatos na

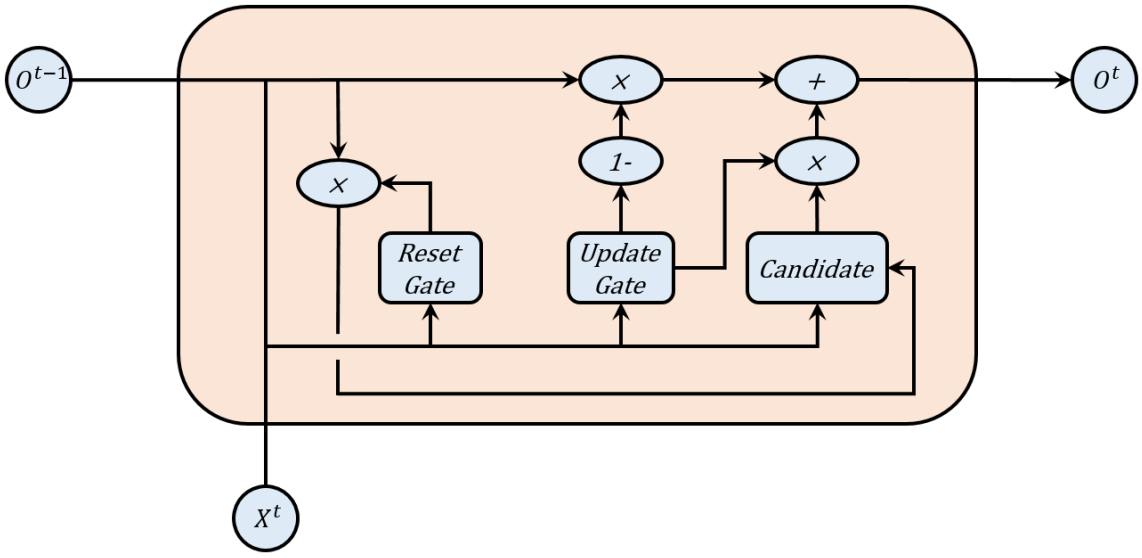


Figura 3.25. Representação de uma célula GRU, onde X^t representa o vetor de entrada no instante t e O^t o vetor com a saída produzida pela célula no instante t.

camada *candidate*. De maneira similar aos portões da célula LSTM, o *reset gate* (r^t) é dado por:

$$r^t = \sigma(\sum_j W_j^r O_j^{t-1} + \sum_j U_j^r X_j^t + b^r)$$

Dessa forma, utilizando o *reset gate*, os valores candidatos (C^t) podem ser obtidos por:

$$C^t = \tanh(\sum_j W_j^c r^t O_j^{t-1} + \sum_j U_j^c X_j^t + b^c)$$

Por sua vez, o *update gate*, é responsável por decidir qual parte da saída anterior vai persistir e qual vai ser atualizada. De maneira análoga ao *reset gate*, o *update gate* (q^t) é dado por:

$$q^t = \sigma(\sum_j W_j^q O_j^{t-1} + \sum_j U_j^q X_j^t + b^q)$$

Utilizando o *update gate* para parametrizar os valores candidatos e o complemento do *update gate* para parametrizar os valores do instante anterior, é possível obter a nova saída O^t através de:

$$O^t = q^t \cdot C^t + (1 - q^t) \cdot O^{t-1}$$

3.6.1. Implementando uma rede LSTM utilizando a base de dados YouTube8M

O YouTube8M [Abu-El-Haija et al. 2016] é um *dataset* de vídeo multi-etiquetado que contém cerca de 6.1 milhões de vídeos do Youtube[12] e um vocabulário com 3820 etiquetas. Para facilitar o desenvolvimento de projetos voltados ao entendimento de vídeo, o Youtube8M fornece as *features* áudio-visuais pré-computadas. As *features* visuais foram extraídas da última camada da rede Inception-Resnet-v2 [Szegedy et al. 2017]

[12]https://www.youtube.com/

pré-treinada com o *dataset* ImageNet [Deng et al. 2009]. Já as *features* de áudio foram extraídas da última camada de versão da rede VGG modificada para áudio [Hershey et al. 2017] pré-treinada com o *dataset* AudioSet [Gemmeke et al. 2017].

Após a extração das *features*, foi aplicado PCA e uma quantização de 8-bits sobre os vetores de *features* para redução de dimensionalidade, resultando em um vetor de 1024 dimensões de *features* visuais e um vetor de 128 dimensões de *features* de áudio. O *dataset* YouTube8M é fornecido em duas versões:

- *Frame-level* (1.71 TB): *features* áudio-visuais extraídas dos primeiros 360 segundos (6 minutos) do vídeo, a uma taxa de 1 *frame* por segundo.
- *Video-level* (31 GB): *features* áudio-visuais extraídas da média do RGB dos *frames* e áudio.

Para efeitos didáticos, apenas uma pequena parte do *dataset* em *frame-level* será utilizado e está disponível no Drive[13] dos autores juntamente com o vocabulário das etiquetas. Com o objetivo de classificar etiquetas, em um vídeo, a arquitetura geral do sistema utilizado é apresentado na Figura 3.26. Nesta arquitetura foi desenvolvido uma rede do tipo LSTM para agregar as *features* áudio-visuais e produzir uma lista de etiquetas para o vídeo. É importante notar que a utilização do *dataset* em *frame-level* é importante para o treinamento deste tipo de rede, uma vez que consiste de uma sequência de informações para cada vídeo, diferente do *dataset* em *video-level*.

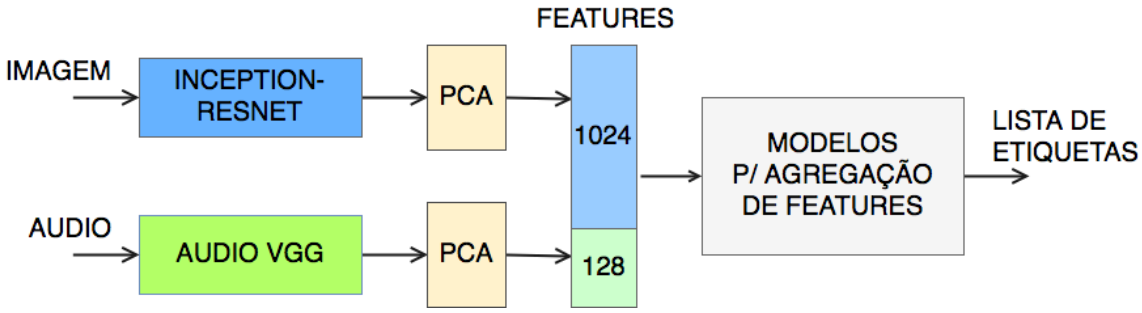


Figura 3.26. Arquitetura geral do sistema de classificação.

A Listagem 3.12 mostra a construção de uma LSTM. A função "build_lstm" recebe como parâmetro o número de camadas que a rede vai ter, o número de células que as camadas LSTM vão ter, o numero de *features* em cada *frame* do vídeo, bem como o número de etiquetas. Entre as linhas 3 e 8, são definidos os *placeholders* da LSTM, onde "X_" vai armazenar a matriz de *features* do vídeo (uma linha por *frame*), "Y_" é o vetor contendo as etiquetas do vídeo e "N_" o número de *frames* do vídeo. Na linha 10 e 11 as camadas de LSTM são construídas e empilhadas na linha 12. Na linha 16 essas camadas são então utilizadas para construir uma RNN desenrolando a matriz de *features*. Por fim o resultado da RNN é conectado a uma camada densa, produzindo um resultado equivalente a multiplicar a saída por pesos e adicionar um *bias*. Na linha 23 a função

[13]https://drive.google.com/open?id=14qkt9y2adFNPg16xpXG-lHoYLZTS_5iL

sigmoid é aplicada ao *logits* para obter a predição final da probabilidade de cada etiqueta estar no vídeo. O restante do código possui as definições da função de custo, otimizador já explicados nos exemplos anteriores deste capítulo.

Listagem 3.12. Construindo uma LSTM.

```
1 def build_lstm(n_layers, n_hidden_cells, n_features, n_labels):
2
3     # matriz de features:
4     X_ = tf.placeholder(dtype=tf.float32, shape=[None, n_features])
5     # vetor de etiquetas:
6     Y_ = tf.placeholder(dtype=tf.float32, shape=[n_labels])
7     # numero de frames:
8     N_ = tf.placeholder(dtype=tf.float32, shape=None)
9
10    lstm_layers = [tf.contrib.rnn.BasicLSTMCell(n_hidden_cells,
11    forget_bias=1.0) for _ in range(n_layers)]
12    lstm_stack = tf.contrib.rnn.MultiRNNCell(lstm_layers)
13
14    model_input = tf.expand_dims(X_, 0)
15    # rnn dinamico para batchs de tamanho variavel
16    outputs, state = tf.nn.dynamic_rnn(lstm_stack, model_input,
17    sequence_length=N_, dtype=tf.float32)
18
19    logits = tf.layers.dense(tf.layers.batch_normalization(outputs[:, -1, :]),
20    n_labels, activation=None,
21 kernel_initializer=tf.contrib.layers.xavier_initializer())
22
23    output = tf.nn.sigmoid(logits)
24
25    # loss function
26    loss = tf.reduce_mean(
27    tf.nn.sigmoid_cross_entropy_with_logits(
28    logits=logits, labels=tf.expand_dims(Y_, 0)))
29
30    # optimizer
31    opt = tf.train.AdamOptimizer(learning_rate=0.001).minimize(loss)
32
33    return opt, loss, output, X_, Y_, N_
```

A Listagem 3.13 mostra o código que carrega o *dataset*, inicia o *TensorFlow* e carrega a LSTM definida na listagem anterior. Na Linha 2 o *dataset* é carregado. Na Linha 8 e 9 o modelo é carregado, utilizando 2 camadas de LSTM com 512 células escondidas cada, o número de *features* é $1024 + 128$, correspondendo respectivamente as *features* visuais e de áudio, o número de etiquetas do *dataset*. Na linha 12 as variáveis do TensorFlow são inicializadas.

Listagem 3.13. Iniciando o TensorFlow e carregando o *dataset* e a LSTM.

```
# Carregando o dataset
load_dataset()

# Iniciando
with tf.Session() as sess:

    # carregando o modelo LSTM
    opt, cost, output, X_, Y_, N_ = build_lstm(n_layers=2,
    n_hidden_cells=512, n_features=1024+128, n_labels=3862)

    # inicializando as variaveis do tensorflow
    tf.global_variables_initializer().run()
```

A Listagem 3.14 mostra o código que realiza o treinamento da LSTM com o *dataset*. Neste exemplo de implementação o modelo é treinado em 5 épocas. Em cada época, como definido nas linha 8, um *batch* é lido. Em seguida, a rede é treinada com o *batch*. O erro do treinamento é impresso a cada época.

Listagem 3.14. Treinando a LSTM.

```
#definindo numero de epocas
num_epochs = 5

for epoch in range(num_epochs):
    last_cost = 0
    #treinando a rede com cada batch
    while True:
        features, labels, n_frames = get_next_train_batch()
        # se n_frames < 0, significa que os batchs acabaram
        # e devemos comecar uma nova epoca
        if n_frames < 0:
            break

        _, last_cost = sess.run([opt, cost],
        feed_dict={X_: features, Y_: labels, N_: n_frames})

        #imprimindo o erro a cada epoca
    print("Erro na epoca", epoch, ":", last_cost)

---------------------------------- OUTPUT ----------------------------------
> Erro na epoca 0 : 0.0039477115
> Erro na epoca 1 : 0.0037346634
> Erro na epoca 2 : 0.00365565
> Erro na epoca 3 : 0.0036228024
> Erro na epoca 4 : 0.0035409592
```

A Listagem 3.15 mostra como utilizar a LSTM recém treinada para realizar classificações multi-etiquetas nos vídeos. Na linha 3 é feita a predição das etiquetas para o

vídeo de entrada. Na linha 5 são obtidos as 5 etiquetas com maior probabilidade. Por fim, a linha 8 mostra a saída do programa.

Listagem 3.15. Usando a LSTM para obter as 5 primeiras *labels* de um vídeo.

```
1 features, labels, n_frames = get_test_example()
2
3 result = sess.run(output, feed_dict={X_: features, Y_: labels, N_: n_frames})
4
5 result_vocabulary = get_top_labels(5, result)
6 print("As 5 primeiras labels sao: ", result_vocabulary)
7 ---------------------------------- OUTPUT ------------------------------------
8 > As 5 primeiras labels sao:  ['Game', 'Cartoon', 'Food', 'Animation',
9 'Slam dunk']
```

3.7. Reconhecimento Facial

O Modelo FaceNet [Schroff et al. 2015] é o estado-da-arte para a tarefa de reconhecimento facial. Este modelo tem um desempenho 99.6% de acurácia no *dataset* LFW (Labeled Faces in the Wild) [Learned-Miller 2014]. A Figura 3.27 ilustra a arquitetura usada pelo FaceNet para realizar o reconhecimento facial. Dada duas imagens de face, o FaceNet extrai um vetor de *features* (*face embedding*) a partir da ativação linear da última camada densa da rede Inception-Resnet-V1 [Szegedy et al. 2017]. Em seguida, a similiraridade entre as imagens de face é calculada pela distancia euclidiana entre os seus vetores de *features*, se a distância for menor que um limiar, então presume-se que as imagens de faces sejam da mesma pessoa.

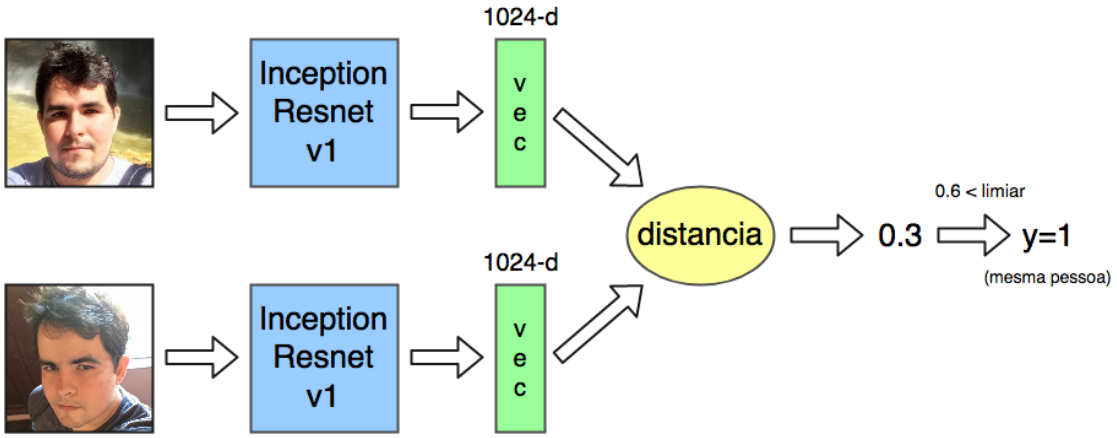


Figura 3.27. Arquitetura usada pelo FaceNet para reconhecimento facial.

Neste minicurso o modelo FaceNet é usado para reconhecimento facial em conjuntos de imagens de faces. Em um sistema empresarial para controle de acesso de funcionários, por exemplo, a câmera de segurança pode capturar a imagem da face do visitante, então o sistema pode verificar se o visitante está registrado como funcionário da empresa para liberar o acesso. Dessa forma, o sistema pode utilizar o FaceNet para extrair o *face embedding* da imagem do visitante, e em seguida, calcular a distância com

os *face embeddings* das imagens dos demais funcionários registrados, como ilustrado na Figura 3.28. A menor distância obtida corresponde a identidade do visitante, se a distância for maior que um limiar m, significa que o visitante não está registrado na base de funcionários.

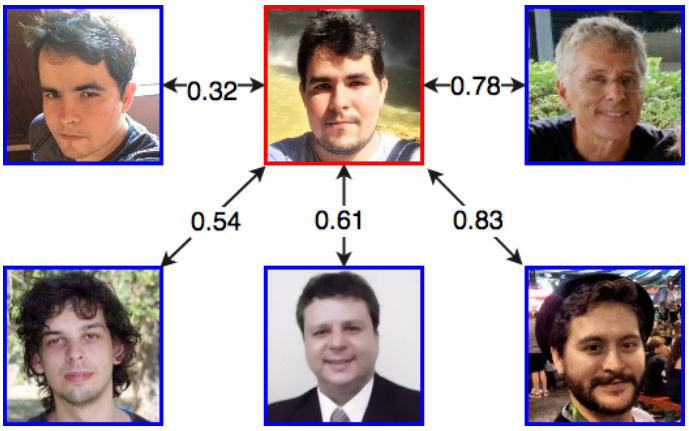


Figura 3.28. Comparação entre as distâncias das imagens do banco de dados (em azul) em relação a imagem de entrada (em vermelho).

A Sub-seção 3.7.1 descreve a função de perda utilizada para treinamento do FaceNet. Em seguida, a Sub-seção 3.7.2 apresenta um cenário de uso onde um modelo pré-treinado do FaceNet é usado na implementação de um sistema de reconhecimento facial.

3.7.1. Função de perda Tríplice

Dada uma imagem x, seu vetor de features faciais é dado por $f(x)$, onde f é a computação realizada pela CNN. A função de perda Trípicle usa um trio de imagens (A,P,N), onde:

- A (âncora) refere-se a uma imagem de face;
- P (positiva) refere-se a imagem de face do mesmo individuo da imagem A;
- N (negativa) refere-se a uma imagem de face de um individuo diferente da imagem A.

O objetivo é fazer a imagem $A^{(i)}$ ficar próxima da sua $P^{(i)}$ e mais distante da $N^{(i)}$ por uma margem α. Dessa forma, a função de perda Tríplice é descrita como:

$$\mathscr{J} = \sum_{i=1}^{m} [|| f(A^{(i)}) - f(P^{(i)}) ||_2^2 - || f(A^{(i)}) - f(N^{(i)}) ||_2^2 + \alpha]_+$$

A notação "$[z]_+$" corresponde a função $max(z,0)$. A contante α é usada para garantir que a rede não tente otimizar para $f(A) - f(P) = f(A) - f(N) = 0$. A Listagem 3.16 descreve a implementação da função tríplice no Tensorflow. Na linha 4 é calculado o primeiro termo da equação, que corresponde a distância entre os vetores de *features* das imagens A e P. Em seguida, na linha 7 é calculado o segundo termo, que corresponde a distância entre os vetores de *features* das imagens A e N. Na linha 10, é calculada a

diferença entre os dois termos anteriores e somado com a margem α. Por fim, na linha 13, todos os erros são somados, ressaltando que os erros menores que 0 são transformados em 0 pela função *max*.

Listagem 3.16. Implementação da função de perda tríplice.

```
1 def triplet_loss(anchor, positive, negative, alpha = 0.2):
2
3     #Passo 1: Calculo da distancia entre A e P
4     pos_dist = tf.reduce_sum(tf.square(tf.subtract(anchor,positive)), axis=-1)
5
6     #Passo 2: Calculo da distancia entre A e N
7     neg_dist = tf.reduce_sum(tf.square(tf.subtract(anchor,negative)), axis=-1)
8
9     #Passo 3: Subtracao dos dois termos anteriores e depois soma com alpha.
10    basic_loss = tf.add(tf.subtract(pos_dist, neg_dist), alpha)
11
12    # Step 4: Somatorio do max entre o basic_loss e 0
13    loss = tf.reduce_sum(tf.maximum(basic_loss, 0.0))
14
15    return loss
```

3.7.2. Cenário de Uso: Sistema de Controle de Acesso com Reconhecimento Facial

A tarefa de reconhecimento facial tenta responder a pergunta "Quem é essa pessoa?". Neste cenário de uso implementamos um sistema de controle de acesso que usa o FaceNet para realizar o reconhecimento facial dos funcionários de uma empresa, e então, permitir seu acesso.

O modelo FaceNet utilizado nesta implementação está disponível para download na sua página oficial no GitHub[14]. Adicionalmente, é possível utilizar modelos pré-treinados no FaceNet. Na página oficial há dois modelos pré-treinados disponíveis, um treinado no dataset CASIA-WebFace[15], e outro treinado no dataset VGGFace2[16].

A Listagem 3.17 mostra como o sistema de reconhecimento utiliza um modelo pré-treinado do FaceNet. Na linha 6 é feito o carregamento do modelo. Em seguida, nas linhas 9-11, são selecionados os tensores necessários para uso do modelo. Na linha 13 é definida a função que usa o FaceNet para extrair o *embedding* de uma imagem facial. Entre as linhas 16-18 é feito o redimensionamento da imagem para garantir a compatibilidade com as dimensões do tensor de entrada. Por fim, na linha 19 são atribuídos os valores dos tensores de entrada (placeholders) do modelo, e na linha 20, o *embedding* da imagem é obtido com a execução do FaceNet.

Listagem 3.17. Usando o FaceNet pré-treinado.

```
1 import facenet
2
```

[14] https://github.com/davidsandberg/facenet
[15] https://drive.google.com/open?id=1R77HmFADxe87GmoLwzfgMu_HY0IhcyBz
[16] https://drive.google.com/open?id=1EXPBSXwTaqrSC0OhUdXNmKSh9qJUQ55-

```
3 sess = tf.Session()
4
5 #Carregando do modelo pre-treinado
6 facenet.load_model("20170512-110547/20170512-110547.pb")
7
8 #Selecionando os tensores necessarios para execucao
9 images_placeholder = tf.get_default_graph().get_tensor_by_name("input:0")
10 embeddings = tf.get_default_graph().get_tensor_by_name("embeddings:0")
11 train_placeholder = tf.get_default_graph().get_tensor_by_name("phase_train:0")
12
13 def get_embedding(img):
14     img_size = 160
15     #Preparando a imagem de entrada
16     resized = cv2.resize(img, (img_size,img_size),interpolation=cv2.INTER_CUBIC)
17     reshaped = resized.reshape(-1,img_size, img_size,3)
18     #Configurando entrada e execucao do FaceNet
19     feed_dict = {images_placeholder: reshaped, train_placeholder: False}
20     embedding = sess.run(embeddings, feed_dict=feed_dict)
21     return embedding
```

Como primeira etapa da implementação do sistema, é necessário realizar o registro dos *embeddings* das imagens faciais dos funcionários. As linhas 2-6 da Listagem 3.18 mostram como usar a função descrita anteriormente para registrar os *embeddings*. Em seguida, nas linhas 9 é definida a função que calcula a distância euclidiana entre dois vetores. Essa função é utilizada para calcular a similaridade entre os *embeddings* das imagens.

Listagem 3.18. Usando o FaceNet para extrair os embeddings das imagens.

```
1 #Registrando os embeddings das pessoas
2 database = {}
3 database["antonio"] = get_embedding("faces/antonio.png")
4 database["lucas"] = get_embedding("faces/lucas.png")
5 database["gabriel"] = get_embedding("faces/gabriel.png")
6 database["sergio"] = get_embedding("faces/sergio.png")
7
8 #Funcao que calcula a distancia euclidiana entre dois vetores
9 def distance(vector1, vector2):
10         return np.sqrt(np.sum((vector1-vector2)**2))
```

Na segunda etapa, é implementada a função que identifica a foto do visitante a partir das fotos dos funcionários registrados. Como pode ser visto na Listagem 3.19, a função "who_is_it" recebe como parâmetro o caminho da imagem facial do visitante e a lista com os *embeddings* dos funcionários registrados. Na linha 6 é calculado o *embedding* da imagem de entrada. Em seguida, nas linhas 9-13 um laço percorre a lista de funcionários registrados, calculando a distância entre o *embedding* do visitante e cada funcionário. Por fim, nas linhas 15-18, é verificado se a menor distância encontrada é

maior que um limiar de 0.7, se sim, significa que a imagem de entrada é da face de uma pessoa não registrada, caso contrário, a identidade é de um funcionário é impressa.

Listagem 3.19. Função que identifica uma imagem de face.

```
1 def who_is_it(visitor_image_path, database):
2
3     min_dist = 1000
4     identity = -1
5
6     #Calculando o embedding do visitante
7     visitor = get_embedding(visitor_image_path)
8     #Calculando a distacia do visitante com os demais funcionarios
9     for index, employee in enumerate(database):
10        dist = distance(visitor, employee)
11        if dist < min_dist:
12                min_dist = dist
13                identity = index
14    #verificando a identidade
15    if min_dist > 0.7:
16        print("Essa pessoa nao esta cadastrada, soltem os caes!")
17    else:
18        print ("Bem vindo(a)",identity ,"!")
```

A Listagem 3.20 exemplifica o uso da função de reconhecimento facial. Na linha 1 foi colocada a imagem facial de uma pessoa que não foi registrada. Em seguida, na linha 2, foi colocada outra imagem facial de uma pessoa registrada. As linhas 5-6 mostram a saída do programa.

Listagem 3.20. Exemplo de uso da função de reconhecimento.

```
1 who_is_it("faces/carlos.png",database)
2 who_is_it("faces/antonio2.png",database)
3
4  ---------------------------------- OUTPUT ----------------------------------
5 > Essa pessoa nao esta cadastrada, soltem os caes!
6 > Bem vindo(a) antonio !
```

3.8. Detecção de Objetos

Nesta seção descrevemos o modelo YOLO (You Only Look Once) [Redmon et al. 2016], considerado o estado-da-arte na tarefa de detecção de objetos. Sua última versão, chamada YOLOv3 [Redmon and Farhadi 2018] obeteve um mAP de 57.9% no dataset COCO [Lin et al. 2014]. O YOLO é ideal para aplicações de tempo-real, visto que é o modelo de detecção de objetos baseado em CNN mais rápido da literatura, chegando a rodar próximo de 30 FPS na GPU Pascal Titan X[17].

[17] https://www.nvidia.com/pt-br/geforce/products/10series/titan-x-pascal/

A Subseção 3.8.1 descreve a arquitetura geral do YOLO. Em seguida, a Subseção 3.8.2 descreve a implementação de um cenário de uso que usa o YOLO para identificar objetos em imagens.

3.8.1. Arquitetura YOLO

O YOLO divide a imagem de entrada em uma grade de $S \times S$ dimensões. Cada célula pode conter B *bounding boxes* (BBs) e *scores* de confiança para cada uma. O score de confiança reflete o quão a rede tem certeza que a BB contém um objeto. Se não existe objetos na célula, então o score de confiança deve ser zero. Caso contrário, o score de confiança deve ser condicionada pela Interseçao sobre União (explicada na Subseção seguinte) entre a BB predita e a BB do *ground truth*, $Pr(Object) * IOU_{pred}^{truth}$.

No YOLO, cada BB contém as seguintes informações: 1) score de confiança da célula conter um objeto; 2) coordenadas da BB (b_x, b_y, b_h, b_w), onde (b_x, b_y) representa o ponto central da BB relativa a uma célula da grade, enquanto (b_h, b_w) representa a altura e largura da BB relativa as dimensões da imagem de entrada; 3) Um vetor de probabilidades $(c_1, c_2, ..., c_n)$ para cada uma das n classes de objetos, onde cada probabilidade de classe é condicionada pela probabilidade da célula conter um objeto, $Pr(Class_i|Object)$.

O *score* da confiança de cada classe em cada BB é dada por:

$$Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth}$$

.

Os *scores* informam: (1) a probabilidade de um objeto de uma classe aparecer na BB, e (2), o quão ajustado está a BB ao objeto. Como ilustra a Figura 3.29, a saída do YOLO é um tensor de dimensões $S \times S \times (B * 5 + C)$. Onde S é a dimensão do *grid* de regiões, B é o número de *bounding boxes* em cada célula, e C é a quantidade de classes no problema. A Figura 3.30 (cima) mostra três visualizações da saída do YOLO. A imagem da esquerda mostra a entrada dividida em uma grade $S \times S$ regiões. A imagem do meio mostra o mapa de probabilidade de classes para cada célula da grade. Por último, a imagem da direita mostra as BBs.

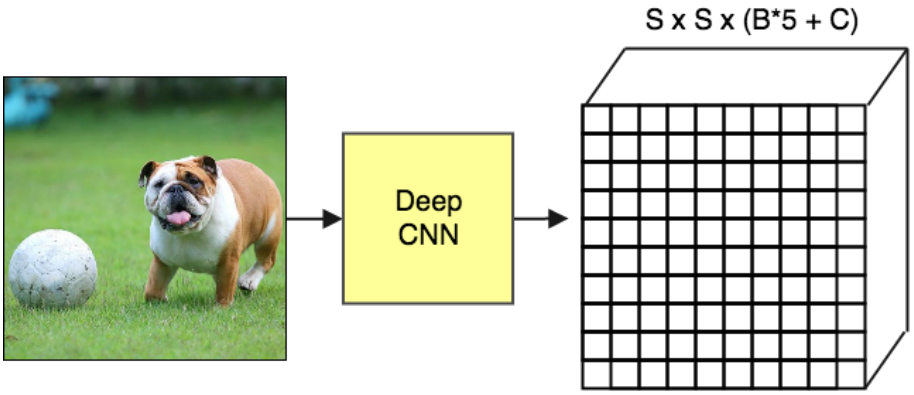


Figura 3.29. Esquema arquitetural do YOLO.

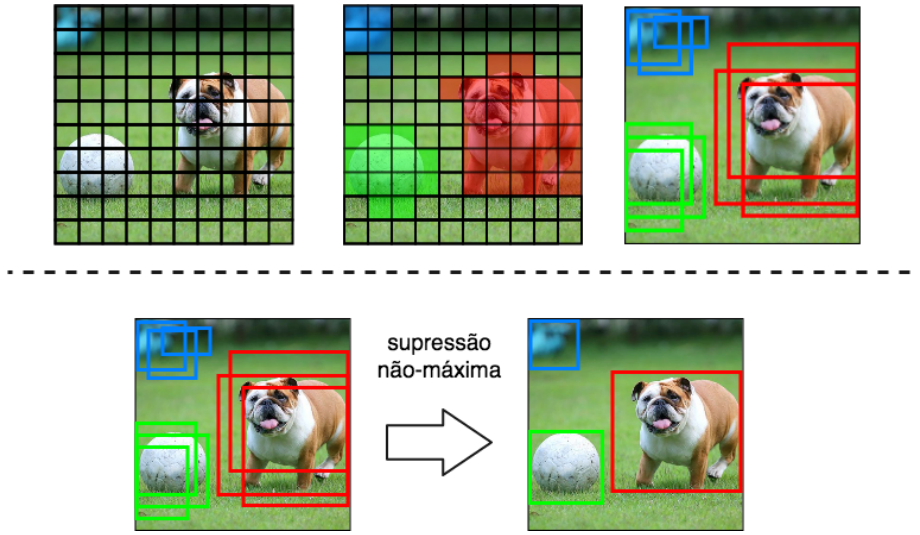


Figura 3.30. (cima) Visualização da saída do YOLO. (baixo) Remoção das BBs sobrepostas com o filtro de supressão não-máxima.

3.8.1.1. Supressão não-máxima

Mesmo com a filtragem pelo *score*, muitas BBs podem ficar sobrepostas uma as outras, como ilustrado na Figura 3.30 (baixo). Um segundo tipo de filtro chamado "supressão não-máxima" é necessário para remover as BBs sobrepostas.

A supressão não-máxima utiliza uma técnica chamada "Interseção sobre União" (em inglês, *IoU - Intersection over Union*). Como pode ser visto na Figura 3.31, essa técnica consiste basicamente em dividir a interseção pela união de duas BBs. A Listagem 3.21 mostra como implementar o IoU, nas linhas 3-7 é calculada a área de interseção entre as BBs. Em seguida, nas linhas 10-12 é calculada a área de união entre as BBs. Por fim, na linha 16 é calculado o Iou pela divisão da área de interseção pela área de união.

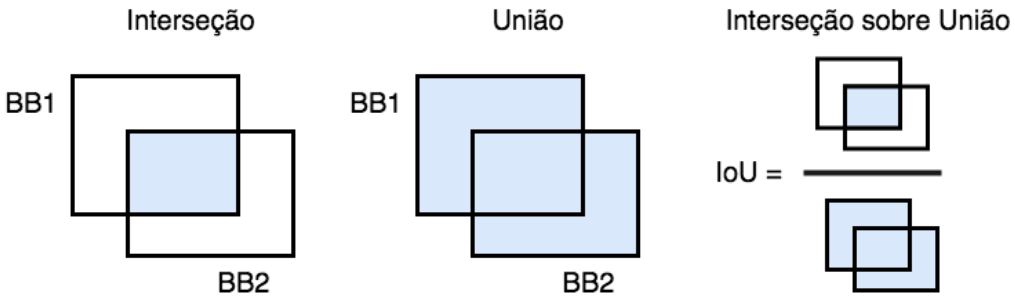


Figura 3.31. Visualização da operação de IoU.

Listagem 3.21. Função de cálculo do IoU.

```
def iou(box1, box2):
    #Calculando a intersecao entre as BBs
    xi1 = max(box1[0], box2[0])
    yi1 = max(box1[1], box2[1])
    xi2 = min(box1[2], box2[2])
    yi2 = min(box1[3], box2[3])
    inter_area = (xi2 - xi1)*(yi2 - yi1)

    #Calculando a uniao usando a formula: Union(A,B) = A + B - Inter(A,B)
    box1_area = (box1[2] - box1[0])*(box1[3] - box1[1])
    box2_area = (box2[2] - box2[0])*(box2[3] - box2[1])
    union_area = box1_area + box2_area - inter_area

    # Calculando o IoU
    iou = inter_area / union_area

    return iou
```

3.8.1.2. Função de perda

Durante o treinamento o YOLO otimiza uma função composta por 5 partes. Cada parte é uma equação que realiza uma tarefa específica.

$$\mathscr{J} = eq1 + eq2 + eq3 + eq4$$

Para aumentar a sua estabilidade, o YOLO aumenta a perda da predições da coordenada das BBs e diminui a perda das BBs que não contém objetos. Para isso, dois parâmetros λ_{coord} e λ_{noobj} são definidos com valores 5 e 0.5, respectivamente.

A equação descrita abaixo calcula a perda relativa a posição **(x,y)** da BB. O 1_{ij}^{obj} denota se a j-ésima BB na i-ésima célula é responsável pela predição do objeto. o YOLO predita múltiplas BB por célula, durante o treinamento somente uma BB é responsável por cada objeto. Então uma BB recebe a responsabilidade de preditar baseado no maior IoU com a BB do *ground truth*.

$$eq1 = \lambda_{coord} \sum_{i=o}^{S^2} \sum_{j=o}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$

A equação descrita abaixo calcula a perda relativa a largura e altura **(w,h)**. A equação é similar a primeira, com a diferença do uso das raízes quadradas para fazer com que pequenas variações em BBs largas importem menos que em BBs pequenas.

$$eq2 = \lambda_{coord} \sum_{i=o}^{S^2} \sum_{j=o}^{B} 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

A equação abaixo calcula a perda associada ao *score* de confiança para cada BB, onde C é o score de confiança e $\hat{C}$ é o IoU entre a BB predita e a BB do *ground truth*. O termo 1_{ij}^{noobj} denota se a j-ésima BB na i-ésima célula não é responsável pelo objeto.

$$eq3 = \sum_{i=o}^{S^2} \sum_{j=o}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=o}^{S^2} \sum_{j=o}^{B} 1_{ij}^{noobj} (C_i - \hat{C}_i)^2$$

A última equação calcula a perda da classificação. Ela é similar a equação de erro de soma quadrada tradicional, mas com a adição do termo 1_i^{obj}. Este termo denota se o objeto aparece na i-ésima célula, é usado para que o erro de classificação não seja penalizado quando o não houver um objeto em uma célula.

$$eq4 = \sum_{i=o}^{S^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

3.8.2. Cenário de Uso: Sistema de Detecção de Objetos

A forma mais fácil para usar o modelo YOLO é importando o *framework* Darkflow (Tensorflow + Darknet[18]). O *framework* Darknet é escrito em C e CUDA[19] e foi usado para a implementação oficial do modelo YOLO. O Darkflow é uma re-implementação em Python do Darknet usando o Tensorflow como base.

Como descreve os comandos abaixo, para instalar o Darkflow basta realizar o download do repositório no Github[20]. E em seguida, realizar a instalação do Darknet usando o Pip. Vale ressaltar que é necessário ter o pacote Cython[21] instalado.

```
git clone https://github.com/thtrieu/darkflow.git
cd darkflow
pip3 install .
```

Após realizar a instalação, para utilizar o pacote basta importa-lo para o projeto, como mostra a Listagem 3.22. Na linha 6, é criado um dicionário chamado *options*, que define os atributos necessários para executar o YOLO pré-treinado no *dataset* COCO. O atributo "model" especifica o caminho do arquivo "yolo.cfg", que define a arquitetura da CNN usada. O atributo "load" define o caminho para o arquivo "yolo.weights", que são os pesos da rede pré-treinada no *dataset* COCO. Esse arquivo pode ser baixado no Drive[22] do autor da rede. O atributo "threshold" define o percentual mínimo para confidencia de detecção de objetos, nesse caso só objetos com pelo menos 10% de *score* de confidencia

[18] https://pjreddie.com/darknet/
[19] https://developer.nvidia.com/cuda-zone
[20] https://github.com/thtrieu/darkflow
[21] http://cython.org/
[22] https://drive.google.com/drive/folders/0B1tW_VtY7onidEwyQ2FtQVplWEU

são retornados. O atributo "gpu" define se o programa pode fazer uso da GPU do sistema. Em seguida, na linha 10, é instanciada uma rede que recebe as opções definidas. É importante ressaltar que tambem é necessário ter o arquivo "coco.names" na pasta "cfg", esse arquivo é encontrado no repositório do Darknet e contém os nomes das classes do *dataset* COCO.

Listagem 3.22. Configurando a aplicação com os dados pré-treinados.

```
1 from darkflow.net.build import TFNet
2 import matplotlib.pyplot as plt
3 import cv2
4 from draw_boxes import *
5
6 options = {"model": "cfg/yolo.cfg",
7            "load": "cfg/yolo.weights",
8            "threshold": 0.1,
9            "gpu": 1.0}
10 tfnet = TFNet(options)
```

A Listagem 3.23 mostra como utilizar o modelo YOLO para detectar objetos. Nas linhas 1 e 2 um imagem é aberta e convertida para RGB. Em seguida, na linha 3, a imagem é usada como entrada da rede. Na linha 4 o resultado é impresso na tela. Por fim, nas linhas 6 e 7 a imagem de entrada é exibida com as BBs identificadas com pelo menos 30% de confidência (Figura 3.32). As linhas 9-10 mostram a saída do programa. Nota-se que o YOLO encontrou seis BBs, das quais apenas duas possuem score de confidência suficiente para ser desenhada.

Listagem 3.23. Usando o YOLO para detectar objetos.

```
1 img = cv2.imread("images/sample1.png")
2 img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
3 result = tfnet.return_predict(img)
4 print(result)
5
6 plt.imshow(boxing(img, result, 0.3))
7 plt.show()
8  ---------------------------------- OUTPUT ----------------------------------
9 [{'label': 'person', 'confidence': 0.15689932, 'topleft': {'x': 314, 'y': 82},
10 'bottomright': {'x': 379, 'y': 253}},
11 {'label': 'person', 'confidence': 0.7260812, 'topleft': {'x': 268, 'y': 64},
12 'bottomright': {'x': 367, 'y': 358}},
13 {'label': 'person', 'confidence': 0.7622834, 'topleft': {'x': 143, 'y': 82},
14 'bottomright': {'x': 235, 'y': 369}},
15 {'label': 'handbag', 'confidence': 0.2231428, 'topleft': {'x': 198, 'y': 178},
16 'bottomright': {'x': 233, 'y': 239}},
17 {'label': 'skis', 'confidence': 0.12105702, 'topleft': {'x': 313, 'y': 103},
18 'bottomright': {'x': 375, 'y': 271}},
19 {'label': 'snowboard', 'confidence': 0.2203493, 'topleft': {'x': 342, 'y': 159},
20 'bottomright': {'x': 371, 'y': 257}}]
```

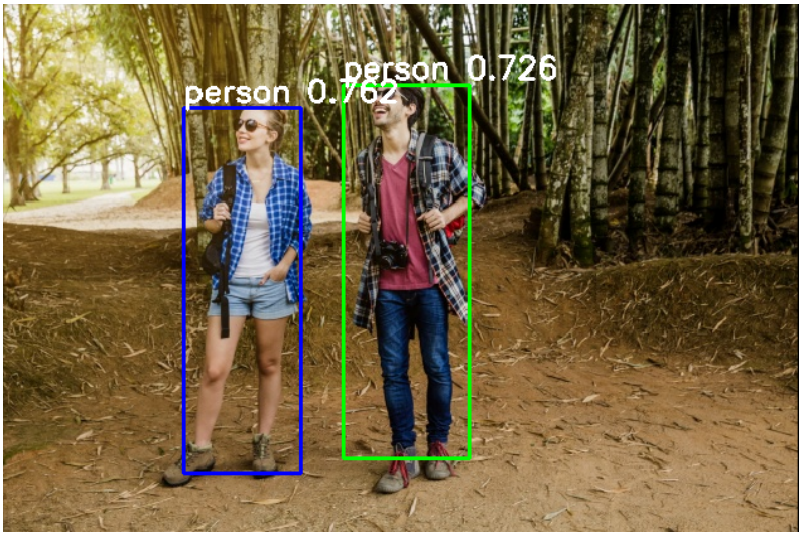


Figura 3.32. Imagem com as *bounding boxes* preditas pelo YOLO.

3.9. Conclusão

Este minicurso apresenta os fundamentos e tecnologias para desenvolver modelos de *Deep Learning*. Em especial, apresenta os modelos de redes neurais baseados em CNNs e LSTMs. Além disso, apresentamos técnicas e métodos de *Deep Learning* para resolução de problemas do domínio de sistemas multimídia. Esperamos que o participante do curso esteja apto para usar *Deep Learning* para realizar classificação de imagens, reconhecimento facial, detecção de objetos e classificação de cenas de vídeo.

Adicionalmente, o minicurso também aborda e evolução das técnicas de *Deep Learning*. Em especial, apresenta a evolução da rede InceptionNet, que é considerada um *milestone* da área e foi a vencedora do desafio ImageNet 2014. As últimas versões da rede apresentada já ultrapassam as capacidades humanas (a acurácia humana no ImageNet está entre 5-10%).

Quanto ao planejamento do curso, foi apresentado um roteiro que inicia com a explicação da instalação e uso básico do *framework* Tensorflow. Em seguida são apresentados os fundamentos de redes neurais e *Deep Learning*, focando principalmente em modelos do tipo CNN e LSTM. O minicurso é concluído com a apresentação de quatro projetos práticos, um de classificação de imagens com sinais de mão usando uma rede CNN, um de classificação de cenas de vídeo usando uma rede LSTM, um de reconhecimento facial usando o modelo FaceNet, e o último, de reconhecimento de objetos usando o modelo YOLO.

Referências

[Abu-El-Haija et al. 2016] Abu-El-Haija, S., Kothari, N., Lee, J., Natsev, P., Toderici, G., Varadarajan, B., and Vijayanarasimhan, S. (2016). Youtube-8m: A large-scale video classification benchmark. *arXiv preprint arXiv:1609.08675*.

[Cho et al. 2014] Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv*

preprint arXiv:1409.1259.

[Chung et al. 2014] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

[Deng et al. 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE.

[Equadex 2018] Equadex (2018). Helpicto. Accessed: 2018-05-18.

[Gemmeke et al. 2017] Gemmeke, J. F., Ellis, D. P., Freedman, D., Jansen, A., Lawrence, W., Moore, R. C., Plakal, M., and Ritter, M. (2017). Audio set: An ontology and human-labeled dataset for audio events. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 776–780. IEEE.

[Goodfellow et al. 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[He et al. 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Hershey et al. 2017] Hershey, S., Chaudhuri, S., Ellis, D. P. W., Gemmeke, J. F., Jansen, A., Moore, C., Plakal, M., Platt, D., Saurous, R. A., Seybold, B., Slaney, M., Weiss, R., and Wilson, K. (2017). Cnn architectures for large-scale audio classification. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

[Kearn and Beeby 2017] Kearn, M. and Beeby, M. (2017). Using cognitive services to make museum exhibits more compelling and track user behavior. Accessed: 2018-05-18.

[Learned-Miller 2014] Learned-Miller, G. B. H. E. (2014). Labeled faces in the wild: Updates and new reporting procedures. Technical Report UM-CS-2014-003, University of Massachusetts, Amherst.

[Lin et al. 2014] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer.

[Ota et al. 2017] Ota, K., Dao, M. S., Mezaris, V., and De Natale, F. G. (2017). Deep learning for mobile multimedia: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 13(3s):34.

[Redmon et al. 2016] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.

[Redmon and Farhadi 2018] Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

[Rosenblatt 1957] Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory.

[Rumelhart et al. 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533.

[Samuel 1959] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229.

[Schroff et al. 2015] Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823.

[Sussillo 2014] Sussillo, D. (2014). Random walks: Training very deep nonlin-ear feed-forward networks with smart ini.

[Szegedy et al. 2017] Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12.

[Szegedy et al. 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

[Szegedy et al. 2016] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

[Thomas and Sadagopan 2016] Thomas, Janki Vora, J. W. and Sadagopan, S. (2016). Use cases for industry cognitive solutions. Accessed: 2018-05-18.

BIO

Antonio José Grandson Busson. Possui graduação (2012) e mestrado (2015) em Ciência da Computação pela Universidade Federal do Maranhão. Atualmente é doutorando em Informática pela Pontifícia Universidade Católica do Rio de Janeiro. Seus interesses de pesquisa incluem: sistemas multimídia/hipermídia, modelos de hiperdocumentos, reconhecimento de padrões e sistemas de TV Digital. Currículo Lattes: `http://lattes.cnpq.br/1857348479447184`.

Lucas Caracas de Figueiredo. Possui graduação (2014) em Ciência da Computação pela Universidade Federal do Maranhão e mestrado (2017) em Informática pela Pontifícia Universidade Católica do Rio de Janeiro. Atualmente é doutorando em Informática pela Pontifícia Universidade Católica do Rio de Janeiro e realiza pesquisa em parceria com o Instituto Tecgraf. Seus interesses em pesquisa incluem: processamento de imagens, sensoriamento remoto, LIDAR, computação gráfica e geometria computacional. Currículo Lattes: `http://lattes.cnpq.br/5353884964040661`.

André Luiz de Brandão Damasceno. Possui graduação (2012) e mestrado (2015) em Ciência da Computação pela Universidade Federal do Maranhão. Atualmente é doutorando em Informática pela Pontifícia Universidade Católica do Rio de Janeiro. Seus interesses de pesquisa incluem: sistemas multimídia, ciência de dados e análise visual. Currículo Lattes: `http://lattes.cnpq.br/0969337931297570`.

Gabriel Noronha Pereira dos Santos. É graduando em Engenharia de Eletricidade na faculdade Wyden. Atualmente trabalha como cientista de dados na Startup Niddu, onde desenvolve projetos de aprendizagem de máquina aplicada a microlearning e sistemas de recomendação. Currículo Lattes: `http://lattes.cnpq.br/8088572506239597`.

Sérgio Colcher. É professor do quadro principal do Departamento de Informática (DI) da PUC- Rio desde 2001 e coordenador do laboratório TeleMídia. Obteve os títulos de Engenheiro de Computação (1991), Mestre em Ciências em Informática (1993) e Doutor em Ciências em Informática (1999), todos pela PUC-Rio, além do Pós-Doutorado (2003) no ISIMA (Institute Supérieur D'Informatique et de Modelisation des Applications — Université Blaise Pascal, Clermont Ferrand, França). Trabalhou no Centro Científico da IBM - Rio e na divisão de desenvolvimento de hardware da COBRA (Computadores Brasileiros S/A). Foi também professor dos cursos de MBA em Gerência de Telecomunicações e MBA em e-Business da Fundação Getúlio Vargas. Suas áreas de interesse incluem redes de computadores, análise de desempenho de sistemas computacionais, sistemas multimídia/hipermídia e sistemas de TV digital. Currículo Lattes: `http://lattes.cnpq.br/1104157433492666`.

Ruy Luiz Milidiú. Bacharel em Matemática pela Universidade Federal do Rio de Janeiro (1974), Mestre em Matemática Aplicada pela Universidade Federal do Rio de Janeiro (1978), M.Sc. em Operations Research - University of California (1983) e Ph.D. em Pesquisa Operacional - University of California (1985). Atualmente, é professor associado da Pontifícia Universidade Católica do Rio de Janeiro e também consultor ad hoc do CNPq, da CAPES, da FAPERJ, da FAPESP e da FINEP. Tem experiência na área de Ciência da Computação, com ênfase em Algorítmica, Aprendizado de Máquina e Complexidade de Computação. Currículo Lattes: `http://lattes.cnpq.br/6918010504362643`.

Capítulo 4

Desenvolvendo Sensores de Vídeo para a Internet das Coisas com o Raspberry Pi

Daniel G. Costa[1]

[1]Universidade Estadual de Feira de Santana (DTEC-UEFS)

danielgcosta@uefs.br

Abstract

The increasing interest for Internet of Things (IoT) technologies has brought a lot of attention to microelectronics and sensors development. With the availability of affordable embedded platforms for countless applications, it is possible to develop low-cost programmable sensors to provide different types of data, specially when cameras are employed. This Chapter covers the fundamentals of visual sensing and presents several technical details when using the Raspberry Pi platform for development of visual sensors. Basic configurations of the Raspberry Pi along with the most popular programs to take image snapshots and to make video streams will be presented. Moreover, the development of Python programs using the Raspberry Pi Camera will be discussed, as well as real-time transmissions of visual data over wireless networks. At the end of this Chapter, readers should know the fundamentals for the creation of highly programmable visual sensors with Raspberry Pi.

Resumo

O crescente interesse por tecnologias ligadas à Internet das Coisas (Internet of Things - IoT) tem trazido muita atenção para o desenvolvimento de sensores e dispositivos microeletrônicos. Com a disponibilidade de plataformas acessíveis para desenvolvimento embarcado, torna-se possível criar soluções de baixo custo para um número incontável de aplicações, especialmente quando câmeras são utilizadas. Este Capítulo aborda os fundamentos do monitoramento por sensores visuais, apresentado diversos detalhes técnicos relacionados à utilização do Raspberry Pi como um sensor visual. Para tanto, configurações básicas do Raspberry Pi e detalhes da utilização de ferramentas para captura de imagens e vídeos serão apresentados. Mais ainda, será discutido o desenvolvimento de programas Python explorando a câmera do Raspberry Pi, além de questões relacionadas a transmissão em tempo real dos dados visuais capturados. Assim, espera-se que ao final deste Capítulo os leitores conheçam os fundamentos para a criação de sensores visuais altamente programáveis.

4.1. Introdução

O desenvolvimento de tecnologias eficientes de sensoriamento e o constante barateamento de dispositivos eletrônicos embarcados vêm permitindo a inserção crescente de milhões de nós inteligentes às redes de comunicação digitais, com previsões de dezenas de bilhões de dispositivos conectados até o final desta década. De fato, já existem mais dispositivos que pessoas conectadas à Internet, permitindo o surgimento de uma nova era para troca e processamento de informações [Atzori et al. 2010, Lin et al. 2017].

De maneira geral, a Internet das Coisas (*Internet of Things - IoT*) é uma das principais fronteiras para o desenvolvimento das redes de comunicação, com potencial para grandes revoluções sociais e econômicas. Nesse novo cenário, além das inúmeras redes privadas que devem ser criadas, governos e organizações irão implantar diversos tipos de dispositivos que poderão ser acessados publicamente na Internet, como sensores pluviométricos, câmeras de segurança, sensores de temperatura, lâmpadas inteligentes, entre muitos outros [Costa et al. 2017]. Esse grande conjunto de informações, gerado e processado diariamente e de forma ininterrupta, irá revolucionar a forma como interagimos com as pessoas, com as cidades e com os dispositivos que nos cercam [Perera et al. 2014, Costa et al. 2018, Andrade et al. 2017]

Um dos elementos centrais do universo IoT é o "sensor", um dispositivo eletrônico com funções bem definidas e que estará frequentemente conectado a uma rede. Em linhas gerais, um sensor é um dispositivo eletrônico desenvolvido para coletar informações do ambiente monitorado, podendo as informações coletadas serem de tipos diferentes de acordo com as unidades de monitoramento utilizadas para a construção do sensor e de sua programação. Os sensores podem então ser interligados de diferentes formas e à diferentes redes, dependendo do seu objetivo no contexto da aplicação desenvolvida [Costa et al. 2015].

Na última década, Redes de Sensores Sem Fio (RSSF) constituíram-se em um dos principais tópicos de pesquisa na área de redes de computadores e comunicação digital. As RSSF são redes *ad hoc* tipicamente compostas por muitos dispositivos operados por bateria e de baixo custo (sensores e/ou atuadores), cada um com recursos de processamento e comunicação sem fio que permitem que operem de forma cooperativa e auto organizável. Como alguns ou todos esses dispositivos podem possuir capacidade para um determinado tipo de monitoramento, as RSSF podem ser utilizadas para uma grande variedade de aplicações inovadoras, permitindo monitoramento em regiões sem infraestrutura, perigosas para acesso humano ou de difícil acesso, revolucionando a forma como dados são obtidos de um ambiente monitorado [Kuo et al. 2018]. Entre as aplicações de monitoramento das RSSF, podemos destacar: controle industrial, monitoramento ambiental e climático, operações de resgate, automação residencial, detecção de poluição, planejamento militar, entre muitas outras [Costa et al. 2017].

Quando sensores são equipados com câmeras de baixo custo e reduzido consumo de energia, dados visuais podem ser obtidos do ambiente monitorado, permitindo um novo escopo de aplicações quando comparado com RSSF tradicionais compostas por sensores de grandezas escalares (temperatura, pressão, luminosidade, umidade, etc). Essas novas Redes de Sensores Visuais Sem Fio (RSVSF) enriquecem a percepção do ambiente monitorado com imagens e/ou vídeos, aperfeiçoando aplicações de monitoramento tradi-

cionais ou mesmo permitindo o surgimento de uma nova gama de aplicações, como em controle de tráfego, localização e rastreamento, segurança pública, previsão climática e monitoramento de desastres, apenas para citar alguns exemplos [Leone et al. 2017].

Independente da forma como serão interligados, da programação relacionada às funções de monitoramento e da quantidade empregada, os sensores serão elementos centrais do universo IoT. E, portanto, disponibilizar formas eficientes e flexíveis de criar sensores torna-se bastante desejável. Nesse cenário, a utilização de plataformas para desenvolvimento embarcado permite que uma extensa gama de sensores possa ser criada de forma facilitada, acelerando atividades de experimentação e prototipação na área de IoT. Este Capítulo, portanto, está destinado a fornecer o *background* necessário para que sensores visuais (também referenciados comumente como sensores de vídeo) sejam facilmente criados na plataforma Raspberry Pi.

De fato, com o advento de plataformas embarcadas para o desenvolvimento de dispositivos eletrônicos genéricos, como, por exemplo, o Raspberry Pi, o BeagleBone e o Intel Edison, é possível criar dispositivos multimídia de monitoramento de forma facilitada [Kruger and Hancke 2014]. Assim, pode-se desenvolver aplicações de monitoramento visual de baixo custo para a Internet das Coisas. Contudo, há diversos detalhes de configuração e operação desses dispositivos que devem ser considerados antes de se escolher qual é a plataforma mais adequada para determinado projeto [Nikhade 2015, Patil et al. 2017]. A Tabela 4.1 apresenta alguns exemplos de plataformas disponíveis para desenvolvimento embarcado que podem ser consideradas para a construção de sensores visuais IoT. Essas plataformas possuem diferentes recursos e preços que podem guiar a escolha da solução mais apropriada para determinado projeto.

Tabela 4.1. Algumas plataformas para construção de sensores visuais IoT.

Plataforma	Lançamento	CPU	RAM	Website
Raspberry Pi 2	2015	900 MHz	1 GB	http://raspberrypi.org
BeagleBone Black	2013	1 GHz	512 MB	http://beagleboard.org
Orange Pi One	2016	1.2 GHz	512 MB	http://www.orangepi.org
CubieBoard 4	2015	2 GHz	2 GB	http://cubieboard.org

Com o crescente aumento do interesse por dispositivos embarcados microprocessados em uma única placa (*single-board computers*), o mercado foi inundando por diferentes modelos de diferentes fabricantes, com significativas variações na capacidade de processamento e armazenamento, na quantidade e tipo de interfaces de entrada/saída e no custo final. Nesse rico cenário, a plataforma Raspberry Pi vem se destacando por sua versatilidade, baixo custo e ampla comunidade de suporte, que coloca essa plataforma como um potencial candidato para o desenvolvimento de dispositivos IoT. Tendo sido inicialmente lançado em 2012 com propósitos educacionais, o Raspberry Pi evoluiu e continua evoluindo, com diferentes modelos sendo lançados e com extensos recursos de software disponíveis. No cenário atual, a plataforma Raspberry Pi oferece amplo suporte para o desenvolvimento de sensores para aplicações da Internet das Coisas [Gupta et al. 2015, Sandeep et al. 2015, Patchava et al. 2015].

A Tabela 4.2 apresenta os principais modelos do Raspberry Pi que foram lançados até meados de 2018. Todos os modelos do Raspberry Pi (com exceção do modelo original,

com 26 pinos) possuem 40 pinos de GPIO (*General Purpose Input Output)* e realizam armazenamento por cartão microSD (os modelos do Raspberry Pi não possuem unidade de armazenamento integrada).

Tabela 4.2. Principais modelos lançados do Raspberry Pi.

Placa	Lançamento	CPU	RAM	Rede	Interfaces E/S
Raspberry Pi Model B	2012	700 MHz	512 MB	Ethernet	HDMI, 2 USB
Raspberry Pi Model A+	2014	700 MHz	256 MB	-	HDMI, 1 USB
Raspberry Pi Model B+	2014	700 MHz	512 MB	Ethernet	HDMI, 4 USB
Raspberry Pi 2 Model B	2015	900 MHz	1 GB	Ethernet	HDMI, 4 USB
Raspberry Pi Zero	2015	1 GHz	512 MB	-	HDMI mini, 1 micro USB
Raspberry Pi 3 Model B	2016	1.2 GHz	1GB	Wi-Fi, Bluetooth, Ethernet	HDMI, 4 USB
Raspberry Pi Zero W	2017	1 GHz	512 MB	Wi-Fi, Bluetooth	HDMI mini, 1 micro USB
Raspberry Pi 3 Model B+	2018	1.4 GHz	1GB	Wi-Fi, Bluetooth, Ethernet	HDMI, 4 USB

A Figura 4.1 apresenta três modelos diferentes do Raspberry Pi. Embora todos os três modelos possam ser utilizados para construir um sensor visual, diferenças na capacidade de processamento e armazenamento podem diretamente interferir na qualidade efetiva do sensor desenvolvido.

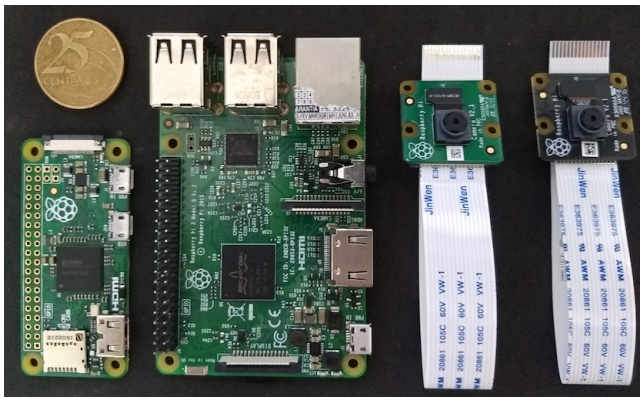


Figura 4.1. Da esquerda para a direita: Raspberry Pi Zero, Raspberry Pi 3 Model B, Pi Camera v2 e Pi Camera v2 NoIR.

4.2. Iniciando no Raspberry Pi

O Raspberry Pi é um pequeno computador construído em uma única placa de circuito, tão pequeno que cabe na palma da mão. Além do tamanho reduzido, o que permite sua utilização prática em diversos cenários da Internet das Coisas, o Raspberry Pi consome pouca energia e possui baixo custo de aquisição, tornando-o um candidato promissor como plataforma versátil para a construção de sensores visuais. De fato, existem muitos detalhes relacionados ao Raspberry Pi e muitos projetos estão sendo criados utilizando essa plataforma, nas mais diversas áreas. Contudo, será dada atenção aos detalhes relacionados a transformar o Raspberry Pi em um sensor de vídeo (que neste Capítulo será considerado como um sensor capaz de transmitir imagens e/ou *streams* de vídeo).

O modelo de referência adotado neste Capítulo é o Raspberry Pi 2 model B (rasp2b), embora os detalhes apresentados também sejam válidos para outros modelos

lançados a partir deste. Para alimentação, recomenda-se utilizar uma fonte fornecendo entre 4.8 V e 5.2 V e 2.5A, sobretudo quando diversos periféricos estejam conectados ao rasp2b. Além do Raspberry, serão necessários neste Capítulo um cartão microSD (classe 10) com pelo menos 4 GB de capacidade de armazenamento e uma câmera HDMI oficial. Periféricos como monitor, teclado e mouse podem ser utilizados de acordo com o nível de conhecimento do leitor e as funcionalidade desejadas, uma vez que o rasp2b pode ser configurado diretamente a partir de um computador auxiliar.

4.2.1. Escolhendo o sistema operacional

Sendo um computador, o Raspberry precisa de um sistema operacional para oferecer serviços aos usuários e coordenar a interação entre processador, memória e unidades de entrada e saída. Para tanto, há diversos sistemas operacionais disponíveis para serem utilizados com o Raspberry Pi, utilizando diferentes sistemas "tradicionais" como referência (por exemplo, Linux, Windows e Android). Entre os sistemas disponíveis, o Raspbian é um sistema operacional baseado no Linux Debian, sendo o sistema operacional oficialmente suportado pela Raspberry Pi Foundation, para todos os modelos Raspberry Pi. Este Capítulo considera o Raspbian como sistema operacional oficial, mais especificamente a versão Raspbian Stretch (lançado em 2017).

Há diversos tutoriais gratuitos online sobre como realizar a instalação do Raspbian. Adicionalmente, a Raspberry Pi Foundation suporta o NOOBS (New Out Of the Box Software), um instalador de sistemas operacionais (algumas opções de SO são disponibilizadas) para facilitar o uso do Raspberry por iniciantes. De qualquer forma, será considerado neste Capítulo a utilização do Raspbian como sistema operacional padrão, sendo o Raspbian Stretch completo (*Desktop*) a versão utilizada como referência para os experimentos realizados.

4.2.2. Configurações iniciais

Há algumas configurações que devem ser feitas logo após a primeira execução do sistema operacional. Essas configurações são necessárias para a correta execução dos exemplos deste Capítulo. Embora a indicação da instalação seja feita quando necessário, nas seções apropriadas, podem-se instalar todos os pacotes necessários logo de início. Assim, a sequência de comandos a seguir instala todos os programas, bibliotecas e dependências necessárias neste Capítulo (permissão de "root" é exigida para execução desses comandos):

```
$ apt-get update
$ apt-get upgrade
$ apt-get install gpac vlc mplayer
$ apt-get install hostapd dnsmasq apache2
```

4.2.3. Comandos básicos

Como o Raspbian é baseado no sistema Debian, a maioria dos comandos disponíveis para o *bash* do Linux estão também disponíveis no Raspberry. A interface de interação por linha de comando na versão simplificada do Raspbian (*Lite*) é, por padrão, o *bash*. Já para a versão completa (*Desktop*), o terminal de linha de comando pode ser facilmente acessado.

De qualquer forma, a utilização de comandos diretamente no *bash* garante mais flexibilidade de configuração, além de permitir acesso e controle remoto do Raspberry através do protocolo SSH. Portanto, conhecer os comandos disponíveis pode facilitar sobremaneira a utilização do Raspberry como sensor de vídeo.

A Tabela 4.3 apresenta alguns dos principais comandos de configuração utilizados na construção de sensores de vídeo com o Raspberry Pi. Para facilitar a organização de tais comandos, a Tabela 4.3 define a seguinte classificação: Sistema e Redes.

Tabela 4.3. Alguns comandos do *bash* para a construção de sensores visuais.

Comando	Tipo	Descrição
apt-get	Sistema	Baixa e instala novos programas e bibliotecas
chmod	Sistema	Altera as permissões de acesso e execução de arquivos
date	Sistema	Exibe informações de data e hora
df	Sistema	Apresenta informações de uso do cartão microSD
hostname	Sistema	Exibe o nome definido para o sistema
free	Sistema	Apresenta o uso da memória do sistema
ip	Rede	Usado para exibir e alterar informações das interfaces de comunicação em rede (Ethernet e Wi-Fi) e da tabela de rotas
iwconfig	Rede	Usado para exibir e alterar informações das interfaces de comunicação em rede sem fio
iwgetid	Rede	Exibe o id da rede sem fio que está oferecendo conectividade ao Raspberry
iwlist	Rede	Lista as redes Wi-Fi atualmente disponíveis
lsusb	Sistema	Lista os dispositivos USB conectados ao Raspberry
netcat (nc)	Rede	Comando versátil utilizado para abrir conexões em rede
nslookup	Rede	Comando utilizado para interação com o serviço DNS
ping	Rede	Utilizado para teste de conectividade (mensagens ICMP)
poweroff	Sistema	Desliga imediatamente o sistema
reboot	Sistema	Reinicia imediatamente o sistema
scp	Rede	Comando utilizado para realizar transferências seguras de arquivos pela rede (protocolo SSH)
ssh	Rede	Permite o acesso seguro via terminal à um dispositivo remoto (protocolo SSH)
tar	Sistema	Programa utilizado para agregar e compactar arquivos
uname	Sistema	Exibe informações do sistema operacional em execução
wget	Rede	Realiza o download de arquivos presentes ma Web

Alguns comandos serão mais frequentemente utilizados para a construção e operacionalização de sensores visuais em aplicações IoT. Por exemplo, os seguintes comandos apresentam igualmente o endereço IP atualmente associado ao Raspberry, mudando apenas a quantidade de informações exibidas:

```
$ ip -4 addr show
$ hostname --all-ip-address
$ ip addr list
```

Já o seguinte comando lista as redes sem fio disponíveis:

```
$ iwlist wlan0 scan
```

Uma rede sem fio pode ser definida diretamente no arquivo de configuração /etc /wpa_supplicant/wpa_supplicant.conf, no caso de uma configuração realizada, por exemplo, dentro de um *script*.

Uma configuração importante para sensores visuais é a definição de um endereço IP fixo. Embora tal configuração possa ser definida em um servidor DHCP, por exemplo associando endereços MAC à endereços IP, usuários podem estabelecer no próprio Raspberry qual endereço IP deve ser utilizado. Uma forma usual é alterando o arquivo "/etc/dhcpcd.conf". Outra forma que pode ser utilizada é apresentada no exemplo a seguir, que altera o endereço IP da interface conectada à rede Wi-Fi (wlan0).

```
$ ip addr add 10.1.1.100 dev wlan0
```

Embora esses e muitos outros comandos sejam bastante úteis, muitos usuários podem se sentir mais confortáveis com a utilização de ferramentas integradas de configuração, como o comando *raspi-config*. Esse comando irá abrir uma tela de configuração para importantes configurações do sistema.

Por fim, assim como ocorre com comandos no Linux, pode-se automatizar a execução de diversos comandos através de *scripts*, diretamente no *bash* ou utilizando a linguagem de programação Python. Contudo, deve-se lembrar que muitos do comandos necessários podem necessitar de acesso privilegiado de super usuário (root). O usuário padrão "sudo" do Raspbian Stretch é *pi* e a senha padrão desse usuário é *raspberry*. Como muitos comandos exigem acesso privilegiado, deve-ser ter bastante cuidado ao executar comandos com permissão de "root".

4.2.4. Câmera do Raspberry Pi

Após o lançamento da plataforma Raspberry e com o rápido sucesso obtido, foi lançado o Raspberry Pi Camera Module em 2013. Essa é uma câmera de baixo consumo de energia porém com "boa" definição, sendo pequena e flexível o suficiente para acompanhar a filosofia de desenvolvimento do Raspberry. Como foi desenvolvida pelo mesmo fabricante do Raspberry, a Raspberry Pi Camera (chamada aqui apenas de RaspCamera) é 100% compatível e pode ser facilmente conectada através de uma interface CSI (*Camera Serial Interface*), presente em todos os modelos atuais. Em 2016, a segunda versão da RaspCamera foi lançada, juntamente com a versão infra-vermelha (NoIR).

A RaspCamera V2 é a referência para os exemplos descritos neste Capítulo. A Figura 4.1 apresenta a RaspCamera V2 "padrão"e a infra-vermelha (NoIR).

Para conectar a RaspCamera, deve-se primeiramente desligar o Raspberry. A câmera é conectada diretamente na interface CSI, removendo o ajuste de plástico e encaixando a câmera de modo que os conectores do cabo e do Raspberry entrem em contato. Por fim, deve-se habilitar a câmera no Raspbian (a câmera é desabilitada por padrão), utilizando o painel de controle ou o comando *raspi-config*. A RaspCamera deve ser habilitada na opção "interface".

A Figura 4.2 apresenta um Raspberry Pi 2 model B com uma RaspCamera V2

conectada à interface CSI.

Figura 4.2. Câmera conectada ao Raspberry pela interface CSI.

Embora câmeras USB convencionais possam ser conectadas ao Raspberry, ou mesmo qualquer tipo de câmera através da GPIO, a RaspCamera trás diversas vantagens para aplicações baseadas na captura de imagens e/ou vídeos. De fato, a comunicação através da CSI, o tamanho reduzido, a "boa" resolução e o baixo consumo de energia colocam essa câmera como uma excelente opção para a construção de sensores visuais com o Raspberry Pi.

A Tabela 4.4 apresenta algumas das especificações da RaspCamera V2.

Tabela 4.4. Características básicas da RaspCamera V2.

Resolução máxima de imagem	3280 x 2464 (8 MP)
Resolução máxima de vídeo	1920 x 1080 (HDMI)
Codecs de imagem	JPEG, GIF, BMP, PNG, YUV420 e RGB888
Codec de vídeo	h.264 (*raw*)
FoV horizontal	62.2°
FoV vertical	48.8°

Como comentário final, para o Raspberry Pi Zero deve-se adaptar o cabo da câmera, porque a interface CSI é menor que no Raspberry Pi 2 e 3. Para tanto, pode-se adquirir adaptadores específicos para esse fim.

4.3. Trabalhando com imagens

Para iniciar no uso da câmera do Raspberry, que é um recurso fundamental para a construção de sensores visuais, serão apresentados exemplos de uso do programa *raspistill*. Esse programa é padrão no Raspbian e possui diferentes parâmetros de configuração para diversas funções relacionadas a obtenção de imagens pela câmera do Raspberry.

Digitando o comando a seguir, é apresentada um tela de “preview” por tempo indeterminado. Essa opção é útil para ver se a câmera está funcionando corretamente e qual é a imagem que será capturada.

```
$ raspistill -t 0
```

Para recuperar uma imagem instantânea atraves da câmera (“foto”), deve-se especificar o nome do arquivo que irá salvar a imagem, através da opção “-o”. O comando a seguir apresenta um exemplo do uso dessa opção. A imagem salva pode depois ser facilmente visualizada clicando sobre ela ou através de qualquer programa de reprodução de imagens.

```
$ raspistill -o foto.jpg
```

Quando a opção “-o” é utilizada da maneira apresentada, a tela de *preview* é exibida e a imagem é salva após 5 segundos, na pasta atual. Para que a imagem seja salva após um tempo específico, sem apresentar a tela de *preview*, deve-se executar o programa *raspitill* como descrito a seguir (para 1 segundo de “intervalo”):

```
$ raspistill -n -t 1000 -o foto.jpg
```

Há diversas operações que ocorrem quando a câmera é iniciada, preparada e utilizada para a captura de uma imagem. Com a opção *verbose* habilitada (“-v”) é possível ver todas as configurações estabelecidas (valores padrões são apresentados, quando não explicitamente definidos). A Figura 4.3 apresenta o *print* de uma captura de uma imagem através do comando *raspistill* no Raspbian *Desktop*.

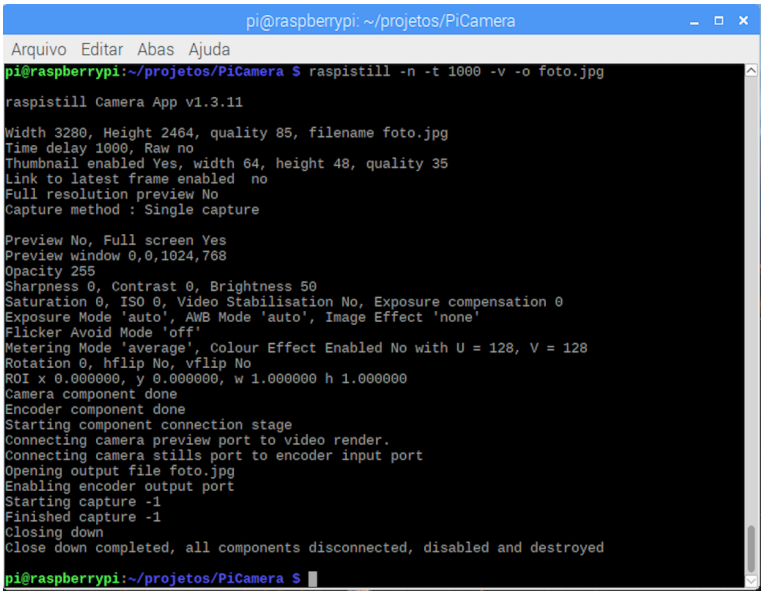


Figura 4.3. Configurações e procedimentos para captura de uma imagem.

Por padrão, o comando *raspistill* salva a imagem na resolução máxima possível, a menos que indicado de outra forma através de parâmetros específicos. O comando a seguir altera a resolução da imagem que será capturada, definindo a imagem no formato VGA.

```
$ raspistill -n -w 640 -h 480 -t 2000 -o foto.jpg
```

Há ainda parâmetros para funções mais complexas. O comando a seguir salva uma imagem P&B no formato PNG (o parâmetro "-cfx" altera o padrão de cores).

```
$ raspistill -n -cfx 128:128 -t 1 -o foto.png -e png
```

Caso a câmera esteja invertida, a imagem pode ser facilmente rotacionada. O comando a seguir apresenta um exemplo que rotaciona a imagem em 180^o.

```
$ raspistill -n -rot 180 -t 500 -o foto.jpg
```

A Tabela 4.5 apresenta uma lista com alguns parâmetros do programa *raspistill*. Para a lista completa de opções, a documentação oficial pode ser consultada gratuitamente online.

Tabela 4.5. Parâmetros de configuração do programa *raspistill*.

Parâmetro	Funcionalidade
-k	Exibe a tela de *preview* e apenas salva a imagem após o usuário pressionar a tecla <ENTER>
-e	Define o codec para a imagem a ser capturada (opções: jpg, bmp, gif e png)
-t	Altera o tempo de *delay* antes de uma imagem ser capturada
-rot	Rotaciona a tela de *preview* e a imagem capturada de acordo com o ângulo informado
-p	Estabelece a dimensão e posição da tela de *preview*
-n	Desabilita a tela de *preview*
-f	Apresenta a tela de *preview* em modo de "tela cheia" (*full screen*)
-ex	Estabelece o modo de exposição da lente (algumas opções: *night*, *backlight*, *spotlight*, *sports*, *snow*, *beach*, *antishake* e *fireworks*)
-q	Indica a qualidade da compressão das imagens JPEG (varia de 0 a 100). Impacta a qualidade da imagem e o tamanho do arquivo
-r	Salva a imagem sem compressão (*raw*)
-br	Altera o brilho da imagem (varia de 0 a 100)
-co	Altera o contraste da imagem (varia de -100 a 100)
-sh	Altera a nitidez da imagem (varia de -100 a 100)
-ifx	Estabelece um filtro para a imagem (algumas opções: *negative*, *solarise*, *whiteboard*, *sketch*, *emboss*, *oilpaint*, *pastel*, *film* e *saturation*)
-v	Exibe na tela informações de configuração e captura

As opções de efeitos na imagem são úteis para diversas aplicações e devem ser consideradas como um recurso poderoso. O exemplo a seguir apresenta o comando para salvar uma imagem com o efeito *watercolour*:

```
$ raspistill -ifx watercolour -o imagem.jpg
```

Um exemplo de uma imagem salva após a execução desse comando é exibida na Figura 4.4.

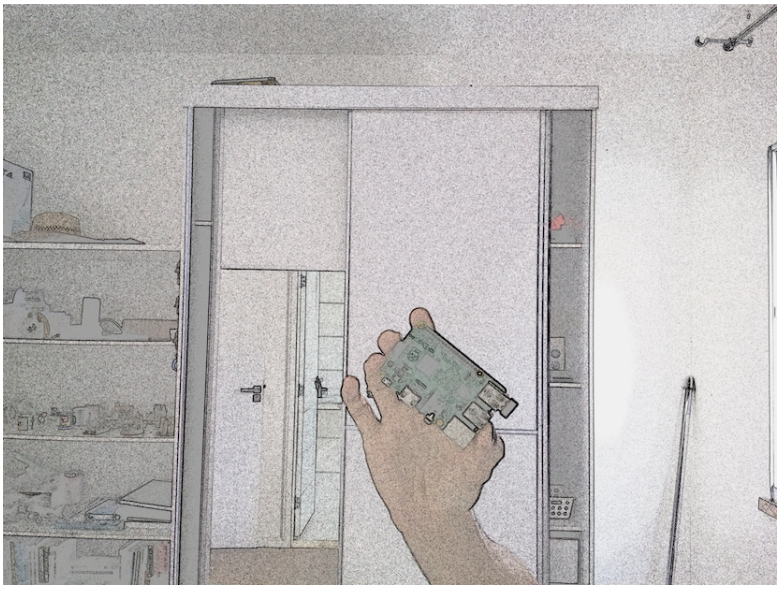


Figura 4.4. Exemplo de imagem com aplicação de filtro.

O comando *raspistill* também pode ser utilizado para capturar diversas imagens em sequência. Para tanto, deve-se utilizar a opção *timelapse* ("-tl"). O comando a seguir captura uma imagem a cada 5 segundos, durante 1 minuto, sendo as imagens salvas em arquivos numerados sequencialmente. Após a execução desse comando, 13 arquivos de imagens serão salvos, de "img0.jpg" a "img12.jpg" (a opção "%2d" define uma contagem com 2 dígitos). A resolução estabelecida é 300 x 300 (a menor resolução possível é 64 x 64).

```
$ raspistill -w 300 -h 300 -t 60000 -tl 5000 -o img%2d.jpg
```

Outro comando que pode ser utilizado para captura de imagens é o *raspiyuv*. Esse comando possui muitas opções semelhantes àquelas do comando *raspistill*, com a diferença que o comando *raspiyuv* produz saídas sem formatação e sem processamento.

4.4. Trabalhando com vídeos

O Raspbian também disponibiliza um programa para gravar vídeo, o *raspivid*. Por padrão, o comando *raspivid* salva vídeos na resolução 720p, a menos que indicado de outra forma através de parâmetros específicos. De fato, há diversos parâmetros de configuração para esse programa, sendo muitos desses parâmetros semelhantes aos disponíveis para o programa *raspistill* (Tabela 4.5).

Vamos iniciar com um exemplo. A seguir é apresentado um comando para gravar 15 segundos (15000 milisegundos) de vídeo no arquivo "video.h264". Esse vídeo é gravado como um *raw stream* h.264 e, portanto, não está inserido em um "container" (como avi ou mpg). Assim, o arquivo de vídeo não contém informações adicionais, como uma trilha de áudio, por exemplo.

```
$ raspivid -t 15000 -o video.h264
```

A Figura 4.5 apresenta o *print* de uma captura de um vídeo através do comando *raspivid* no Raspbian *Desktop*, quando a opção "-v" é utilizada. É interessante notar as configurações realizadas automaticamente para a captura do vídeo: resolução 1920 x 1080 e 30fps.

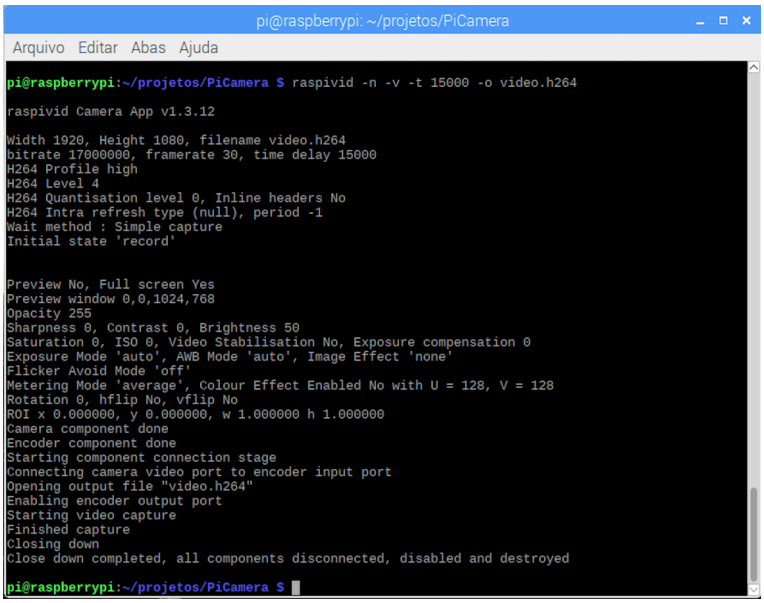


Figura 4.5. Configurações e procedimentos para captura de um vídeo.

As opções "-w" e "-h" podem ser utilizadas para alterar a resolução do vídeo que será gravado, naturalmente alterando o tamanho do arquivo final. O exemplo a seguir apresenta a captura de um vídeo de 20s na resolução 200 x 200.

```
$ raspivid -t 20000 -w 200 -h 200 -o video2.h264
```

Outros parâmetros bastante úteis é "-b", para alterar a taxa de bit do arquivo de vídeo, e "-fps", para alterar o número de quadros por segundo.

Após salvar um arquivo de vídeo, ele pode ser transmitido para outro dispositivo, para reprodução e/ou processamento, ou mesmo ser considerado localmente. Entre as opções para reproduzir o vídeo localmente, a ferramenta "omxplayer" apresenta-se como uma boa opção, além de já estar disponível nativamente no Raspbian.

Um exemplo da utilização da ferramenta "omxplayer" é apresentado a seguir. Há diversas opções que podem ser ativadas apertando determinados botões, por exemplo pausando a reprodução do vídeo ou alterando sua velocidade de reprodução.

```
$ omxplayer video.h264
```

Como o vídeo gravado com o programa *raspivid* é *raw*, ele poderá ser reproduzido com restrições e perdas de qualidade em determinadas ferramentas. Uma solução para esse "problema" é encapsulá-lo em algum "container" (por exemplo, mpg). O programa "MP4Box" permite realizar tal processamento. Para tanto, deve-se instalar o pa-

cote gratuito “gpac”. O seguinte comando encapsula o vídeo gerado pelo *raspivid* em um “container” específico (MPEG4).

```
$ MP4Box -add video.h264:fps=30 video.mp4
```

Além do “omxplayer”, podem-se utilizar ferramentas mais robustas. Um bom exemplo nesse sentido é a ferramenta VLC.

4.5. Criando sensores visuais com Python

Embora os programas *raspistill* e *raspivid* sejam bastante úteis e possam ser inclusive automatizados em *scripts* Bash ou Python, por exemplo, há diversas ações relativas a um sensor visual que podem ser melhor desempenhadas com recursos de programação mais completos. Nesse sentido, a câmera do Raspberry Pi se torna tão poderosa quanto for sua API de utilização. Neste Capítulo iremos apresentar a API “picamera”, disponível para a linguagem Python e instalada por padrão no Raspbian.

De fato, a linguagem Python é bastante utilizada pela comunidade em torno do Raspberry Pi, principalmente devido a sua flexibilidade, facilidade de programação e extensa gama de API disponíveis, permitindo um controle profundo dos recursos do Raspberry. A referência neste Capítulo é o Python 3.

4.5.1. Automatizando sensores visuais

Com a flexibilidade de programação trazida com o uso da linguagem Python, um número incontável de possibilidades são apresentadas ao Raspberry Pi e aos hardwares conectados. Essa subseção apresenta diversos exemplos para auxiliar na construção de sensores visuais explorando a API picamera.

O código a seguir rotaciona a câmera em 180^o e captura uma imagem, salvando-a no arquivo “pic.jpg”.

```
1 import picamera
2
3 camera = picamera.PiCamera()
4 camera.rotation = 180
5 camera.capture ("pic.jpg")
```

Esse pequeno código é bastante simples, porém apresenta características interessantes do uso da API picamera. Inicialmente, na linha 1, é realizada a “importação” da API para o código. Na linha 3 é criado um objeto do tipo PiCamera(): é através desse objeto que a câmera é acessada. Na linha 5, uma imagem é capturada e salva no arquivo “pic.jpg” (não é exibida a tela de *preview*).

A seguir é apresentado um código para exibir a tela de *preview* por 15 segundos, sem salvar qualquer imagem. Nesse código é utilizado o tratamento de exceções nativo da linguagem Python.

```
1 import picamera
2 import time
3
4 camera = picamera.PiCamera()
```

```
5  try:
6      camera.start_preview()
7      time.sleep(15)
8      camera.stop_preview()
9  finally:
10     camera.close()
```

Uma sequência de fotos é capturada no código a seguir, definindo para tanto a resolução 640 x 480 para as imagens salvas. Ao final da execução desse código, 10 arquivos de imagens serão salvos, cada um sendo criado com intervalo de 1 segundo entre as capturas.

```
1  from picamera import PiCamera
2  from time import sleep
3
4  #2 segundos de preparacao
5  sleep(2)
6
7  with PiCamera() as cam:
8          cam.resolution = (640, 480)
9      for n in range(1,11):
10         cam.capture("pic" + str(n) + ".jpg")
11         sleep (1)
```

Utilizando a API picamera, diversos métodos podem ser empregados para interação com a câmera do Raspberry. A Tabela 4.6 apresenta alguns desses métodos que podem ser utilizados.

Seguindo a ideia de explorar os métodos disponíveis para interação com a câmera, o exemplo a seguir apresenta diversas variáveis que alteram dinamicamente propriedades da captura de dados visuais, modificando a forma como a imagem é exibida na tela de *preview*. Para cada uma de quatro propriedades (*brightness*, *contrast*, *saturation* e *sharpness*), o código uniformemente varia o valor possível entre o mínimo e o máximo, em intervalos de 10 segundos. Deve-se lembrar que alguns atributos, como por exemplo *resolution*, só podem ser alterados quando a câmera estiver ociosa.

```
1  from picamera import PiCamera
2  from time import sleep
3
4  with PiCamera() as camera:
5      camera.resolution = (640, 480)
6      camera.start_preview()
7
8      try:
9          for i in range(101):
10             camera.brightness = i
11             sleep(0.1)
12
13         camera.brightness = 50
```

Tabela 4.6. Alguns métodos da API picamera.

Método	Descrição
capture()	Captura uma imagem e salva no arquivo especificado
capture_continuous()	Captura uma sequência contínua de imagens
capture_sequence()	Captura uma sequência de imagens
close()	Libera os recursos alocados para a câmera
start_recording()	Inicia a gravação de um novo *stream* de vídeo
stop_recording()	Interrompe a gravação de vídeo corrente
wait_recording()	Define o tempo de duração de uma gravação de vídeo
resolution()	Configura a resolução de imagem ou vídeo
image_effect()	Define um efeito para a imagem
start_preview()	Exibe a janela de *preview*
stop_preview()	Fecha a janela de *preview*

```
14
15          for i in range(-100, 101):
16              camera.contrast = i
17              sleep(0.05)
18
19          camera.contrast = 0
20
21          for i in range(-100, 101):
22              camera.saturation = i
23              sleep(0.05)
24
25          camera.saturation = 0
26
27          for i in range(-100, 101):
28              camera.sharpness = i
29              sleep(0.05)
30
31      finally:
32          camera.stop_preview()
33          camera.close()
```

É importante chamar o método *close()* quando a câmera não for mais necessária, uma vez que a câmera consome em média 250mA quando está em execução, o que inicia com a criação do objeto "PiCamera".

Outras configurações interessantes são realizadas no código a seguir. Nesse có-

digo é alterado o tamanho da janela de *preview* e o modo de exposição da câmera. Além disso, é aplicado um filtro à imagem. Por fim, a imagem é salva no formato PNG.

```
1  from picamera import PiCamera
2  from time import sleep
3
4  with PiCamera() as camera:
5      camera.resolution = (320, 320)
6
7      camera.exposure_mode = "beach"
8      camera.image_effect = "pastel"
9
10     camera.preview_fullscreen = False
11     camera.preview_window = (100, 100, 320, 320)
12
13     camera.start_preview()
14
15     #Preparar a camera
16     sleep (2)
17     camera.capture ("foto.png", format="png")
18     camera.close()
```

Uma sequência de imagens também pode ser criada utilizado o método *capture_sequence()*, como apresentado a seguir.

```
1  from picamera import PiCamera
2  from time import sleep
3
4  with PiCamera() as camera:
5      camera.resolution = (320, 320)
6      sleep (1)
7
8      #sequencia 1
9      camera.capture_sequence
       (["img1.jpg","img2.jpg","img3.jpg"])
10
11     sleep (1)
12     #sequencia 2
13     camera.capture_sequence
       (("img%1d.jpg" % i for i in range (4,11)))
14
15     camera.close()
```

Já o código a seguir salva imagens de forma ininterrupta, a cada 0,5 segundos, gravando a descrição da hora de captura no nome de cada arquivo.

```
1  from picamera import PiCamera
2  from time import sleep
3
```

```
4  with PiCamera() as camera:
5
6      sleep (1)
7
8      for arquivo in camera.capture_continuous
       ('pic_{timestamp}.jpg',use_video_port=False):
9          print ("Arquivo %s salvo" % arquivo)
10         sleep (0.5)
```

Vídeos também podem ser facilmente criados com a API picamera. No geral, as configurações necessárias são semelhantes, com a diferença que devem-se chamar outros métodos para isso. O código a seguir cria um vídeo de 10 segundos de duração, com 15fps e com resolução de 128 x 128.

```
1  from picamera import PiCamera
2  from time import sleep
3
4  camera = PiCamera()
5  camera.framerate = 15
6
7  camera.resolution = (128,128)
8  camera.preview_fullscreen = False
9  camera.preview_window = (0, 0, 128, 128)
10
11 camera.start_preview()
12
13 #Inicia a captura do video
14 camera.start_recording("video.h264")
15
16 sleep(10)
17
18 #Encerra a captura do video
19 camera.stop_recording()
20
21 camera.stop_preview()
22
23 camera.close()
```

No exemplo apresentado, a janela de *preview* é exibida no canto superior esquerdo da tela, exatamente do mesmo tamanho do vídeo criado.

O próximo exemplo grava um vídeo usando o método *wait_recording()*, na resolução padrão (1080p) e por 20 segundos. Esse método é mais interessante por retornar um erro caso o vídeo não possa ser gravado (por exemplo, for falta de espaço de armazenamento), sendo uma opção melhor que utilizando o método *time.sleep()*.

```
1  from picamera import PiCamera
2
3  camera = PiCamera()
```

```
4  camera.framerate = 30
5
6  camera.start_preview()
7  camera.start_recording("video2.h264")
8
9  #grava um video de 20 segundos
10 camera.wait_recording(20)
11
12 camera.stop_recording()
13
14 camera.stop_preview()
15 camera.close()
```

Há ainda muitos outros métodos úteis para utilizar a API picamera. Contudo, os métodos e exemplos apresentados já oferecem uma boa base para a construção de sensores visuais IoT.

4.5.2. Interagindo com a GPIO

Um dos recursos mais poderosos do Raspberry é a disponibilidade de 40 pinos de interação com outros hardwares, chamados simplesmente de GPIO (*General Purpose Input/Output*). Com esse pinos, o Raspberry pode se conectar com uma infinidade de outros dispositivos, ampliando significativamente as possibilidades no desenvolvimento de sensores de vídeo com o Raspberry Pi. Com a adoção do modelo de GPIO com 40 pinos, que se mantém nas versões mais recentes, inclusive no pequeno Raspberry Pi Zero, diversos fabricantes puderam desenvolver hardwares adicionais ao Raspberry com maior facilidade.

Cada pino da GPIO possui uma funcionalidade bem específica, que é padrão, como apresentado na Figura 4.6. Para facilitar as atividades que serão desenvolvidas nesta seção, os pinos GPIO estão apresentados de forma simplificada, não diferenciando os recursos adicionais que podem estar presentes em alguns pinos. Deve-se notar que a identificação numérica dos pintos (de 1 a 40) é diferente da sua utilização. Por exemplo, o pino 11 é o pino GPIO 17. A Figura 4.6 é uma ótima referência para utilizar corretamente cada pino.

Dos 40 pinos da GPIO, os pinos 2 e 4 fornecem 5V, os pinos 1 e 17 fornecem 3.3V e os pinos 6, 9, 14, 20, 25, 30, 34 e 39 são pinos "terra"(GND). Os demais pinos possuem diversas funcionalidades, podendo não apenas interagir diretamente com outros hardwares, mas também realizarem comunicações utilizando os padrões UART, I2C e SPI (pinos em “azul” na Figura 4.6), devendo esses recursos serem consultados em material complementar. Contudo, os pinos em verde na Figura 4.6 são exclusivos para entradas e saídas genéricas, sendo esses pinos utilizados prioritariamente neste Capítulo.

De fato, cada um dos pinos GPIO podem ser definidos como pinos de entrada ou saída, operando todos eles com apenas dois valores bem definidos: LOW (0V) ou HIGH (3.3V). Basicamente, todos os pinos estão no modo de “entrada” por padrão quando o sistema é iniciado, sendo a configuração correta dos pinos fundamental para sua operação apropriada.

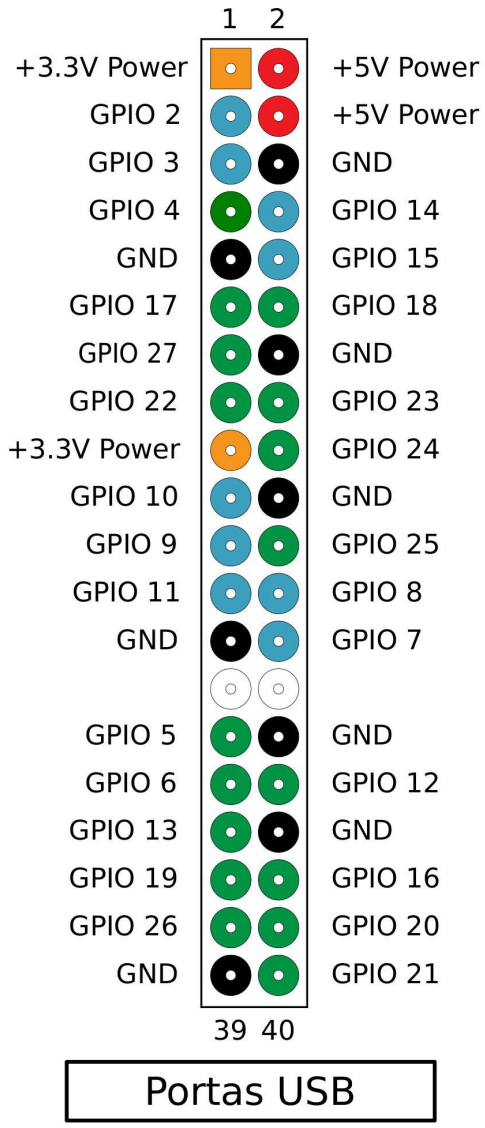


Figura 4.6. Funções de cada pino da GPIO.

Um lembrete importante é que como a entrada é feita com 3.3V no estado HIGH, deve-se ter cuidado com a utilização de componentes que utilizam uma voltagem maior, sobretudo porque muitos componentes operam com 5V. Para tanto, deve-se dimensionar apropriadamente o circuito desenvolvido, usando resistores quando necessário.

Neste Capítulo serão apresentados alguns exemplos de utilização da GPIO através da programação em Python, integrando as funcionalidades apresentadas com o uso da câmera. De fato, existem algumas bibliotecas no Python para interação com a GPIO, com níveis diferentes de recursos e complexidades. Contudo, durante toda esta seção será utilizada a API "gpiozero", que é uma API que facilita sobremaneira o uso de diversos componentes comuns da eletrônica, como LEDs e botões (*push-buttons*). Dessa forma, ao invés de se preocupar em configurar individualmente cada pino que será usado (por exemplo, indicando se o pino será de entrada ou saída), abstrações de alto nível serão

utilizadas.

A Tabela 4.7 apresenta alguns dos principais componentes modelados pela API gpiozero.

Tabela 4.7. Componentes modelados pela API gpiozero.

Classe	Tipo	Descrição	Métodos comuns
LED	Saída	LED padrão	on() off() blink()
RGBLED	Saída	LED rgb	on() off() blink() color()
Buzzer	Saída	Componente *buzzer*	on() off() beep()
Motor	Saída	Motor genérico	forward() backward() stop()
Button	Entrada	Botão (*push-button*)	wait_for_press() wait_for_release()
MotionSensor	Entrada	Sensor de movimento	wait_for_motion() wait_for_no_motion()
LightSensor	Entrada	Sensor de luminosidade	wait_for_light() wait_for_dark()
DistanceSensor	Entrada	Sensor de distância	wait_for_in_range() wait_for_out_of_range()

4.5.2.1. Controlando a câmera com botões

O acesso aos pinos da GPIO será feito diretamente a partir de um programa Python, utilizando para tanto a biblioteca gpiozero (instalada por padrão no Raspbian). Essa biblioteca permite o acesso facilitado há diversos pinos da GPIO, com abstrações de alto nível.

O primeiro exemplo a ser considerado irá utilizar um *push-button* conectado a uma *breadboard*, que será então conectado ao pino GPIO 18. A conexão é bastante simples: o pino 6 (GND) é conectado a uma linha do botão, sendo a linha oposta conectada ao pino GPIO 18. Para interagir com o botão criado, pode-se utilizar o código apresentado a seguir, que define o método "fotografar()".

```
1 import gpiozero
2 import picamera
3 import signal
4
5 camera = picamera.PiCamera()
6
7 def fotografar():
```

```
8           camera.capture ("imagem.jpg")
9
10  botao = gpiozero.Button (18)
11
12  botao.when_pressed = fotografar
13
14  signal.pause()
```

A classe gpiozero.Button já encapsula todos os detalhes necessários para interagir com um botão, bastando apenas informar qual é o pino que será associado ao botão desejado no processo de criação do objeto, como realizado na linha 10. A partir daí, o botão real já pode ser "acessado" através desse objeto.

A instrução na linha 14, "signal.pause()", evita que o programa seja encerrado antes do usuário apertar o botão. Quando em execução, toda vez que o usuário apertar o botão, uma nova imagem será salva no arquivo "imagem.jpg".

O próximo exemplo não apenas salva imagens com o clique do botão, mas também acende um LED. Para tanto é considerado o circuito apresentado na Figura 4.7.

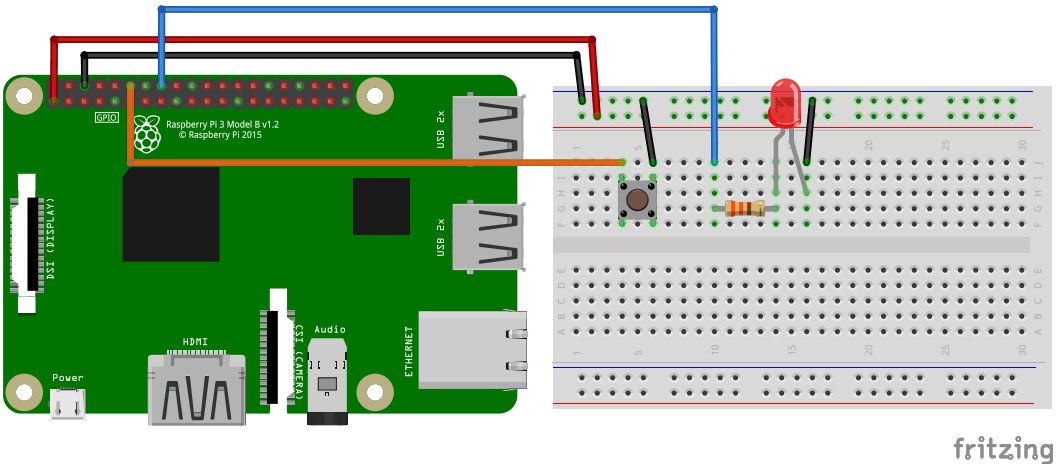


Figura 4.7. Circuito para captura de imagem a partir de um botão.

Para o circuito apresentado, pode-se utilizar o código descrito a seguir. Enquanto o botão está automaticamente configurado para a GPIO 18, como um *input*, o LED está configurado no pino GPIO 23, com um *output*. Além disso, nas funções definidas, será executado um trecho de código quando o botão for pressionado ("when_pressed") e outro código quando o botão for liberado ("when_released").

```
1  from picamera import PiCamera
2  from gpiozero import Button, LED
3  from signal import pause
4  from time import sleep
5
6  camera = PiCamera()
```

```
7
8  #acessando via GPIO
9  botao = Button (18)
10 led = LED (23)
11
12 #capturar imagem
13 def fotografar():
14     camera.capture("imagem.jpg")
15
16 #acender LED por 2 segundos
17 def ligarled():
18     led.on()
19     sleep(2)
20     led.off()
21
22 botao.when_pressed = ligarled
23 botao.when_released = fotografar
24
25 pause()
```

No próximo exemplo são utilizados dois botões e dois LEDs. O primeiro botão é utilizado para disparar a captura de uma imagem, acionando em seguida um LED e um *buzzer* (para emitir um sinal sonoro). Já o segundo botão dispara a captura de um vídeo de 10 segundos, também deixando um LED específico ligado por 10 segundos. O código para esse exemplo é apresentado a seguir.

```
1  from picamera import PiCamera
2  from gpiozero import Button, LED
3  from signal import pause
4  from time import sleep
5
6  camera = PiCamera()
7
8  botao1 = Button (18)
9  botao2 = Button (23)
10
11 led1 = LED (24)
12 led2 = LED (25)
13
14 def filmar():
15     camera.start_recording("video.h264")
16     camera.wait_recording (10)
17     camera.stop_recording()
18
19 def fotografar():
20     camera.capture("imagem.jpg")
21
```

```
22 def ligarledVideo():
23     led1.on()
24     sleep(10)
25     led1.off()
26
27 def ligarledCamera():
28     led2.on()
29     sleep(1)
30     led2.off()
31
32 botao1.when_pressed = ligarledCamera
33 botao1.when_released = fotografar
34
35 botao2.when_pressed = ligarledVideo
36 botao2.when_released = filmar
37
38 pause()
```

O circuito relacionado a esse exemplo é apresentado na Figura 4.8.

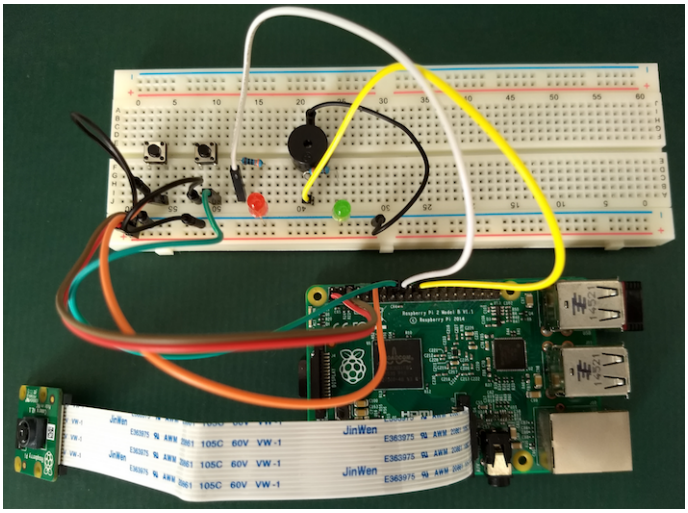


Figura 4.8. Circuito para captura de imagem e vídeo a partir de diferentes botões.

Por fim, para exemplificar o uso de botões através da GPIO, a tabela 4.8 apresenta métodos comuns que podem ser utilizados por objetos do tipo Button.

4.5.2.2. Tirando fotos com um sensor de distância

Outro exemplo interessante do uso da GPIO é a conexão com um sensor de distância (proximidade). Para tanto, foram escritos dois códigos de exemplo para conexão a um sensor ultra-sônico HC-SR04, que é um sensor de distância barato e fácil de operar. Esse sensor possui quatro pinos, sendo um deles utilizado para controle do sensor (*trigger*) e outro para receber informações sobre a distância dos objetos localizados (*echo*). Contudo,

Tabela 4.8. Alguns métodos de gpiozero.Button.

Método	Descrição
wait_for_press()	Pausa a execução do *script* até o botão ser pressionado (*timeout* também pode ser definido)
wait_for_release()	Pausa a execução do *script* até o botão ser liberado (*timeout* também pode ser definido)
held_time()	Tempo em segundos que o botão esteve pressionado
is_pressed()	Retorna "True" se o botão estiver pressionado
when_held()	Indicação da função que será chamada quando o botão for pressionado pelo tempo indicado
when_pressed()	Indicação da função que será chamada quando o botão for pressionado (sai do estado "inativo" para o estado "ativo")
when_released()	Indicação da função que será chamada quando o botão for liberado (sai do estado "ativo" para o estado "inativo")

como esse sensor opera com 5V, deve-se utilizar resistores para a entrada de dados no Raspberry, uma vez que o nível HIGH da GPIO é 3.3V. A Figura 4.9 apresenta o esquema de interconexão, utilizando dois resistores de $1K\Omega$.

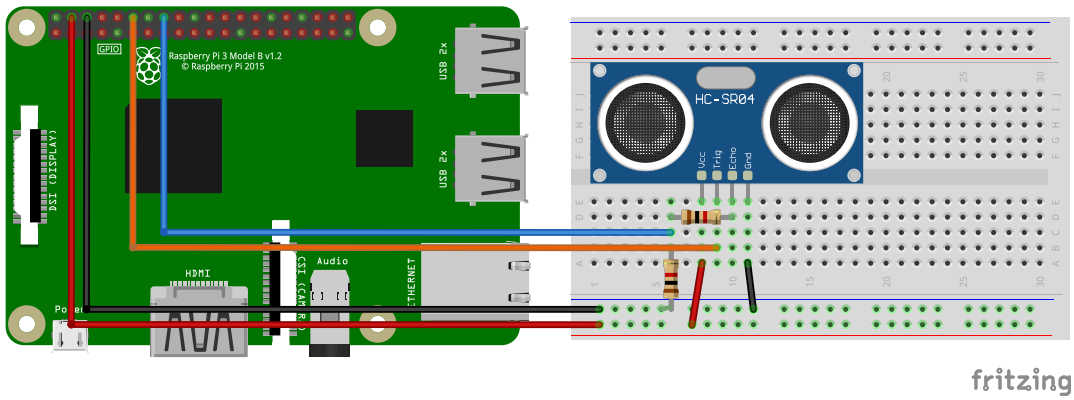


Figura 4.9. Conectando um sensor HC-SR04 ao Raspberry.

O código apresentado a seguir captura uma imagem quando um objeto está em uma distância de até 80cm em relação ao sensor. Para tanto, o pacote gpiozero cria a abstração DistanceSensor, que facilita bastante a utilização desse tipo de sensor.

```
1 from gpiozero import DistanceSensor
2 from picamera import PiCamera
3 from time import sleep
4 from datetime import datetime
5
6 camera = PiCamera()
```

```
7
8  def fotografar():
9      hora = datetime.now()
10     camera.capture(str(hora) + ".jpg")
11     print ("Imagem capturada: " + str(hora))
12
13 sensor = DistanceSensor (echo=23, trigger=18,
   max_distance=2.0)
14
15 while True:
16         distancia = sensor.distance
17
18         #distancia em metros
19         if (distancia <= 0.8):
20             fotografar()
21
22         sleep (1)
```

Quando um objeto gpiozero.DistanceSensor é criado, deve ser especificado quais são os pinos *trigger* e *echo*. A partir daí, pode-se acessar dinamicamente o valor da distância de objetos localizados pelo sensor. Há diversos métodos disponíveis que são úteis quando é utilizado o objeto DistanceSensor, o que oferece mais flexibilidade ao processo de programação. Por exemplo, o código a seguir possui a mesma funcionalidade do código anteriormente apresentado (detecção de objetos na distância de até 80cm), estando a diferença nos métodos que são empregados.

```
1  from gpiozero import DistanceSensor
2  from picamera import PiCamera
3  from time import sleep
4  from datetime import datetime
5
6  camera = PiCamera()
7
8  def fotografar():
9      hora = datetime.now()
10     camera.capture(str(hora) + ".jpg")
11     print ("Imagem capturada: " + str(hora))
12
13 sensor = DistanceSensor (echo=23, trigger=18,
   max_distance=0.8)
14
15 while True:
16         sensor.wait_for_in_range()
17         fotografar()
```

A Figura 4.10 apresenta o circuito criado seguindo o modelo definido.

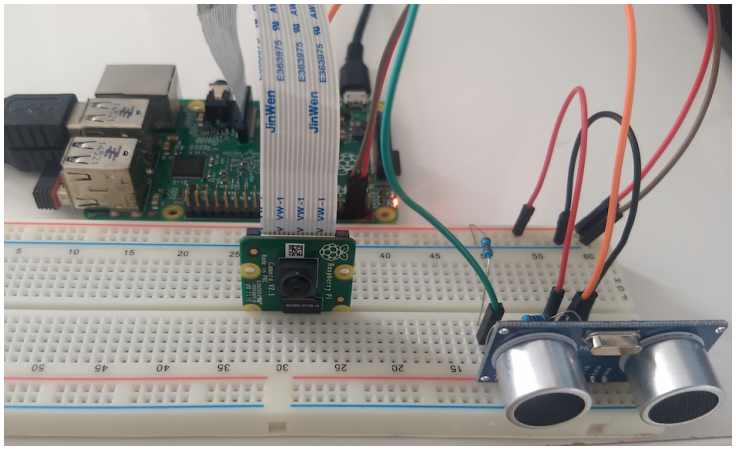


Figura 4.10. Sensor de vídeo ativado por presença.

4.5.3. Automatizando a execução de *scripts*

Em diversas situações, a execução de *scripts* pode ser automatizada para que seja realizada em momentos definidos pelo usuário. E esse tipo de serviço é útil em diversas situações, garantindo maior flexibilidade e controle avançado em inúmeras aplicações desenvolvidas com o Raspberry Pi, incluindo aquelas destinadas ao monitoramento por câmera.

A execução automatizada de *scripts* e/ou programas em geral é um serviço oferecido pelo sistema operacional. Como este Capítulo é baseado no Raspbian, recursos básicos do Linux Debian podem ser utilizados para essa função.

Considerando a necessidade de automatizar a execução de *scripts* que seja válida para o sistema como um todo e que ocorra sempre que o Raspberry for iniciado, pode-se utilizar o arquivo "/etc/profile". As instruções presentes nesse arquivo serão então executadas na inicialização do sistema, sendo isso ideal quando sensores visuais precisam entrar em operação automaticamente após serem ligados.

O exemplo a seguir apresenta um código Python que vai ligar um LED (GPIO 18) toda vez que o sistema for iniciado. O circuito que deve ser montado é muito simples, sendo responsável apenas pela ativação desse LED.

```
1 from gpiozero import LED
2 from time import sleep
3
4 led = LED (18)
5
6 #Por padrao, o pino GPIO18 esta LOW quando iniciado
7 led.on ()
8 sleep (10)
9 led.off ()
```

Considerando o nome do arquivo como sendo "ligarled.py", estando localizado na pasta *home* do usuário padrão do sistema, podemos inserir a seguinte linha de comando ao final do arquivo "/etc/profile":

```
python /home/pi/ligarled.py
```

Após a configuração do arquivo "/etc/profile", toda vez que o sistema for iniciado ou quando houver um novo *login*, o LED indicado será ligado por 10 segundos, sendo apagado em seguida. Deve-se perceber que como há uma chamada ao método "time.sleep()", o fluxo de processamento só será devolvido ao sistema após a execução do *script* indicado.

Já para executar *scripts* ou programas em geral em uma frequência definida, pode-se utilizar o serviço "crontab". O comando *$ crontab -e* irá permitir a configuração do arquivo de configuração desse serviço. A partir daí é possível alterar diretamente o arquivo de configuração, que possui um padrão de formatação simples que indica a frequência de execução de determinado comando (1 vez a cada hora, 3 vezes ao dia, 1 vez por semana, etc). O exemplo a seguir define a execução do *script* "ligarled.py"a cada minuto, para todos os dias.

```
* * * * * python /home/pi/ligarled.py
```

Há, de fato, diversas opções para configuração do crontab, sendo elas bastante úteis na configuração de sensores visuais. Por exemplo, a opção "@reboot"pode ser usada para indicar que o comando indicado será executado a cada inicialização do sistema. Para maiores informações sobre a configuração desse serviço, bibliografia complementar deve ser consultada, ou mesmo a documentação oficial do Raspbian.

4.6. Transmitindo a partir do Raspberry

Frequentemente, a captura de imagens e vídeos feitas por um Raspberry deverá ser transmitida para outro computador, para processamento e/ou reprodução. Esse cenário é, de fato, muito comum em aplicações IoT [Jyothi and Vardhan 2016, Sruthy and George 2017]. Para tanto, há diversos recursos disponíveis, sendo alguns deles descritos nesta seção.

De fato, *streaming* pode ser realizado de diferentes formas possíveis, podendo utilizar como base o TCP (confiabilidade) ou UDP (tempo-real). Os próximos exemplos apresentam soluções com diferentes abordagens.

4.6.1. Transmissões com programas padrões

O vídeo capturado a partir da câmera do Raspberry pode ser facilmente transmitido utilizando o comando *raspivid*. Para tanto, utilizam-se recursos já disponíveis no Raspbian. Aqui será apresentado um exemplo simples dessa possibilidade.

No exemplo apresentado a seguir, considerando uma comunicação cliente-servidor, o Raspberry irá se comportar como um cliente (que produz o vídeo e se conecta à um servidor para enviar esse vídeo), enquanto o servidor irá apenas reproduzir o vídeo recebido. O comando a seguir, que deve ser executado no lado Servidor, aguarda uma conexão na porta 15000 (usa o comando "nc"para isso). Quando a conexão é iniciada, o fluxo de bytes é direcionado ao *player* de vídeo "vlc", que é configurado para processar um fluxo de vídeo no formato h.264.

```
$ nc -l 15000 vlc –demux h264 -
```

No lado do cliente (Raspberry), pode-se digitar o comando apresentado a seguir. O comando *raspivid* é usado para transmitir por 60 segundos um pequeno vídeo com resolução 128 x 128, que é então direcionado para a conexão criada pelo comando "nc". Neste exemplo é considerado o endereço IP do servidor como sendo "10.0.0.10".

```
$ raspivid -w 128 -h 128 -t 60000 -o - nc 10.0.0.10 15000
```

Deve-se perceber que o comando *netcat* ("nc") está criando um fluxo de dados com o protocolo TCP, o que garante confiabilidade na transmissão em detrimento do tempo da comunicação. Dessa forma, o vídeo recebido é armazenado em cache (bufferização) e só é reproduzido quando isso for possível de forma contínua. Contudo, para diversas aplicações, um fluxo de dados sem perdas de transmissão pode ser desejado.

4.6.2. Transmissões com o MJPEG-streamer

O MJPEG-streamer é um conjunto de bibliotecas e programas para permitir a transmissão de imagens e vídeos em rede, de forma facilitada. De fato, essa opção é bastante flexível e permite transmissões de diversas formas possíveis. Nesta seção será apresentado um exemplo de como utilizar esse recurso.

Inicialmente, algumas bibliotecas e dependências devem ser instaladas, como apresentado a seguir:

```
$ apt-get install build-essential libjpeg8-dev
imagemagick libv4l-dev cmake
```

O MJPEG-streamer pode ser baixado do endereço "http://lara.uefs.br" (há outras fontes na web, inclusive no Github). O arquivo ".zip" que será baixado deve ser descompactado e compilado da seguinte forma:

```
$ make
$ make install
```

Se tudo estiver correto e não houver problemas na compilação, o programa mjpeg-streamer será instalado na pasta "/usr/local/bin/", que é uma pasta que já está no $PATH do sistema. A partir daí, esse programa poderá ser acessado livremente.

O exemplo a seguir inicia o *streaming* a partir da câmera do Raspberry Pi.

```
$ mjpg_streamer -i "input_raspicam.so -d /dev/video0
-fps 15 -q 80" -o "output_http.so -p 8080
-w /usr/local/share/mjpg-streamer/www"
```

Na execução desse comando é indicado como entrada (*input*, "-i") a interface representada no arquivo "/dev/video0", sendo a transmissão feita com 15 frames por segundo e "qualidade" jpeg estabelecida em 80. A câmera do Raspberry deve está indicada pelo sistema operacional no arquivo /dev/video0, o que faz como que ela se comporte como uma câmera USB qualquer. Caso isso não ocorra, pode-se digitar o comando "modprobe bcm2835-v4l2", carregando o módulo necessário. Por fim, a opção de saída (*output*, "-o") é estabelecida como um fluxo HTTP na porta 8080. Assim, pode-se acessar facilmente o fluxo transmitido em um navegador Web.

A Figura 4.11 apresenta o acesso em um computador ao fluxo transmitido pelo Raspberry. O acesso é feito digitando o endereço IP do Raspberry e indicando a porta 8080 (por exemplo, http://10.3.1.1:8080).

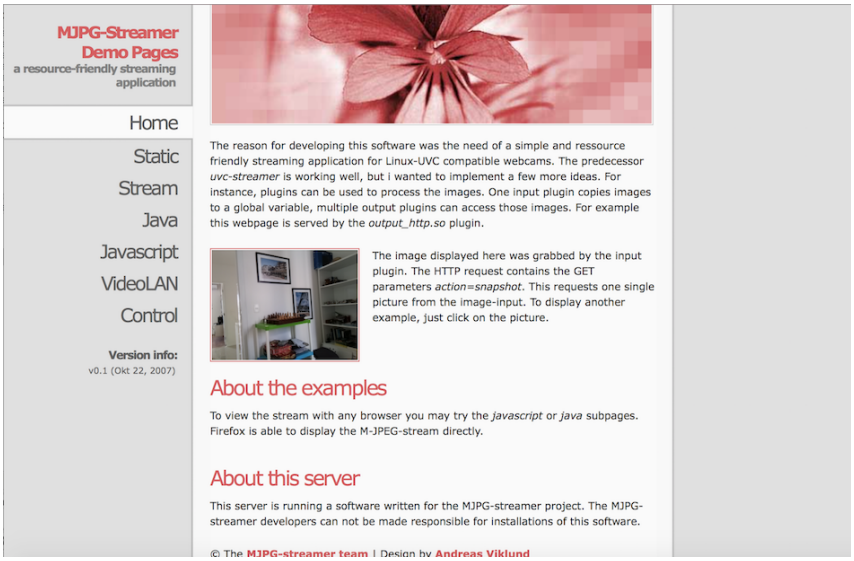


Figura 4.11. Tela principal quando o Raspberry é acessado pelo navegador Web na porta 8080.

Na tela apresentada há algumas opções para interação com o fluxo transmitido. Nas opções apresentadas, “Static” permite visualizar uma imagem instantânea, capturada naquele momento, enquanto “Stream” exibe em tempo real um fluxo de vídeo.

4.6.3. Transmissões pela API picamera

A API picamera também oferece recursos para realizar a transmissão de imagens e vídeos em tempo real, como alternativa ao armazenamento local de arquivos. O exemplo a seguir apresenta um código para realizar *streaming* por 60 segundos, enviando o vídeo capturado para o endereço “10.0.0.10” na porta 15000.

```
1  import socket
2  import picamera
3  import time
4
5  ip = "10.0.0.10"
6  porta = 15000
7
8  socket_cliente = socket.socket()
9  socket_cliente.connect ((ip, porta))
10
11 conexao = socket_cliente.makefile ('wb')
12
13 try:
14     camera = picamera.PiCamera()
15     camera.resolution = (320, 320)
```

```
16      camera.framerate = 15
17
18      time.sleep(2)
19
20      camera.start_recording (conexao,format="h264")
21      camera.wait_recording (60)
22      camera.stop_recording ()
23
24  except:
25
26      print ("Erro no envio do video")
27
28  finally:
29      camera.close()
30      socket_cliente.close()
```

Nesse exemplo é utilizado o pacote "socket", que é um pacote para conexões diversas na Internet (pilha TCP/IP). Através de um objeto "socket" é iniciada uma conexão com o endereço IP e a porta TCP especificados e, portanto, a parte "servidor' da conexão já deve estar esperando novas conexões. De fato, o lado servidor pode ser criado como na subseção anterior, utilizando o comando a seguir (o programa "vlc" é substituído pelo programa "mplayer"):

```
$ nc -l -p 15000 mplayer -fps 15 -cache 1024 -
```

Outra opção viável para o lado servidor é criar um *script* Python para reproduzir o vídeo. Para este exemplo, será feita uma comunicação direta entre dois Raspberry conectados à mesma rede Wi-Fi: um deles (cliente) irá transmitir o vídeo para o outro Raspberry (servidor), que irá reproduzir o vídeo recebido. O código do lado cliente pode ser o mesmo do apresentando anteriormente. Já o código para o lado do servidor é apresentando a seguir.

```
1  import socket
2  import subprocess
3
4  porta = 15000
5
6  socket_servidor = socket.socket()
7  socket_servidor.bind(('0.0.0.0', porta))
8  socket_servidor.listen(0)
9
10 conexao = socket_servidor.accept()[0].makefile('rb')
11 try:
12     comando = ['vlc', '--demux', 'h264', '-']
13
14     player = subprocess.Popen(comando,
       stdin=subprocess.PIPE)
15
```

```
16      while True:
17          # Ler 1k de dados e enviar ao player
18          data = conexao.read(1024)
19
20          if not data:
21              break
22
23          player.stdin.write(data)
24
25  finally:
26      conexao.close()
27      socket_servidor.close()
28      player.terminate()
```

O exemplo apresentado utiliza o programa "vlc", que é externo ao Python, para reproduzir o vídeo recebido. Essa é, de fato, uma estratégia simples e fácil de implementar, porém bibliotecas adicionais poderiam ser utilizadas para reproduzir o vídeo pelo próprio Python.

4.7. Copiando arquivos do Raspberry

Frequentemente, os arquivos de imagem e vídeo salvos em um Raspberry deverão ser copiados para outro Raspberry, computador convencional ou mesmo um celular, permitindo assim que os arquivos possam ser reproduzidos, processados, armazenados ou mesmo distribuídos. E isso pode ser feito de diferentes formas, como apresentado nesta seção.

4.7.1. Conexão direta via SSH

Uma solução simples para obter dados visuais capturados pelo Raspberry é conectando-se a ele diretamente. A partir daí, arquivos de imagem e vídeo podem ser facilmente copiados. De fato, o acesso direto a um Raspberry através do protocolo SSH, por exemplo, ou mesmo usando soluções de acesso remoto de Desktop (como o VNC), é uma tarefa simples caso seja conhecido o endereço IP do Raspberry e este esteja acessível.

Considerando que o Raspberry Pi em questão e o computador que irá acessá-lo estão na mesma rede (ou acessíveis através de uma rede IP), pode-se realizar uma conexão SSH direta para permitir a cópia de arquivos. Deve-se lembrar, contudo, que o acesso SSH deve estar habilitado no Raspberry (por exemplo, usando o comando *raspi-config*).

A Figura 4.12 apresenta uma conexão via SSH à um Raspberry, que neste caso está acessível pelo endereço "192.168.1.12".

Para acessar os arquivos através do protocolo SSH, existem diversos programas que podem ser utilizados. Uma forma simples é realizar a cópia de arquivos por linha de comando usando programas como o *scp*, nativamente presente em diversos sistemas operacionais. Outra opção é através de programas populares com diversos recursos disponíveis em interfaces interativas, como o "PuTTY" e o "WinSCP".

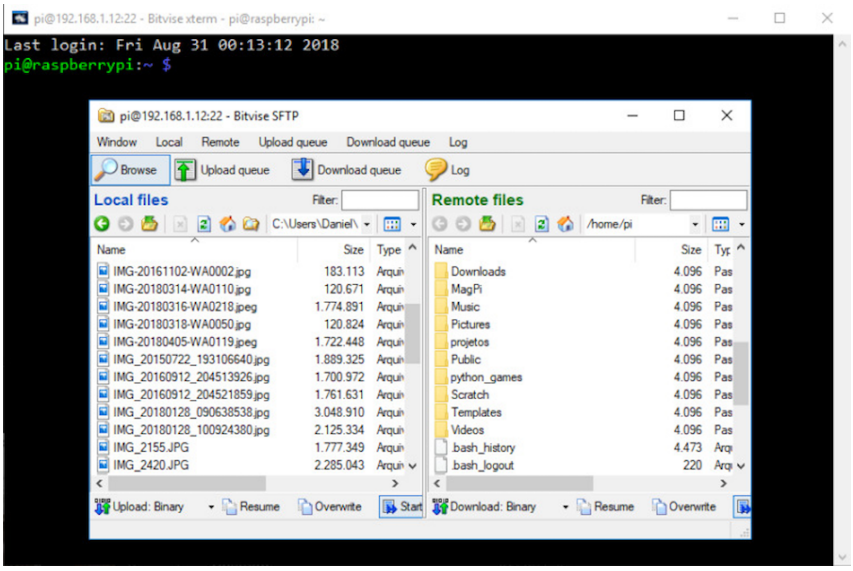


Figura 4.12. Conexão SSH à um Raspberry.

4.7.2. Transformando o Raspberry num AP

Quando houver muitos Raspberry operando de forma concorrente em determinado ambiente, uma solução simples e prática é transformar cada um dos dispositivos em um Wi-Fi Access Point (AP). Assim, para se conectar a um determinado Raspberry, basta selecionar o ID da rede Wi-Fi desejada e fazer a associação à rede. Para diversos cenários de monitoramento, quando for necessário se conectar individualmente à cada Raspberry, está será uma opção prática e rápida, sobretudo porque não é necessário configurar a adesão dos Raspberry à uma mesma rede.

Para configurar o Raspberry como um AP, serão utilizados dois serviços: "hostapd" e "dnsmasq'. Ambos os serviços podem ser facilmente instalados com o comando *apt-get*.

O primeiro passo para configurar um AP é definir um endereço IP estático para a interface Wi-Fi (wlan0). Para isso, considerando o sistema operacional Raspbian Stretch, deve-se alterar o arquivo /etc/dhcpcd.conf e incluir/alterar a seguinte informação (os dados utilizados são apenas exemplos e devem ser substituídos pelos valores apropriados).

```
interface wlan0
static ip_address=10.3.1.1/24
```

Como o objetivo não é conectar o Raspberry à Internet mas sim agir apenas como um Access Point offline, pode-se utilizar qualquer endereçamento válido na faixa de endereços IP. Por questões de conformidade com redes IP já existentes e por padronização, optou-se aqui por usar um endereço na rede privada 10.3.1.0/24, mas qualquer endereçamento poderia ser usado aqui. Neste exemplo, escolheu-se a seguinte padronização: o bloco de endereço "10", seguido pelo número "3" (uma vez que está sendo utilizado um Raspberry Pi 3 B) e o número "1", indicando que esse é o sensor número 1. Dessa forma, pode-se facilmente saber qual o sensor de vídeo que está sendo acessado.

Após a instalação do serviço "hostapd", deve-se configurar o arquivo /etc/hostapd/hostapd.conf. É neste arquivo que serão descritas as configurações da rede sem fio sendo criado. A seguir é apresentado um exemplo de configuração desse arquivo.

```
interface=wlan0
hw_mode=g
channel=1
wpa=2
wpa_key_mgmt=WPA-PSK
ssid=rasp3_1
wpa_passphrase=rasp3sensor1
```

No exemplo apresentado, apenas algumas configurações foram estabelecidas, deixando diversas outras configurações com valores padrões. Entre as opções definidas, é interessante notar a indicação do padrão IEEE 802.11g para comunicação, uso do canal 1 do padrão Wi-Fi (poderia ser automaticamente definido) e o padrão de criptografia baseado em WPA. Para o ID da rede (ssid), foi definido o nome "rasp3_1", sendo a senha definida como "rasp3sensor1". Como as redes Wi-Fi podem implementar criptografia, que é, inclusive, um serviço comumente utilizado, a adesão à determinada rede é geralmente controlada por senha. Contudo, pode-se configurar uma rede aberta (sem senha), porém se for utilizado WPAv2, a senha definida deve ter no mínimo 8 caracteres.

A terceira configuração necessária é definir o serviço DHCP, para que qualquer computador, celular ou dispositivo que se conecte ao Raspberry AP receba automaticamente um endereço IP para comunicação. Para tanto, deve-se adicionar ao arquivo /etc/dnsmasq.conf a seguinte informação.

```
interface=wlan0
dhcp-range=10.3.1.2,10.3.1.5,255.255.255.0,24h
```

Na configuração definida, o range de IP foi estabelecido entre 10.3.1.2 e 10.3.1.5, sendo então possíveis apenas quatro endereços IP (10.3.1.2, 10.3.1.3, 10.3.1.4 e 10.3.1.5).

Por fim, devem-se iniciar o serviços "hostapd" e "dnsmasq", como apresentado a seguir.

```
$ systemctl start hostapd
$ systemctl start dnsmasq
```

Após a realização das configurações apresentadas, a rede "rasp3_1" pode ser acessada por qualquer dispositivo habilitado para a rede Wi-Fi. A rede fica disponível como uma rede convencional, como apresenta a Figura 4.13.

Figura 4.13. Rede disponível para conexão ao Raspberry.

Após a conexão à rede criada, pode-se facilmente acessar o Raspberry e fazer cópia de arquivos, alterar configurações e/ou executar comandos. Pensando no cenário de

cópia de arquivos obtidos pelo sensor (imagens e vídeos), há duas abordagens bem interessantes. A primeira delas é usando o serviço SSH, como descrito na seção anterior. Uma outra opção é instalar um servidor Web no Raspberry, a exemplo do Apache. Instalando o servidor, os arquivos podem ser acessados diretamente através de um navegar convencional, como o "Firefox", o "Chrome" ou o "Safari". No caso do Apache, ele poder ser configurado de várias formas para exibir os arquivos de imagem e/ou vídeo capturados pelo Raspberry. Como uma solução viável, arquivos podem ser armazenados na pasta "/var/wwww/html", permitindo fácil e rápido acesso aos arquivos de imagem e/ou vídeo produzidos pelo Raspberry.

A Figura 4.14 apresenta um acesso via um *smartphone* ao endereço "10.3.1.1/imagens", considerando que o celular está conectado à rede "rasp3_1" definida previamente.

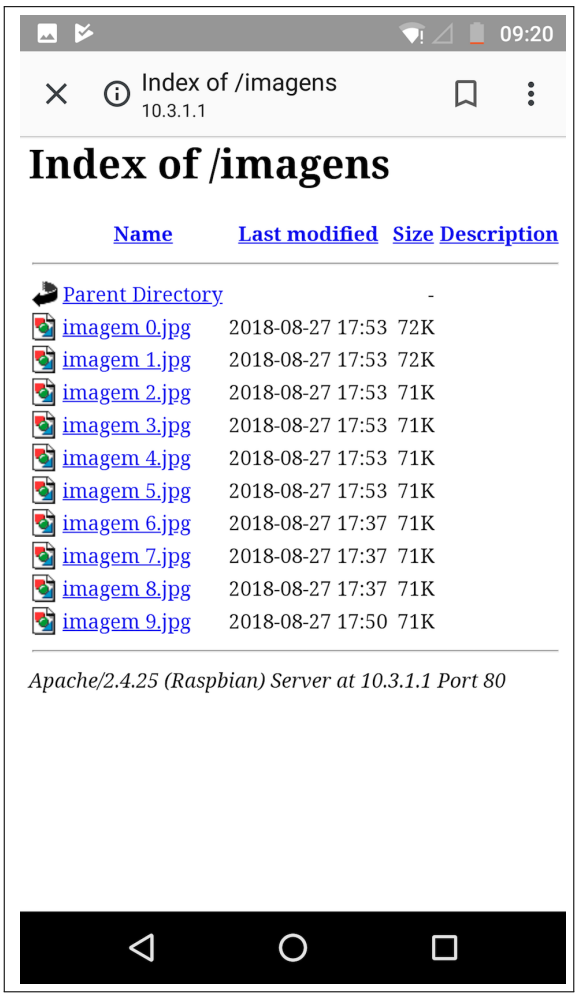


Figura 4.14. Acessando arquivos de imagens a partir de um celular.

4.8. Conclusão

O desenvolvimento tecnológico em diversas áreas, como microeletrônica, redes de comunicação, processamento paralelo, inteligência artificial e ciência de dados, vêm transformando significativamente o cenário da computação e das tecnologias da informação em geral. Nesse contexto, a Internet das Coisas surge como um grande escopo que integra

diversas tecnologias, revolucionando a forma como dados são capturados, distribuídos e processados. Ao final desta década, a Internet das Coisas já será o cenário dominante.

No mundo IoT, encontrar plataformas acessíveis para desenvolvimento e prototipação é de grande importância. O Raspberry surge como uma dessas plataformas, permitindo uma ampla gama de possibilidades de desenvolvimento. Com o uso de uma câmera, sensores de vídeo podem ser facilmente criados, o que aumenta sobremaneira as possibilidades de utilização do Raspberry em aplicações IoT.

Este Capítulo abordou a construção de um sensor de vídeo com o Raspberry Pi, apresentando alguns dos mais importantes detalhes de configuração e operação. Além disso, muitos exemplos mostraram usos práticos do Raspberry como sensor visual, permitindo sua utilização prática em diversas aplicações. Contudo, há ainda muitos detalhes que não foram abordados. Para o leitor, há muito material gratuito na Web, com inúmeros tutoriais e projetos constantemente mostrando novos usos promissores para o Raspberry. Apenas como referência, o site da Raspberry Foundation, "https://www.raspberrypi.org/", e da revista MagPi, "https://www.raspberrypi.org/magpi/", podem ser gratuitamente consultados para esse propósito.

Referências

[Andrade et al. 2017] Andrade, D. C., Costa, D. G., and ao B. Rocha-Junior, J. (2017). Adaptive sensing relevance exploiting social media mining in smart cities. In *Proceedings of the 23rd Brazillian Symposium on Multimedia and the Web*, pages 405–408.

[Atzori et al. 2010] Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer Networks*, 54(15):2787 – 2805.

[Costa et al. 2017] Costa, D. G., Collotta, M., Pau, G., and Duran-Faundez, C. (2017). A fuzzy-based approach for sensing, coding and transmission configuration of visual sensors in smart city applications. *Sensors*, 17(1):1–17.

[Costa et al. 2018] Costa, D. G., Duran-Faundez, C., Andrade, D. C., Rocha-Junior, J. B., and Just Peixoto, J. P. (2018). Twittersensing: An event-based approach for wireless sensor networks optimization exploiting social media in smart city applications. *Sensors*, 18(4).

[Costa et al. 2015] Costa, D. G., Guedes, L. A., Vasques, F., and Portugal, P. (2015). Research trends in wireless visual sensor networks when exploiting prioritization. *Sensors*, 15:1760–1784.

[Gupta et al. 2015] Gupta, M. S. D., Patchava, V., and Menezes, V. (2015). Healthcare based on iot using raspberry pi. In *Green Computing and Internet of Things (ICGCIoT)*, pages 796–799.

[Jyothi and Vardhan 2016] Jyothi, S. N. and Vardhan, K. V. (2016). Design and implementation of real time security surveillance system using iot. In *2016 International Conference on Communication and Electronics Systems (ICCES)*, pages 1–5.

[Kruger and Hancke 2014] Kruger, C. P. and Hancke, G. P. (2014). Benchmarking internet of things devices. In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, pages 611–616.

[Kuo et al. 2018] Kuo, Y., Li, C., Jhang, J., and Lin, S. (2018). Design of a wireless sensor network-based iot platform for wide area and heterogeneous applications. *IEEE Sensors Journal*, 18(12):5187–5197.

[Leone et al. 2017] Leone, G. R., Moroni, D., Pieri, G., Petracca, M., Salvetti, O., AzzarÃ, A., and Marino, F. (2017). An intelligent cooperative visual sensor network for urban mobility. *Sensors*, 17(11).

[Lin et al. 2017] Lin, J., Yu, W., Zhang, N., Yang, X., Zhang, H., and Zhao, W. (2017). A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet of Things Journal*, 4(5):1125–1142.

[Nikhade 2015] Nikhade, S. G. (2015). Wireless sensor network system using raspberry pi and zigbee for environmental monitoring applications. In *Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM)*, pages 376–381.

[Patchava et al. 2015] Patchava, V., Kandala, H. B., and Babu, P. R. (2015). A smart home automation technique with raspberry pi using iot. In *Smart Sensors and Systems (IC-SSS)*, pages 1–4.

[Patil et al. 2017] Patil, N., Ambatkar, S., and Kakde, S. (2017). Iot based smart surveillance security system using raspberry pi. In *Communication and Signal Processing (ICCSP)*, pages 0344–0348.

[Perera et al. 2014] Perera, C., Zaslavsky, A., Christen, P., and Georgakopoulos, D. (2014). Sensing as a service model for smart cities supported by internet of things. *Transactions on Emerging Telecommunications Technologies*, 25:81–93.

[Sandeep et al. 2015] Sandeep, V., Gopal, K. L., Naveen, S., Amudhan, A., and Kumar, L. S. (2015). Globally accessible machine automation using raspberry pi based on internet of things. In *Advances in Computing, Communications and Informatics (ICACCI)*, pages 1144–1147.

[Sruthy and George 2017] Sruthy, S. and George, S. N. (2017). Wifi enabled home security surveillance system using raspberry pi and iot module. In *2017 IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems (SPICES)*, pages 1–6.

Capítulo 5

Dados de Múltiplas Fontes da Web: coleta, integração e pré-processamento

Natércia A. Batista[1], Michele A. Brandão[1,2], Michele B. Pinheiro[1], Daniel H. Dalip[3] e Mirella M. Moro[1]

[1]Universidade Federal de Minas Gerais (UFMG)

[2]Instituto Federal de Minas Gerais (IFMG)

[3]Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)

{natercia, micheleabrandao, mibrito, mirella}@dcc.ufmg.br

hasan@decom.cefetmg.br

Abstract

Web data are heterogeneous and unstructured, which defines challenges for data crawling, integration and preprocessing. Different studies are "data-oriented" (i.e. based on the available data) but their results are restricted to their specific data. In contrast, there are various problems prior to identifying what data is needed to solve them, and often multiple data sources are needed. In this context, crawling, integrating and preprocessing data appropriately enables to create datasets for solving such problems. Therefore, this short course addresses these three activities by discussing challenges and practical solutions.

Resumo

Atividades de coleta, integração e pré-processamento representam diferentes desafios para pessoas que necessitam de lidar com dados extraídos da Web por serem heterogêneos e não estruturados. Ademais, existem diferentes fontes de dados na Web que podem ser sites e aplicativos, mídias e redes sociais e até mesmo bancos (ou bases) de dados já construídos e disponibilizados. Considerar dados dessas diferentes fontes pode parecer irrelevante quando avaliados de forma isolada. Entretanto, quando combinados, conhecimentos novos, integrados e úteis podem ser descobertos. Tais dados podem ser aplicados na solução de problemas em diferentes campos, como sistemas inteligentes, ao permitir a ampliação dos dados utilizados como treinamento; marketing, ao possibilitar a identificação de público alvo; sistemas de recomendação, ao viabilizar a construção do

perfil de usuários, entre muitos outros. Nesse contexto, coletar, integrar e pré-processar dados adequadamente de múltiplas fontes permite a criação de conjuntos de dados enriquecidos que possibilitam a solução de problemas reais. Assim, este minicurso aborda essas três tarefas e apresenta seus principais desafios.

5.1. Introdução

A Web (do inglês *World Wide Web*, ou Rede Mundial de Computadores) é um sistema de documentos dos mais variados formatos que são interligados e acessados (ou executados) via Internet. Desde sua criação, a Web tem evoluído constantemente. Por exemplo, ela foi sendo gradualmente expandida para gerenciar os mais diversos tipos de documentos que vão muito além do original hipertexto (e.g., vídeo, som, imagem e afins) como também permitir que o próprio usuário crie e publique seu conteúdo. Com tantos *dados* disponíveis, a comunidade científica e a indústria de modo geral logo começaram a explorá-los das mais variadas formas e com propósitos imensuráveis.

De fato, existem diferentes fontes de dados da Web, incluindo sites e aplicativos, mídia e redes sociais e até mesmo bancos de dados completos. Os dados de tais múltiplas fontes são geralmente não estruturados, desnormalizados, inconsistentes, duplicados, incompletos e de qualidade variada [Farnadi et al., 2018, Geerts et al., 2018, Wang et al., 2018b]. Essas fontes incluem planilhas do Excel, arquivos *Comma Separated Values* (CSV), bancos de dados relacionais e não relacionais, armazém de dados e diferentes plataformas da Web (e.g., sites e aplicativos sociais).

Nesse vasto contexto, qualquer pesquisa orientada a dados Web requer o estabelecimento de uma relação entre eles para melhor combiná-los e analisá-los [Moro et al., 2009, Wang et al., 2017]. Ou seja, ao considerar múltiplas fontes de dados, pesquisadores e desenvolvedores adquirem uma visão maior sobre o contexto estudado, promovendo a descoberta de informações complementares, as quais permitem realizar inferências mais precisas, ou ainda, identificar padrões que só se tornam visíveis quando essas múltiplas fontes estão conectadas. Como exemplo prático e real, considere duas das maiores plataformas sociais online que são independentes uma da outra: Facebook (rede social de amizade) e GitHub (plataforma de desenvolvimento de software colaborativo). Pesquisas para compreender diferentes perfis de colaboração podem integrar dados de ambas a fim de, por exemplo: analisar como os relacionamentos pessoais influenciam o desenvolvimento de software; verificar se um desenvolvedor é popular no GitHub por também estar em outra mídia social; ou se um desenvolvedor é capaz de influenciar a comunidade desenvolvedora pela criação novos padrões de desenvolvimento, bem como disseminação desse novos padrões em outra rede social; entre muitas outras possibilidades interessantes.

Outro exemplo prático e real é usar dados de múltiplas fontes para apoiar a tomada de decisões, nesse caso considerando principalmente o aspecto financeiro [Geerts et al., 2018]. Em tal contexto, uma equipe de comércio eletrônico pode analisar o perfil de seus usuários em diferentes redes sociais para descobrir interesses e, em seguida, recomendar vendas combinadas. Para fazê-lo de uma maneira eficiente e eficaz, é necessário novamente coletar, integrar e pré-processar dados, que por sua vez são tarefas que representam maneiras de extrair *valor* (e então ganho financeiro) de tais dados.

Porém, para atingir os objetivos desses dois exemplos (desenvolvimento colabora-

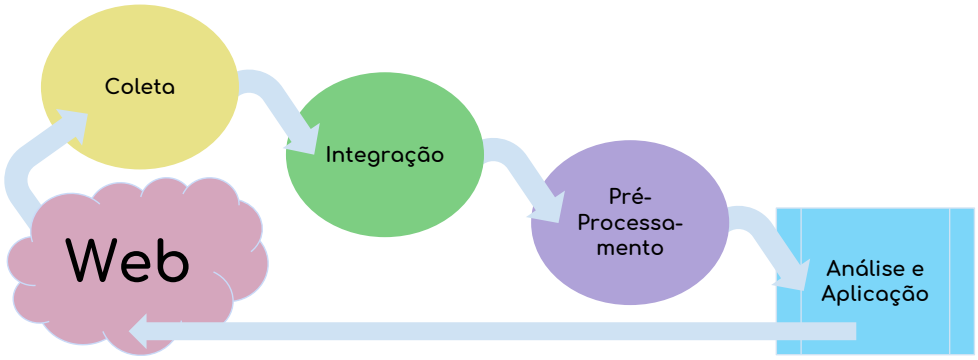


Figura 5.1: Processo de dados Web e tópicos abordados neste capítulo

tivo e marketing dirigido), é necessário obter os dados da Web. Tecnicamente, o primeiro problema é definir a estratégia de coleta (ou *crawling*), que pode ser classificada de acordo com o período e a forma como a semente (por onde a coleta inicia) é definida. A escolha de uma estratégia para integrar os dados de diferentes fontes também é importante para propiciar uma visão uniforme para usuários ou aplicativos, bem como armazenamento adequado para permitir consultas eficientes mais tarde. Finalmente, o pré-processamento de dados também pode ser necessário, o que ocorre antes ou depois da integração de dados, e envolve a resolução de dados ausentes e duplicados, normalização, etc.

Em resumo, como a Figura 5.1 ilustra, para desenvolver qualquer pesquisa ou aplicação com dados Web é necessário: *coletar* tais dados de fontes diversas (geralmente espalhadas pela própria Web), *integrar* tais dados, muitas vezes realizar *pré-processamentos* diferentes para então conseguir *analisar* e *aplicar* quaisquer que sejam as técnicas sendo pesquisadas ou desenvolvidas, que comumente têm seus resultados divulgados ou armazenados novamente na Web (fechando o ciclo). Nesse contexto, coletar, integrar e pré-processar dados de várias fontes (frequentemente heterogêneas) apresentam diferentes desafios, incluindo: coletar dados em tempo real, gerenciar ferramentas de coleta, decidir sobre questões de privacidade, padronizar dados diferentes, resolver dados duplicados, trabalhar com dados não uniformes, padrões úteis de mineração e qualidade, entre outros.

Apesar de conhecidas, tais tarefas são muitas vezes complexas e dependentes de contexto. Por exemplo, apenas o pré-processamento de dados requer cerca de 80% do tempo de cientistas de dados de acordo com a pesquisa publicada em [Tyagi et al., 2010]. Na prática, não apenas cientistas de dados, mas também programadores, pesquisadores, estudantes, empresas e usuários podem se beneficiar da solução desses desafios. Em especial, quando tais soluções retornam à comunidade através da publicação ou disponibilização de dados padronizados e completos na Web, o ganho é global.

Como exemplos, considere as seguintes publicações de tal retorno no âmbito da edição de 2017 deste evento (WebMedia). Araujo et al. [2017] utilizam de técnicas de coleta e pré-processamento de dados para predizer o sucesso de um álbum de música baseado em comentários de redes sociais online. Já Freitas et al. [2017] propõem uma estratégia de integração de dados baseada em ontologias e dados conectados (tecnologias a serem explicadas mais adiante neste capítulo) para calcular a probabilidade do risco de óbito maternos e infantil no Brasil. Ainda em integração, o conceito de ontologia é tão

versátil que Veiga et al. [2017] o utilizam para dados provenientes de redes de sensores e internet das coisas. Já utilizando um framework muito comum no contexto de dados Web, Maia and Oliveira [2017] compilam as pesquisas sobre o vírus Zika e analisam a reputação de seus pesquisadores no contexto da saúde mundial.

A motivação deste capítulo é: auxiliar pesquisadores e desenvolvedores que precisam de dados Web e diminuir o tempo de obtenção de tais dados através da divulgação de soluções existentes para vários dos desafios mencionados. Este capítulo aborda então as três questões de forma integrada (coleta, integração e pré-processamento) e apresenta soluções que podem ser aplicadas à pesquisa e ao desenvolvimento de aplicações comerciais. A organização deste capítulo segue a ordem supra-citada após a Seção 5.2 que resume fontes de dados Web; a Seção 5.3 discute coleta de dados Web; a Seção 5.4 resume algumas estratégias para integração de dados; a Seção 5.5 apresenta os principais problemas durante o pré-processamento de dados; e a Seção 5.6 detalha algumas aplicações reais de dados múltiplas fontes Web. Finalmente, a seção 5.7 apresenta considerações finais e ponteiros para outras fontes de informação sobre os assuntos aqui tratados.

5.2. Fontes de Dados Web

O processamento de dados da Web requer o prévio conhecimento das fontes disponíveis. Com tal entendimento, é possível realizar um planejamento da coleta e definir estratégias que podem melhorar sua eficiência e cobertura. Nesta seção, são apresentados alguns dos tipos de fontes de dados mais comumente encontrados na Web, sendo essas os dados abertos, dados conectados, APIs e páginas da Web. Sobre cada fonte, são também apresentados os desafios a serem considerados em seu processamento.

5.2.1. Dados Abertos

Os dados abertos, como o próprio nome expressa, são dados disponibilizados abertamente. Uma organização sem fins lucrativos *Open Knowledge Foundation* (OKF)[1] foi criada para incentivar e estipular um conjunto de diretivas para a publicação de dados abertos. Ela define, de forma geral, que os dados abertos devem ser disponibilizados de forma completa, em formato legível por máquina e sem restrições de utilização. Especificamente, dentre as diretivas apresentadas pela fundação citam-se:

- disponibilidade e acesso – definem que os dados abertos devem estar disponíveis de forma completa com um custo não maior do que o custo de reprodução, e de preferência disponíveis para download;
- reúso e distribuição – definem que os dados devem ser distribuídos com licença que permita sua reutilização incluindo a mistura com outras bases de dados, além de estar em um formato legível por máquinas; e
- participação universal – define que os dados devem viabilizar a participação universal, ou seja devem estar aptos a serem utilizados, reutilizados e distribuídos por qualquer pessoa sem discriminação a campos de atuação e grupos de pessoas, como restrições para fins não comerciais ou propósitos educacionais apenas.

Também houve um grande envolvimento de vários países na abertura de dados oficiais como forma de transparência do estado. Assim, surgiram grandes portais de pu-

[1]Open Knowledge Foundation: `https://okfn.org/`

blicação desses dados. A publicação desses dados na Web possui seus próprios desafios, como a indexação, catalogação e recuperação desses *datasets*. Geralmente, esses desafios são decorrentes da forma que os dados são publicados – como já mencionado, em formatos arquivos legíveis por máquina como XML (*Extensible Markup Language*), CSV (*Comma-separated Values*) e JSON (*JavaScript Object Notation*). Assim a organização dos dados é realizada via de regra através dos metadados que são informados sobre esses arquivos. Portanto, todas as operações de indexação, catalogação e recuperação de informação são realizadas em relação a seus metadados. Ao longo dos anos, surgiram diversas aplicações destinadas à publicação de catálogos de dados, que apesar de ainda não permitirem a recuperação de informações presentes dentro dos *datasets*, viabiliza a recuperação de informações sobre seus metadados. No universo de aplicações para dados de uso geral, destacam-se as ferramentas como CKAN[2] e Socrata[3]. São exemplos dessas grandes portais de dados públicos o portal Brasileiro[4], o Americano[5] e o Europeu[6].

Por outro lado, existem ainda bases de dados que possuem um tipo de dado específico, que são os dados espaciais. Para esse tipo de dados, existem aplicações específicas que permitem tanto a pesquisa sobre os metadados quanto a visualização dos dados espaciais, e até mesmo a sua visualização em conjunto por composição sobre formas de camadas. Essas aplicações recebem o nome de Infra-estruturas de Dados Espaciais (IDEs, ou em inglês SDI - *Spatial Data Infrastructure*). Existem diversas alternativas de implementações de IDEs, dentre elas estão o Geoserver[7], mais popularmente utilizada e o Mapserver[8]. Nesse caso os dados são disponibilizados sobre o um padrão serviços e formatos definidos pela *Open Geospatial Consortium* (OGC)[9].

5.2.2. Dados Conectados

Dados conectados estão no contexto de Web Semântica. Portanto, para compreender sua representação, é preciso primeiramente entender o propósito da Web Semântica, bem como a contextualização de pontos chave em sua de sua estruturação.

Web Semântica. A Web Semântica é uma extensão da Web tradicional que inclui informações sobre sentido e significado dos dados em suas páginas ou publicados abertamente. Sentido e significado dos dados são então introduzidos de forma que possam ser interpretados por aplicações, permitindo assim que executem tarefas mais complexas e de forma mais autônoma. Dessa forma, através desse tipo de informação, uma aplicação que coleta páginas da Web pode identificar sentido do termo "Bertha Lutz" como sendo o nome de uma pessoa, endereço (rua, avenida, bairro, etc), lugar (escola, hospital, etc), por exemplo. Partindo desse conhecimento, a aplicação de coleta de dados pode então executar ações mais precisas sobre esses dados, como armazená-los como dado espacial, descartar caso não seja o tipo de dado a qual a aplicação esteja interessada em coletar, ou ligar um

[2]CKAN: `https://ckan.org`
[3]Socrata: `https://socrata.com/`
[4]Portal Brasileiro de Dados Abertos: `http://dados.gov.br`
[5]data.gov: `https://www.data.gov`
[6]*European Data Portal*: `http://europeandataportal.eu`
[7]GeoServer: `http://geoserver.org`
[8]MapServer: `https://mapserver.org`
[9]OGC: `http://www.opengeospatial.org`

dado a outras informações encontradas anteriormente pelo processo de coleta, gerando assim um conjunto de dados mais complexo e com mais informações sobre seu contexto.

Na tarefa de representar o contexto e o significado dos dados, a Web Semântica propõe dois conceitos importantes: Ontologia e Vocabulário. O conceito de ontologia vem da filosofia e implica no estudo da natureza da existência e propriedade do seres e das "coisas". Para a Computação, as ontologias são modelos de dados que representam a descrição de seres, objetos e coisas existentes no mundo. Dessa forma, as ontologias na Web Semântica têm a função de descrever a que um dado se refere. No caso do exemplo anterior, sobre o termo "Bertha Lutz", o lugar pode ser descrito por ontologias como rua, avenida, bairro, escola, hospital. Dentro dos conceitos de Web Semântica, também foram definidas linguagens para a descrição de ontologias, como a mais comumente utilizada OWL (*Web Ontology Language*). Através da OWL é possível descrever ontologias utilizando componentes como: Classes, Atributos, Indivíduos, Relações, Termos funcionais, Restrições, Regras, Axiomas e Eventos [Sikos, 2015].

Por outro lado, os vocabulários são coleções de termos utilizados para descrever uma área de interesse. Por exemplo, um dos vocabulários mais utilizados para descrever "Pessoas" é o chamado FOAF (*Friend of a Friend*)[10], e apresenta termos que representam classes como *OnlineAccount*, *PersonalProfileDocument*, propriedades como *homepage*, *knows*, *accountName*, dentre outras. Os vocabulários em geral podem ser referenciados durante a publicação dos dados através de *namespaces*, que por sua vez são representados através de atributos *"xmlns"*.

Em resumo, os conceitos de ontologia e vocabulário são semelhantes. Inclusive, de acordo com a W3C[11] não existe uma separação clara entre os dois. Porém, é usualmente referido a modelos de descrição de dados mais complexos como Ontologias.

Linked Data. Os dados conectados, conforme próprio nome, representam dados que estão relacionadas entre si. Estão no contexto de Web Semântica por explicitarem os sentido dos dados, assim como o significado das relações representadas. Assim, são capazes de representar de forma ampla ambos o sentido dos dados e o contexto de suas relações.

Nesse contexto, um dos formatos para a representação dos dados mais utilizados é o RDF (*Resource Description Framework*). Esse formato é capaz de representar a descrição de qualquer tipo de dado presente na Web. Para isso, o *framework* possui um vocabulário próprio que permite descrever informações sobre os dados como literais, classes, propriedades, *statements*, listas, conjuntos e sequências [Sikos, 2015]. O RDF é estruturado através de sentenças que seguem a forma sujeito-predicado-objeto (ou recurso-propriedade-valor), também conhecidas por *triplas RDF*. As triplas, dessa forma, são capazes de expressar as propriedades de um conteúdo presente na Web, bem como seu contexto. Expressar as propriedades de um recurso é mais simples, pois as triplas representam a sentença recurso-propriedade-valor. Assim, esse formato de tripla pode ser utilizado para caracterizar o recurso através de suas propriedades. Por outro lado, o RDF pode representar uma relação entre recursos presentes na Web, utilizando a triplas da forma sujeito-predicado-objeto, onde o objeto é outro recurso também disponível na

[10]FOAF: `http://xmlns.com/foaf/spec/`
[11]W3C: `https://www.w3.org/standards/semanticWeb/ontology`

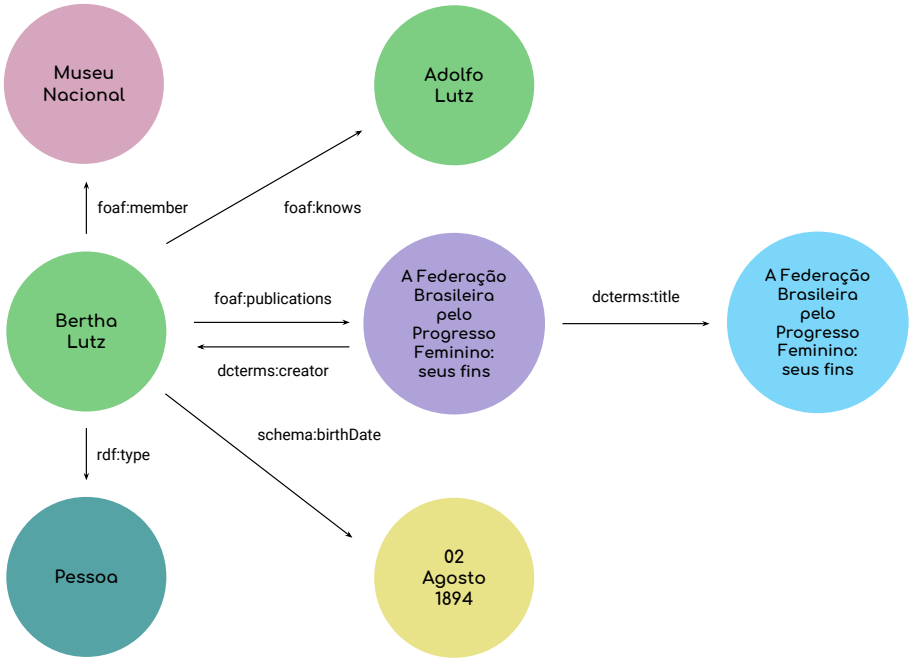


Figura 5.2: Representação gráfica do exemplo apresentado utilizando RDF em N-Triples

Web, e o predicado especifica o sentido da relação estabelecida entre ambos.

Voltando ao exemplo da Bertha Lutz, podemos criar um arquivo sobre o formato N-Triple que descreve a informação relacionada à bióloga brasileira, assim como outras informações relacionadas ao seu contexto. O Código 5.1 (a seguir) permite verificar as duas formas de utilização das triplas; ou seja, aplicada na descrição das propriedades de um recurso, e na descrição de sua conexão com outros recursos.

Código 5.1: Exemplo de RDF utilizando N-Triple

```
<https://www.wikidata.org/wiki/Q1264246> <rdf:type> <foaf:Person> .
<https://www.wikidata.org/wiki/Q1264246> <foaf:knows>
<https://www.wikidata.org/wiki/Q199652> .
<https://www.wikidata.org/wiki/Q1264246> <schema:birthDate> "02-08-1894"^^<xsd:date> .
<https://www.wikidata.org/wiki/Q1264246> <foaf:publications>
    <http://memoria.bn.br/DocReader/178691_05/36862> .
<https://www.wikidata.org/wiki/Q10301958> <dcterms:title>
    "A Federacao Brasileira pelo Progresso Feminino: seus fins" .
<https://www.wikidata.org/wiki/Q10301958> <dcterms:creator>
    <https://www.wikidata.org/wiki/Q1264246> .
```

O Código 5.1 utiliza uma forma de representar o RDF conhecido como N-Triples, mas existem outros formatos que seguem os conceitos propostos pelo RDF, como N-Quad e Turtle, que fazem parte da família das linguagens RDF chamada de *Turtle*, assim como o RDF/XML, RDFa e JSON-LD. A descrições dessas linguagens podem ser vistas na página da W3C sobre a utilização dos formatos para RDF[12].

Para compreender as relações estabelecidas pelas triplas RDF, o exemplo anterior é ilustrado na Figura 5.2. Assim, visualiza-se como as triplas estabelecem relações para: as propriedades de um recurso sobre a Bertha Lutz como sua a data de nascimento; e a

[12] https://www.w3.org/TR/rdf11-primer/#section-graph-syntax

relação entre recursos, como o predicado *foaf:publications*, que indica que Bertha Lutz publicou o artigo *A Federacao Brasileira pelo Progresso Feminino: seus fins*.

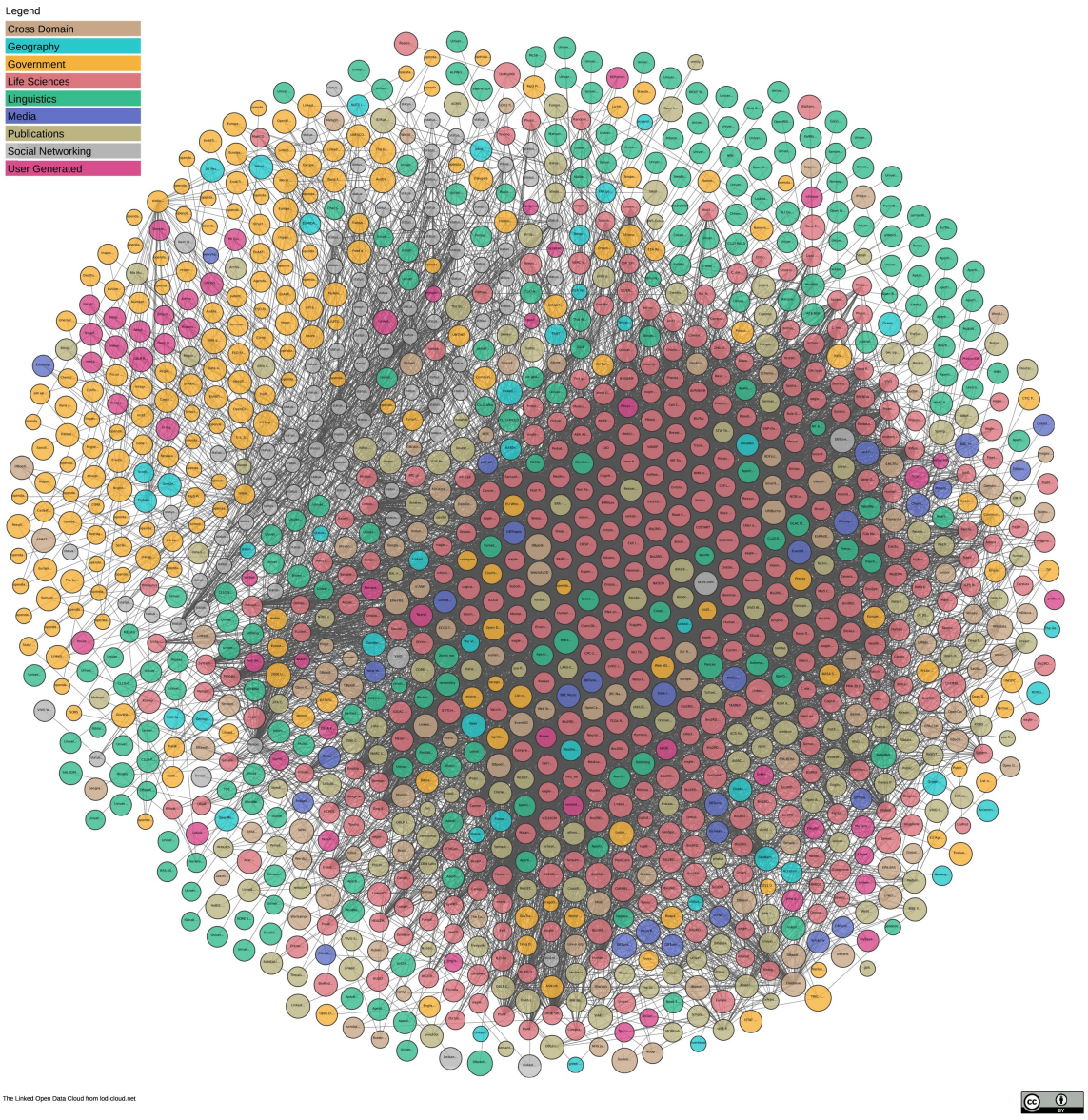


Figura 5.3: Distribuição das associações entre bases de dados conectados abertos. Fonte: *Linked Open Databases Cloud*

Os dados conectados recebem esse nome por representar e descrever as conexões entre recursos presentes na Web. Assim, é possível expressar as ligações entre conteúdos de contextos diferentes, como pessoas a objetos através de relações de pertencimento, interesses, etc. Nesse sentido existe um esforço também para a publicação desse tipo de dados de forma aberta. Uma vez que esses dados têm a capacidade de estarem interligados em diferentes contextos, a combinação dessas bases tem o potencial de gerar inúmeras aplicações complexas, uma vez que elas também carregam a semântica desses dados e relacionamentos. Nesse sentido, existe um esforço de padronização do processo de publicação desses tipos de dados sobre a forma de dados abertos. Os dados abertos

dessa natureza receberam o nome de *Linked Open Data* (LOD). Já existem diversas bases de dados LOD, como ilustrado na Figura 5.3[13], que começaram a ser publicadas em meados de 2007, com destaque para DBpedia, GeoNames e FreeBase.

5.2.3. Páginas da *Web*

Páginas da Web são a forma mais comum de dados que podem ser coletados. Tais páginas foram criadas para permitir a disseminação de informações na Web, com principal finalidade de exibir tais informações aos usuários obedecendo elementos visuais definidos pelos seus autores. Dessa forma, criou-se o *HyperText Markup Language* (HTML), o qual permite que as páginas sejam disseminadas mantendo a formatação estipulada pelos produtores de conteúdo. Os navegadores são então capazes de interpretar os arquivos HTML e realizar a diagramação do conteúdo da forma descrita por esses arquivos.

Devido ao propósito inicial da Web, suas páginas foram desenvolvidas de forma menos estruturada, pois apesar de os elementos visuais serem interpretados, não existe um modelo de dados comum entre páginas que garanta uma estrutura única entre as mesmas. Ou seja, ao desconsiderar os elementos de renderização, as páginas da Web não foram desenvolvidas visando a interpretação de seus dados por máquinas, e sim por humanos.

A coleta de dados dessas páginas, organizadas da forma tradicional, requer conhecimento de ferramentas de navegação e consulta em HTML, como o XPath (linguagem de consulta para documentos XML [Moro et al., 2009]) que permite a navegação na árvore da estrutura HTML. Além disso, apesar de não existir um modelo de dados bem definido para a Web como um todo, algumas vezes é possível encontrar um modelo de dados específico para alguns domínios. Isso ocorre quando páginas Web constituem uma forma de exibição de dados que já estão estruturados nas bases desses sites. Exemplos incluem perfis de usuários em redes sociais, páginas de venda de produtos, dados de diferentes publicações em uma biblioteca digital, entre outros.

Com o surgimento da Web Semântica, também houve um avanço nas convenções de utilização das *tags* HTML e em seus possíveis atributos a fim de compreender a descrição dos dados compartilhados. No caso das páginas da Web, a inserção de anotações semânticas permite que aplicações clássicas sejam melhoradas. Exemplos de tais aplicações são as máquinas de busca, que passaram a compreender melhor o conteúdo das páginas Web, aprimorando a indexação e a qualidade dos resultados das consultas. Dentre as principais alterações do HTML, destaca-se a inserção de metadados através de microformatos, RDFa, *microdata* para HTML5 e JSON-LD.

Os microformatos são uma das primeiras tentativas de inclusão da informação semântica nas páginas da Web e seguem ideias como, por exemplo, as do chamado *Plain Old Semantic HTML*[14]. Portanto, se baseiam na estratégia de reutilizar algumas partes dos atributos das *tags* já existentes para HTML (como *rel*, *class* e *rev*) para anotar o sentido e significado dos dados presentes em uma página. Existem também alguns padrões[15] para descrição de diversas entidades através dos microformatos, dentre os mais conhecidos pode-se citar o *hCalendar* para a descrição de eventos e o *hCard* para informações

[13]LOD Cloud: `https://lod-cloud.net/`
[14]Plain Old Semantic HTML: `http://microformats.org/wiki/posh`
[15]Microformats wiki: `http://microformats.org/wiki/Main_Page`

de contato de pessoas, companhias e organizações. Retomando novamente o exemplo da Bertha Lutz, pode-se publicar os dados sobre o evento da fundação da Liga para Emancipação Intelectual da Mulher, em que a mesma estava envolvida, assim como os dados sobre informações básicas da pesquisadora, como pode ser visto no Código 5.2.

Código 5.2: Exemplo de utilização dos padrões de microformato *hCalendar* e *hCard*.

```
<!-- hCalendar -->
<span class="vevent">
 <span class="summary">
  Fundacao da Liga para Emancipacao Intelectual da Mulher
 </span> on <span class="dtstart">1919</span>
 ocorrida no
 <span class="location">Brasil</span>
 organizada por <span class="organizer">Bertha Lutz</span>.
</span>

<!-- hCard -->
<div class="vcard">
 <span class="fn given-name">Bertha</span>
 <span class="fn family-name">Lutz</span>
 <div class="adr>
  <span class="type">Nascida em</span>
  <span class="locality">Sao Paulo</span>,
  <abbr class="region" title="Sao Paulo">SP</abbr>
  <span class="country-name">BRA</span>
 </div>
</div>
```

O RDFa é outra forma de anotar as informações semânticas em páginas HTML, especificamente voltado para o conceito de dados conectados. O RDFa propõe a implementação dos conceitos de triplas do RDF através de novos atributos (*vocab*, *typeof*, *property*, *resource*, e *prefix*) que são inseridos em *tags* HTML tradicionais. De forma geral, a página que contém o RDFa representa o sujeito das triplas, quando este não é destacado pelo atributo *resource*, como apresentado no Código 5.3 através da utilização do atributo na *tag div* mais externa. No caso do predicado, este é indicado pelo atributo *property*; e o objeto a que esse predicado se refere é o conteúdo existente dentro da *tag*, como foi utilizado na *tag h2* que contém o título do artigo da Bertha Lutz no Código 5.3. Os demais atributos indicam informações presentes no RDF, como vocabulário, tipos e prefixo, o qual é utilizado no RDF para referenciar um *namespace* ou vocabulário, exemplificado na *tag body* (Código 5.3).

Código 5.3: Exemplo de utilização do RDFa

```
<html>
  <head>
    ...
  </head>
  <body vocab="https://bib.schema.org/Thesis">
   <div
    resource="/bertha_lutz/publications/thesis"
    typeof="Thesis">
    <h2 property="https://schema.org/headline">
     A Nacionalidade da Mulher Casada perante
     o Direito Internacional Privado
    </h2>
    <h3
     property="https://schema.org/author"
     resource="#me">
      Bertha Lutz
    </h3>
    <div property="text">
```

```
      <!-- article content -->
     </div>
    </div>
   </body>
</html>
```

Outra forma de anotar a semântica dos dados em uma página da Web é por meio de *Microdata* para HTML5. Essa abordagem foi desenvolvida especificamente para HTML5, e portanto é compatível com a mesma (ao contrário dos Microformatos, por exemplo, que em alguns casos apresentam incompatibilidade no reuso de alguns atributos em elementos quando se utiliza HTML5 para codificar uma página). O *Microdata* utiliza o padrão chave-valor também através do atributos próprios introduzidos por esse formato, os quais incluem *itemscope*, *itemtype*, *itemprop*, *itemid*,*id*, *itemref*. Especificamente, o *itemprop* deve ser utilizado para indicar um novo escopo de objeto, como pode ser observado na primeira *tag section* Código 5.4. O atributo *itemprop* em geral deve ser acompanhado da informação semântica desse objeto através do atributo *itemtype*, o qual pode assumir valores que são uma referência a um vocabulário ou ontologia.

No Código 5.4, a primeira *tag section* é acompanhada da informação de que se trata de uma pessoa, descrita pelo vocabulário FOAF, identificado por esse atributo *itemtype*. O atributo *itemid* pode ser utilizado para indicar qual o identificador do objeto descrito no contexto do site ou aplicação Web. Portanto, pode ser utilizado por sites que exibem objetos que estão catalogados em um banco de dados, por exemplo. Esse atributo é utilizado no Código 5.4 como atributo da *tag section*, com valor 3309.

Por fim, os atributos de *id* e *itemref* são utilizados quando se deseja relacionar dois objetos em uma mesma página, quando esses objetos não estão aninhados, por exemplo. O *id* não necessariamente deve assumir o mesmo valor semântico do atributo *itemid*, apresentado anteriormente, mas deve ser consistente com sua referência utilizando *itemref*. Um exemplo de utilização dessas refernências pode ser observado no Código 5.4 quando é realizada uma descrição das publicações de Bertha Lutz. Os elementos que descrevem as informações básicas da feminista são apontados na primeira *section*; posteriormente, nas segunda *section*, essa referência é retomada para a descrição de suas publicações.

Código 5.4: Exemplo de utilização do *microdata*

```
<html>
 <head>
  <title>Bertha Lutz</title>
 </head>
 <body>
  <section itemscope itemtype="foaf:person"
   id="berthaLutz" itemid='3309'>
   <h1>
    <span itemprop="foaf:firstName">Bertha</span>
    <span itemprop="foaf:surname">Lutz</span>
   </h1>
  </section>
  <section itemref="berthaLutz" itemscope
   itemtype="foaf:publications">
   <div itemprop="schema:article">
    <a itemprop="rdf:resource"
     href="http://memoria.bn.br/docreader/178691_05/36862">
      A Federacao Brasileira pelo Progresso Feminino: seus fins
    </a>
    <span itemprop="dcterms:creator">Bertha Lutz</span>
    <span itemprop="dcterms:creator">Carmem de Carvalho</span>
    <span itemprop="dcterms:creator">Orminda Bastos</span>
```

```
    </div>
   </section>
  </body>
</html>
```

Dentre os formatos mencionados para a publicação de dados em páginas da Web com sua informação semântica, o JSON-LD é o que mais se diferencia. Esse formato utiliza a *tag* de *script* da linguagem HTML para disponibilizar todos os objetos referenciados na página utilizando JSON (*JavaScript Object Notation*). Para isso, a tag de *script* é acompanhada do atributo *type* que indica se tratar de um código no formato JSON-LD. Seguindo a lógica de anotação semântica, a chave *@context* e *@type* definem o vocabulário ou ontologia a qual o objeto se trata. Além dessas chaves, *@id* é outra chave definida pelo modelo e deve ser utilizada para anotar o identificador desse objeto no respectivo conjunto de dados. Além de tais propriedades, podem ser utilizadas quaisquer outras propriedades definidas pela ontologia. Um exemplo de sua utilização pode ser visto através do exemplo da Bertha Lutz no Código 5.5.

Código 5.5: Exemplo de utilização do JSON-LD

```
{
  "@context": "https://json-ld.org/contexts/person.jsonld",
  "@id": "https://pt.wikipedia.org/wiki/Bertha_Lutz",
  "name": "Bertha Lutz",
  "born": "1894-08-02",
  "father": "https://pt.wikipedia.org/wiki/Adolfo_Lutz"
}
```

5.2.4. APIs

Em meados de 2007, a Web passou por uma transformação na forma de organizar suas páginas. Essa transformação aconteceu pelo surgimento dos conceitos da Web 2.0 que mudam a forma com que os dados publicados na Internet são produzidos. Especificamente, a Web 2.0 trouxe uma nova visão sobre como melhor utilizar os recursos já disponíveis na Web para incluir os até então apenas *viewers* das páginas Web na produção do conteúdo que circula na rede. Assim também surgiu a ideia de *aplicações Web*, que agem como software tradicional para *desktop*, incluem as funções de criação, modificação, persistência dos dados e visualização conhecido pelo padrão CRUD (*Create*, *Retrieve*, *Update* e *Delete*). As aplicações Web se diferenciam das aplicações em *desktop* por permitirem acesso de qualquer navegador Web, estando disponíveis sem a necessidade da instalação de algum programa.

Por se tratar de um ambiente diferente do ambiente *desktop*, houve a necessidade do desenvolvimento de novos conceitos sobre a arquitetura de aplicações. Então, Fielding [2000] propõe a arquitetura em camadas, que em linhas gerais é amplamente utilizada atualmente nesse tipo de aplicação. Essa arquitetura divide as aplicações em componentes que estão distribuídos em três tipos principais de camadas: Aplicação, Serviços e Persistência. A Figura 5.4 ilutstra tal arquitetura. A camada de aplicação contém todas as aplicações que podem ser criadas com um mesma lógica do conjunto de dados presente na camada de Persistência. As APIs surgem então como componentes que integram a camada de Serviços (que é responsável por organizar toda a lógica das operações de CRUD) e controlam as permissões de execução desses tipos de operação em relação aos usuários que requisitam a operação, dentre outras atribuições de segurança.

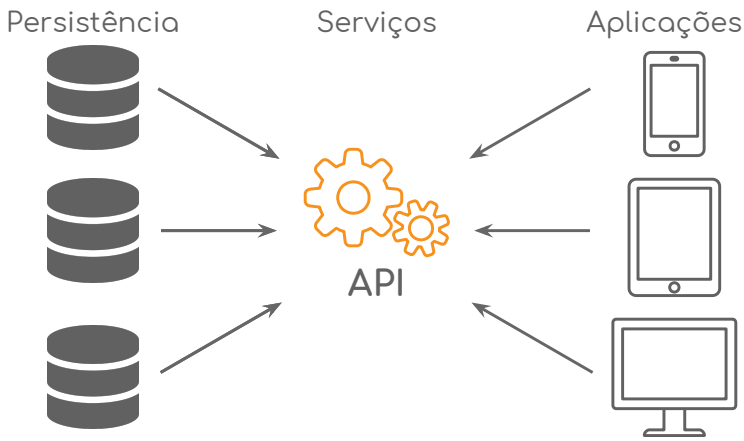


Figura 5.4: Arquitetura em camadas de aplicações Web

Para gerar aplicações secundárias que revertam em uma maior utilização dessas aplicações Web, as empresas comumente concedem o acesso a APIs que permitem a manipulação até certo ponto de seus dados através de requisições ao serviço. Essa estratégia permite que essas aplicações secundárias possam ser desenvolvidas sem que as empresas necessariamente tenham que despender recursos para isso. Através dessa concessão, essas APIs se tornam outra fonte de dados na Web.

Além disso, as APIs expandiram sua utilização com o conceito de arquitetura orientada a serviços (em inglês: *Service-Oriented Architecture* - SOA). Nesse sentido, o SOAP (*Simple Object Access Protocol*) foi o primeiro padrão de serviço a ser formalizado. Esse padrão estabelece que as requisições ao serviço devem ser realizadas através do protocolo HTTP (*Hypertext Transfer Protocol*) e RPC (Remote Procedure Call), e as mensagens devem ser trocadas utilizando o formato XML. Anos mais tarde, houve a proposta de um nova arquitetura de serviços chamada de REST (*Representational State Transfer*) [Fielding, 2000], ou RESTful. A grande maioria dos serviços implementados com o padrão REST utiliza também o formato JSON (*JavaScript Object Notation*) para as mensagens trocadas através do protocolo HTTP.

Note que o acesso a essas APIs é realizado com a *concessão* das suas empresas fornecedoras. Portanto ao contrário dos dados anteriores de acesso irrestrito, o acesso aos dados fornecidos pelas APIs requer um cadastro prévio. Esse cadastro em geral é realizado em nome das aplicações que desejam utilizar a API e não em nome dos desenvolvedores; e nele, os desenvolvedores garantem que essa aplicação seguirá as políticas de acesso e utilização dos dados impostas pelas empresas. Após o cadastro, as aplicações recebem uma chave (ou conjunto de chaves) de acesso que deve ser utilizada para autenticar as requisições realizadas à API. Através dessa autenticação, as empresas podem controlar quais aplicações requisitam suas informações e aplicar suas políticas de acesso.

As políticas de acesso variam entre APIs, mas em geral elas tentam garantir que o serviço não seja sobrecarregado por requisições, e que parte dos dados dos usuários só possam ser acessados se o próprio usuário concedeu essa permissão à aplicação. O primeiro caso de política de acesso é comum em praticamente todas as APIs, e é definido por medidas como número de requisições por intervalo de tempo (minutos, horas, dias,

por exemplo), ou por intervalo de tempo entre requisições (segundos, minutos, por exemplo). Em alguns casos, o descumprimento dessa política pode acarretar em penalizações às aplicações, em geral sobre a forma do interrompimento temporário da resposta às suas requisições. Dessa forma essa política deve ser observada pelas aplicações que consomem os dados das APIs, para que não impeça seu funcionamento.

O segundo tipo de política de acesso é relativa a aplicações que apresentem dados de usuários, em geral aplicações que implementam um protocolo de autenticação conhecido como OAuth, que atualmente possui duas versões. Esse protocolo de autenticação permite que os usuários concedam permissão às aplicações terceiras para acessarem seus dados. Em alguns casos, as empresas exigem que os desenvolvedores especifiquem quais dados as aplicações desejam acessar dos usuários, para que estes sejam informados ao conceder o acesso as seus dados.

Uma das vantagens dos dados presentes em APIs é que os mesmos apresentam um formato e um esquema claros e documentados para que seus usuários possam as utilizar. Por essas APIs seguirem um protocolo bem definido, o formato das suas mensagens em geral são sempre em XML e JSON. Por outro lado, o esquema dos dados fornecidos é característico de cada API, e em geral as empresas disponibilizam uma documentação sobre esses esquemas. Além do formato dos dados, no caso das APIs é importante entender o formato das requisições, tanto para realizar o processo de autenticação, quanto para compreender quais *endpoints* estão disponíveis para realizar as ações de coleta e manipulação dos dados pelas aplicações desenvolvidas. A documentação cobre todas essas informações, bem como as restrições de utilização desses *endpoints*, como as políticas de acesso, que também podem variar em uma mesma API.

5.3. Coleta de Dados Web

A partir dos tipos existentes de publicação de dados na Web (discutidos na seção anterior), existem diversas formas de coletá-los. Nos dois primeiros tipos de dados apresentados (Dados abertos e Dados conectados), ambos podem ser obtidos de forma completa pelos sites de dados abertos. Esses dados são disponibilizados de forma estruturada, sobre formatos legíveis por máquina como XML, CSV e JSON, para os dados abertos de forma geral; e em forma de RDF/XML, N-Triples, N-Quad no caso de dados abertos conectados (LOD). Normalmente é possível obter todos os dados sobre determinado assunto de uma fonte através de seu download, uma vez que esses sites já oferecem esse conteúdo organizado para os usuários.

De um outro modo, páginas da Web e APIs consideram dados fragmentados, ou espalhados, e requerem a utilização de uma aplicação para coletá-los e organizá-los em uma base estruturada pelo desenvolvedor. No caso das páginas da Web, os dados estão espalhados em diversos arquivos, que dependendo da intenção de sua utilização pelo usuário, podem estar espalhados inclusive sobre domínios diferentes, como o caso da indexação de documentos para máquinas de busca. No caso das APIs, apesar de o conteúdo estar bem formatado, o relacionamento entre as informações disponibilizadas é que pode não estar junto. Por exemplo, como as relações de amizade em redes sociais e os perfis dos usuários que fazem parte da relação de amizade. Além disso, o tamanho da base de dados que essas fontes cobrem é muito grande, e é possível que nem todos os dados des-

ses contextos sejam necessários para a análise desejada. Por esses motivos, nesses casos são utilizadas aplicações que realizam a coleta de forma organizada, com definição clara sobre quais informações serão coletadas, assim como, de que forma as mesmas serão armazenadas para que o usuário possa utilizá-las. Essa aplicações de coleta de dados são comumente chamados de coletores ou, em inglês, *crawlers*.

Coletores Web. Os coletores são comumento utilizados em diversas aplicações de áreas como recuperação de informação e redes complexas. Do ponto de vista da recuperação de informação, Baeza-Yates e Ribeiro-Neto [2011] enumeram seis tipos de de aplicações específicas desse contexto: busca na Web de forma geral, busca por tópicos específicos, arquivos de páginas da Web, caracterização da Web, análise de sites Web, e espelhos da Web. Do ponto de vista de redes complexas, os coletores podem ser utilizados em estudos sobre redes sociais on-line (*On-line Social Networks*, ou OSNs, em inglês) envolvendo a análise das relações entre seus usuários, por exemplo. Além disso, assim como a recuperação de informação, existem estudos de redes complexas que analisam o comportamento do grafo formado pelas páginas da Web.

Baeza-Yates e Ribeiro-Neto [2011] apresentam ainda três aspectos que os coletores exploram: atualização, qualidade e volume. O primeiro aspecto é tratado por coletores que estão constantemente funcionando e mantendo as bases atualizadas. O segundo presa por porções da Web que possuam melhor qualidade (que sua definição pode variar entre aplicações) nos dados disseminados, e que essa qualidade seja homogênea entre as páginas. Por fim, o volume trata da quantidade de páginas coletadas e pode sofrer redução em favor de se manter apenas dados atualizados, ou de melhor qualidade.

Desafios para Coleta. A coleta de dados possui ainda problemas específicos que devem observados como: tempo entre requisições, erro soft-404, identificação dos padrões de URL das páginas e extração dos dados do código fonte. O primeiro caso está presente tanto em páginas da Web quanto em APIs. No caso das páginas da Web, essa informação é disponibilizada pelos administradores dos sites em um arquivo chamado *robots.txt*. Esse arquivo formaliza as linhas gerais de boas práticas que os coletores devem seguir quando estão coletando suas páginas. Assim, nesses arquivos estão presentes as informações de tempo de requisição, bem como quais as seções do site o administrador permite que sejam coletadas, e quais gostaria que não fossem verificadas.

A segunda questão trata do erro soft-404. O erro 404 ocorre quando se tenta acessar uma página que não existe, e em geral essa resposta é dada diretamente pelo servidor. Existe uma pequena parcela de sites (29% dos links de páginas ausentes de acordo com [Baeza-Yates and Ribeiro-Neto, 2011]) que retornam uma página (com código de resposta 200 do servidor), porém essa página contém apenas a informação de que a página não foi encontrada. Esse problema aumenta a complexidade dos coletores e piora a eficiência do tempo de coleta, uma vez que somente é possível verificar se a página retornada é válida quando a mesma é aberta.

A identificação de padrões de URL permite que o coletor possa ser implementado utilizando a técnica de tentativa e erro. Em várias casos, os sites representam uma visualização dos dados presentes em bancos de dados. É comum utilizar uma padronização das URLs em função dos identificadores dos registros presentes nessas bases. Assim, sabendo

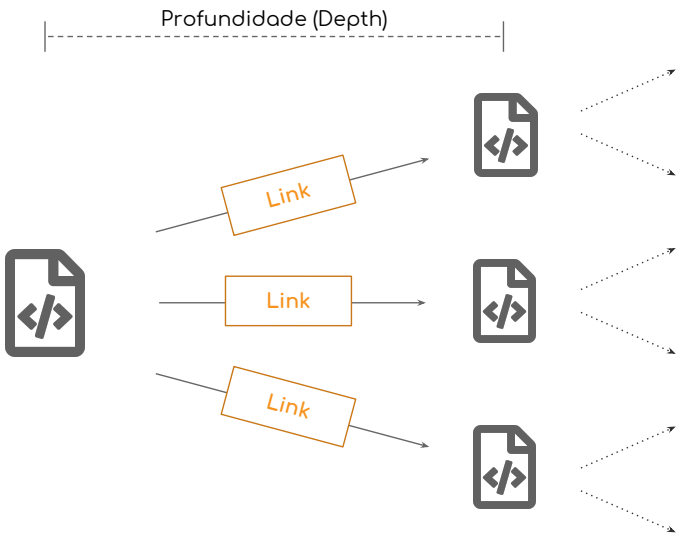


Figura 5.5: Caminhamento no grafo de páginas da Web

o número máximo de registros, é possível prever qual a estatura da chave desses registros (como registros sequenciais). Partindo dessa informação, o coletor fica responsável por verificar quais dessas chaves previstas levam a dados realmente existentes.

Finalmente a extração de código fonte é outra tarefa que depende do conhecimento de ferramentas de *parsing* ou caminhamento na árvore HTML das páginas Web. A inclusão da etapa de *parsing* durante a coleta reduz o volume de dados que serão armazenados, uma vez que todo o código HTML é removido.

Principais Técnicas de Coleta. Os coletores se baseiam em duas principais técnicas: caminhamento em grafo e amostragem probabilística. As técnicas baseadas em caminhamento em grafo realizam a coleta percorrendo o grafo de relações existente entre os objetos coletados. Esse tipo de coleta é muito útil quando não se tem informação sobre quantos objetos existem no conjunto de dados a ser coletado, assim como o endereço exato onde cada objeto se localiza ou qual identificador que pode ser utilizado para recuperá-lo. Sabe-se apenas que esses objetos estão conectados sobre algum tipo de relação.

Um exemplo de aplicação dessa técnica é a coleta de páginas da Web por máquinas de busca. Nesse caso, o tamanho do conjunto de dados é definido por todas as páginas da Web, e em geral não é possível saber antes da coleta o endereço Web de todas as suas páginas. Sabe-se que páginas da Web estão interligadas através de hiperlinks, que estabelecem uma relação de citação entre páginas. Assim é possível desenvolver um coletor que partindo de uma página inicial (comumente chamada de semente) seja capaz de encontrar novas páginas na Web através dos endereços citados por meio de hiperlinks. Dessa forma a coleta é realizada percorrendo o grafo formado pelas páginas da Web e seus hiperlinks, como o exemplo genérico ilustrado na Figura 5.5.

Existem três técnicas baseados em caminhamento em grafo para a coleta de dados: busca em largura (*Breadth-First Search*, BFS), *Snowball Sampling* e busca em profundidade (*Depth-First Search*, *DFS*). A BFS e a DFS são as técnicas mais simples de serem implementadas. Conforme o Algoritmo 1, a cada iteração, a BFS possui um conjunto de

Algoritmo 1 Algoritmo de busca em largura (BFS)

```
1: L conjunto de links da iteração i, inicialmente contém as sementes
2: LNext conjunto de links da iteração i+1, inicialmente vazia
3: AllLinks conjunto de todos os links obtidos
4: dMax altura máxima
5: d ← 0 altura atual
6: while d < dMax do
7:     for all l ∈ L do
8:         if l ∉ AllLinks then
9:             p ← collect(l)
10:            LNext.insert(p.links)
11:            AllLinks.insert(l)
12:    L ← LNext
13:    LNext ← ∅
14:    d ← d + 1
```

Algoritmo 2 Algoritmo de busca em profundidade (DFS)

```
1: S pilha de links, inicialmente contém apenas as sementes
2: AllLinks conjunto de todos os links obtidos
3: dMax altura máxima
4: d ← 0 profundidade atual
5: while d < dMax ∨ P = ∅ do
6:     l ← S.pop()
7:     if l ∉ AllLinks then
8:         p ← collect(l)
9:         S.push(p.links)
10:        AllLinks.insert(l)
11:    d ← d + 1
```

links de páginas que serão coletadas, sendo o conjunto inicial chamado de semente. A cada link, é verificado se a coleta já foi realizada para esse link; caso não tenha sido coletada, é então realizada sua coleta. Após a análise de cada página, obtem-se um conjunto de links que podem ser utilizados para acessar novas páginas (também chamadas de páginas vizinhas). Esse conjunto de links será utilizado em uma nova iteração do algoritmo.

O algoritmo de *Snowball Sampling* é bem similar ao BFS; a única diferença é que considera apenas uma amostra de k links presentes em cada página, limitando assim a largura máxima da árvore a k ramificações [Goodman, 1961]. No *Snowball* também é realizada a verificação se o novo conjunto de links já foi coletado.

Por fim, o algoritmo de busca em profundidade DFS, como o nome sugere, prioriza a exploração dos filhos das árvores antes de seus irmãos. O Algoritmo 2 apresenta o pseudo-código para DFS utilizando pilhas. Novamente, o algoritmo inicia com uma conjunto de links para páginas semente, que estão presentes na pilha S. Antes de coletar uma página, o algoritmo verifica se essa página já foi explorada. Os links da página coletada são então extraídos e inseridos no topo da pilha. Na próxima execução, o link a ser desempilhado é o primeiro filho da última página coletada. Dessa forma o algoritmo sempre

Figura 5.6: Principais etapas para integração de dados de múltiplas fontes utilizando ETL.

prioriza a coleta do primeiro filho de maior profundidade até que a profundidade máxima seja atingida. Após esse ponto, são desempilhados primeiramente os irmãos do primeiro filho de maior profundidade até que todos os ramos da árvore tenham sido explorados.

5.4. Integração dos Dados

Após a coleta de dados de múltiplas fontes, o próximo passo é *integrar*. Em resumo, a integração de dados consiste em combinar dados de diferentes fontes para obter informações valiosas. Tal tarefa tem sido foco de muitos estudos devido à ampla quantidade de dados heterogêneos disponíveis na Web [Doan et al., 2018, Freitas et al., 2017, Golshan et al., 2017]. A integração é importante para permitir que usuários tenham uma visão unificada de dados heterogêneos e consultem facilmente diferentes informações sobre os mesmos [Bouzeghoub et al., 2002]. Além disso, essa integração permite considerar várias definições/visualizações sobre um objeto. Por exemplo, Ma et al. [2017] usam dados de múltiplas fontes para identificar diversos efeitos colaterais de drogas; e Freitas et al. [2017] propõem um modelo baseado em ontologias e ligação de dados (*Linked Data*) para integrar conjuntos de dados de forma a permitir o cálculo da probabilidade do risco de óbito materno e infantil. Assim, os usuários podem descobrir melhor o conhecimento a partir de múltiplos dados e, então, essa integração pode fornecer suporte para a tomada de decisões, entre várias outras aplicações.

No entanto, existem diferentes desafios na integração de dados de múltiplas fontes, principalmente porque a maioria dos dados da Web são heterogêneos, não estruturados ou semi-estruturados. Além disso, os dados de várias fontes da Web possuem modelos distintos, diferentes representações de objetos do mundo real e nem sempre são confiáveis. Outro desafio relevante é manter o esquema de dados consistente após a integração, pois a cada alteração na fonte de dados é necessário verificar se as mudanças precisam ser propagadas pelo esquema, se novas consultas aos dados precisam ser elaboradas ou se o esquema precisa ser reescrito [Bouzeghoub et al., 2002, Laender et al., 2009].

Uma abordagem bastante utilizada e comum para integração de dados é a ETL: extração, transformação/limpeza e carregamento (do inglês *Extract*, *Transform* e *Load*) [Azeroual et al., 2018, Bansal, 2014], sumarizada na Figura 5.6. Considere diferentes conjuntos de dados, extraídos de fontes distintas, fornecidos como entrada para realização de um determinado estudo, os quais precisam ser integrados. Para tal, existem três etapas principais: (*i*) *extração* representa a aquisição de dados de múltiplas fontes; (*ii*) *transformação e limpeza* referem-se à padronização e limpeza dos dados, sendo nem sempre obrigatórias, mas boas práticas; e (*iii*) *carregamento* refere-se à inserção dos dados em

Tabela 5.1: Exemplos de abordagens para integração de dados e respectivas aplicações.

Abordagem	Aplicação
Sistema de mediação	Fornece uma visão única dos dados em formatos distintos para o usuário
Processamento de linguagem natural	Processa dados em formato de texto para permitir a padronização e posterior integração de tais dados
Abordagem Bayesiana	Lida bem com dados incompletos e inconsistentes de modo a garantir que a base resultante seja confiável

um sistema de organização incluindo, por exemplo, planilhas (apesar de não serem muito recomendadas para armazenamento e gerenciamento de grandes volumes de dados), armazém de dados, sistemas de gerenciamento de bancos de dados, etc.

É importante enfatizar que a ETL e a integração de dados são conceitos distintos. ETL são ferramentas de software, enquanto integração de dados é uma arquitetura[16]. Assim, a ETL pode ser utilizada como parte da etapa de integração de dados, mas não necessariamente representar toda a integração. Além da ETL, a tarefa de integração de dados também inclui modelagem de dados, criação de perfil dos dados, processamento de dados estruturados e não-estruturados, integração em tempo real, governança de dados, entre outras [Doan et al., 2018]. Ou seja, a área de integração de dados é bastante ampla e abrangente, e apenas as principais estratégias são abordadas aqui.

Na prática, pode-se coletar dados de cada fonte e armazená-los separadamente para posterior integração; ou armazenar todos os dados em um único local de forma integrada à medida que cada coleta de dados (por meio de rastreamento) é realizada. Há muitas vantagens e desvantagens das duas estratégias e diferentes formas de armazenamento. Especificamente sobre a primeira abordagem, pode ocorrer o armazenamento de diferentes volumes de dados em um único repositório chamado Lago de Dados (ou *Data Lake*). Tal armazenamento permite que os usuários façam diferentes consultas aos dados.

Entre muitas estratégias para integração de dados de múltiplas fontes, citamos: sistema de mediação [Bouzeghoub et al., 2002] que é uma abordagem para mapear diferentes fontes de dados em um esquema global; processamento de linguagem natural [Ma et al., 2017] que permite converter texto de diferentes fontes em códigos de identificadores exclusivos; e abordagem bayesiana [Wang et al., 2017, Zhao et al., 2012] que fornece uma maneira de integrar informações de confiabilidade em vários níveis.

Uma análise da definição e aplicação dessas abordagens revela que existem diferentes situações em que cada uma delas pode ser melhor utilizada para integrar dados, conforme mostra a Tabela 5.1. Ou seja, é necessário avaliar prós e cons para uma escolha mais acertada. Além disso, note que tais abordagens podem ser utilizadas de forma combinada. Por exemplo, o processamento de linguagem natural pode ser utilizado para identificar características relevantes no texto e possibilitar o armazenamento de tais dados de forma padronizada. Em seguida, a abordagem Bayesiana pode ser aplicada para inte-

[16]Diferença entre ETL e integração de dados: `https://www.passionned.com/is-data-integration-becoming-the-new-etl/`. Acessado em 20 de agosto de 2018.

Figura 5.7: Exemplo de integração de dados horizontal: as duas tabelas superiores representam clientes e os produtos comprados por eles, respectivamente. A tabela inferior é a combinação dessas duas tabelas. Observe que são perdidas informações individuais dos clientes e duplicatas são inseridas (exemplo adaptado de [Berthold et al., 2010]).

id	**nome**	**sobrenome**	**gênero**
c2	Joana	Oliveira	F
c5	Isis	Lima	F
c7	Rafael	Silva	M
...	...	...	...

cliente_id	**item_id**	**preço**
c2	i254	12,50
c5	i4245	1,99
c5	i32123	1,29
c5	i254	12,50
c5	i21435	5,99
c7	i254	12,50
...	...	...

item_id	**preço**	**nome**	**sobrenome**	**gênero**
i254	12,50	Joana	Oliveira	F
i4245	1,99	Isis	Lima	F
i32123	1,29	Isis	Lima	F
i254	12,50	Isis	Lima	F
i21435	5,99	Isis	Lima	F
i254	12,50	Rafael	Silva	M
...	...	...	...	...

grar os dados de forma a garantir a confiabilidade. Por fim, o sistema mediador pode ser utilizado para garantir uma visão única dos dados.

Outras estratégias usuais de integração de dados são específicas para armazenamento em bancos de dados relacionais [Berthold et al., 2010]. Um primeiro tipo de estratégia aborda o problema em uma perspectiva vertical, na qual as tabelas com essencialmente as mesmas informações são concatenadas. Por exemplo, unir uma tabela que representa fornecedores de equipamentos eletrônicos com uma que armazena fornecedores de baterias pode resultar em apenas uma tabela chamada *fornecedores* com as mesmas colunas das outras duas. Já um segundo esquema aborda o problema em uma perspectiva horizontal, na qual diferentes tipos de dados são combinados com o objetivo de enriquecer uma tabela existente, conforme mostra a Figura 5.7. Em geral, ambas as estratégias são simples, mas não isentas de questões relevantes. Por exemplo, lidar com duplicatas é necessário na estratégia vertical, enquanto o tratamento de excesso de representação e de explosão de dados é necessário na horizontal.

Esses dois últimos problemas estão presentes na Figura 5.7. Por exemplo, se a tabela resultante fosse utilizada para determinar o gênero dos compradores, o resultado seria um alto número de compradores do sexo feminino devido ao excesso de representação. Nesse caso, o ideal é não fazer consulta a tal tabela para buscar informações referentes aos compradores. Em relação à explosão de dados, também é um problema visível, pois as duplicatas que claramente não são necessárias para manter as informações essenciais.

Assim, o ideal é encontrar uma configuração do banco de dados para mantê-lo normalizado. A junção mostrada nessa figura é apenas um exemplo simples do que é possível realizar em um sistema real.

A mineração de dados relacionais (ou mineração em bancos de dados multirelacionais) consiste em encontrar informação em múltiplas tabelas [Berthold et al., 2010, Flach, 2001]. Nesse contexto, a integração de dados consiste em uma forma de transformar um problema difícil com múltiplas tabelas em um possível de lidar com métodos já existentes para uma única tabela. Entretanto, tais métodos, em geral, são bem especializados e consideram que existe uma única tabela bem formada com todos os dados relevantes. Assim, essa é uma simplificação, e a integração de dados requer muito mais tempo e esforço do que a própria análise dos dados.

5.5. Pré-Processamento dos Dados

Dados Web fornecem recursos inestimáveis para muitos pesquisadores e desenvolvedores. No entanto, esses dados podem vir com muitos problemas, especialmente quando coletados de várias fontes da Web, incluindo: valores ausentes, dados falsos (veracidade dos dados), dados duplicados, redução de dados e falta de padronização. Tais problemas geralmente são resolvidos na etapa de pré-processamento de dados, que requer cerca de 80% do tempo de cientistas de dados (como já foi apontado por muitos estudos como o de Tyagi et al. [2010]). Além disso, a representação dos dados coletados pode não ser ideal para o processamento dos algoritmos e análises sobre os mesmos. Dessa forma, é necessário converter os dados obtidos para representações adequadas.

As estratégias de pré-processamento de dados variam de acordo com o contexto e o tipo de dados. Para cada tipo de problema há algumas sugestões de solução; entretanto, a aplicação da solução aos dados deve ser analisada para cada caso. Por exemplo, no caso de valores ausentes é possível escolher completar os valores buscando de alguma outra fonte (quando disponível) ou eliminando os registros que não têm a informação completa. A seguir, discutimos cada um dos problemas apontados, apresentando sugestões de solução a partir de exemplos disponíveis na literatura.

5.5.1. Valores Ausentes

Os valores ausentes (*missing values*) ocorrem quando nenhum valor é armazenado para uma variável (ou um atributo) da base de dados. Os valores ausentes podem ocorrer em apenas alguns registros (quando existem registros que possuem o valor e registros que não possuem, por exemplo), ou podem ser dados importantes para a análise que não estão disponíveis para nenhum registro da base de dados. A questão, então, é se é possível extrair os dados de alguma outra fonte ou o banco de dados deve ser limpo para remover tais valores [Alves et al., 2016]. Em alguns casos, é possível buscar outra fonte de dados que tenha a informação específica disponível e que seja de fácil integração com a base atual. Avaliar a viabilidade da integração dos dados é importante, uma vez que fontes diferentes podem não apresentar um elo de ligação dos registros para que eles sejam relacionados. Como uma terceira opção, pode ainda ser encontrado um valor padrão (*default*) para substituir tais dados faltantes, quando necessário.

Um exemplo de *missing values* foi tratado por Alves et al. [2016] no trabalho re-

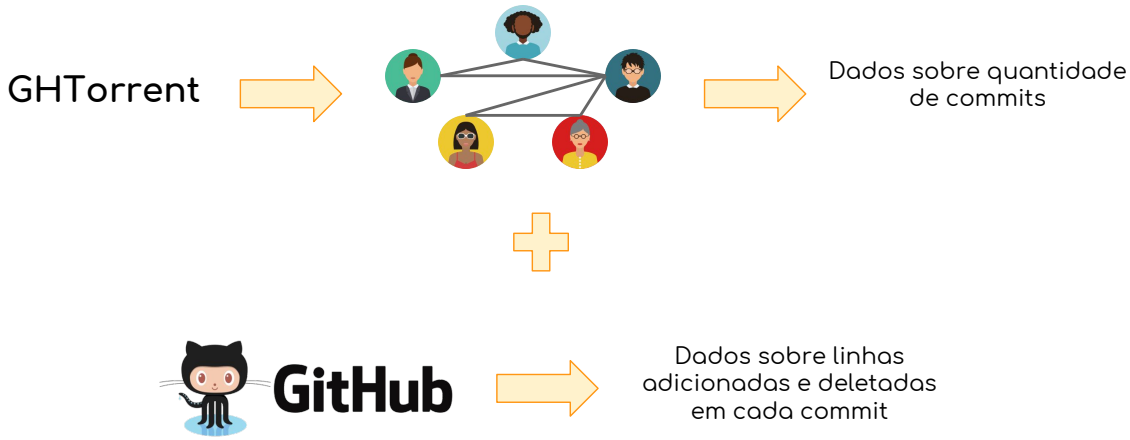


Figura 5.8: Correção de valores ausentes na rede de colaboração do GitHub [Alves et al., 2016]

lacionado ao GitHub. O trabalho apresenta uma métrica para a força de interação entre desenvolvedores do GitHub. Para isso, a partir de uma coleta pelo GHTorrent[17] foi construída a rede de colaboração. Entretanto, para uma métrica específica, observou-se que avaliar a quantidade de commits de um par de desenvolvedores não era suficiente e não estava claro o impacto desses commits. Para detalhar as alterações feitas efetivamente no código, foi necessário avaliar o número de linhas inseridas e excluídas em cada um desses commits. Desta forma, os dados de linhas adicionadas e retiradas foram coletados através da API do GitHub e agregados à base de dados, como resumido na Figura 5.8.

5.5.2. Veracidade dos Dados

O problema de veracidade dos dados se refere à avaliação e melhoria da precisão dos dados [Geerts et al., 2018]. Normalmente, a veracidade dos dados é comprometida pela presença de viés, anormalidades e ruído nos dados, que frequentemente estão presentes nos dados coletados na Web. Ao criar e manter bases de conhecimento, deve-se validar os dados e fornecer suas fontes a fim de garantir a exatidão e rastreabilidade das informações.

Como a veracidade é uma questão importante que afeta diretamente as informações e os conhecimentos extraídos dos dados, existem diferentes estratégias para melhorar ou garantir isso. Geerts et al. [2018] propõem o modelo DeFacto (*Deep Fact Validation*) que busca realizar a validação de fatos encontrando fontes confiáveis para os mesmos na Web. Para alcançar esse objetivo, o DeFacto fornece ao usuário trechos relevantes de páginas na Web, informações adicionais úteis e uma pontuação para a confiança da corretude do dado de entrada.

5.5.3. Remoção de Dados Duplicados

Dados duplicados aparecem em muitos contextos, especialmente ao coletar dados online. Por exemplo, nomes de autores duplicados ocorrem entre fontes de bibliotecas distintas ou até dentro da mesma fonte.

[17]GHTorrent: projeto que reúne um conjunto de dumps do GitHub (`http://ghtorrent.org/`)

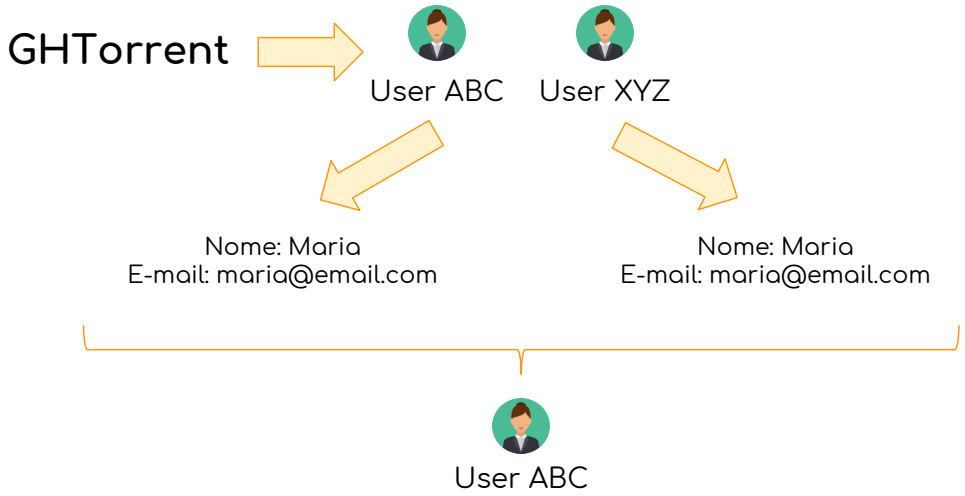


Figura 5.9: Deduplicação de nomes de usuários artificiais presentes no GitHub [Vasilescu et al., 2015]

A duplicidade de dados pode ocorrer de forma mais simples quando existem mais cópias dos mesmos dados em uma base. Esse é um caso mais fácil de ser resolvido, pois basta identificar valores idênticos e removê-los. Já o caso de registros que não são completamente idênticos é mais difícil, pois identificá-los requer entender se realmente aqueles registros dizem respeito à mesma informação. Por exemplo em nomes próprios com e sem abreviação ou omissão de algum dos sobrenomes: Mirella M. Moro, Mirella Moro e Mirella Moura Moro. Nesses casos, é possível usar recursos de contexto para identificar duplicatas, como nomes do usuário ou e-mail [Vasilescu et al., 2015] ou mesmo outras características importantes para o contexto da base [de Souza Silva et al., 2018].

Especificamente, o trabalho de Vasilescu et al. [2015] utiliza dados de desenvolvedores do GitHub coletados do GHTorrent. Há uma peculiaridade nesses dados em relação aos *usernames* dos usuários: quando por algum motivo específico o *username* não pode ser recuperado, o GHTorrent cria um usuário artificial com um *username* aleatório. Entretanto, usuários com *username* diferentes poderiam ser o mesmo usuário na realidade. Dessa forma, os autores realizaram uma deduplicação que impactou boa parte da base de dados utilizando informações como nome do usuário e e-mail de cadastro para identificar as duplicatas entre os usuários artificiais, como ilustra a Figura 5.9.

Enquanto isso, de Souza Silva et al. [2018] apresentam estratégias para melhorar o processo de remoção de duplicatas. A Figura 5.10 apresenta o processo para identificação de duplicadas em um banco de dados. Inicialmente, é feita a *indexação* dos registros a fim de realizar uma divisão dos mesmos em grupos iniciais. Em seguida, são realizadas *comparações* entre registros de mesmo grupo e, finalmente, é feita a *classificação* para identificar os dados em duplicidade. Os autores identificaram que na primeira etapa, de indexação, a escolha do atributo utilizado é de grande impacto para todo o processo. Então, propuseram uma melhor forma de escolher este atributo considerando aspectos como duplicidade, distinção, densidade e repetição.

Note que para ambos os exemplos apresentados, as estratégias de deduplicação

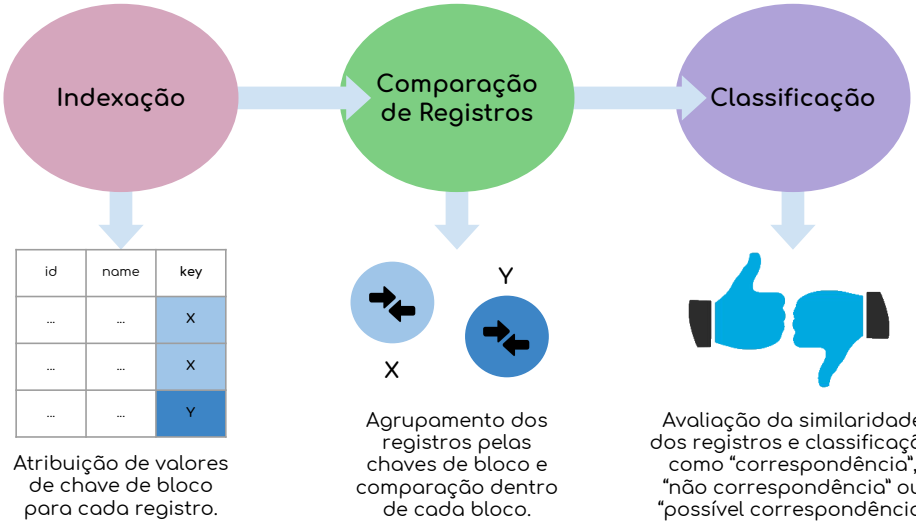


Figura 5.10: Processo de deduplicação de dados [de Souza Silva et al., 2018]

escolhidas estão diretamente relacionadas ao contexto em que os dados estão inseridos. Note que a identificação de duplicatas numa base de dados quase sempre depende de quais informações estão disponíveis para a correta classificação. Também é possível encontrar dados muito parecidos, mas que não são duplicatas, como por exemplo o nome "José P. Amaral", que pertence a pessoas distintas: "José Pedro Amaral"e "José Pereira Amaral". Por este motivo, a tarefa de deduplicação é delicada e requer um alto nível de precisão.

5.5.4. Redução de Dados

A redução de dados aborda o problema de minimizar a quantidade de dados que serão armazenados no conjunto de dados. Dependendo do volume de dados, armazená-los em um banco comum ou processá-los é um problema relacionado tanto a espaço quanto a tempo. As soluções geralmente utilizadas incluem: minimizar os dados ou armazená-los considerando outra forma de armazenamento [Liu and Ram, 2018], ou extrair amostras da base completa sem perder a generalidade das análises [Batista et al., 2017a].

Por exemplo, Batista et al. [2017a] utilizam ambas as estratégias visando a redução de dados armazenados. Especificamente, ao analisar toda a rede do GitHub, foram encontrados alguns desafios em relação ao volume de dados e algumas especificidades do contexto. Dessa forma, optou-se por dividir a rede de colaboração por linguagens de programação, selecionando as linguagens com maior quantidade de repositórios, conforme apresentado na Figura 5.11. Foram escolhidas as linguagens JavaScript, Java e Ruby. A partir das análises, os autores perceberam que os comportamentos dos desenvolvedores dentro da rede de cada linguagem é semelhante, o que permitiu que os resultados fossem obtidos dessa maneira. Ainda assim, o volume de dados para armazenamento era bastante expressivo. Dessa forma, foi escolhido armazenar os dados utilizando o MongoDB[18] ao invés do MySQL, utilizado anteriormente. A compressão de dados do MongoDB e a nova modelagem dos dados permitiram armazenar os mesmos dados com um espaço em disco menor e melhorar alguns filtros e consultas realizados na base.

[18]MongoDB: `https://www.mongodb.com/`

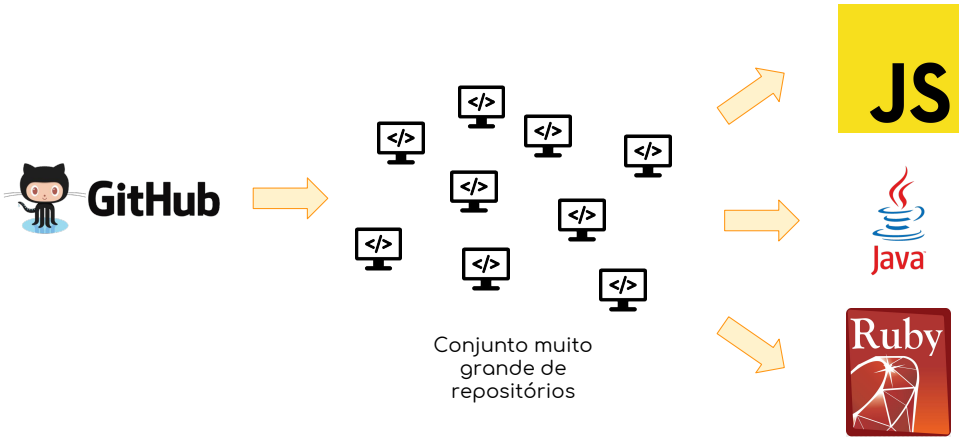


Figura 5.11: Escolha de repositórios por linguagem de programação

5.5.5. Ausência de Padronização

A padronização de dados é o processo de reestruturação de dados em um formato comum. Ter dados coletados de várias fontes da Web pode gerar um conjunto de dados que não é apenas heterogêneo, mas também em formato diferente. Então, iniciar a pesquisa real requer primeiro padronizar todos os dados.

Por mais que os dados coletados sejam suficientes para as análises que serão realizadas, se os mesmos estiverem desorganizados ou sem um padrão, eles dificultam a análise ao invés de auxiliar. Desta forma, é importante avaliar a base de dados no sentido de encontrar a forma que esses dados devem ser organizados para gerar os resultados desejados. Em alguns casos, propor uma nova modelagem para os dados coletados é uma forma de organizá-los formalmente e padronizá-los como um todo [Batista et al., 2017a].

5.6. Aplicações Reais

Existem diversos estudos que combinam fontes de dados para diferentes propósitos. A maioria deles depende da rica informação disponível apenas pela coleta de dados de fontes distintas. Esta seção cobre uma pequena fração de tais estudos em diferentes domínios.

5.6.1. Estimativa da Qualidade em Documentos Colaborativos

Dalip et al. [2017] propuseram uma abordagem para estimativa da qualidade de conteúdo colaborativo – como enciclopédias colaborativas e fórum de perguntas e respostas. Assim, foi necessário identificar quais dimensões de qualidade são importantes nesta tarefa, por exemplo, organização, legibilidade, importância e maturidade do texto. Baseado em trabalhos anteriores sobre qualidade de informação e nas orientações de publicação fornecidos por repositórios colaborativos (como StackOverflow[19] e Wikipedia[20]) foi adaptada a lista de dimensões de qualidade apresentada por Tejay et al. [2006], proposta originalmente no domínio de dados estruturados.

[19]StackOverflow help: `http://meta.stackoverflow.com/help/how-to-answer`
[20]Wikipedia Assessing Articles: `https://en.wikipedia.org/wiki/Wikipedia:Assessing_articles`

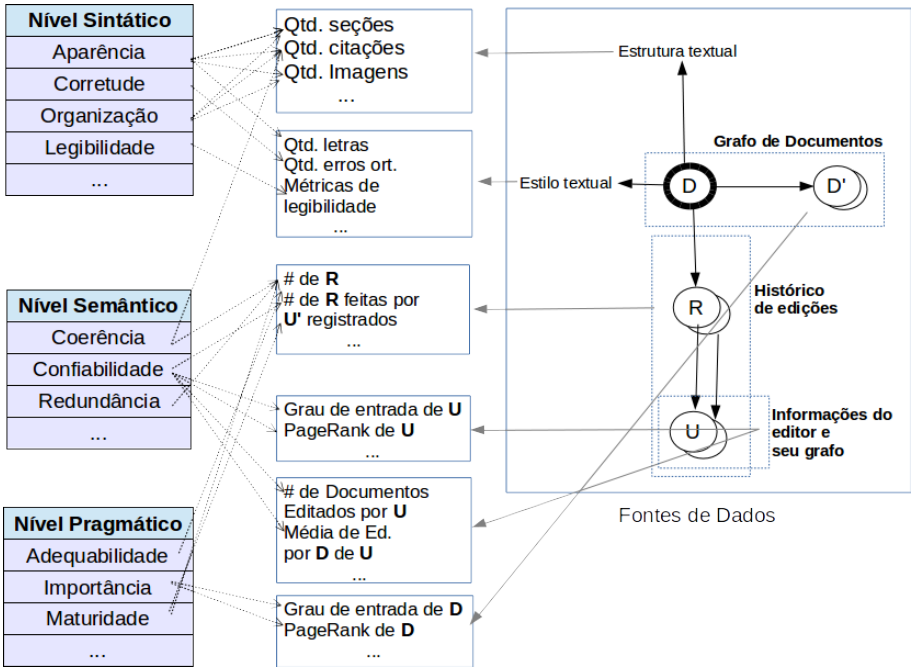


Figura 5.12: Uso de dados de múltiplas fontes para prever a qualidade de conteúdo em enciclopédias colaborativas (adaptado de Dalip et al. [2017])

Para isso, Tejay et al. [2006] propuseram um arcabouço conceitual visando agrupar as dimensões de qualidade em níveis semióticos: sintático, semântico e pragmático. De acordo com os autores, a semiótica pode ajudar a organizar as dimensões, pois ela estuda como um signo é criado, processado e usado. No presente contexto, signo é o próprio dado. Dimensões sintáticas são relacionadas em como o texto é apresentado. Dimensões semânticas relacionam o conteúdo do texto com o seu significado. Dimensões pragmáticas são relacionadas com a intenção do autor/leitor em um determinado contexto.

Posteriormente, foram definidos os indicadores de qualidade. Um indicador é um valor contendo uma medida estatística correlacionada com uma dimensão de qualidade. Por exemplo, o número de caracteres em um texto (indicador) pode ser correlacionado com a concisão do mesmo (dimensão de qualidade). A Figura 5.12 demonstra uma visão geral de como dimensões, indicadores e fontes estão relacionados no contexto de enciclopédias colaborativas. Neste domínio, Dalip et al. [2017] combinaram indicadores de qualidade do texto, do histórico de revisões e do grafo de links entre os artigos.

Para a Wikipédia, tais indicadores foram coletados de diversas fontes dessa enciclopédia colaborativa. Então, para cada fonte, precisou-se obter os dados de uma forma distinta. Inicialmente, extraiu-se a edição atual de todos os artigos da Wikipédia para a criação de uma amostra a partir deles. Para isso, foi feito o download do XML 'enwiki-20080101-pages-meta-current.xml.bz2' que estava disponível em `https://dumps.wikimedia.org/enwiki`[21]. A Wikipédia avalia manualmente seus artigos per meio

[21] A partir deste link `https://dumps.wikimedia.org/backup-index.html` pode-se obter os últimos *dumps* de diversas Wikis providas pela Wikimedia. Os dados extraídos deste estudo pode ser obtido em: `http://www.lbd.dcc.ufmg.br/lbd/collections/wiki-quality`

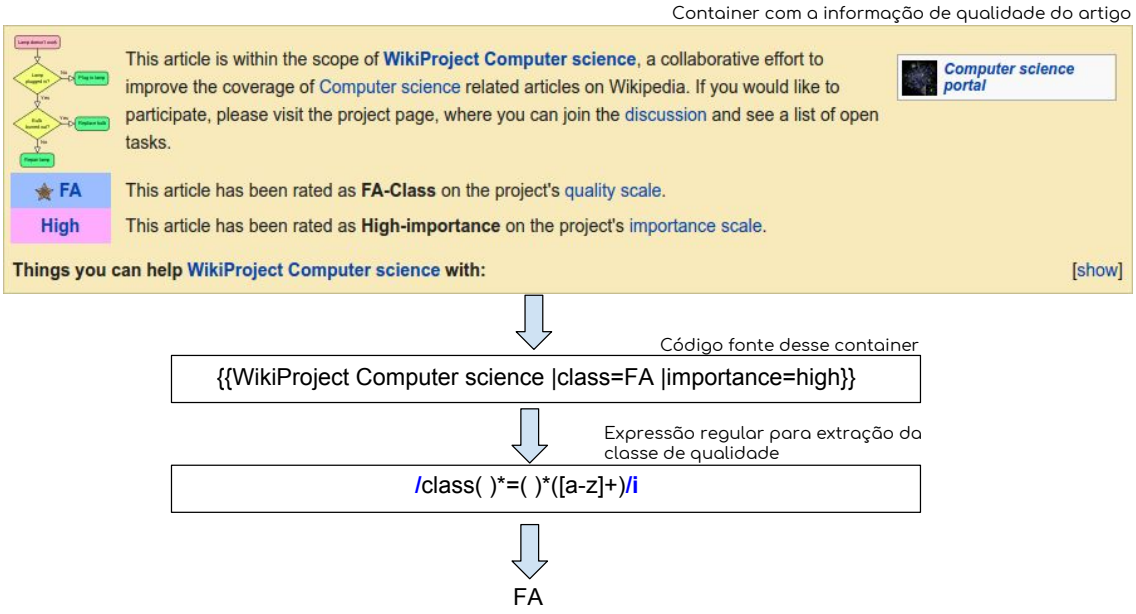


Figura 5.13: Exemplo de extração da classe de qualidade do artigo *Binary Search*.

de classes de qualidade. Assim, por meio deste link, foi coletado também a página de discussão de cada artigo com o objetivo de extrair a qualidade do mesmo. Um exemplo do funcionamento da extração dessa classe é ilustrado na Figura 5.13. A extração foi feita por meio de expressão regular para obter a sigla correspondente à classe de qualidade.

A partir desse mesmo link, extraiu-se também o grafo por meio do arquivo com sufixo 'pagelinks.sql.gz'. A partir desta página de *dumps*, também é possível extrair um XML contendo todas as edições de cada artigo da Wikipédia e, assim, pode-se extrair os indicadores de histórico de revisões (arquivo sufixo 'pages-meta-history.xml.bz2'). Porém, como o tamanho do XML é grande, optou-se por criar a amostra e usar a API da Wikipédia para coletar, para cada artigo dessa amostra, todas as suas edições[22].

Nessa API, é possível obter o XML completo de revisões por meio de requisições. Esse XML possui, para cada página, o texto de suas revisões e metadados (título, autor, data de revisão etc.). Como há um limite de quantidade de revisões por requisição, para cada página, deve-se extrair uma quantidade de revisões por vez. O Algoritmo 3 demonstra como foi feito a coleta por Dalip et al. [2017]. Nesse algoritmo, é necessário determinar a data l para que sejam coletadas as revisões até essa data e o conjunto de páginas P a serem coletadas. Assim, para cada página, são feitas requisições para extrair o conjunto de revisões (função *request_wiki*). Para cada revisão, é possível extrair seus dados para processamento dos indicadores (função $process\_features$) e obter seus metadados como, por exemplo, a data da revisão (função $get\_timestamp$)[23].

Após extrair todos os indicadores e a classe de qualidade correspondente de cada artigo, é possível utilizar a abordagem proposta por Dalip et al. [2017] para combinar tais fontes e, assim, estimar a qualidade de documentos colaborativos. A descrição completa

[22] `https://en.wikipedia.org/wiki/Special:Export`
[23] Exemplo da implementação: `https://github.com/lab-csx-ufmg/webmedia2018`

Algoritmo 3 Coleta de dados por meio da API da Wikipédia

Require: Conjunto P de páginas a serem coletadas (representadas pelo seu título)
Require: Data l representa a data limite de uma revisão a ser coletada. Ou seja, são coletadas todas as revisões de uma página até a data l.
Considere R o conjunto de revisões de uma página $p \in P$
Considere d_p a data da mais antiga revisão coletada para uma determinada pagina p
for all $p \in P$ **do**
    $has\_revision \leftarrow True$
    $d_p \leftarrow l$
    while $has\_revision = True$ **do**
        $R \leftarrow request\_wiki(p, d_p)$
        for all $r \in R$ **do**
            $process\_revision(r)$
            $d_p \leftarrow get\_timestamp(r)$
        if $|R| = 0$ **then**
            $has\_revision \leftarrow False$

da abordagem per se foge do escopo deste capítulo.

5.6.2. Estimativa da Força de Relacionamentos no GitHub

Social Coding é uma abordagem de desenvolvimento de software colaborativa para desenvolvedores, que incentiva a discussão e compartilhamento de ideias e conhecimento [Dabbish et al., 2012]. Essa metodologia tem alterado a forma de desenvolvimento de software, pois colaboradores geograficamente distantes podem acessar plataformas colaborativas remotamente. Alguns exemplos de sites que permitem o *Social Coding* são o Google Code[24] e o GitHub[25]. Os dados disponíveis nesses sites permitem definir redes sociais que conectam desenvolvedores a partir das suas atividades colaborativas em repositórios de software, formando uma grande rede implícita de codificação social.

Um tipo específico de aspecto social nessa rede é a força da interação entre desenvolvedores, ou a força do relacionamento no contexto de codificação social [Dabbish et al., 2012]. Nesse contexto, a força dos relacionamentos tem sido investigada no GitHub para analisar a produtividade de desenvolvedores em projetos [Casalnuovo et al, 2015], prever a colaboração entre desenvolvedores [Bartusiak et al., 2016] e investigar a aceitação de solicitações *pull requests* [Tsay et al., 2014].

Trabalhos que exploram extensivamente diferentes aspectos da força dos relacionamentos sociais entre desenvolvedores foram propostos em nosso grupo de pesquisa por Alves et al. [2016], Batista et al. [2017b] e Oliveira et al. [2018]. A Figura 5.14 apresenta uma visão esquemática do processo adotado nos trabalhos propostos, desde a fase inicial de coleta (GHTorrent e da API do GitHub), seguida da modelagem da rede e dos relacionamentos até, por fim, a fase das análises considerando propriedades semânticas e topológicas para analisar relacionamentos de codificação social.

Tais estudos foram realizados através da integração de dados coletados de base de

[24]Google Code: `code.google.com`
[25]GitHub: `github.com`

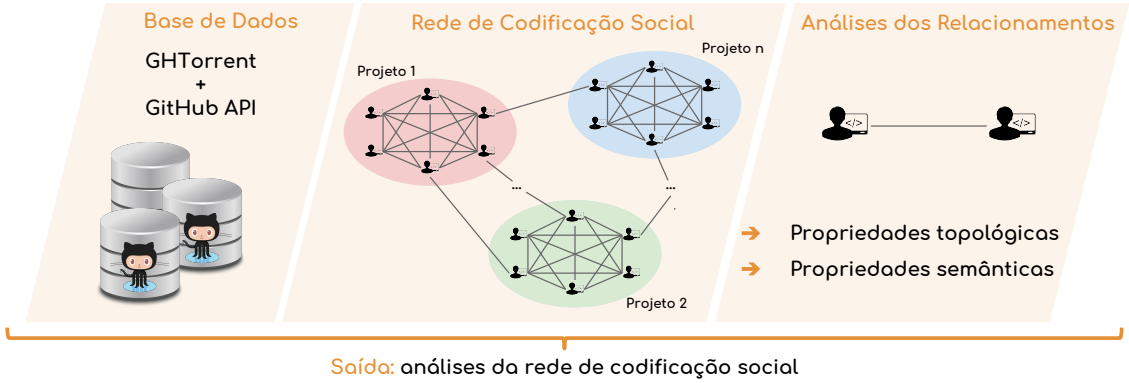


Figura 5.14: Visão geral do processo de coleta, integração e análise dos dados do GitHub

dados disponível na Web, chamada GHTorrent[26], e dados coletados por meio da API do GitHub. O GHTorrent fornece dados sobre repositórios, projetos, desenvolvedores, commits, pull requests, entre outros. Entretanto, Alves et al. [2016] e Batista et al. [2017b] precisam do número de linhas dos commits em uma das métricas para a força dos relacionamentos; e Oliveira et al. [2018] necessitam da posição de desenvolvedores em ranks gerados pelo GitHub. Tais dados foram obtidos por meio da API do GitHub ou por coleta diretamente ao Git Awards[27].

Para a coleta de informações adicionais, foram desenvolvidos coletores para recuperar dados de fontes diferentes, além dos dados do GHTorrent. A necessidade do número de linhas adicionadas e apagadas em cada commit teve objetivo de detalhar uma das métricas da rede que considera a quantidade de commits realizada por um par de desenvolvedores. A partir das análises, notou-se diferentes comportamentos entre desenvolvedores: alguns realizavam grandes commits em períodos maiores de tempo e outros realizavam vários commits durante o dia com pequenas alterações. Dessa forma, é importante saber o real impacto de cada commit ao repositório que ele se enquadra analisando o número de linhas de código que foram efetivamente inseridas ou excluídas pelo mesmo. Já as informações de rankings extraídas do Git Awards são importantes no sentido de utilizar um baseline para comparação da classificação de desenvolvedores na rede.

A base de dados, que está disponível online[28], foi capaz de reunir diversos dados sobre os repositórios, os usuários e, principalmente, as colaborações no GitHub considerando o tempo de contribuição entre usuários. A partir das coletas realizadas, foram propostas bases de dados com diferentes modelagens e versões [Batista et al., 2017b,a, Oliveira et al., 2018], sendo esta base atualizada incrementalmente.

A integração de dados foi um grande desafio no processo de construção da base. Por terem datas de referência diferentes, as bases possuíam uma série de divergências entre si. Projetos existentes no GHTorrent podiam não estar mais disponíveis para coleta dos números de linhas dos commits (os motivos principais são repositórios que foram excluídos ou transformados em repositórios privados, impossibilitando a coleta por

[26]GHTorrent: `http://ghtorrent.org/`
[27]Git Awards: `http://git-awards.com/`
[28]Dataset GitHub: `https://homepages.dcc.ufmg.br/~mirella/projs/apoena`

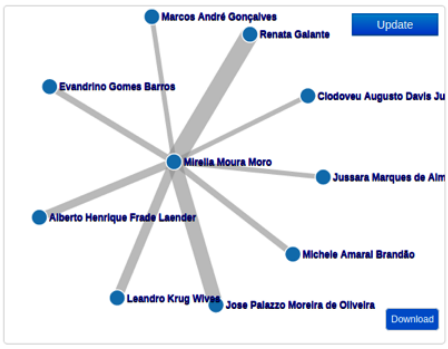


Figura 5.15: Rede centrada no pesquisador – adaptada de Brandão et al. [2018].

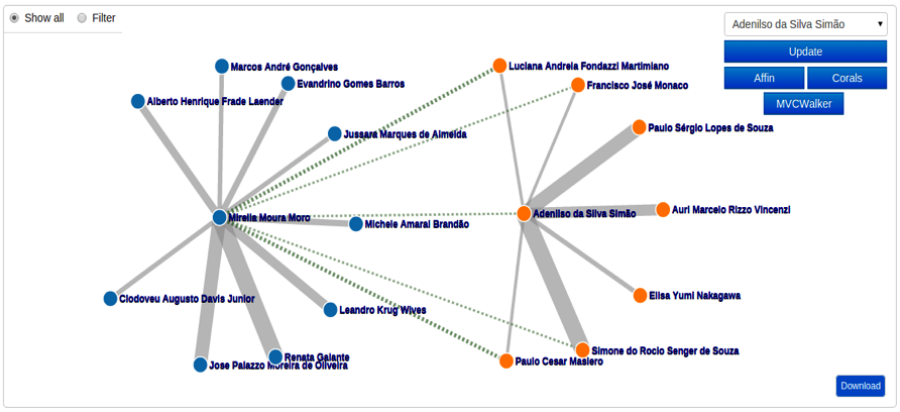


Figura 5.16: As linhas verdes representam colaborações recomendadas: quanto mais intenso, mais foi recomendado pelo algoritmo. As recomendações são geradas clicando em uma das opções com o nome dos algoritmos (figura extraída de Brandão et al. [2018]). A geração das recomendações e tal visualização só foi possível devido a integração de dados de fontes distintas.

meio de *crawlers*). No contexto dos rankings, alguns usuários que constavam no ranking, não necessariamente estavam também na base inicial, prejudicando a análise da sua classificação. Em sua maioria, os dados faltantes precisaram ser ignorados para que não impossibilitassem a extração de resultados e conhecimento a partir da base consolidada.

5.6.3. Caracterização e Visualização de Pesquisadores

No contexto de colaborações científicas, Brandão et al. [2018] combinam dados da DBLP[29] e dados coletados da biblioteca digital da ACM[30] para propor visualizações mais completas para pesquisadores, tais como rede centrada no pesquisador (Figura 5.15), recomendações de colaborações (Figura 5.16), rede de coautoria global e métricas de redes complexas. Dessa forma, foi proposta a ferramenta CNARe[31] que objetiva auxiliar pesquisadores a escolher colaboradores através de recomendações automáticas, visualizar recomendações, comparar os resultados de diferentes algoritmos de recomendação e ana-

[29]DBLP: `dblp.uni-trier.de`
[30]ACM digital library: `dl.acm.org`
[31]CNARe: `http://homepages.dcc.ufmg.br/~mirella/Tools/CNARe/`

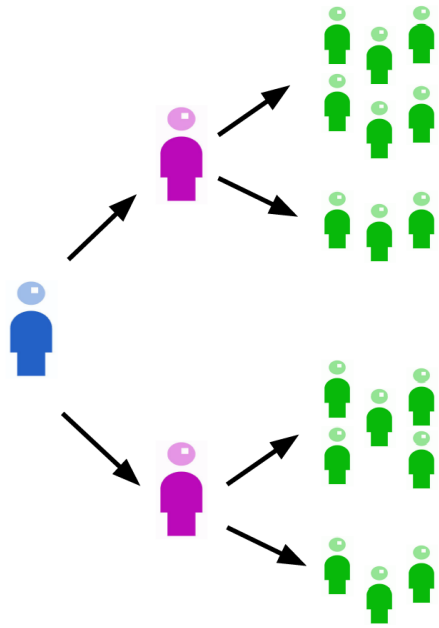


Figura 5.17: Exemplo de coleta utilizando a estratégia bola de neve.

lisar o impacto dos pesquisadores recomendados em sua rede atual.

O banco de dados inicial da ferramenta CNARe incluía apenas dados de publicações da área de Ciência da Computação, mas para montar as redes sociais e gerar recomendações é necessário dados sobre pesquisadores. Assim, informações disponíveis na página dos pesquisadores na biblioteca digital da ACM foram coletadas utilizando a estratégia de amostragem bola de neve (*snowball sampling*) [Goodman, 1961]. A Figura 5.17 mostra um exemplo de como novos dados de pesquisadores são coletados utilizando tal estratégia. A partir de um pesquisador semente conhecido (ou um conjunto de sementes), os coautores de tal pesquisador são coletados. Em seguida, os coautores desses coautores também são coletados até que a quantidade de dados disponível seja suficiente para análise. Em Brandão et al. [2018], o limite de coleta foi atingir os top 100 pesquisadores com maior quantidade de publicações, pois os algoritmos de recomendações implementados na CNARe não possuem bom desempenho para grandes volumes de dados.

A biblioteca da ACM foi escolhida por apresentar a área de cada publicação de acordo com o *ACM Classification System*. A página de cada pesquisador tem uma lista de publicações, na qual cada publicação tem o DOI (Digital Object Identifier System), a lista de coautores, a data e o local da publicação. A partir do DOI, a URL especificada é acessada para obter a área de pesquisa de cada publicação e informações sobre cada coautor (pesquisador): instituição, número total de publicações e nome do pesquisador (já que a lista de coautores fornece o nome no formato de citação). Após inserir os coautores em um banco de dados diferente do CNARe, uma nova consulta é executada para obter o coautor com o maior número de publicações cuja página ainda não foi visitada. Então, o processo de coleta começa novamente a partir da página deste pesquisador.

CNARe utiliza um banco de dados relacional, e o esquema do banco de dados

Tabela 5.2: Um exemplo de conjuntos de dados que representam os efeitos colaterais do Thyroxine, adaptado de Ma et al. [2017].

FAERS		**HealthBoards**	
ID Usuário	**Efeito colateral**	**ID Usuário**	**Efeito colateral**
110696642	Disfagia	2918	Enxaqueca
108294651	Disfagia	3171	Disfagia
108294651	Náusea	3171	Náusea
108294651	Mudança de humor	3171	Anemia
108325471	Disfagia	6871	Mudança de humor
108325471	Náusea	6871	Desidratação
108325471	Enxaqueca	27417	Desidratação

inicial foi alterado para receber os dados coletados da ACM. Antes de realizar a inserção dos dados no banco de dados do CNARe, foi necessário processá-los, que incluiu as atividades de verificar valores ausentes, remover dados duplicados e lidar com a ausência de padronização. A quantidade de valores ausentes era inferior a 10% dos dados coletados, o que permitiu que eles apenas fossem ignorados sem prejudicar o estudo. Além disso, alguns dados de publicações e autores foram coletados mais de uma vez, o que exigiu uma verificação antes de integrar os dados. Finalmente, em relação à falta de padronização, o principal problema foi com o nome dos autores. Por exemplo, o nome de um mesmo autor pode estar no formato Mirella M. Moro ou M. M. Moro. Também existem autores distintos com mesmo nome abreviado, por exemplo, C. J. Xia. Nesses casos, foi necessário verificar se dados como instituição e publicações correspondiam ao mesmo autor ou se eram diferentes para então fazer a integração.

5.6.4. Predição de Efeitos Colaterais de Medicamentos

Uma preocupação mundial na área da saúde são os efeitos que podem ser causados por medicamentos. Nesse contexto, Ma et al. [2017] propõem um modelo de grafos probabilísticos para prever efeitos colaterais de medicamentos. Para isso, foram consideradas três diferentes fontes de dados: (*i*) SIDER, base de dados contendo pares medicamento e efeitos colaterais; (*ii*) plataforma FAERS, que contém informação sobre cada efeito colateral enviado ao FDA (do inglês, *U.S. Food and Drug Administration*); e (*iii*) HealthBoards, uma plataforma que possui milhões de mensagens relacionadas a medicamentos e seus efeitos colaterais.

Especificamente, Ma et al. [2017] também mostram os benefícios de considerar múltiplas fontes de dados para obter os verdadeiros efeitos colaterais. A Tabela 5.2 exemplifica os efeitos colaterais da Thyroxine extraídos de dois conjuntos de dados distintos, FAERS e Healthboards. Cada linha na tabela representa um efeito colateral reportado por um usuário. É possível observar que disfagia, náusea e desidratação são efeitos colaterais verdadeiros, enquanto que os outros três são incorretos. Ao minerar os efeitos colaterais de apenas um conjunto de dados, por exemplo apenas do FAERS, dois efeitos colaterais podem ser obtidos: disfagia e náusea. Entretanto, ao utilizar FAERS e Healthboards, é

possível obter três efeitos colaterais corretos.

5.6.5. Predição de Informações de Usuários em Redes Sociais

Devido ao grande volume de dados disponíveis em redes sociais, diferentes estudos estão sendo realizados e abordam, por exemplo, marketing viral [Subramani and Rajagopalan, 2003], detecção de comunidades [Brandão and Moro, 2017, Kim and Hastak, 2018] e privacidade e segurança [Akcora et al., 2012, Yuan et al., 2010]. Dentre tais estudos, também há a predição de informações em redes sociais, que podem ser sobre usuários (nós em uma rede social que pode ser modelada como um grafo) e/ou seus relacionamentos (links ou arestas na rede social) [Brandão et al., 2013].

Nesse contexto, Farnadi et al. [2018] combinam múltiplas fontes de informações sociais usando redes neurais profundas com o objetivo de prever informações do perfil do usuário como idade, gênero e características da personalidade. Para isso, os autores usaram os dados do Facebook de 5.670 usuário e combinaram três diferentes fontes: (*i*) textual por meio das mensagens em seu *status*; (*ii*) visual pela foto do perfil; e (*iii*) dados relacionais através das páginas na quais o usuário indicou sua preferência. Por meio de experimentos, os autores demonstraram que combinando tais indicadores eles poderiam melhorar o desempenho para prever informações em seu perfil.

Em relação à predição de relacionamentos, o objetivo é inferir quais conexões são possíveis de inferir em um futuro próximo. Por exemplo, Lu et al. [2010] propõem um framework de aprendizagem supervisionada para predição de links que aprende de redes sociais dinâmicas (considera o aspecto temporal dos relacionamentos) na presença de redes auxiliares. Em outras palavras, os relacionamentos em uma rede A são preditos utilizando dados de redes auxiliares B, C e D. Para tal estudo, foram utilizadas redes sociais de coautoria construídas com as bases de dados do arXiv[32] com publicações de 1992 a 2003, CiteSeer[33] com publicações de 1995 a 2003, e SIAM (*Society of Industrial and Applied Mathematics*)[34] com publicações de 1999 a 2004.

Há também a predição de informações sobre os relacionamentos. Por exemplo, Wang et al. [2018a] propõem um framework para predizer o sinal do sentimento (positivo ou negativo) de relacionamentos entre usuários em uma rede heterogênea na ausência de dados sobre sentimentos. A realização desse estudo considerou dois conjuntos de dados reais: Weibo Tweets, uma das redes sociais online mais populares na China, e a base de conhecimentos Microsoft Satori.

5.7. Conclusões

Neste capítulo, abordamos três questões relacionadas à utilização de dados provenientes de diferentes fontes Web: coleta, integração e pré-processamento. A seguir, resumimos as principais metodologias para cada etapa, bem como discutimos potenciais variáveis que ficaram de fora desse estudo.

O capítulo iniciou descrevendo quatro principais fontes de dados Web: dados abertos, dados conectados, páginas Web e APIs. Considerando a primeira fase de pro-

[32]arXiv: `https://www.arxiv.org/`
[33]CiteSeer: `https://citeseerx.ist.psu.edu/`
[34]SIAM: `https://www.siam.org/`

cessamento de tais dados, foram abordados conceitos em relação ao processo de coleta de dados, discutindo principalmente os tipos de fontes de dados e as respectivas técnicas de coleta. Especificamente, apresenta-se um conjunto de classificações para os coletores, suas principais aplicações, os principais desafios e três técnicas simples de coleta baseados no caminhamento em grafo. Dentre essas técnicas de coleta estão *Breadth-First Search*, *Snowball Sample* e *Depth-First Search*.

Logo após, este capítulo discutiu como a integração de dados de múltiplas fontes é uma tarefa que permite a extração de informações de forma mais realista para diferentes finalidades. Sem tal integração, as chances de interpretar os dados de forma equivocada é muito maior, conforme mostrado na aplicação de efeitos colaterais de drogas. Por isso, conforme apresentado, diferentes estudos foram realizados e diferentes estratégias foram desenvolvidas para a integração de dados, por exemplo, sistema de mediação, processamento de linguagem natural e abordagem bayesiana. Também apresentou-se as diferenças entre ETL e integração de dados, bem como exemplos para duas estratégias de integração de dados para bancos de dados relacionais: a vertical e a horizontal.

Após a integração dos dados, diferentes tipos de problemas que podem ser identificados na base são tratados na etapa de pré-processamento. Nessa parte do capítulo, foram apresentados então alguns dos problemas mais comuns, incluindo: valores ausentes, veracidade dos dados, remoção de duplicatas, redução de dados e ausência de padronização. Há diversas formas de tratamento para cada um dos problemas, e as soluções ideais variam de acordo com o contexto e disponibilidade dos dados. Foram sumarizados exemplos de trabalhos que necessitaram de algum dos tipos de tratamento e a forma com que cada um lidou com o problema.

Para contextualizar as três etapas, também foram apresentadas diversas aplicações. Por exemplo, apresentou-se como foi feita a coleta de diversas fontes na Wikipédia por meio de dumps e de sua API. Demonstrou-se também que o uso diferentes fontes foi útil para melhorar a predição de efeitos colaterais de medicamentos e apresentar visualizações mais completas de pesquisadores. Além disso, foi demonstrado como identificar informações sociais (idade, gênero e características da personalidade) de perfis de usuários no Facebook e prever relações de amizade.

O tratamento dos dados até a geração dos resultados e análises dos mesmos são processos fundamentais para diversos tipos de pesquisa. A geração correta de conhecimento a partir de dados depende do processo completo de coleta, integração e pré-processamento a fim de compor informações robustas e mais próximas da realidade do assunto estudado. Em tal contexto, diversos dos trabalhos conduzidos pelo nosso grupo de pesquisa estão disponíveis na página do Projeto Apoena[35], com as bases de dados modeladas e tratadas também disponíveis. Os laboratórios que colaboram com essas pesquisas são o Lab CS+X[36] na UFMG e o Piim-Lab[37] no CEFET-MG.

Finalmente, é importante notar que este capítulo possui limitações que são intrínsecas a qualquer trabalho deste tipo. Especificamente, não foram discutidos aspectos de

[35]Projeto Apoena: `http://bit.ly/proj-apoena`
[36]Lab CSX: `http://www.labcsx.dcc.ufmg.br`
[37]Piim-Lab: `http://piim-lab.decom.cefetmg.br`

armazenamento dos dados ou tipos de sistemas de gerência para tais dados. Porém, é necessário clarificar que são muitas variáveis envolvidas em tais aspectos e que devem ser avaliadas caso a caso. Também poderíamos explorar melhor outros problemas inerentes aos grandes volumes de dados que trafegam na Web diariamente. Igualmente importante seria o processamento de strings, imagens e vídeos. Em resumo, pesquisadores e desenvolvedores que necessitam utilizar dados provenientes da Web têm outros desafios que não são abordados aqui e que precisam ser considerados para bom planejamento e execução de seus projetos.

Agradecimentos. As pesquisas que resultaram na escrita deste capítulo foram financiadas por CAPES, CNPq e FAPEMIG.

Referências

C. G. Akcora, B. Carminati, and E. Ferrari. Privacy in Social Networks: How Risky is Your Social Graph? In *IEEE International Conference on Data Engineering (ICDE)*, pages 9–19, Washington, DC, USA, 2012. doi: 10.1109/ICDE.2012.99.

G. B. Alves, M. A. Brandão, D. M. Santana, A. P. C. da Silva, and M. M. Moro. The Strength of Social Coding Collaboration on GitHub. In *Simpósio Brasileiro de Banco de Dados (SBBD)*, pages 247–252, Salvador, Brasil, 2016.

C. V. Araujo, R. M. Neto, F. G. Nakamura, and E. F. Nakamura. Predicting Music Success Based on Users' Comments on Online Social Networks. In *Proceedings of the 23rd Brazillian Symposium on Multimedia and the Web*, pages 149–156, Gramado, RS, Brazil, 2017. doi: 10.1145/3126858.3126885.

O. Azeroual, G. Saake, and E. Schallehn. Analyzing data quality issues in research information systems via data profiling. *International Journal of Information Management*, 41:50–56, 2018. doi: 10.1016/j.ijinfomgt.2018.02.007.

R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval: The Concepts and Technology Behind Search*. Addison-Wesley Publishing Company, USA, 2nd edition, 2011. ISBN 9780321416919.

S. K. Bansal. Towards a semantic extract-transform-load (etl) framework for big data integration. In *Proceedings of IEEE International Congress on Big Data (BigData Congress)*, pages 522–529, Anchorage, AK, USA, 2014.

R. Bartusiak, T. Kajdanowicz, A. Wierzbicki, L. Bukowski, O. Jarczyk, and K. Pawlak. Cooperation prediction in github developers network with restricted boltzmann machine. In *Asian Conference on Intelligent Information and Database Systems (ACIIDS)*, pages 96–107, Vietnam, 2016. doi: 10.1007/978-3-662-49390-8_9.

N. A. Batista, G. B. Alves, A. L. Gonzaga, and M. A. Brandão. GitSED: Um Conjunto de Dados com Informações Sociais Baseado no GitHub. In *Dataset Showcase Workshop, Simpósio Brasileiro de Banco de Dados (SBBD)*, pages 224–233, Salvador, Brazil, 2017a.

N. A. Batista, M. A. Brandão, G. B. Alves, A. P. C. da Silva, and M. M. Moro. Collaboration strength metrics and analyses on GitHub. In *Proceedings of the International Conference on Web Intelligence*, pages 170–178, Leipzig, Germany, 2017b. doi: 10.1145/3106426.3106480.

M. R. Berthold, C. Borgelt, F. Höppner, and F. Klawonn. *Guide to intelligent data analysis: how to intelligently make sense of real data*. Springer Science & Business Media, 2010. doi: 10.1007/978-1-84882-260-3.

M. Bouzeghoub, B. F. Lóscio, Z. Kedad, and A. Soukane. Heterogeneous data source integration and evolution. In *Proceedings of International Conference on Database and Expert Systems Applications (DEXA)*, pages 751–757, Aix-en-Provence, France, 2002. doi: 10.1007/3-540-46146-9_74.

M. A. Brandão and M. M. Moro. Social professional networks. *Computer Communications*, 100(C):20–31, 2017. doi: 10.1016/j.comcom.2016.12.011.

M. A. Brandão, M. M. Moro, G. R. Lopes, and J. P. M. de Oliveira. Using link semantics to recommend collaborations in academic social networks. In *International Conference on World Wide Web (WWW), Companion Volume*, pages 833–840, Rio de Janeiro, Brazil, 2013. doi: 10.1145/2487788.2488058.

M. A. Brandão, M. A. Diniz, G. A. de Sousa, and M. M. Moro. Visualizing co-authorship social networks and collaboration recommendations with cnare. In N. Meghanathan, editor, *Graph Theoretic Approaches for Analyzing Large-Scale Social Networks*, pages 173–188. IGI Global, 2018. doi: 10.4018/978-1-5225-2814-2.ch011.

C. Casalnuovo et al. Developer onboarding in github: The role of prior social links and language experience. In *Proceedings of Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Sympo. Foundations of Software Engineering*, pages 817–828, Bergamo, Italy, 2015. doi: 10.1145/2786805.2786854.

L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in GitHub: transparency and collaboration in an open software repository. In *ACM Conference on Computer Supported Cooperative Work*, pages 1277–1286, Seattle, USA, 2012. doi: 10.1145/2145204.2145396.

D. H. Dalip, M. A. Gonçalves, M. Cristo, and P. Calado. A general multiview framework for assessing the quality of collaboratively created content on web 2.0. *Journal of the Association for Information Science and Technology*, 68(2):286–308, 2017. doi: 10.1002/asi.23650.

L. de Souza Silva, F. Murai, A. P. C. da Silva, and M. M. Moro. Automatic identification of best attributes for indexing in data deduplication. In *Proceedings of the 12th Alberto Mendelzon International Workshop on Foundations of Data Management*, Cali, Colombia, 2018.

A. Doan, P. Konda, A. Ardalan, J. R. Ballard, S. Das, Y. Govind, H. Li, P. Martinkus, S. Mudgal, E. Paulson, et al. Toward a system building agenda for data integration (and data science). *IEEE Data Eng. Bull.*, 41(2):35–46, 2018.

G. Farnadi, J. Tang, M. De Cock, and M.-F. Moens. User profiling through deep multimodal fusion. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 171–179, 2018. doi: 10.1145/3159652.3159691.

R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

P. A. Flach. Multi-relational data mining: a perspective. In *Portuguese Conference on Artificial Intelligence*, pages 3–4. Springer, 2001. doi: 10.1007/3-540-45329-6_2.

R. Freitas, C. Rocha, O. Braga, G. Lopes, O. Monteiro, and M. Oliveira. Using Linked Data in the Data Integration for Maternal and Infant Death Risk of the SUS in the GISSA Project. In *Proceedings of the 23rd Brazillian Symposium on Multimedia and the Web*, pages 193–196, Gramado, RS, Brazil, 2017. doi: 10.1145/3126858.3131606.

F. Geerts, P. Missier, and N. Paton. Editorial: Special issue on improving the veracity and value of big data. *J. Data and Information Quality*, 9(3):13:1–13:2, 2018. doi: 10.1145/3174791.

B. Golshan, A. Halevy, G. Mihaila, and W.-C. Tan. Data integration: After the teenage years. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 101–106, Chicago, Illinois, USA, 2017. doi: 10.1145/3034786.3056124.

L. A. Goodman. Snowball sampling. *The annals of mathematical statistics*, pages 148–170, 1961.

J. Kim and M. Hastak. Social network analysis: Characteristics of online social networks after a disaster. *International Journal of Information Management*, 38(1):86–96, 2018. doi: 10.1016/j.ijinfomgt.2017.08.003.

A. H. F. Laender, M. M. Moro, C. Nascimento, and P. Martins. An x-ray on web-available XML schemas. *SIGMOD Record*, 38(1):37–42, 2009. doi: 10.1145/1558334.1558338.

J. Liu and S. Ram. Using big data and network analysis to understand wikipedia article quality. *Data & Knowledge Engineering*, 115:80–93, 2018. doi: 10.1016/j.datak.2018.02.004.

Z. Lu, B. Savas, W. Tang, and I. S. Dhillon. Supervised link prediction using multiple sources. In *Proceedings of IEEE 10th International Conference on Data Mining (ICDM)*, pages 923–928, 2010. doi: 10.1109/ICDM.2010.112.

F. Ma, C. Meng, H. Xiao, Q. Li, J. Gao, L. Su, and A. Zhang. Unsupervised discovery of drug side-effects from heterogeneous data sources. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 967–976, 2017. doi: 10.1145/3097983.3098129.

L. F. M. P. Maia and J. Oliveira. Investigation of research impacts on the zika virus: An approach focusing on social network analysis and altmetrics. In *Proceedings of the 23rd Brazillian Symposium on Multimedia and the Web*, pages 413–416, Gramado, RS, Brazil, 2017. doi: 10.1145/3126858.3131593.

M. M. Moro, V. Braganholo, C. F. Dorneles, D. Duarte, R. de Matos Galante, and R. dos Santos Mello. XML: some papers in a haystack. *SIGMOD Record*, 38(2):29–34, 2009. doi: 10.1145/1815918.1815924.

G. P. Oliveira, N. A. Batista, M. A. Brandão, and M. M. Moro. Tie strength in github heterogeneous networks. In *Brazilian Symposium on Multimedia and the Web (WebMedia)*, 2018.

L. Sikos. *Mastering structured data on the Semantic Web: From HTML5 microdata to linked open data*. Apress, 2015.

M. R. Subramani and B. Rajagopalan. Knowledge-sharing and influence in online social networks via viral marketing. *Communications of the ACM*, 46(12):300–307, 2003. doi: 10.1145/953460.953514.

G. Tejay, G. Dhillon, and A. G. Chin. Data Quality Dimensions for Information Systems Security: A Theoretical Exposition. In *Security Management, Integrity, and Internal Control in Information Systems*, pages 21–39. Springer, 2006.

J. Tsay, L. Dabbish, and J. Herbsleb. Influence of social and technical factors for evaluating contribution in github. In *Procs. of the 36th International Conference on Software Engineering*, pages 356–366, Hyderabad, India, 2014. doi: 10.1145/2568225.2568315.

N. K. Tyagi, A. Solanki, and S. Tyagi. An algorithmic approach to data preprocessing in web usage mining. *International Journal of Information Technology and Knowledge Management*, 2(2):279–283, 2010.

B. Vasilescu, A. Serebrenik, and V. Filkov. A data set for social diversity studies of github teams. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, pages 514–517, 2015. doi: 10.1109/MSR.2015.77.

E. F. Veiga, M. F. Arruda, J. A. B. Neto, and R. d. F. Bulcão Neto. An ontology-based representation service of context information for the internet of things. In *Proceedings of the 23rd Brazillian Symposium on Multimedia and the Web*, pages 301–308, Gramado, RS, Brazil, 2017. doi: 10.1145/3126858.3126894.

H. Wang, F. Zhang, M. Hou, X. Xie, M. Guo, and Q. Liu. Shine: Signed heterogeneous information network embedding for sentiment link prediction. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 592–600, Marina Del Rey, USA, 2018a. doi: 10.1145/3159652.3159666.

L. Wang, R. Pan, X. Wang, W. Fan, and J. Xuan. A bayesian reliability evaluation method with different types of data from multiple sources. *Reliability Engineering & System Safety*, 167:128–135, 2017. doi: 10.1016/j.ress.2017.05.039.

R. Wang, W. Ji, M. Liu, X. Wang, J. Weng, S. Deng, S. Gao, and C.-a. Yuan. Review on mining data from multiple data sources. *Pattern Recognition Letters*, 2018b. doi: 10.1016/j.patrec.2018.01.013.

M. Yuan, L. Chen, and P. S. Yu. Personalized Privacy Protection in Social Networks. In *Proceedings of Very Large Data Base Endowment*, pages 141–150, 2010. doi: 10.14778/1921071.1921080.

B. Zhao, B. I. P. Rubinstein, J. Gemmell, and J. Han. A bayesian approach to discovering truth from conflicting sources for data integration. *Proc. VLDB Endow.*, 5(6):550–561, 2012. ISSN 2150-8097. doi: 10.14778/2168651.2168656.

Biografia Resumida dos Autores

Natércia A. Batista. Natércia A. Batista. É aluna de mestrado em Ciência da Computação na Universidade Federal de Minas Gerais, Bacharel em Sistemas de Informação pela Universidade Federal de Minas Gerais (2017), e técnica em Informática Industrial pelo Centro Federal de Educação Tecnológica de Minas Gerais (2011). Atualmente trabalha no Laboratório de Computação Interdisciplinar CS+X e seus principais interesses são nas áreas de análise e gerenciamento de dados, e análise de redes sociais.

Michele A. Brandão É professora do Instituto Federal de Minas Gerais. É Doutora e Mestre em Ciência da Computação pela UFMG, Bacharel em Ciência da Computação pela Universidade Estadual de Santa Cruz (UESC, Bahia). Foi professora substituta na UFMG e PUC/Minas (Pontifícia Universidade Católica de Minas Gerais) e bolsista de Pós-Doutorado Júnior (PDJ-CNPq) no Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais (UFMG). Seus principais interesses de pesquisa estão nas áreas de mineração de dados, análise e gerenciamento de dados, sistemas de recomendação, predição de links e redes sociais. Seu projeto de pesquisa atual visa aplicar seu conhecimento para auxiliar no avanço da forense digital.

Michele B. Pinheiro Possui bacharelado (2013) e mestrado (2016) em Ciência da Computação pela Universidade Federal de Minas Gerais. Realizou pesquisas envolvendo *crowdsourcing*/*crowdsensing* ativo no contexto de dados geográficos, como contribuição voluntária geográfica. Atualmente trabalha em projetos interdisciplinares que envolvem o grupo de pesquisa CS+X (Departamento de Ciência da Computação - UFMG) e o grupo Indisciplinar (Escola de Arquitetura - UFMG), sobre a coleta de dados realizada por cidadãos e políticas públicas junto ao INCT de Tecnopolíticas (Indisciplinar).

Daniel H. Dalip É professor do Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG). É Doutor (UFMG/2015), Mestre (UFMG/2009) e Bacharel (Uni-BH/2006) em Ciência da Computação. Realiza pesquisas nas áreas de banco de dados e recuperação de informação. Tem experiência de docência nas disciplinas de Programação Web, Algoritmos, Recuperação de Informação, Pesquisa Operacional. Já lecionou na PUC-MG e Uni-BH. Durante seu mestrado e o doutorado, desenvolveu pesquisas sobre o uso de aprendizagem de máquina para avaliar automaticamente a qualidade em documentos colaborativos na Web.

Mirella M. Moro É professora associada do Departamento de Ciência da Computação (DCC) da Universidade Federal de Minas Gerais (UFMG). Possui doutorado em Ciência da Computação pela *University of California in Riverside* (2007), e graduação e mestrado em Ciência da Computação pela Universidade Federal do Rio Grande do Sul (UFRGS). Após o seu doutoramento, foi bolsista CNPq PDJ (PosDoc Junior) no Instituto de Informática da UFRGS. Foi membro do *Education Council* da ACM (*Association for Computing Machinery*), Diretora de Educação da SBC (Sociedade Brasileira de Computação, 2009-2015), editora-chefe da revista eletrônica SBC Horizontes (2008-2012), editora associada do JIDM (Journal of Information and Data Management, 2010-2012) e coordenadora da Comissão Especial de Bancos de Dados (CE-BD) da SBC (2015). Seus interesses de pesquisa estão na área de Banco de Dados, incluindo tópicos como processamento de consultas, redes sociais, recomendação, bibliometria e NoSQL.

Capítulo 6

Do device à cloud com a Plataforma SOFT-IoT: sua infraestrutura IoT em poucas horas

Leandro Andrade[1], Cleber Lira[3,1], Brenno Mello[1], Andressa Andrade[1], Antonio Coutinho[2,1], Fabíola Greve[1], Cássio Prazeres[1]

[1]Universidade Federal da Bahia. Departamento de Ciência da Computação (UFBA/DCC)

[2]Universidade Estadual de Feira de Santana. Departamento de Tecnologia (UEFS/DTEC)

[3] Instituto Federal de Educação, Ciência e Tecnologia da Bahia. (IFBA)

(leandrojsa, brenno.mello, fabiola, prazeres)@ufba.br,

dsandrade@dcc.ufba.br, cleberlira@ifba.edu.br, acoutinho@uefs.br

Abstract

The Internet of Things (IoT) is playing a great role in the technology scenario due to its high potential and impact on different society segments. Currently, several IoT initiatives have simultaneously developed new architectures, platforms and applications which has resulted in the creation of several parallel IoT ecosystems. This diversity makes access to IoT devices, and their integration in potential applications, difficult. In this chapter, we present the SOFT-IoT platform, which introduces the Fog of Things (FoT) concept in order to exploit the processing, storage and network capacity of local resources, allowing for the integration of different devices in a seamless IoT architecture.

Resumo

A Internet das Coisas (Internet of Things - IoT) vem desempenhando um importante papel no cenário tecnológico devido ao seu alto potencial e impacto em diferentes segmentos da sociedade. Recentemente, diferentes iniciativas em IoT desenvolveram simultaneamente novas arquiteturas, plataformas e aplicações, o que resultou na criação de diversos ecossistemas IoT paralelos. Essa diversidade dificulta o acesso a dispositivos IoT e sua integração em potenciais aplicações. Neste capítulo, apresentamos a plataforma SOFT-IoT, que introduz o conceito de Névoa das Coisas (Fog of Things - FoT) visando explorar o processamento, o armazenamento e a capacidade de rede dos recursos locais, permitindo a integração de diferentes dispositivos em uma arquitetura IoT estruturada.

6.1. Introdução

A comunidade acadêmica e a indústria têm feito significantes progressos em Internet das Coisas (*Internet of Things* – IoT) e áreas correlatas, em diferentes direções, incluindo o desenvolvimento de novas arquiteturas, plataformas e aplicações. Entretanto, esses progressos têm resultado na criação de diversos ecossistemas IoT paralelos, dificultando a criação de um ecossistema global, que permitiria habilitar o acesso a um vasto número de dispositivos existentes (e futuros), aumentando, dessa forma, a quantidade e o escopo de potenciais aplicações IoT [Sundmaeker et al. 2010]. Assim, o IERC (*European Research Cluster on the Internet of Things*) publicou um relatório contendo um conjunto de desafios de pesquisa, melhores práticas, recomendações e próximos passos na direção de prover interoperabilidade na Internet das Coisas [Serrano et al. 2015a]. Nesse relatório, os autores apontam que interoperabilidade e a entrega de serviços e de dados são importantes desafios que devem ser tratados por novas soluções para a Internet das Coisas.

Por um lado, interoperabilidade em IoT deve garantir meios de habilitar anotação semântica, identificação, registro e descoberta de dados provenientes dos dispositivos na IoT [Serrano et al. 2015b]. Por outro lado, serviços IoT devem garantir a entrega e utilização desses dados a usuários, aplicações ou mesmo outros serviços [Bassi et al. 2013]. Diversas soluções baseadas em Computação em Nuvem (*Cloud Computing*) têm sido propostas pela indústria e na academia com foco em entrega de serviços e interoperabilidade. Entretanto, soluções baseadas em nuvem apresentam algumas limitações [Abdelshkour 2015]: conectividade com a Internet é um pré-requisito essencial; dados e aplicações são processados na nuvem, tarefas que consomem muito tempo em se tratando de grandes volumes de dados; muito uso da largura de banda, pois é preciso enviar dados coletados de dispositivos IoT por segundos para a nuvem; tempo de resposta alto e problemas de escalabilidade, dado a dependência de servidores localizados em locais remotos.

Um novo conceito de infraestrutura para dados e serviços foi proposto inicialmente pela CISCO [Bonomi et al. 2012][Bonomi et al. 2014]: Computação em Névoa (*Fog Computing*, *Fog Networking* ou *Fogging*). Na IoT, a Computação em Névoa visa tirar parte da complexidade da nuvem e trazer para perto dos dispositivos, aplicações e/ou usuários, funcionando como uma espécie de "nuvem local privada" (névoa). Portanto, a Computação em Névoa propõe que [Abdelshkour 2015]: a conectividade ininterrupta com a Internet não seja essencial; o processamento de dados e a execução de aplicações também possam ocorrer nos limites da rede local; o uso da largura de banda seja otimizado; a melhora do tempo de resposta e da escalabilidade aconteça através do emprego de "pequenos servidores" locais próximos aos dispositivos e/ou usuários; e outros.

A Computação em Nuvem e a Computação em Névoa não são mutuamente exclusivas. De fato, em IoT, elas são complementares em diversos aspectos como, por exemplo: dispositivos (atuadores e sensores) continuam funcionando na névoa mesmo sem conectividade e quando a conectividade for possível, dados são enviados para a nuvem; antes de serem enviados à nuvem os dados são processados em servidores locais na névoa; análises de dados podem ser realizados na névoa e visualizados na nuvem ou realizados na nuvem e executados (atuadores) na névoa; usuários/aplicações locais podem acessar dispositivos e dados diretamente via névoa e usuários/aplicações remotos via nuvem, provendo, dessa forma, melhor qualidade de experiência (*Quality of Experience* - QoE); dentre outros.

Nesse contexto, este capítulo apresenta a plataforma SOFT-IoT (*Self-Organizing Fog of Things for the Internet of Things*), que provê soluções relacionadas à interoperabilidade e entrega de serviços na Internet das Coisas, desde a borda da rede até a nuvem. Para prover essas soluções, a plataforma SOFT-IoT apresenta o paradigma da "Névoa de Coisas" (*Fog of Things* - FoT) auto-organizável para entrega de serviços e produção de dados interoperáveis na borda da rede.

6.2. Plataforma SOFT-IoT

Com o objetivo de aproveitar as vantagens que a Computação em Névoa pode propiciar para IoT, em [Prazeres and Serrano 2016] é apresentado um novo paradigma chamado de *Fog of Things* (FoT). Dessa forma, como na Computação em Névoa, o FoT permite que parte da capacidade de processamento de dados e operações de entrega de serviços sejam processados em *gateways* (pequenos servidores) e/ou em servidores, ambos na borda da rede. O paradigma FoT vai além da Computação em Névoa nos seguintes aspectos:

- Explora a capacidade de processamento da borda da rede por meio do processamento de dados e da entrega de serviços em dispositivos, *gateways* IoT e/ou servidores locais;
- Define serviços IoT na borda da rede;
- Distribui esses serviços na borda da rede através de um *middleware* orientado para mensagens e serviços.

A Figura 6.1 ilustra uma plataforma baseada no paradigma *Fog of Things*, que é composta por uma combinação ou pela totalidade dos seguintes componentes: aplicações; dispositivos; *gateways*; servidores; *middleware* orientado a serviços de mensagens; e provedores de segurança. Esses componentes formam o núcleo de organização de uma FoT e são descritos em detalhes a seguir:

- *FoT-Device* - reusa e estende o conceito proposto por [Bassi et al. 2013], em que os dispositivos IoT são artefatos físicos que realizam a integração do mundo real com o mundo digital da Internet. Diante disso, no trabalho de [Bassi et al. 2013] são definidos três tipos de dispositivos IoT: sensor (por exemplo, um sensor de temperatura), atuador (por exemplo, um interruptor liga/desliga e etiquetas (por exemplo, etiquetas RFID). Os *FoT-Devices* têm algumas funcionalidades além das propostas por [Bassi et al. 2013], pois, podem: realizar transformação de dados brutos em conteúdo estruturado seguindo as diretrizes do *Linked Data*; agrupar/agregar dados antes de serem enviados ao gateway; formatar a mensagem a ser enviada ao gateway; dentre outras coisas, a depender das suas capacidades de processamento e memória. Além disso, os dispositivos FoT não requerem conectividade com a Internet porque eles realizam comunicação diretamente com os *gateways* que os gerenciam. Desse modo, os dispositivos podem continuar monitorando (sensores) e/ou atuando (atuadores) no ambiente, mesmo sem uma conexão com a Internet ativa, e todos os dados são enviados para serem processados nos *gateways*;

- *FoT-Gateway* - é um nó da rede FoT que tem como objetivo principal mapear os protocolos da camada de comunicação (Ethernet, Wi-Fi, ZigBee, Bluetooth e outros) para HTTP (camada de aplicativo da Web). Sendo assim, um *FoT-Gateway* oferece serviços Web RESTful para acesso as funcionalidades dos *FoT-Devices* (*Thing as a Service paradigm* – TaaS) e a outros serviços da IoT. Assim, as aplicações podem abstrair o protocolo de comunicação com os dispositivos e acessá-los através de uma API REST padronizada para cada tipo de dispositivo, como é o funcionamento atual da maioria dos aplicativos da Web;

- *FoT-Server* - servidores mais robustos que o *FoT-Gateway* e que podem ser de dois tipos: um tipo especial de *gateway* aprimorado com recursos de gerenciamento; um tipo especial de recurso fornecendo informações específicas que não são suportadas por dispositivos e *gateways*, como, por exemplo, dados históricos de longo prazo de dispositivos (servidor de dados) ou credenciais de identificação e autorização (servidor provedor de segurança);

- *Application* - são aplicações que utilizam a API REST baseada em HTTP (RESTful Web Services) para acessar os serviços IoT fornecidos pela FoT, via os *FoT-Gateways*, e que oferece a interação com serviços para os usuários. Os aplicativos podem ser Web, dispositivos móveis (Android, iPhone ou Windows Phone) ou mesmo aplicativos *desktop* tradicionais;

- *FoT-Profile* - define com quais tipos de serviços que os nós da FoT (FoT-Gateway e FoT-Servers) podem ser configurados. Como exemplo, alguns dos perfis que os nós da FoT podem assumir são: composição; descoberta; segurança; armazenamento; dentre outros.

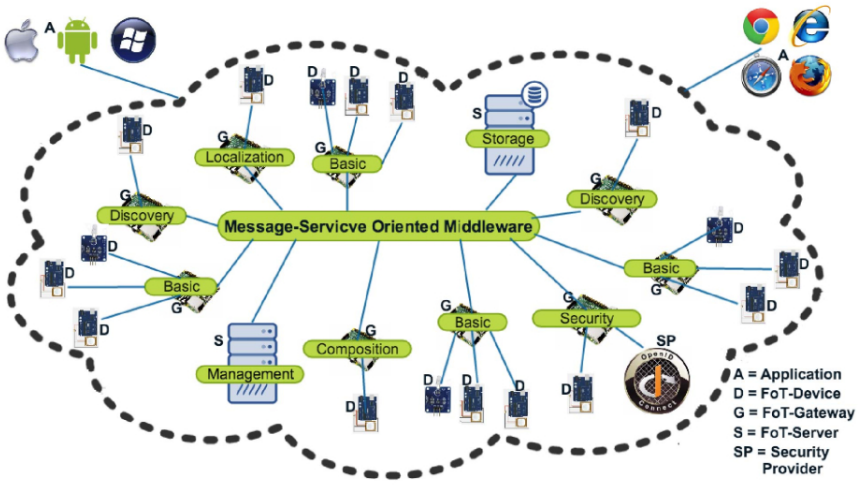


Figure 6.1. Visão Geral SOFT-IoT. Obtida de [Prazeres and Serrano 2016].

Como forma de validar a proposta do paradigna FoT, [Prazeres and Serrano 2016] apresentam a plataforma SOFT-IoT (*Self-Organizing Fog of Things for The Internet of Things*), que é uma implementação concreta do paradigma *Fog of Things*. Os *FoT-Devices* na plataforma SOFT-IoT são nós de sensores ou atuadores embutidos com um

driver (TATUDevice) que é implementado a partir de um protocolo leve denominado TATU (*The Accessible Thing Universe Protocol*). O protocolo TATU estende o protocolo MQTT através da padronização das mensagens para comunicação entre *FoT-Devices* e *FoT-Gateways*. Além disso, *TATUDevice* também implementa o protocolo *Zeroconf*[1] e uma descrição semântica do dispositivo baseada em uma taxonomia (ontologia) para dispositivos IoT.

O *gateway* da SOFT-IoT implementa um *middleware* composto por duas camadas, uma camada orientada a serviços e outra orientada a mensagens [Prazeres et al. 2017]. A camada do *middleware* orientada a serviços fornece comunicação entre os aplicativos e os serviços implantados nos *gateways*, utilizando a tecnologia ESB (Enterprise Service Bus) com base na especificação OSGi[2]. Por outro lado, a camada orientada a mensagens fornece comunicação entre os *FoT-Gateways* e os *FoT-Devices* da SOFT-IoT. Portanto, um *FoT-Gateway* na plataforma SOFT-IoT é, em geral, um dispositivo de baixo custo com processamento e recursos de memória limitados. Além disso, o *FoT-Gateway* utiliza versões reduzidas do sistema operacional Linux, uma máquina virtual Java, uma implementação da especificação OSGi (parte orientada para o serviço) e um servidor MQTT (parte orientada para mensagens). As tecnologias utilizadas no *FoT-Gateway* são mostradas na Figura 6.2. Na arquitetura mostrada na Figura 6.2, a camada do *middleware* orientada a serviços fornece interfaces para aplicativos acessarem os dispositivos via Web Services RESTful (TaaS), utilizando a tecnologia Apache CXF. Por sua vez, o *broker* MQTT serve para fornecer comunicação através de mensagens entre o *gateway* e os dispositivos e também entre os diversos gateways. Por fim, existe o componente *Mapping Devices* que tem como objetivo intermediar a comunicação entre os dispositivos e os *gateways*, sendo implementado seguindo a especificação OSGI e implantado no servidor Apache Karaf.

A plataforma SOFT-IoT propõe três tipos de servidores FoT, são eles: servidor de gerenciamento; servidor de armazenamento; e um servidor provedor de segurança. O primeiro é um servidor FoT que funciona como um tipo especial de *gateway* e implementa todos os serviços relacionados com a auto-organização dinâmica da plataforma SOFT-IoT. Os dois últimos são *FoT-Servers* que funcionam como um tipo especial de recurso e implementa serviços relacionados aos aspectos de armazenamento e segurança. A plataforma SOFT-IoT, por se basear em serviços, é uma plataforma suficientemente flexível para implantação dinâmica de novos tipos de servidores FoT, que, portanto, podem oferecer seus serviços para fins específicos. Por exemplo, os serviços IoT para análise de grandes volumes de dados (*Big Data Analytics*) podem ser desenvolvidos e implementados em *FoT-Servers* para análise de dados localmente (na borda da rede).

A plataforma SOFT-IoT também propõe recursos de auto-organização que devem ser implantados nos servidores da FoT, por exemplo, os servidores de gerenciamento devem implementar todos os serviços relacionados à auto-organização da plataforma SOFT-IoT. Os principais serviços que podem ser implementados nos servidores de gerenciamento são [Sousa and Prazeres 2018][Prazeres et al. 2017]: serviço de monitoramento auto-organizado; serviço de implantação de *gateways*; serviço de recuperação de falhas; serviço de gerenciamento e balanceamento dos perfis.

[1]http://www.zeroconf.org/

[2]https://www.osgi.org/developer/architecture/

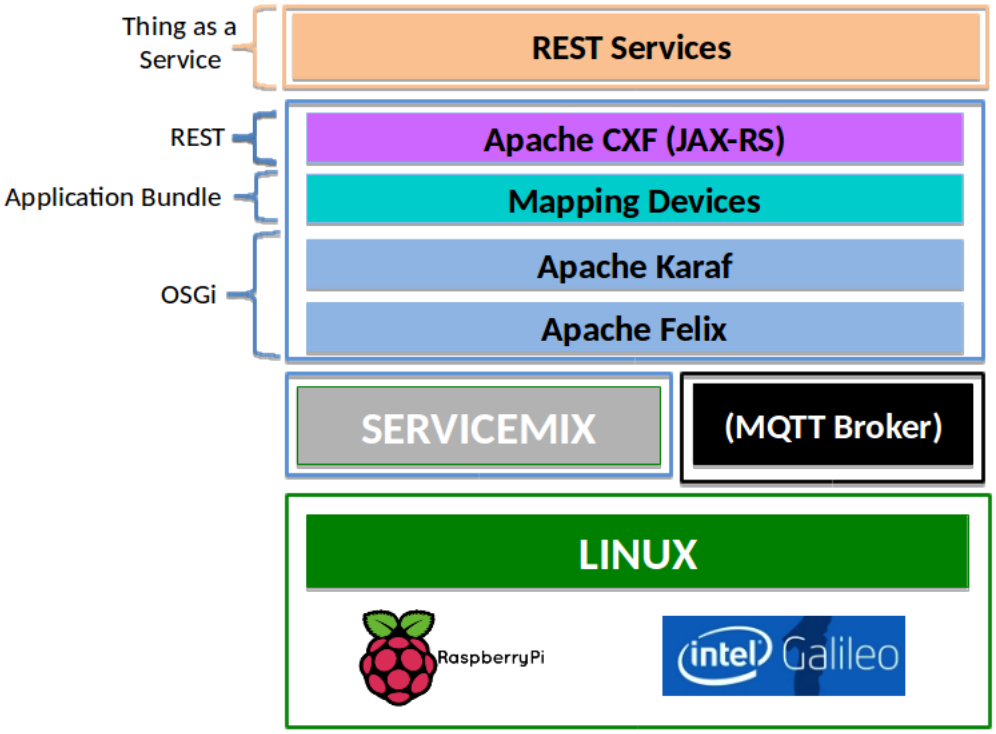


Figure 6.2. Gateway da Fog of Things para plataforma SOFT-IoT, adaptado de [Prazeres and Serrano 2016].

O paradigma *Fog of Things* e sua implementação concreta na plataforma SOFT-IoT suporta a implantação em diversos domínios da IoT, como, por exemplo, rede de sensores básicos, casas inteligentes, veículos autônomos e assistência médica. Além disso, permite a utilização de diferentes configurações de arquiteturas na Computação em Névoa e/ou na Computação em Nuvem [Andrade et al. 2018], como apresentado na Figura 6.3. Com a SOFT-IoT, podem ser criadas as seguintes arquiteturas computacionais: somente redes de sensores de área pessoal; rede de sensores de área pessoal combinada com gerenciamento de FoT-Gateways, criando assim uma rede local; rede de sensores de área pessoal com FoT-Gateways e FoT-Servers, criando assim uma infraestrutura mais robusta de rede local; somente Cloud Servers gerenciando os dispositivos locais; implantação de todo infraestrutura desde a rede global até a rede de área pessoal. Ou seja, a arquitetura é flexível e configurável de forma a atender requisitos de infraestrutura e de aplicações específicas ou genéricas.

6.3. Arquitetura IoT e Plataforma SOFT-IoT

Dado a diversidade de plataformas, infraestruturas, arquiteturas e aplicações propostas em IoT, tanto pela academia como pela indústria, alguns pesquisadores e mesmo a indústria têm trabalhado no sentido de propor padronizações e especificações compartilhadas. Khan et al. [Khan et al. 2012], Bassi et al. [Bassi et al. 2013] e Al-Fuqaha et al. [Al-Fuqaha et al. 2015] são exemplos dessas propostas. Tal como apresentado na Seção 6.2, a plataforma SOFT-IoT reusa os conceitos definidos no modelo conceitual de IoT proposto por Bassi et al. [Bassi et al. 2013]. Em relação à arquitetura, a plataforma SOFT-IoT utiliza conceitos definidos por Khan et al. [Khan et al. 2012] e Al-Fuqaha et al. [Al-Fuqaha

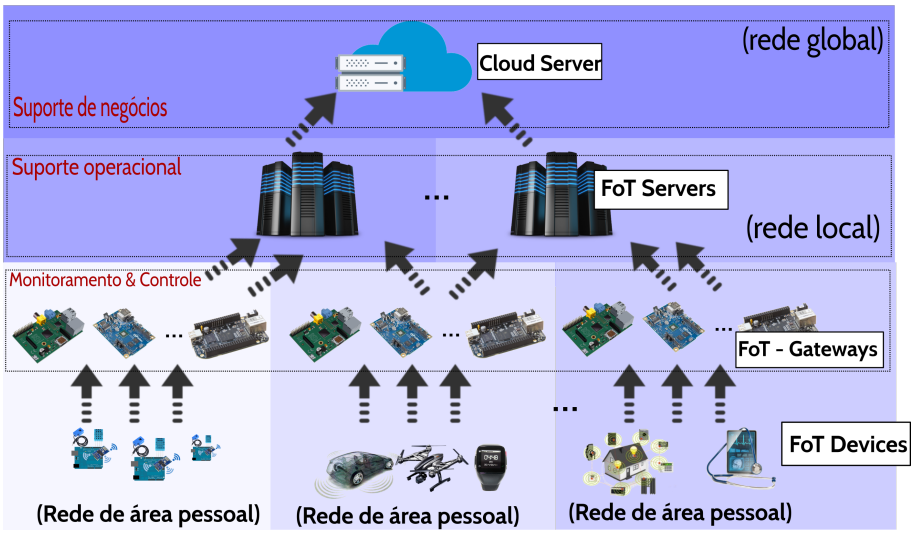


Figure 6.3. Paradigma Fog of Things, adaptado de [Andrade et al. 2018]

et al. 2015], que propuseram a divisão da estrutura da IoT em cinco camadas:

- A **Camada de Percepção** também é denominada camada de dispositivo. Essa camada é composta por objetos físicos e dispositivos como os sensores, atuadores e etiquetas, que trata basicamente da identificação e coleta de informações através de sensores. Dependendo do tipo de sensor, as informações podem ser sobre localização, temperatura, orientação, movimento, vibração, aceleração, umidade, mudanças químicas no ar, dentre outras. As informações coletadas são então passadas para a Camada de Rede para sua transmissão segura ao nó de rede responsável pelo processamento das informações.

- A **Camada de Rede** transfere com segurança as informações dos sensores para o sistema que processará as informações. O meio de transmissão pode ser com fio ou sem fio e a tecnologia pode ser 3G, 4G, UMTS, Wifi, Bluetooth, infravermelho, ZigBee, dentre outros, a depender da tecnologia utilizada no sensor. Assim, a Camada de Rede transfere as informações da camada de percepção para a camada de middleware.

- Na **Camada de Middleware** os dispositivos IoT implementam diferentes tipos de serviços. Cada dispositivo se conecta e se comunica apenas com outros dispositivos que possuam o mesmo tipo de serviço. A principal responsabilidade dessa camada é o gerenciamento de serviços e e o oferecimento de conexão com alguma base de dados. As informações obtidas da Camada de Rede são, dessa forma, armazenadas na base de dados. Além disso, essa camada realiza o processamento de informações e toma decisões automáticas com base nos resultados desse processamento.

- A **Camada de Aplicação** fornece serviços solicitados pelos usuários. Por exemplo, a Camada de Aplicação pode fornecer medições de temperatura e umidade do ar ao usuário que solicitar essas informações. A importância dessa camada para a IoT é que ela possui a capacidade de fornecer serviços inteligentes, com qualidade, visando atender as necessidades dos usuários. A camada de aplicação cobre

vários mercados verticais, como casas inteligentes, prédios inteligentes, transporte, automação industrial e saúde inteligente, dentre outros.

- A **Camada de Negócio** gerencia as atividades e serviços gerais de um sistema IoT. As responsabilidades dessa camada são construir um modelo de negócios, gráficos, fluxogramas, etc., com base nos dados recebidos da Camada de Aplicação. Assim, a Camada de Negócios possibilita o suporte a processos de tomada de decisão com base na análise de *Big Data*. Além disso, o monitoramento e o gerenciamento das quatro camadas subjacentes são obtidos nessa camada. Por fim, a Camada de Negócio também é responsável por comparar a saída de cada camada, com a saída esperada, com a finalidade de melhorar os serviços e manter a privacidade dos usuários.

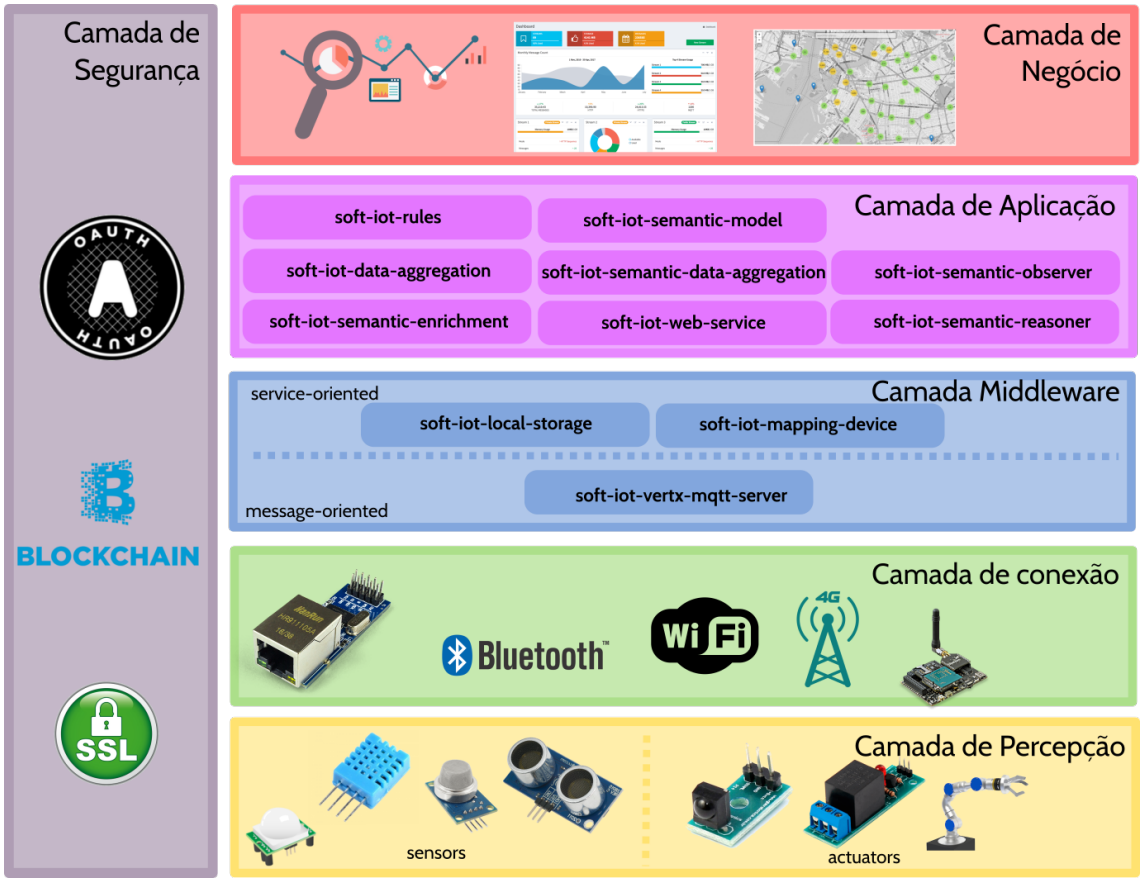


Figure 6.4. Arquitetura IoT com a plataforma SOFT-IoT

A Figura 6.4 mostra uma visão da plataforma SOFT-IoT implementada na arquitetura IoT descrita anteriormente. No restante desta seção, é explicado o desenvolvemos de aplicações IoT a partir da adoção da arquitetura IoT apresentada na Figura 6.4: são descritos os aspectos e os mecanismos de segurança em desenvolvimento na plataforma SOFT-IoT. Ainda no restante desta seção, são descritas as características e os detalhes de configuração e instalação da plataforma SOFT-IoT dentro de cada camada da Figura 6.4.

Para ter uma visão centralizada e resumida da instalação e configuração da plataforma acesse o link abaixo:

`https://github.com/WiserUFBA/soft-iot-platform`

6.3.1. Camada de Percepção

A camada de percepção contém os *FoT-Devices* que são responsáveis por captar informações do ambiente, por meio de sensores, ou atuar no ambiente, por meios de atuadores. Existem vários dispositivos que podem ser utilizados para esta finalidade. Na Figura 6.5 observa-se o nó de sensores Sonoff SC da ITEAD[3]. Esse nó de sensores é composto por cinco sensores: som, temperatura, umidade, poeira e luminosidade, que, na Figura 6.5, estão destacados em amarelo.

Além disso, o nó conta com o microcontrolador ATmega328, em destaque na cor vermelha na Figura 6.5. Esse microcontrolador, que foi desenvolvido pela Atmel Corporation (adquirida pela Microchip Technology), possui baixo custo, porém, também tem um baixo poder de processamento, por ter um microprocessador de 8 bits. No Sonoff SC ilustrado na Figura 6.5, o ATmega328 é responsável pelo interfaceamento entre os sensores com o ESP8266 (em destaque na cor rosa na Figura 6.5).

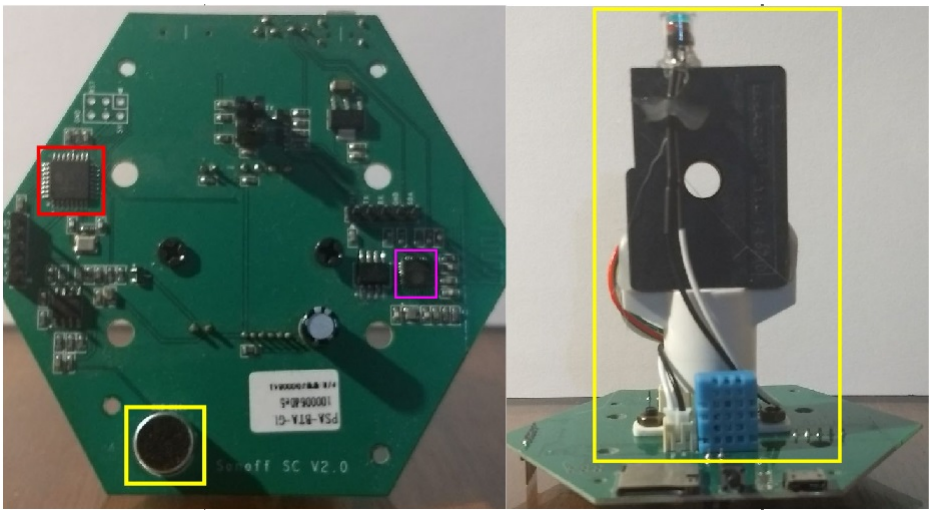


Figure 6.5. Sonoff SC da ITEAD.

O ESP8266 foi desenvolvido pela Espressif e possui poder de processamento maior que o ATmega328, por ter um microprocessador de 32 bits, além de possuir mais memória disponível. Por essas características e pelo seu baixo custo, o ESP8266 representa uma boa escolha como adaptador Wi-Fi e intermediador entre *FoT-Devices* e *FoT-Gateways*.

6.3.1.1. ESP8266

O primeiro passo para possibilitar o uso do ESP8266, presente no Sonoff SC, na plataforma SOFT-IoT, é trocar o *firmware*, que vem de fábrica, antes de sua primeira utilização. Essa

[3]https://www.itead.cc/

ação permite reprogramar o dispositivo utilizando um Ambiente de Desenvolvimento Integrado (Integrated Development Environment - IDE, em inglês), como por exemplo a Arduino IDE[4].

Para esse procedimento, é necessário um conversor serial/USB, como o apresentado na Figura 6.6, onde, em verde, estão destacados a chave seletora de tensão da comunicação serial e os pinos de alimentação (VCC), tanto 3.3V quanto 5V. A posição da chave seletora e qual pino utilizar para alimentar dependem da tensão de operação do dispositivo a ser programado.

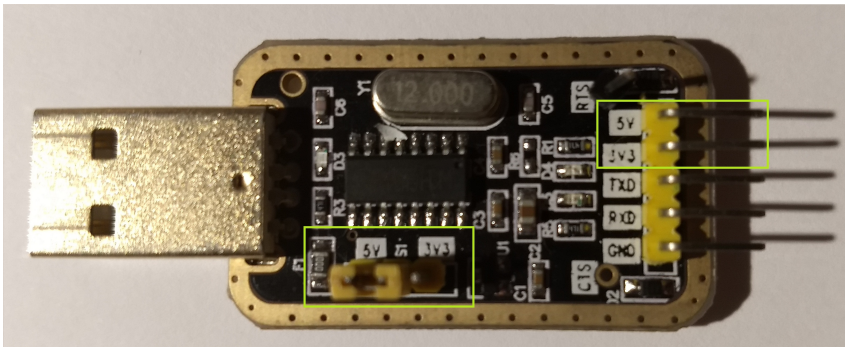


Figure 6.6. Conversor USB/Serial 3.3V/5V CH340G.

O importante, ao decidir qual conversor escolher, é observar o componente a ser programado. Por exemplo, o microcontrolador ATmega328 opera em uma tensão de 5V e necessita de 5 pinos para ser programado: 2 pinos para alimentação (VCC-5V e terra), 2 pinos para comunicação serial (RX, TX) e um pino para reiniciar(RESET). No caso do ESP8266, que opera em uma tensão de 3.3V e seria danificado caso alimentado com tensões maiores, são usados 4 pinos: 2 pinos para alimentação (VCC-3.3V e terra) e 2 pinos para comunicação serial (RX, TX). Portanto, o ideal é escolher um conversor que possa operar nas duas tensões e que possua o pino extra necessário para programar o ATmega328.

Na Figura 6.7, que apresenta a parte traseira do nó de sensores Sonoff SC, estão destacados os pinos de programação do ESP8266 (rosa), do ATmega328 (amarelo) e os dois jumpers (branco), que interligam e permitem a comunicação serial entre esses dois componentes. Esses jumpers são peças pequenas de plástico e metal que devem ser removidos no momento da programação.

Para a troca do *firmware* do ESP8266 é necessário baixar um programa chamado ESP8266 Flash Downloader. O ESP8266 Flash Downloader é um programa para o sistema operacional Windows. Porém, outros programas podem ser utilizados para outros sistemas operacionais, tal como o ESPtool para Linux. Além disso, é necessário um arquivo binário contendo o novo *firmware* (o arquivo e o programa estão disponíveis na página do Github[5]) e 5 fios jumpers Fêmea X Fêmea para interligar o conversor USB/Serial com os pinos de programação de cada componente, devido a necessidade de 5 pinos do microcontrolador ATmega328.

[4]https://www.arduino.cc/en/Main/Software

[5]https://github.com/WiserUFBA/soft-iot-devices/tree/master/Programa_Flash

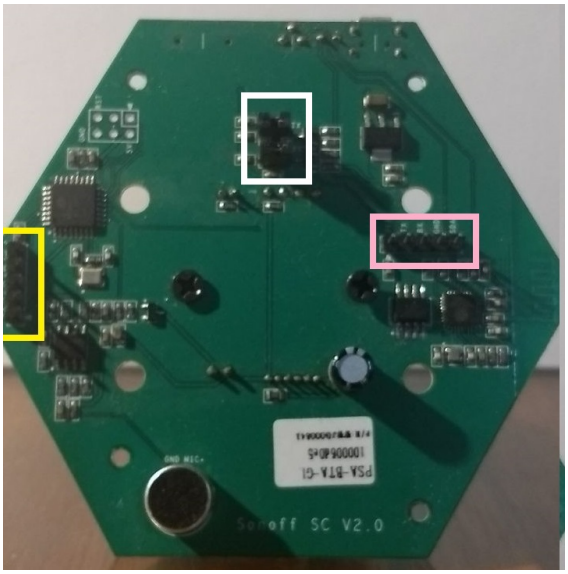


Figure 6.7. Pinos de programaçao do ESP8266 e do ATmega328, respectivamente e localização dos jumpers a serem removidos durante programação.

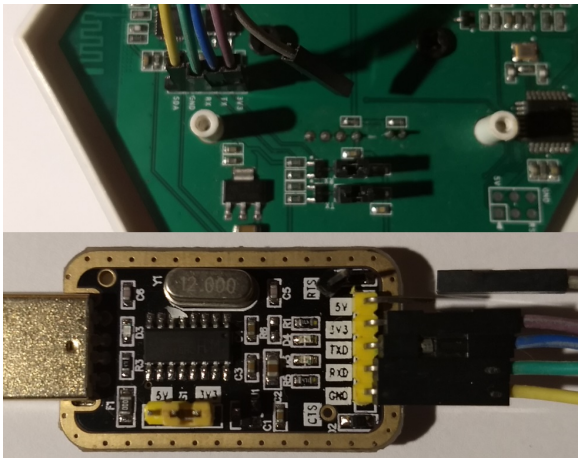


Figure 6.8. Interligação de fios do ESP8266 com conversor USB/Serial.

A Figura 6.8 ilustra a conexão entre o conversor USB/Serial e os pinos de programação do ESP8266 do Sonoff SC. O mais importante é se atentar aos pinos de alimentação e na mudança da chave seletora de tensão da comunicação serial para 3.3V. Os pinos de comunicação podem variar sua conexão a depender do conversor utilizado. Alguns seguem o padrão RX-TX e TX-RX, mas há casos de conversores que requerem TX-TX e RX-RX. Nesse caso, é importante observar a correta posição da chave seletora de tensão para o componente a ser programado e inverter os fios RX e TX em casos de falha ao programar ou trocar de *firmware*.

Outra questão importante sobre a programação dos componentes é que alguns necessitam de procedimentos extras para entrarem no modo de programação. O ESP8266, por exemplo, necessita que o pino GPIO0 seja conectado ao pino RESET. No nó de sensores Sonoff SC essa ação é realizada pressionando e mantendo pressionado o botão disponível na placa no momento de conexão do conversor USB/Serial com o computador.

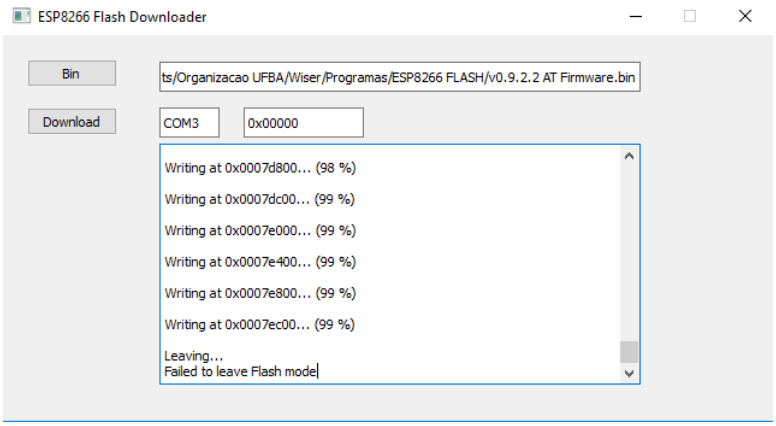


Figure 6.9. Programa de trocar de firmware ao finalizar ação.

Deve-se observar também a porta que o conversor está inserido e modificar para a porta correta no programa, por padrão o programa inicia com a porta "COM0". Na Figura 6.9, observa-se a janela do programa ao finalizar o processo de troca de firmware.

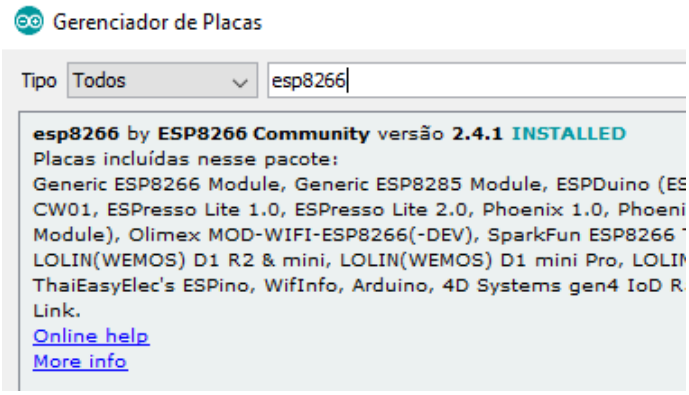


Figure 6.10. Gerenciador de placas da Arduino IDE.

Antes de programar os componentes é necessário configurar a Arduino IDE. O primeiro passo é instalar, por meio do gerenciador de placas da IDE, a placa ESP8266. Para tal, é preciso seguir as abas *Arquivo* > *Preferências*, e no campo "URLs Adicionais para Gerenciadores de Placas" adicionar o seguinte endereço: *http://arduino.esp8266.com/stable/package_esp8266com_index.json*. Dessa forma, a placa aparecerá na busca realizada em *Ferramentas* > *Placa:* > *Gerenciador de Placas...*, como visto na Figura 6.10.

O segundo passo é instalar as bibliotecas utilizadas pelos códigos dos dois componentes (ATmega328 e ESP8266), a Arduino IDE possui um gerenciador próprio para isso que pode ser visualizado na Figura 6.11. As bibliotecas do TATU precisam ser baixadas pela página do grupo de pesquisa WISER(Web, Internet and Intelligent Systems Research) disponível na página do Github[6]. As demais bibliotecas podem ser instaladas pelo gerenciador seguindo as seguintes abas: *Sketch* > *Incluir Biblioteca* > *Gerenciador Bibliotecas...*

[6]https://github.com/WiserUFBA/soft-iot-devices/tree/master/Sonoff_SC

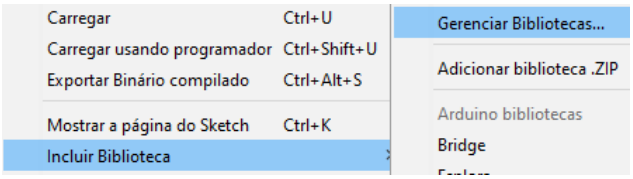


Figure 6.11. Gerenciador de bibliotecas da Arduino IDE.

Elas são utilizadas nas seguintes versões:

1. DHT sensor library da Adafruit - versão 1.0.0
2. PubSubCliente do Nick O'Leary - versão 2.6.0
3. Bounce2 do Thomas O Fredericks - versão 2.52.0
4. ArduinoJson do Benoit Blanchon - versão 5.13.2
5. WiFiManager de tzapu - versão 0.12.0

Com a Arduino IDE configurada, o passo seguinte é programar o ESP8266 com o código chamado "SonoffSC_ESP 8266"[7]. Mesma conexão entre pinos de programação do ESP8266 e conversor USB/Serial utilizada para trocar o firmware da placa, além de novamente usar o procedimento com o botão e a remoção dos jumpers que interligam ESP8266 e ATmega328. Na Arduino IDE, selecione a placa "Generic ESP8266" e defina a porta para a que está sendo utilizada pelo conversor no computador, conforme mostrado na Figura 6.12.

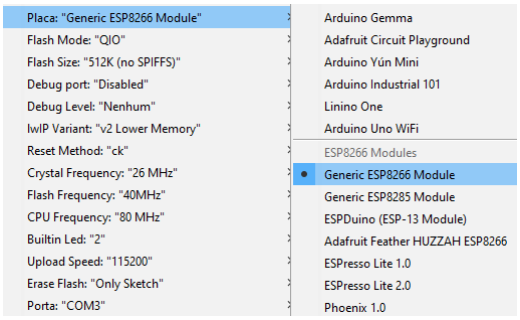


Figure 6.12. Definição da placa a ser utilizada.

Existem 2 pontos a serem modificados antes de carregar o código no ESP8266 do Sonoff SC que podem ser vistos na Figura 6.13. Primeiramente, é necessário definir: nome do dispositivo, usuário do MQTT, senha do MQTT e porta. E depois definir qual será o nome e senha da rede Wi-Fi a ser acessada (não deve ser uma rede existente). Essa rede servirá para configurações dinâmicas do dispositivo que são possíveis devido a biblioteca mencionada anteriormente, WiFiManager. Essas configurações permitem alterar dados, como rede de internet a se conectar, sem a necessidade de reprogramar o dispositivo. Com esses dois pontos definidos, pode-se enviar o código para a placa com o ícone "Carregar" no canto superior esquerdo ou pela combinação de teclas "Crtl + U". Assim que o código estiver carregado, a rede criada anteriormente irá aparecer como disponível.

[7]Disponível em: https://github.com/WiserUFBA/soft-iot-devices/tree/master/Sonoff_SC

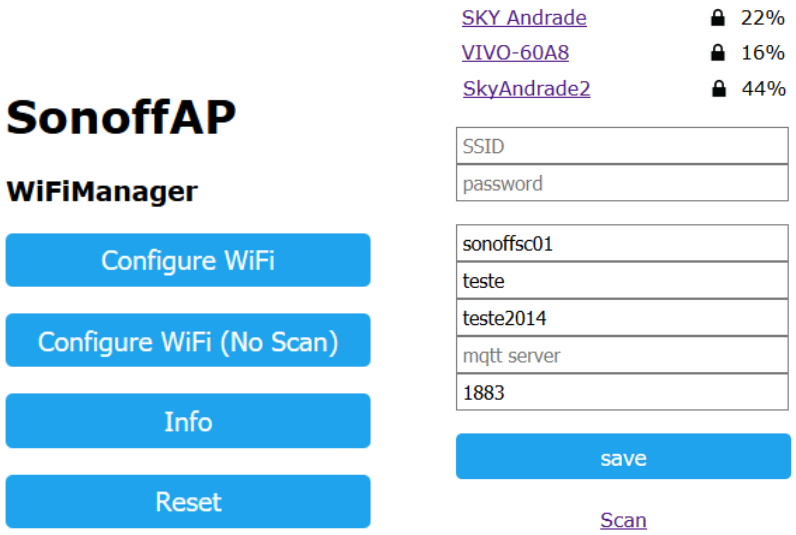
```
SonoffSC_ESP8266
//
char vector_response[1024];

#define stringSize 20
char device_name[stringSize]    = "sonoffsc01";
char mqtt_user[stringSize]      = "teste";
char mqtt_pass[stringSize]      = "teste2014";
char port[stringSize]      = "1883";
char mqtt_server[40];
(...)
wifiManager.startConfigPortal("SonoffAP", "teste2014");
```

Figure 6.13. Parte do código do Sonoff SC para o ESP8266.

Ao se conectar a esta rede de acesso, é preciso entrar em um navegador WEB e acessar o seguinte endereço: *http://192.168.4.1*. Dessa forma, aparecerá uma página como a Figura 6.15(a). As opções são: Configurar WiFi (nesta opção são listadas redes disponíveis), Configurar WiFi sem escaneamento (não são listadas as redes disponíveis), Informações e Reiniciar.

Figure 6.14. Configurando dinamicamente o dispositivo.



(a) Página inicial.

(b) Página exibida ao clicar em "Configure WiFi".

Após selecionar a opção "Configurar WiFi", é apresentada uma página como a Figura 6.15(b) onde as redes disponíveis e qualidade de sinal serão listadas. Assim, é só clicar na rede desejada e digitar a senha. É necessário também informar o endereço do FoT-Gateway com o qual o ESP8266 irá se comunicar. Outras opções de configuração possíveis envolvem alterar: nome do dispositivo, usuário MQTT, senha MQTT e porta. Finalizado todas as configurações a serem realizadas, basta salvar. Por fim, aparecerá uma página informando que os dados foram salvos.

Outro fato interessante da configuração dinâmica, é a possibilidade de não perder esses dados em casos de reinício do nó de sensores. Quando esse evento ocorre, o nó

se conecta automaticamente na última rede configurada. Esta rede pode ser alterada ao apertar o botão presente no Sonoff SC e entrando novamente na rede de acesso das configurações dinâmicas.

6.3.1.2. ATmega328

O microcontrolador ATmega328 também precisa ser reprogramado para que o Sonoff SC possa ser utilizado como um FoT-Device na plataforma SOFT-IoT. Esse processo é mais simples, para isso é necessário carregar código chamado "SonoffSC_Arduino"[8] nele. As interligações entre os pinos de programação do ATmega328 e do conversor USB/Serial estão mostradas na Figura 6.15.

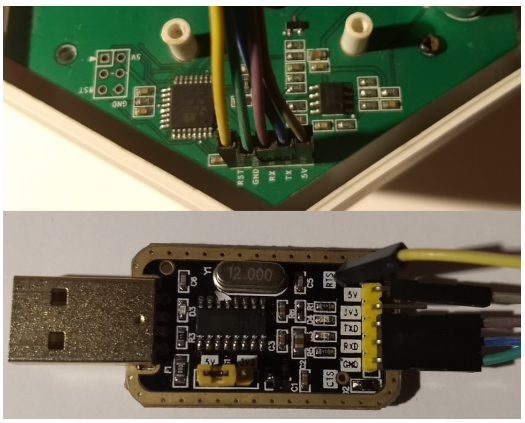


Figure 6.15. Interligação de fios do ATmega328 com conversor USB/Serial.

Com as interligações feitas, é preciso configurar a Arduino IDE para programar o ATmega328. É preciso ir em *Ferramentas* > *Placa:* e selecionar a opção "Arduino Mini". Lembrando-se sempre de confirmar se a porta está configurada corretamente para a porta que o conversor está utilizando no computador. Antes de enviar o código, é válido analisar parte do código do ATmega328, mostrado na Figura 6.16. O nome do dispositivo deve ser o mesmo configurado no ESP8266, assim como a porta MQTT. Observado essas informações, pode-se enviar o código. Finalizado esta tarefa, o Sonoff SC estará definitivamente pronto para exercer o seu papel de FoT-Device, bastando recolocar os jumpers removidos para programação e ligá-lo a uma fonte de alimentação.

SonoffSC_Arduino

```
#include <stdint.h>
#include <string.h>
#include <DHT.h>

// Constants to connection with the broker
#define DEVICE_NAME   "sonoffsc01"
#define MQTT_PORT     1883
```

Figure 6.16. Parte do código do Sonoff SC para o ATmega328.

[8]Disponível em: https://github.com/WiserUFBA/soft-iot-devices/tree/master/Sonoff_SC

6.3.1.3. Protocolo TATU

Como protocolo de comunicação entre FoT-Devices e FoT-Gateway é utilizado o protocolo TATU (The Acessible Thing Universe). Este protocolo é uma extensão do protocolo MQTT (Message Queuing Telemetry Transport) desenvolvida para atender algumas das necessidades de um sistema IoT, como por exemplo, métodos que facilitem a edição das variáveis e dos pinos, além de informar os valores destes pinos, métodos que retornem informações do dispositivo (nome, id, localização) e a possibilidade de atribuir nomes a variáveis.

Este protocolo contém 3 métodos:

(i) GET: retorna o valor de uma variável, pino ou propriedade do sistema.

(ii) SET: edita o valor de uma variável ou pino.

(iii) POST: é o retorno de qualquer um dos métodos descritos anteriormente, representado por uma mensagem no formato JSON.

O GET é utilizado em FoT-Devices que possuem sensores, de modo que os dados desses sensores possam ser capturados e processados por outros componentes da FoT. O SET é utilizado em FoT-Devices que possuem atuadores para editar pinos específicos desses atuadores com a finalidade de realizar a mudança definida por outros componentes da FoT.

Além disso, o protocolo TATU disponibiliza ao FoT-Gateway e FoT-Device dois tipos de fluxos de dados [Batista et al. 2018a][Batista et al. 2018b]:

(i) GET: requisita um dado de algum sensor, sem periodicidade definida. Ilustrado na Figura 6.18(a).

(ii) FLOW: requisita uma amostra de dados de algum sensor que deve ser enviada a cada Y unidades de tempo. Nesse fluxo, o FoT-Device irá coletar dados para formar uma amostra a cada X unidades de tempo e quando atingir Y unidades de tempo irá enviar esses dados em conjunto para o FoT-Gateway. Ilustrado na Figura 6.18(b).

As requisições de dados e as respostas obtidas seguem os padrões descritos na Figura 6.19(a) e 6.19(b), respectivamente. Além disso, exemplos de cada uma das formas de requisições no protocolo TATU são apresentadas no código 6.1. O tópico das requisições é o *"\dev"* e o dispositivo retornará respostas no tópico *"\dev\nome_dis e positivo\RES"*.

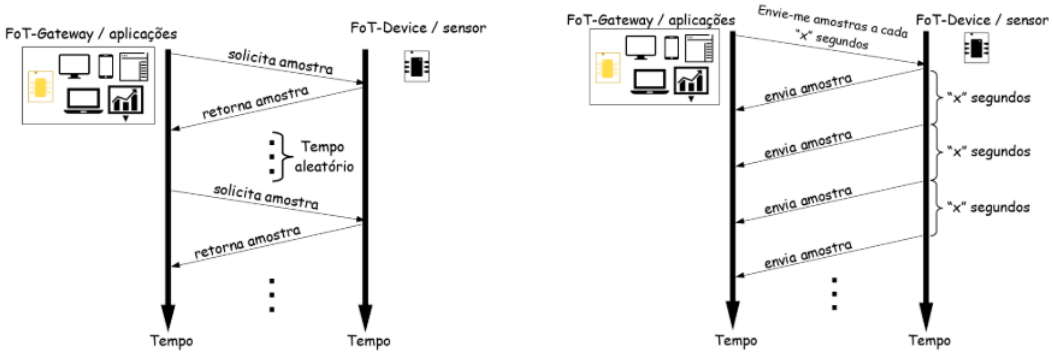
6.3.2. Camada Middleware

A camada middleware na SOFT-IoT é responsável por intermediar a comunicação com os dispositivos IoT (FoT-Device) com os demais serviços da plataforma. O middleware pode ser implantado em dispositivos de capacidade limitada (como Raspberry Pi[9] e BeagleBone[10]), e também servidores em rede local ou remota (Cloud). A SOFT-IoT define

[9]Raspberry Pi - https://www.raspberrypi.org/

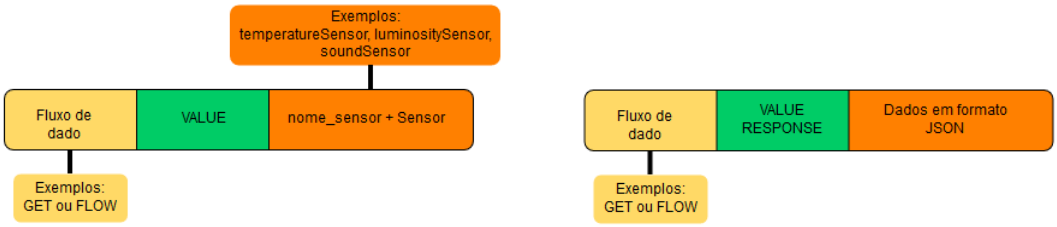
[10]BeagleBone - https://beagleboard.org/bone

Figure 6.17. Ilustrações dos fluxos de dados.



(a) Fluxos baseados em Requisições GET (b) Fluxos baseados em Requisições FLOW

Figure 6.18. Padrão de formato para requisições e respostas.



(a) Formato das requisições. (b) Formato das respostas.

```
//Exemplo de requisicao baseada em GET com sensor de temperatura
GET VALUE temperatureSensor
//Resposta para requisicoes GET
GET VALUE RESPONSE {"CODE":"POST","HEADER":{"NAME":"device_name"
    },"METHOD":"GET","BODY":{"temperaturaSensor":25}}

//Exemplo de requisicao baseada em FLOW, coletas a cada 200 ms e
     publicacoes a cada 1000 ms
FLOW VALUE temperatureSensor {"collect":200, "publish":1000}
//Resposta para requisicoes FLOW, uma aresta com dados da
    amostra obtida.
FLOW VALUE RESPONSE {"CODE":"POST","HEADER":{"NAME":"device_name
    "},"METHOD":"GET","BODY":{"temperaturaSensor"
    :[25,25,27,25,26],"FLOW":{"collect":200,"publish":1000}}}
```

Código 6.1. Exemplos de requisicoes e respostas no protocolo TATU

que o middleware é implementado no FoT-Gateway, e este quando alocado em dispositivos/servidores locais, não dependente do acesso à internet (somente comunicação de rede local) para coleta, armazenamento e acesso aos dados produzidos pelos sensores. Quando implantados em servidores remotos precisam de conectividade estabelecida para captura dos dados dos sensores.

A implementação do middleware na plataforma SOFT-IoT é realizada utilizando o Apache ServiceMix, que funciona como um Enterprise Service Bus (ESB – barramento de serviços), baseado na especificação OSGi. Além disso, o middleware é dividido em duas partes, as quais são [Prazeres et al. 2017]:

- **Orientada a mensagem**: provê o ponto de comunicação com os sensores e atuadores (FoT-Device) através de mensagens MQTT com protocolo de comunicação TATU;
- **Orientada a serviço**: tem a função de viabilizar e oferecer interfaces de acesso aos dados dispositivos pela camada de aplicação.

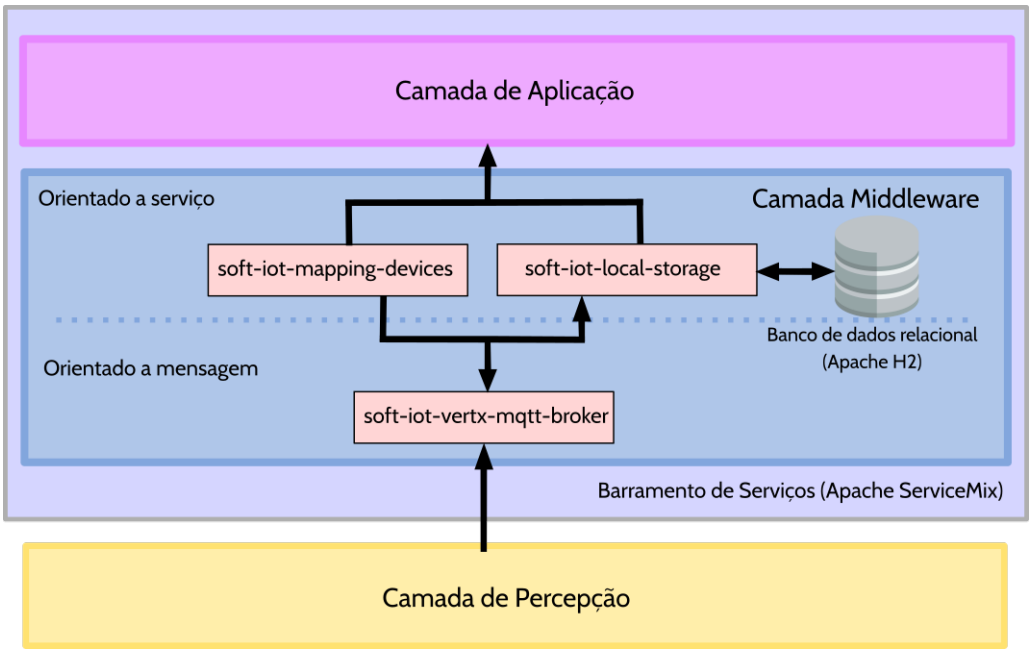


Figure 6.19. Componentes e interações da camada Middleware da SOFT-IoT

A Figura 6.19 mostra a camada middleware, seus módulos e interações. Nota-se que a camada middleware é implementada sobe o ESB com o Apache ServiceMix e possui módulos para prover a comunicação com os dispositivos da camada de Percepção (*soft-iot-vertx-mqtt-broker*), para mapeamento dos sensores (*soft-iot-mapping-devices*) e para coleta e armazenamento dos dados (*soft-iot-local-storage*). Além disso, o módulo *soft-iot-local-storage*, junto com o *soft-iot-mapping-devices*, fornece uma interface de acesso aos dados dos sensores para outros módulos implementados na camada de aplicação.

As subseções a seguir detalham o funcionando da infraestrutura e dos módulos que compõem a camada middleware, tal como um guia para sua instalação e configuração.

6.3.2.1. Barramento de Serviços - ESB

A utilização de um barramento de serviços permite a implantação dinâmica de softwares através de uma infraestrutura base, a qual fornece suporte a comunicação entre aplicações. O ServiceMix, baseado na especificação OSGi, é utilizado como barramento de serviço na plataforma SOFT-IoT.

O Apache ServiceMix[11] é um software código aberto, implementado em Java, no qual serviços nativos da arquitetura ESB/OSGi oferecem toda a infraestrutura necessária para o suporte dos outros serviços da plataforma SOFT-IoT. Com o ServiceMix é possível implantar serviços (também chamados de bundles) em tempo de execução e permite que estes partilhem dados e objetos através de relacionamentos e dependências.

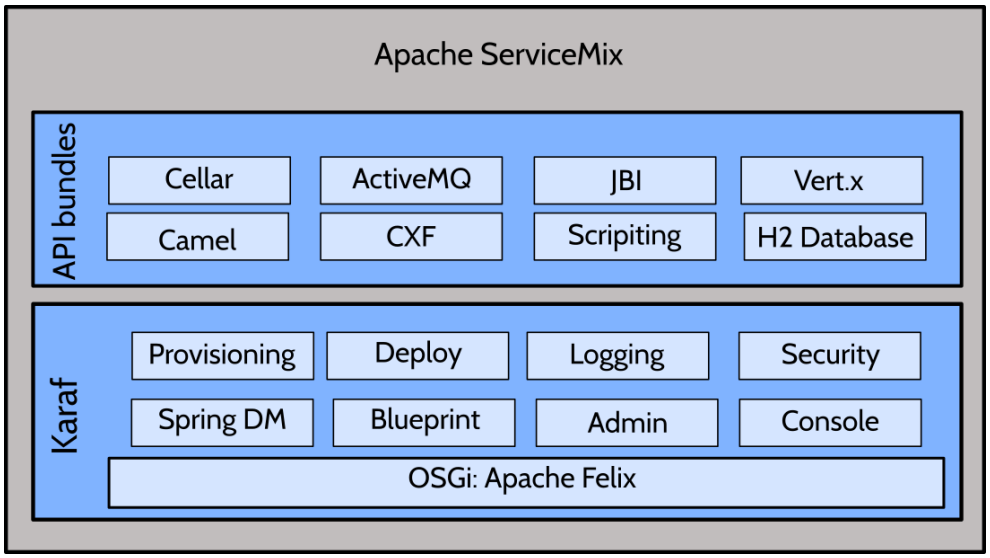


Figure 6.20. Componentes do Apache ServiceMix

A Figura 6.20 representa o ServiceMix e os seus principais módulos. O ServiceMix é leve e portável, tendo como requisito básico o suporte ao Java 1.7. Além disso, possui suporte ao Spring Framework e Blueprint e é compatível com o Java SE ou com um servidor de aplicativos Java EE. O Karaf é o núcleo principal do ServiceMix e oferece o conceito de recursos, onde coleções de pacotes podem ser instalados como um grupo em um ambiente OSGi em execução. Sobe o Karaf, o ServiceMix usa o ActiveMQ para fornecer serviço de mensagens, CXF para suporte serviços RESTful, Cellar para comunicação e interações entre diferentes instalações do ServiceMix e Camel para integração e troca de dados através de rotas. Por fim, o ServiceMix contém módulos adicionais como H2 Database, que gerencia o sistema de banco de dados relacional baseado em arquivos.

A plataforma SOFT-IoT foi baseada na versão 6.10 do ServiceMix. Para seu uso e instalação em um ambiente Linux é necessário somente baixar o pacote do programa, o qual pode ser encontrado em:

`https://servicemix.apache.org/downloads/servicemix-6.1.0.html`

[11] http://servicemix.apache.org/

Após a descompactação do pacote que contém o ServiceMix é possível observar uma conjunto de diretórios com funções específicas, dentre os quais podemos destacar: `etc/`: contém arquivos de configuração dos bundles (incluirá também os arquivos de configuração dos bundles da SOFT-IoT; `bin/`: contém os executáveis para inicializar o ServiceMix através do Karaf; `data/`: armazena dados produzidos pelos bundles, no caso do SOFT-IoT, nesse diretório que serão armazenados os dados obtidos dos sensores; `system/`: contém os arquivos básicos que suportam o ServiceMix.

Para inicializar o ServiceMix em sistemas Linux é preciso executar dentro do diretório do base o seguinte arquivo:

```
$ ./bin/servicemix
```

Para inicialização em sistemas operacionais Windows é preciso executar o arquivo *servicemix.bat*. A Figura 6.21 exibe o terminal do ServiceMix, no qual é possível executar comandos, como por exemplo para instalação e remoção de bundles, para listar dados, e também para verificar status das aplicações que estão operando. Ao completar a inicialização do ServiceMix o terminal é inicializado e disponibilizando para o usuário. Eventuais erros de execução nos bundles também são exibidos nesse terminal.

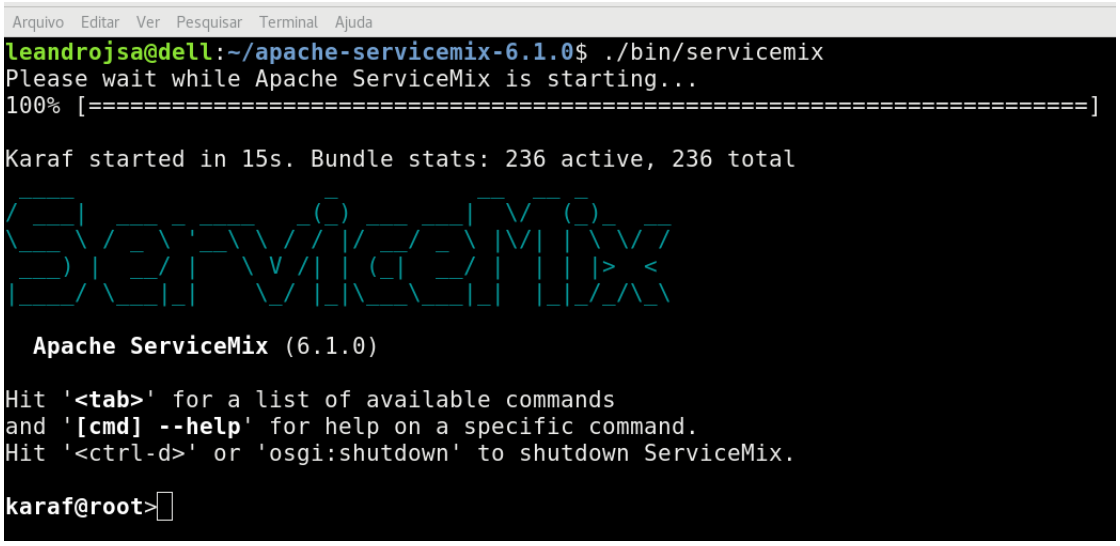


Figure 6.21. Terminal do Apache ServiceMix

O ServiceMix também pode ser gerenciado através de uma aplicação Web chamado webconsole. Nela é possível gerenciar por meio de uma interface gráfica, funções como instalação, atualização, remoção e informações de bundles e também funções de debug. Para instalar o webconsole no ServiceMix basta executar o seguinte comando no terminal:

```
karaf@root()> feature:install webconsole
```

Por padrão aplicação Web inicializada pelo webconsole é configurada na porta 8181 com login e senha `karaf` e pode ser acessada pelo endereço:

`http://localhost:8181/system/console`

A Figura 6.22 mostra um recorte do webconsole. Observe que é possível através dele ter um panorama geral dos bundles em execução e também alterar seus status. O

webconsole, além disso, permite instalar novos bundles e depurar o estado e eventuais erros das aplicações em execução.

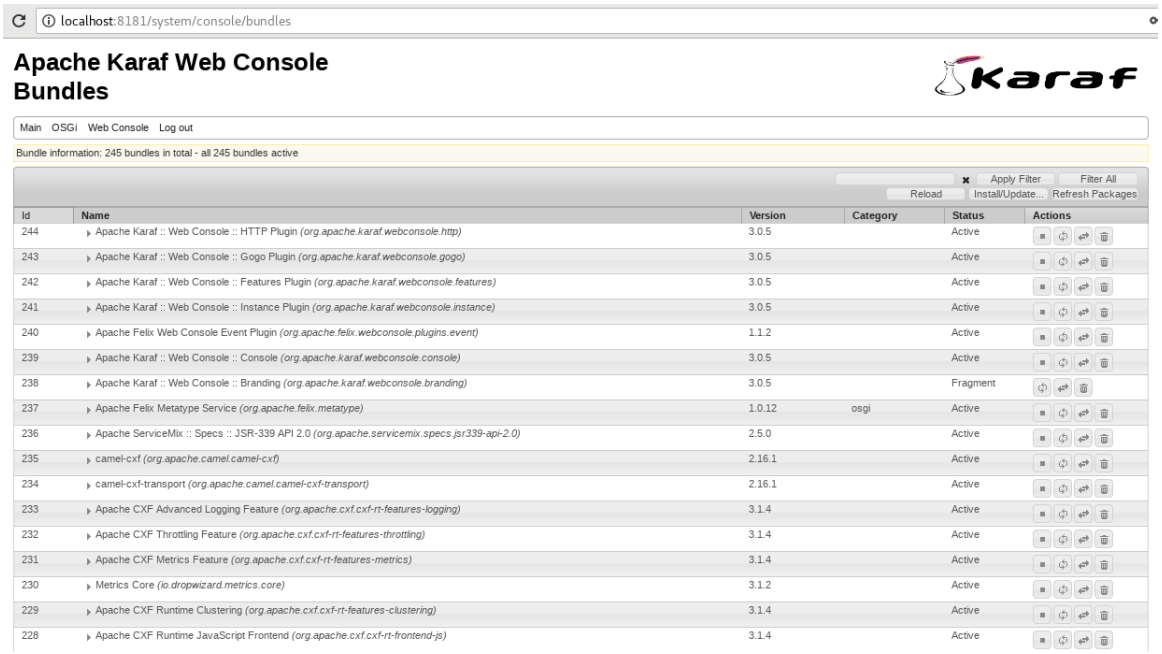


Figure 6.22. Webconsole do Apache ServiceMix (Karaf)

Por fim, é possível configurar o ServiceMix para permitir a instalação dos módulos da plataforma SOFT-IoT através do terminal. Isso é possibilitado pela inserção do repositório Maven do SOFT-IoT nas configurações do ServiceMix, habilitando a instalação dos bundles através de instruções de terminal. Para tal é preciso executar os seguintes comandos:

```
karaf@root()> config:edit org.ops4j.pax.url.mvn
karaf@root()> config:property-append org.ops4j.pax.url.mvn.repositories
",https://github.com/WiserUFBA/wiser-mvn-repo/raw/master/releases@
id=wiser"
karaf@root()> config:update
```

6.3.2.2. Módulo soft-iot-vertx-mqtt-broker

O componente MQTT Broker fornece um servidor que é capaz de lidar com conexões, comunicação e troca de mensagens com clientes MQTT remotos. Na plataforma SOFT-IoT o broker MQTT é implemetado no módulo **soft-iot-vertx-mqtt-broker**[12] uma implementação baseada em OSGI como um *bundle* do ServiceMix.

Este módulo é responsável por habilitar comunicações com clientes MQTT remotos, que no caso da plataforma SOFT-IoT são dispositivos conectados. Além das vantagens de usar uma arquitetura modular, o uso da API Vert.x MQTT Server torna possível escalar o agente MQTT reativo de acordo com, por exemplo, o número de núcleos do sistema e, assim, possibilitar a escalabilidade horizontal.

[12]https://github.com/WiserUFBA/soft-iot-vertx-mqtt-broker

Antes instalar o módulo soft-iot-vertx-mqtt-broker é necessário introduzir ao ServiceMix alguns módulos ao Vert.x. As dependências estão localizadas no endereço:

```
https://github.com/WiserUFBA/soft-iot-vertx-mqtt-broker/tree/master/
src/main/resources/dependencies
```

As dependências são bibliotecas .jar que devem ser incluídas no ServiceMix, podendo ser introduzidas através do webconsole, conforme pode ser visto na Figura 6.22 através do botão `Install/Update`. Além da instalação das dependências é necessário também incluir arquivos referente ao certificado de segurança do soft-iot-vertx-mqtt-broker. Tais arquivos estão disponíveis em:

```
https://github.com/WiserUFBA/soft-iot-vertx-mqtt-broker/tree/master/
src/main/resources/certificates
```

O certificado de segurança possui um arquivo de configuração disponível em:

```
https://github.com/WiserUFBA/soft-iot-vertx-mqtt-broker/blob/master/
src/main/resources/configuration/
br.ufba.dcc.wiser.soft_iot.gateway.brokers.cfg
```

Os arquivos do certificado de segurança, incluindo o arquivo de configuração devem ser armazenados em:

```
<diretorio_servicemix>/etc
```

Para instalação do soft-iot-vertx-mqtt-broker é necessário executar os seguintes comandos no terminal do ServiceMix:

```
karaf@root()> bundle:install mvn:br.ufba.dcc.wiser.soft_iot/
soft-iot-vertx-mqtt-broker/1.0.0
```

6.3.2.3. Módulo soft-iot-mapping-devices

A plataforma SOFT-IoT tem um módulo específico para o mapeamento dos dispositivos e tradução das mensagens comunicação TATU entre o FoT-Device e FoT-Gateway. O **soft-iot-mapping-devices**[13] implementa as interfaces virtuais dos sensores e atuadores conectados na plataforma. Também funciona como tradutor das mensagens TATU trocadas entre os dispositivos sensoriais (FoT-Device) e o FoT-Gateway, as quais permitem requisitar e responder solicitações por dados dos sensores e ações dos atuadores.

O soft-iot-mapping-devices instancia um conjunto de objetos que representam os dispositivos, sensores e atuadores do FoT-Device. Esses objetos contém informações relacionadas aos dispositivos tais como identificador único, tipo do sensor/atuador e dados sobre localização. Além disso, estão disponíveis para acesso pelos demais módulos da plataforma, servindo como interface de acesso aos sensores e/ou atuadores conectados na plataforma.

Para instalação do soft-iot-mapping-devices é necessário executar os seguintes comandos no terminal do ServiceMix:

```
karaf@root()> bundle:install mvn:br.ufba.dcc.wiser.soft_iot/
soft-iot-mapping-devices/1.0.0
```

[13]https://github.com/WiserUFBA/soft-iot-mapping-devices

Após a instalação do soft-iot-mapping-devices é necessário gerar um arquivo de configuração que irá conter informações referentes aos dispositivos conectados a plataforma. Um modelo do arquivo pode ser encontrado em:

```
https://github.com/WiserUFBA/soft-iot-mapping-devices/blob/master/src
/main/resources/br.ufba.dcc.wiser.soft_iot.gateway.mapping_devices.cfg
```

E deve ser armazenado no seguinte diretório:

```
<diretorio_servicemix>/etc
```

O código abaixo apresenta um exemplo do arquivo de configuração do soft-iot-mapping-devices para alguns dispositivos com sensores:

```
#JSON with array of devices:
DevicesConnected=[{"id":"mydevice01", "latitude":53.290411,
"longitude":-9.074406,
 "sensors":[{"id":"temp01", "type":"temperatureSensor",
 "collection_time":30000, "publishing_time": 60000},
 {"id":"humd01", "type":"humiditySensor",
 "collection_time":30000,  "publishing_time": 60000}]},
 {"id":"sonoff01", "latitude":53.290457, "longitude":-9.074423,
  "sensors":[{"id":"temp01", "type":"temperatureSensor",
  "collection_time":30000,  "publishing_time": 60000},
 {"id":"humd01", "type":"humiditySensor", "collection_time":30000,
 "publishing_time": 60000},
 {"id":"sound01", "type":"SoundSensor", "collection_time":30000,
 "publishing_time": 60000},
 {"id":"light01", "type":"LightSensor", "collection_time":30000,
 "publishing_time": 60000},
 {"id":"dust01", "type":"AirPollutantSensor", "collection_time":30000,
 "publishing_time": 60000}]}]

# Habilita ou desabilitar o debug mode
debugMode=false
```

A configuração descreve que existem dois dispositivos conectados a plataforma com seguintes identificadores: `mydevice01`, e `sonoff01`. Cada dispositivo possui dados em relação a localização (latitude e longitude) e seus respectivos sensores. Já cada sensor dos dispositivos conectados possui um identificador, um tipo e os tempos de coletas de dados e publicação no broker MQTT.

6.3.2.4. Módulo soft-iot-local-storage

O **soft-iot-local-storage**[14] é responsável por promover a coleta, armazenamento e acesso interno dos dados produzidos pelos sensores do FoT-Device. Neste módulo concentra-se as funcionalidades referentes ao armazenamento interno dos dados sensoriais, tal como o acesso a estes dados pelos demais módulos da plataforma SOFT-IoT.

Com o suporte do soft-iot-mapping-devices para tradução das mensagens TATU e identificação dos dispositivos conectados a plataforma, o soft-iot-local-storage requisita fluxo dados de cada sensor conectado na plataforma em uma frequência definida pelo

[14]https://github.com/WiserUFBA/soft-iot-local-storage

usuário. As mensagens de requisição e respostas são trocada no broker MQTT implementado pelo módulo soft-iot-vertx-mqtt-broker para por fim serem armazenadas no banco de dados relacional Apache H2.

A Figura 6.23 apresenta um diagrama de sequência das principais operações do soft-iot-local-storage com os seus relacionamentos. Com os dispositivos conectados na plataforma, através da configuração do usuário (*1.set a JSON with connected devices*, Fig. 6.23), o soft-iot-local-storage obtém do módulo soft-iot-mapping-devices os sensores conectados e seus respectivos fluxos de coleta de dados (*2. getConnectedDevices()*, Fig. 6.23) para publicar no soft-iot-vertx-mqtt-broker as requisições de dados do sensores (*3.setFLOWSenrsorsDevices()*, Fig. 6.23). Com a requisição dos dados, os sensores irão publicar no soft-iot-vertx-mqtt-broker as informações capturadas na frequência configurada. O soft-iot-local-storage por sua vez irá coletar tais dados (*4.getDataSensors()*, Fig. 6.23) e armazená-los no banco de dados Apache H2 (*5.StoreSensorData()*, Fig. 6.23). O soft-iot-local-storage também implementa um procedimento para remoção de dados antigos, o qual a frequência pode ser configurada pelo usuário (*6.cleanOldData()*, Fig. 6.23) e módulos da camada de aplicação podem solicitar dados dos sensores através do soft-iot-local-storage (7.*requestDataSensor(), Fig. 6.23*).

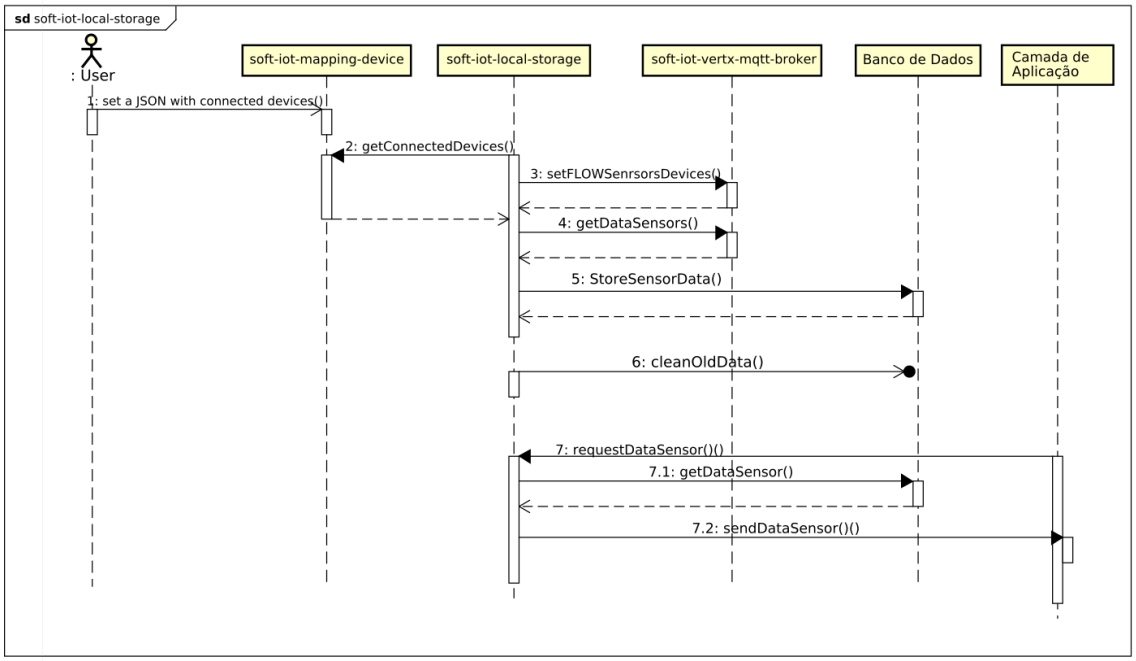


Figure 6.23. Diagrama de sequência com as operações e relacionamentos do soft-iot-local-storage

Antes de propriamente instalar o módulo soft-iot-local-storage é necessário introduzir ao ServiceMix alguns módulos que darão suporte ao banco de dados Apache H2. Assim é necessário executar os seguintes comandos no terminal do ServiceMix:

```
karaf@root()> feature:repo-add mvn:org.ops4j.pax.jdbc/pax-jdbc
-features/0.8.0/xml/features
karaf@root()> feature:install transaction jndi pax-jdbc-h2 pax-jdbc
-pool-dbcp2 pax-jdbc-config
```

Com Apache H2 instalado é necessário também criar um arquivo de configuração para o banco de dados das informações coletadas nos sensores. O arquivo é:

```
<diretorio_servicemix>/etc/org.ops4j.datasource-gateway.cfg
```

E deve ter o seguinte conteúdo:

```
osgi.jdbc.driver.name=H2-pool-xa
url=jdbc:h2:${karaf.data}/soft-iot-local-storage
dataSourceName=soft-iot-local-storage
```

Este arquivo é responsável como indicar um driver para operar as transações no banco de dados, tal como o arquivo em que as inserções serão armazenadas.

É importante observar que o soft-iot-local-storage é dependente dos módulos soft-iot-mapping-devices e soft-iot-vertx-mqtt-broker, para correto funcionamento. Por fim, para instalação do soft-iot-local-storage é necessário executar os seguintes comandos no terminal do ServiceMix:

```
karaf@root()> bundle:install mvn:br.ufba.dcc.wiser.soft_iot/
soft-iot-local-storage/1.0.0
```

Após a instalação do soft-iot-local-storage é necessário gerar um arquivo de configuração que irá conter informações referentes aos dispositivos conectados a plataforma. Um modelo do arquivo pode ser encontrado em:

```
https://github.com/WiserUFBA/soft-iot-local-storage/
src/main/resources/br.ufba.dcc.wiser.soft_iot.local_storage.cfg
```

E deve ser armazenado no seguinte diretório:

```
<diretorio_servicemix>/etc
```

O código abaixo apresenta um exemplo do arquivo de configuração do soft-iot-local-storage:

```
#Endereco do broker MQTT
MQTTHost=localhost

#Porta utilizada pelo broker MQTT
MQTTPort=1883

#Usuario e senha do broker MQTT. Se o broker MQTT
#aceita conexoes anonimas, eh possivel manter os campos vazios
MQTTUsername=
MQTTPassword=

#Nome da conexecao estabelecida com MQTT broker.
#Usado somente em opercoes internas.
MQTTServerId=FoTGatway

#Tempo padrao para coleta de dados, caso o sensor nao possua
#configuracao propria
DefaultCollectionTimeSensorData=300000
#Tempo padrao para publicacao de dados, caso o sensor nao possua
#configuracao propria
```

```
DefaultPublishingTimeSensorData=900000

#Numero de horas de dados que o sistema ira manter armazenado
NumberOfHoursDataStored=24

# Habilitar ou desabilitar o debug mode
debugMode=false
```

6.3.3. Camada de Aplicação e Negócios

A camada de aplicação na plataforma SOFT-IoT é responsável por fornecer os serviços solicitados pelos usuários. As aplicações podem ser implantados em dispositivos de capacidade limitada (e.g. Raspberry Pi (FoT-Gateway)) e servidores que estão localizados na borda rede (FoT-Server) ou em servidores localizados na Cloud (Cloud-Server).

Na SOFT-IOT as aplicações são construídas em linguagem de programação Java, empacotadas como um bundle OSGI e funcionam em um barramento de serviços conforme explicado na Seção 6.3.2.1. O bundle é a unidade de implementação principal ao usar o OSGi. É basicamente um arquivo jar contendo alguns cabeçalhos adicionais no MANIFEST usado pelo *framework* OSGi, no caso da SOFT-IOT, o Karaf[15]. A Figura 6.4 (veja camada de aplicação) ilustra as aplicações implementadas a partir da plataforma SOFT-IOT.

O módulo **soft-iot-data-aggregation** é responsável por agregar dados em FoT-Gateways. O módulo **soft-iot-semantic-enrichment** enriquece os dados do sensor, no FoT-Gateway, com descrições da Web Semântica. O módulo **soft-iot-data-aggregation** fornece agregação de dados a partir de dados semânticos. O módulo **soft-iot-web-service** fornece serviços como a API RESTful para acesso aos dispositivos IoT. O módulo **soft-iot-rules** permite aos usuários a criação de regras no padrão ECA (Event-Condition Action) para posterior execução de ações na borda rede. O módulo **soft-iot-semantic-observer** é responsável por observar alterações feitas em um modelo semântico para posterior tomada de decisão. O módulo **soft-iot-semantic-model** é responsável em obter informações semanticamente descritas por triplas de RDF que estão implantados no FoT-Server (Apache Fuseki). O módulo **soft-iot-semantic-reasoner** é responsável por fazer inferências sobre as regras criadas (a partir do módulo soft-iot-rules) pelo usuário.

Nas próximas seções explicaremos em detalhes essas aplicações e o processo de implantação na SOFT-IOT.

6.3.3.1. Módulo soft-iot-data-aggregation

Módulo da plataforma SOFT-IoT para agregar dados no FoT-Gateway. Esse módulo coleta dados não agregados do banco de dados, aplica funções de agregação e armazena os dados resultantes novamente no banco de dados.

Este módulo possui como pré-requisito a instalação dos módulos soft-iot-gateway-mapping-devices (veja Seção 6.3.2.3) e soft-iot-gateway-local-storage (veja Seção 6.3.2.4).

Para instalar este bundle execute os seguintes comandos no prompt do Karaf:

[15]https://karaf.apache.org/

```
karaf@root()> bundle:install mvn:br.ufba.dcc.wiser.soft_iot/soft-iot--
mapping-devices/1.0.0
karaf@root()> bundle:install mvn:br.ufba.dcc.wiser.soft_iot/soft-iot-lo
cal-storage/1.0.0
karaf@root()> bundle:install -s mvn:br.ufba.dcc.wiser.soft_iot.soft-iot
-data-aggregation/1.0.0
```

O módulo soft-iot-data-aggregation [16] possui um arquivo de configuração, onde é possível definir informações sobre o tempo de execução do procedimento de agregação e configurar a função de agregação para cada sensor.

Finalmente, para a correta execução do módulo você precisa copiar o arquivo:

```
https://github.com/WiserUFBA/soft-iot-data-aggregation/blob/master/src
/main/resources/br.ufba.dcc.wiser.soft_iot.data_aggregation.cfg
```

Para o diretório:

```
<servicemix_directory>/etc
```

6.3.3.2. Módulo soft-iot-semantic-enrichment

O módulo soft-iot-semantic-enrichment[17] possui como objetivo enriquecer os dados do sensor, no FoT-Gateway, com anotações semânticas. Possui como pré-requisito a instalação da biblioteca do Jena 3.1.0. Esta biblioteca não possui uma versão estável do bundle jena-osgi. Por essa razão, precisamos instalá-lo manualmente através da versão compilada no diretório *fot-gateway-semantic-enrichmen/jena-gateway.kar*.

Então, é necessário copiar:

```
https://github.com/WiserUFBA/soft-iot-semantic-enrichment
/tree/master/jena-gateway.kar
```

Para o diretório:

```
<servicemix_directory>/deploy
```

Além disso, este módulo possui também como pré-requisito a instalação dos módulos soft-iot-gateway-mapping-devices (veja Seção 6.3.2.3) e soft-iot-gateway-local-storage (veja Seção 6.3.2.4).

Para instalar este bundle execute os seguintes comandos no prompt do Karaf:

```
karaf@root()>bundle:install mvn:br.ufba.dcc.wiser.soft_iot
/soft-iot-mapping-devices/1.0.0
karaf@root()>bundle:install mvn:br.ufba.dcc.wiser.soft_iot
/soft-iot-local-storage/1.0.0
karaf@root()>bundle:install mvn:br.ufba.dcc.wiser.soft_iot
/soft-iot-semantic-enrichment/1.0.0
```

Finalmente, para a correta execução do módulo você precisa copiar o arquivo:

[16] https://github.com/WiserUFBA/soft-iot-semantic-data-aggregation

[17] https://github.com/WiserUFBA/soft-iot-semantic-enrichment

```
https://github.com/WiserUFBA/soft-iot-semantic-enrichment
/blob/master/src/main/resources/
br.ufba.dcc.wiser.soft_iot.semantic_enrichment.cfg
```

Para o diretório:

```
<servicemix_directory>/etc
```

6.3.3.3. Módulo soft-iot-server-semantic-data-aggregation

O módulo soft-iot-server-semantic-data-aggregation[18] realiza a agregação de dados, onde são coletados dados de um banco de dados semântico. O resultado do procedimento de agregação também é armazenado na base de dados semântico. Além disso, o módulo soft-iot-server-semantic-data-aggregation permite configurar a função de agregação para cada tipo de sensor.

Para instalar este bundle, você precisa instalar previamente estas dependências no karaf:

```
karaf@root()>feature:repo-add mvn:org.ops4j.pax.jdbc/
pax-jdbc-features/0.8.0/xml/features
karaf@root()>feature:install transaction jndi
pax-jdbc-h2 pax-jdbc-pool-dbcp2 pax-jdbc-config
```

Esse módulo necessita da biblioteca do Jena 3.1.0 para funcionar corretamente. Esta biblioteca não possui uma versão estável do bundle jena-osgi. Assim, precisamos instalá-lo manualmente através da versão compilada no diretório soft-iot-server-semântica-data-aggregation/jena-gateway.kar. Então, necessitamos copiar:

```
https://github.com/WiserUFBA/soft-iot-semantic-data-aggregation
/tree/master/jena-gateway.kar
```

Para o diretório:

```
<servicemix_directory>/deploy
```

Depois disso, será necessário criar um arquivo com a configuração do banco de dados. O nome do arquivo é **etc/org.ops4j.datasource-server.cfg** (no diretório servicemix). O conteúdo do arquivo é:

```
osgi.jdbc.driver.name=H2-pool-xa
url=jdbc:h2:${karaf.data}/fot-server-control
dataSourceName=fot-server-control
```

Finalmente, para a correta execução do módulo é necessário copiar o arquivo:

```
https://github.com/WiserUFBA/
soft-iot-semantic-data-aggregation/blob/master/src/main/resources/
br.ufba.dcc.wiser.soft_iot.semantic_data_aggregation.cfg
```

Para o diretório:

```
<servicemix_directory>/etc
```

[18]https://github.com/WiserUFBA/soft-iot-semantic-data-aggregation

6.3.3.4. Módulo soft-iot-web-service

O módulo soft-iot-web-service[19] expõe os dados do sensor do sistema IoT por meio de um serviço web RESTful. Ele acessa os dados armazenados no banco de dados local, gerenciados pelo armazenamento local do módulo soft-iot-local-storage (veja Seção6.3.2.4), permitindo que os usuários obtenham dados e informações no formato JSON sobre os sensores.

Para instalar este bundle, é necessário instalar previamente estas dependências no karaf:

```
karaf@root()>bundle:install mvn:org.codehaus.jackson/jackson-jaxrs/
1.9.2
karaf@root()>bundle:install mvn:org.codehaus.jackson/jackson-core-asl/
1.9.2
karaf@root()>bundle:install mvn:org.codehaus.jackson/jackson-mapper-asl/
1.9.2
```

Este módulo possui também como pré-requisito a instalação dos módulos soft-iot-gateway-mapping-devices (veja Seção 6.3.2.3) e soft-iot-gateway-local-storage (veja Seção 6.3.2.4).

Para instalar este bundle execute os seguintes comandos no prompt do Karaf:

```
karaf@root()>bundle:install mvn:br.ufba.dcc.wiser.soft_iot/
soft-iot-mapping-devices/1.0.0
karaf@root()>bundle:install mvn:br.ufba.dcc.wiser.soft_iot/
soft-iot-local-storage/1.0.0
karaf@root()>bundle:install mvn:br.ufba.dcc.wiser.soft_iot/
soft-iot-iot-service/1.0.0
```

Este módulo cria um Serviço Web RESTful que é possível coletar dados de sensores e informações sobre os dispositivos conectados.

Para obter os dispositivos conectados:

```
http://<servicemix-urls>:<servicemix-port>/cxf
/iot-service/device/{device_id}
```

Para obter os dados do sensor:

```
http://<servicemix-urls>:<servicemix-port>/cxf/iot-service/
device/{device_id}/{sensor_id}
```

Para obter dados do sensor em um intervalo de tempo:

```
http://<servicemix-urls>:<servicemix-port>/cxf/iot-service
/device/{device_id}/{sensor_id}/{start_datetime}/{end_datetime}
```

Para obter mais informações sobre a descrição sintática do serviço web RESTful, acesse:

```
http://<servicemix-urls>:<servicemix-port>/cxf/iot-service?_wadl
```

[19]https://github.com/WiserUFBA/soft-iot-iot-service

6.3.3.5. Módulo soft-iot-rules-editor

O módulo soft-iot-rules-editor[20] é uma aplicação construída em Android e possibilita ao usuário criar regras visualmente no formato EVENT, CONDITION e ACTION (ECA). Na Figura 6.24, o usuário escolhe em uma lista o sensor para o qual deseja criar regras e informa os valores apropriados.

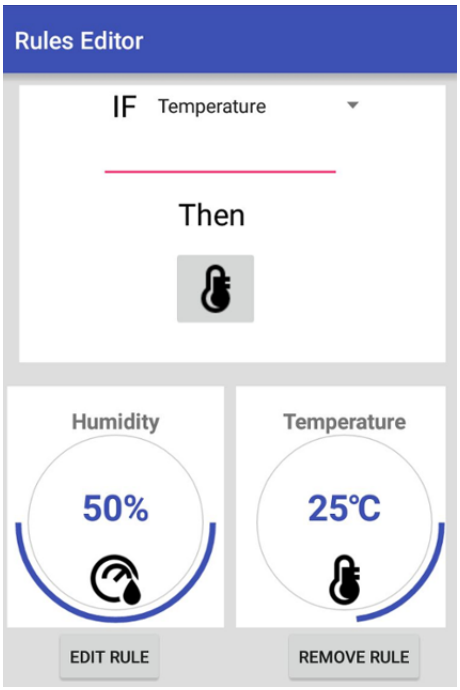


Figure 6.24. Aplicação mobile para criação de regras no formato ECA.

Essa versão do módulo soft-iot-rules-editor é uma versão de teste e está disponibilizada em:[21]. Este módulo possui como pré-requisito a instalação dos módulos soft-iot-semantic-observer (veja Seção 6.3.3.6), soft-iot-semantic-reasoner (veja Seção 6.3.3.7) e soft-iot-rules-execution (veja Seção 6.3.3.8).

6.3.3.6. Módulo soft-iot-semantic-observer

O módulo observador semântico[22] é responsável por observar as alterações feitas em um modelo semântico. Para fazer isso, o módulo soft-iot-semantic-observer usa consultas SPARQL, como mostra a Listagem 6.2. Conforme mostrado na Listagem 6.3, na linha 10, inicialmente, a propriedade isSettingFor é **false** e, após executar uma regra, essa propriedade pode ser alterada para **true** pelo módulo soft-iot-semantic-reasoner. Então, o módulo Semantic Observer, ao identificar mudanças feitas na propriedade isSettingFor, notifica o módulo de Execução de Regras (Seção 6.3.3.8), que é responsável por informar ao atuador e ativar o dispositivo (por exemplo, ativar um ar condicionado,ligar uma luz).

[20]https://github.com/WiserUFBA/Rules-Editor
[21]https://github.com/WiserUFBA/Rules-Editor/blob/master/Rules-Editor.apk
[22]https://github.com/WiserUFBA/Semantic-Observer

Código 6.2. Consulta SPARQL para observar mudanças realizadas no modelo semântico.

```
PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX dul: <http://www.loa-cnr.it/.../DUL.owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?hasDataValue
WHERE { <http://wiser.dcc.ufba.br/smartUFBA/...>
#obsValue_14915308050001491530865086>
ssn: ObservationValue ;
dul: hasDataValue '37'^^xsd:double ;
dul: isSettingFor true . };
```

Código 6.3. Trecho das triplas RDF armazenadas no servidor Fuseki.

```
<http://wiser.dcc.ufba.br/smartUFBA/devices/temp01>
dul: hasIntervalDate "2017-04-03T20:14:11.054Z" .
<http://wiser.dcc.ufba.br/smartUFBA/devices/...>
a dul: TimeInterval ;
dul: hasIntervalDate "2017-04-03T20:16:11.207Z" .
 <http://wiser.dcc.ufba.br/smartUFBA/devices/...>
a ssn: ObservationValue ;
dul: hasDataValue "29"^^xsd:double .
dul: isSettingFor false.
```

A instalação desse bundle pode ser realizada a partir do console do karaf ou do web console do servicemix conforme ilustrado na Figura 6.22.

6.3.3.7. Módulo soft-iot-semantic-reasoner

O módulo soft-iot-semantic-reasoner[23] é responsável por fazer inferências, a partir do *framework* Jena, sobre as regras criadas pelo usuário.

Quando o módulo soft-iot-semantic-reasoner executa uma regra e infere que uma condição foi satisfeita, ela executa uma atualização no modelo. Esta atualização é executada apenas nos triplas de RDF que atendem à regra usando o SPARQL UPDATE de acordo com a Listagem 6.4. Na linha 4, as informações contidas no modelo são excluídas e, na linha 5, a mesma propriedade é inserida com um novo valor.Na linha 6, uma restrição é aplicada para identificar corretamente o valor a ser alterado.

A instalação desse bundle pode ser realizada a partir do console do karaf ou do web console do service mix conforme ilustrado na Figura 6.22.

Código 6.4. Atualização SPARQL para atualizar o modelo semântico.

```
PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX dul:
<http://www.loa-cnr.it/ontologies/DUL.owl#>
PREFIX xsd:
<http://www.w3.org/2001/XMLSchema#>
DELETE { <%s> dul:isSettingFor false .}
INSERT { <%s> dul:isSettingFor true .}
WHERE { <%s> dul:isSettingFor false .}
```

[23]https://github.com/WiserUFBA/Rules-Semantic-for-IoT

6.3.3.8. Módulo soft-iot-rules-execution

O módulo soft-iot-rules-execution[24] executa os serviços Web RESTful para ativar os atuadores (por exemplo, condicionador de ar, janela, alarme, semáforo, etc.) enviando mensagens por meio do MQTT protocolo.

Para ativar um dispositivo:

```
http://<servicemix-urls>:<servicemix-port>/cxf/lamp/devices/actuator/lamp"
```

6.3.4. Camada de Segurança

Ao mesmo tempo que a IoT possibilita uma gama de novas aplicações, ela apresenta desafios em diferentes níveis de sua infra-estrutura. Os sistemas de IoT integram objetos físicos, dados de sensores e recursos de computação em redes amplas através da Internet. Particularmente, o grande volume de dados gerados e a exposição a que esses dados estão sujeitos ocasionam diferentes problemas de segurança e privacidade.

Nesse cenário, possíveis vulnerabilidades de segurança e violações de privacidade precisam ser resolvidas com base na confiança entre as partes envolvidas e pelo uso de mecanismos adequados. Dessa forma, os desenvolvedores podem empregar tais mecanismos para criar soluções escaláveis, distribuídas e confiáveis.

As arquiteturas de referência apresentadas em [Bassi et al. 2013] e [Pöhls et al. 2014] discutem os possíveis problemas de segurança e definem modelos de confiança, segurança e privacidade para sistemas IoT. Atualmente, os serviços de segurança IoT, como autenticação, controle de acesso e gerenciamento de identidade estão apoiadas em uma arquitetura cliente-servidor [Zhu and Badr 2018]. Eles se baseiam na suposição de que os usuários devem depositar toda a confiança em entidades terceiras confiáveis (provedores de identidade) que possuem dados pessoais de usuários e podem ver todas as transações entre os usuários e os provedores de serviços.

Neste sentido, Blockchain é uma tecnologia emergente que oferece suporte distribuído confiável e seguro para realização de transações entre membros que não necessariamente têm confiança entre si e que estão dispersos em larga escala numa rede P2P. É considerada uma tecnologia disruptiva, pois cria digitalmente uma entidade de confiança descentralizada, eliminando a necessidade de uma terceira parte de confiança. Os recentes avanços nessa tecnologia criam perspectivas para desenvolvimento de sistemas robustos e seguros em diversos escopos da computação, particularmente em IoT e Névoa.

Em [Greve et al. 2018], Blockchain é definida como o resultado de uma engenhosa combinação de técnicas robustas provenientes da computação distribuída confiável (tolerância a falhas bzyantinas, sistemas P2P), criptografia (chave assimétrica, funções hash, desafios criptográficos) e teoria dos jogos (mecanismos de incentivos). Agrega elementos eficazes para a implementação de um sistema de acordo em escala global, trazendo respostas para importantes desafios computacionais, dentre os principais: (i) realização de consenso numa rede aberta, com participantes desconhecidos, em escala planetária; (ii) tolerância a falhas byzantinas, com participantes anônimos; (iii) resistência ao ataque de duplo-gasto (*double-spending*), garantindo que ativos transacionados (como moedas

[24]https://github.com/WiserUFBA/Rules-Execution

digitais) não sejam gastos duplamente, para além do seu valor em posse; (iv) resistência a ataques Sybil, garantindo que usuários maliciosos não poderão se personificar em outros (a menos que tenham acesso a sua chave privada); (v) garantia da auditabilidade, autenticidade, não-repúdio e integridade em escala global de todas as transações validadas e armazenadas no livro-razão (*ledger*) distribuído.

O projeto SOFT-IoT vem investigado estratégias de confiança, segurança e privacidade que exploram a consistência eventual e as garantias oferecidas pelas tecnologias de livro-razão distribuído. Por exemplo, sem um gerenciamento de identidade apropriado para IoT, outros serviços de segurança voltados à computação em névoa não podem ser construídos corretamente. Os avanços na aplicação da tecnologia Blockchain em IoT podem revelar uma possível solução para garantir o gerenciamento de confiança em sistemas descentralizados [Abadi et al. 2018]. Pesquisas atuais vem fomentando soluções de gerenciamento de confiança baseadas em Blockchain com foco em sistemas de gerenciamento de identidade (IDMS) para a IoT [Zhu and Badr 2018]. No entanto, a integração da IoT com Blockchain é recente e carece de uma maior investigação.

A Figura 6.4 apresenta a integração da arquitetura SOFT-IoT com a Blockchain através de uma camada computacional para compartilhamento e análise segura dos dados, com garantias de privacidade. Essa camada pode ser utilizada para autenticar, autorizar, controlar e auditar os dados gerados pelos objetos inteligentes [Christidis and Devetsikiotis 2016]. Sua disposição vertical evidencia a possibilidade do provimento e integração de serviços baseados em Blockchain em diferentes níveis da arquitetura SOFT-IoT.

6.4. Tópicos de pesquisas relacionados a SOFT-IoT

Nesta seção, alguns trabalhos de pesquisa relacionados à arquitetura SOFT-IoT são apresentados. O objetivo é oferecer uma visão geral do desenvolvimento da plataforma visando uma aprendizagem dos conceitos e inspiração para novos projetos.

6.4.1. Microserviços Reativos

Os Microsserviços permitem a criação de um sistema a partir de uma coleção de serviços pequenos e isolados, capazes de gerenciar seus próprios dados [Lewis and Fowler 2014]. O REST, que produz comunicação síncrona entre os serviços, é frequentemente usado no desenvolvimento de Microsserviços, entretanto, a comunicação entre Microsserviços deve se basear no uso de mensagens assíncronas. Isso é necessário para estabelecer um limite assíncrono entre cada serviço, com o objetivo de desacoplar o fluxo de comunicação de serviços das perspectivas de tempo e espaço. O fluxo de comunicação desacoplado no tempo permite a simultaneidade, enquanto o desacoplamento no espaço permite a mobilidade e a distribuição que são indispensáveis para o desenvolvimento de aplicativos em cenários de IoT. Operações assíncronas e sem bloqueio reduzem o congestionamento de recursos no sistema e produzem escalabilidade e baixa latência. Além disso, a comunicação baseada em mensagens assíncronas produz sistemas com duas características:

- **Resiliência**, que está associada à capacidade do sistema de lidar com falhas;
- **Elasticidade**, que está associada à capacidade do sistema de dimensionar horizontalmente. Sistemas que têm seu design construído a partir dessas perspectivas são considerados reativos.

Este tópico estende a SOFT-IOT e propõe uma abordagem para modelar o design e a implementação de Microsserviços reativos em cenários IoT.

Para esse fim, definimos um modelo arquitetural para o desenvolvimento de Microserviços e criamos uma nova plataforma baseada no Vert.x[25]. Vert.x é um toolkit que permite a criação de aplicações reativas na JVM. Outra característica é a possibilidade de trabalhar com o Vert.x em diferentes linguagem de programação como Java, JavaScript, Groovy, Ruby, Ceylon, Scala and Kotlin.

6.4.2. Análise de Data Stream utilizando a SOFT-IoT

Os ambientes que são potencializados pela Internet das Coisas devem produzir uma enorme quantidade de dados, criando assim, uma nova área de atuação dentro do ecossistema da IoT que é denominada *IoT Analytics*. Essa área é caracterizada pela análise de dados de diversas fontes de dados IoT, que podem ser sensores, atuadores, dispositivos inteligentes e outros objetos conectados à Internet, sendo considerada como uma das principais partes que pode realizar todo potencial de mercado da IoT [Soldatos 2016].

Os dados de várias fontes e domínios produzidos pela IoT são em alguns casos fluxos de dados, como por exemplo, dados numéricos de diferentes sensores ou entradas de texto de mídias sociais. Os fluxos de dados comuns geralmente seguem a distribuição Gaussiana durante um longo período. Porém, os dados IoT são produzidos em curto espaço de tempo e em grandes quantidades, apresentando uma variedade de distribuições esporádicas ao longo do tempo. Além disso, em alguns casos em tempo real ou próximo do tempo real [Puschmann et al. 2017].

Uma tendência das aplicações para Internet das Coisas que tratam do conceito de *IoT Analytics* é a utilização da Computação em Névoa que pode descentralizar o processamento de fluxos de dados IoT e, apenas realizar a transferência de dados IoT filtrados dos dispositivos da borda da rede para a nuvem [Andrade et al. 2017]. Diante disso, a associação da análise de fluxos dados com a Computação em Névoa permite que as empresas explorem em tempo real os dados produzidos pela IoT e, com isso produzam valor de negócio. Essas análises na borda da rede podem resolver o problema das aplicações que necessitam de resposta em tempo real, como por exemplo, os sistemas que monitoram saúde de pacientes e devem tomar decisões em curto espaço de tempo. Outros tipos de análises podem descobrir preferências dos clientes, padrões ocultos nos dados, entre outras informações que podem ajudar as organizações a tomarem decisões mais fundamentadas e em curto espaço de tempo.

Diante disso, um módulo chamado **FoT-Gateway-StreamAnalytics-Soft_Iot**[26] foi desenvolvido para realizar mineração de fluxo de dados IoT na *Fog Computing* utilizando como base a plataforma SOFT-IoT. Uma arquitetura para análise de *data stream* com a SOFT-IoT tem como objetivo utilizar toda capacidade computacional dos dispositivos empregados na *Fog of Things* para o processamento e análise de dados, sem necessidade de utilização constante da *Cloud Computing*. Diante disso, esta proposta pode minimizar o volume de dados que precisam ser encaminhados para a nuvem, o que pode ter um grande impacto nos tempos de resposta e nos custos de transmissão dos dados na

[25]https://vertx.io/

[26]https://github.com/WiserUFBA/FoT-Gateway-StreamAnalytics-Soft_Iot

rede, além disso, permite análises de dados em tempo real na *Fog Computing*.

Para a instalação do FoT-Gateway-StreamAnalytics-Soft_Iot é necessário ter configurado o módulo soft-iot-mapping-devices como apresentado na Seção 6.3.2.3. A instalação do FoT-Gateway-StreamAnalytics-Soft_Iot pode ser feita executando os seguintes comandos no terminal do ServiceMix:

```
karaf@root()> bundle:install mvn:br.ufba.dcc.wiser.soft_iot.analytics/
fot-gateway-streamanalytics-soft_iot/1.0.0
```

Logo após a instalação do FoT-Gateway-StreamAnalytics-Soft_Iot é necessário gerar um arquivo de configuração (seguindo o padrão da seção 6.3.2.3) que irá conter a configuração dos seguintes parâmetros do módulo: dispositivos que devem ser conectados no módulo; ativação do modo de depuração; endereço e configuração do *broker* MQTT; tempo de publicação e coleta de dados dos sensores. Um modelo do arquivo pode ser encontrado em:

```
FoT-Gateway-StreamAnalytics-Soft_Iot/src/main/resources/br/ufba/dcc/
wiser/soft_iot/analytics/
br.ufba.dcc.wiser.soft_iot.gateway.stream_analytics.cfg
```

O arquivo de configuração deve ser implantado no sub-diretório do ServiceMix:

```
<diretorio_servicemix>/etc
```

6.4.3. Ambientes de Testes para Computação em Névoa

É esperado que os ambientes de névoa suportem milhões de dispositivos IoT e de usuários finais [Coutinho et al. 2016]. Eles podem envolver um grande número de aplicativos, domínios de redes e nós de névoa. Além disso, suas diferentes aplicações podem ter uma grande quantidade de componentes que usam tanto recursos de nuvem quando de névoa para fornecer soluções inteligentes e avançadas.

Um problema atual nesta área é a falta de ambientes de suporte para prototipagem e teste de serviços, componentes e aplicações em larga escala [Mouradian et al. 2017]. Neste momento, simuladores de rede e *middleware* de nuvem são adaptados para permitir uma avaliação experimental de soluções de névoa em ambientes limitados, cenários específicos e condições restritivas. No entanto, testar e validar uma arquitetura com poucos dispositivos e nós não garante que ela seja propriamente executada sobre um ambiente em larga escala, no que diz respeito à qualidade e desempenho da solução desenvolvida.

Apoiado pelo projeto SOFT-IoT, o *framework* Fogbed [Coutinho et al. 2018a] foi desenvolvido para permitir a prototipagem rápida e o teste escalável de componentes e aplicativos de névoa. Seu projeto é compatível com tecnologias do mundo real e atende aos requisitos de configuração flexível e de baixo custo, suportando a integração de sistemas de terceiros por meio da implementação de interfaces padronizadas.

O Fogbed é baseado em tecnologias que são amplamente empregadas por desenvolvedores de software livre e pesquisadores como contêineres Docker[27]. e o emulador de rede Mininet [de Oliveira et al. 2014]. Usando uma abordagem *desktop*, o Fogbed estende o *framework* Mininet para criar ambientes de experimentação (*testbeds*) de névoa

[27]https://www.docker.com

virtualizados. Ele permite a implementação de nós de névoa como contêineres Docker sob diferentes configurações de rede. A API do Fogbed fornece funcionalidades para adicionar, conectar e remover contêineres dinamicamente da topologia da rede.

Essas características possibilitam a emulação da infraestrutura de nuvem e névoa, na qual é possível iniciar e interromper instâncias de computação em qualquer momento. Além disso, é possível alterar as limitações de recursos em tempo de execução para um contêiner, como o tempo da CPU e memória disponível. Utilizando a API de emulação Maxinet [Wette et al. 2014], esses contêineres podem ser executados em uma ou mais (*cluster*) máquinas hospedeiras (*hosts*) através de uma rede local [Coutinho et al. 2018b].

6.4.3.1. Instalação do Ambiente Fogbed

Na versão atual, existem duas opções para instalação do Fogbed em uma máquina hospedeira: (*i*) pode ser instalado nativamente (*bare metal*) sobre o sistema operacional; ou (*ii*) pode ser executado como um contêiner Docker privilegiado (*privileged Docker container*). Em ambas as opções é requerido no mínimo o Linux Ubuntu versão 16.04 LTS.

Na opção (*i*), uma instalação automática é fornecida através de um *playbook* da ferramenta Ansible. A seguinte sequência de comandos pode ser aplicada em um terminal do sistema operacional Linux hospedeiro, onde a senha do usuário administrador pode ser solicitada para permitir a execução do comando *sudo*:

```
$ sudo apt-get install ansible git aptitude
$ git clone https://github.com/fogbed/fogbed.git
$ cd fogbed/ansible
$ sudo ansible-playbook -i "localhost," -c local install_metis.yml
$ sudo ansible-playbook -i "localhost," -c local install_docker.yml
$ sudo ansible-playbook -i "localhost," -c local --skip-tags \
"notindocker" install_fogbed.yml
```

A menos que se esteja tentando obter o melhor desempenho de emulação, é recomendado realizar a instalação em uma máquina virtual (*Virtual Machine* - VM) ao invés de uma máquina física. Este cuidado deve-se ao fato do processo de instalação alterar as configurações do sistema, podendo afetar seu funcionamento caso seja mal sucedido.

Na opção (*ii*), o ambiente Fogbed é executado em um contêiner Docker privilegiado. Como qualquer aplicação baseada em Linux, o Fogbed também pode ser conteinerizado como uma imagem Docker. Essas imagens do Fogbed podem então ser replicadas, distribuídas e instanciadas em diferentes máquinas hospedeiras. Uma imagem Docker do Fogbed pré-compilada está disponível pela Internet através da biblioteca pública de imagens Docker Hub. O seguinte comando pode então ser aplicado em um terminal do sistema Linux hospedeiro para construção de uma imagem do Fogbed localmente:

```
$ docker build -t fogbed .
```

Também é possível utilizar a imagem construída mais recentemente:

```
$ docker pull fogbed/fogbed
```

Por fim, o seguinte comando é usado para instanciar um contêiner do Fogbed:

```
$ docker run --name fogbed -it --rm --privileged --pid='host' -v \
/var/run/docker.sock:/var/run/docker.sock fogbed /bin/bash
```

6.4.3.2. Configurando o Ambiente de Névoa

Uma emulação no *Fogbed* é configurada através da definição de instâncias virtuais, nós virtuais, comutadores virtuais e conexões virtuais. Estas definições são empregadas para a criação de um ambiente de névoa virtual executado em uma máquina hospedeira. A configuração flexível do ambiente é alcançada através do uso de imagens de contêiner *Docker*. Cada nó virtual é instanciado a partir de uma imagem de contêiner pré-configurada e que compreende parte de uma aplicação distribuída, bem como seus serviços e protocolos. Diferentes tipos de imagens de contêiner podem ser usados para instanciar nós virtuais. Perceba que se o ambiente Fogbed estiver sendo executado como um contêiner privilegiado, os nós virtuais estarão sendo executados como contêineres dentro de contêineres ou contêineres aninhados (*nested container deployment*).

Antes de iniciar uma emulação, é necessário criar um *script* que define a topologia da névoa e seus elementos virtuais. Um *script* de topologia pré-definido é fornecido para demostrar a configuração básica do ambiente. Assim, após a instalação do Fogbed em uma máquina virtual ou contêiner Docker, é possível iniciar uma emulação através do seguinte comando aplicado em um terminal da máquina convidada (*guest*):

```
$ python examples/virtual_instance_example.py
```

No *script* de topologia, a definição do ambiente é realizada usando a linguagem Python. Primeiramente, todos os elementos virtuais são descritos e, após sua inicialização, um experimento predefinido pode ser executado sobre o ambiente. Se os procedimentos de instalação do Fogbed foram realizados corretamente, a execução do *script* deve instanciar a topologia em névoa nele definida e executar alguns comandos básicos de teste (*ifconfig* e *ping*) em nós virtuais desta topologia. No exemplo, verificando o conteúdo do *script* de topologia, encontramos em suas primeiras linhas as seguintes declarações:

```
topo = FogTopo()
c1 = topo.addVirtualInstance("cloud")
f1 = topo.addVirtualInstance("fog")
e1 = topo.addVirtualInstance("edge")
```

Nas primeiras declarações temos o instanciamento de uma topologia em névoa seguida pela definição de três instâncias virtuais. Uma instância virtual é uma abstração do Fogbed que permite o gerenciamento de um conjunto de nós virtuais e comutadores virtuais relacionados como uma única entidade. Por exemplo, uma instância virtual pode ser formada por um ou mais nós virtuais conectados por um único comutador virtual, onde durante a emulação novos nós virtuais podem ser alocados em uma instância virtual.

A seguir, nas próximas linhas do *script* de topologia, são definidos três diferentes modelos de recursos. Cada modelo é atribuído para uma das instâncias virtuais:

```
erm = EdgeResourceModel(max_cu=20, max_mu=2048)
frm = FogResourceModel()
crm = CloudResourceModel()
e1.assignResourceModel(erm)
f1.assignResourceModel(frm)
c1.assignResourceModel(crm)
```

Cada instância virtual segue o modelo de recursos associado a ela, que define quantos recursos essa instância possui para distribuir entre os seus nós virtuais (contêineres) e qual a estratégia de gerenciamento desses recursos. Cada modelo de recurso permite definir um valor *max_cu* e *max_mu*, que representam as unidades máximas de processamento e de memória atribuídas à instância virtual. Assim, é possível determinar quantas unidades de processamento *cu* e de memória *mu* os contêineres podem consumir durante a emulação, calculando seus valores como frações proporcionais ao total de recursos disponíveis na instância virtual onde os contêineres estão sendo executados. Esses valores são convertidos em tempo real de CPU e limite disponível de memória. Por exemplo, se para um contêiner *a* for atribuído quatro unidades de processamento, e para um contêiner *b* duas unidades de processamento, e se ambos estiverem na mesma instância virtual, o contêiner *a* terá duas vezes mais tempo de CPU do que o contêiner *b*.

Existem três tipos de modelos de recursos pré-definidos no Fogbed: *EdgeResourceModel*, *FogResourceModel* e *CloudResourceModel*, onde os valores *defaults* de *max_cu* e *max_mu* são respectivamente 32 e 2048, respectivamente. A estratégia de gerenciamento utilizada em*EdgeResourceModel* possui um limite fixo em que, se um contêiner solicita recursos mas toda a capacidade da instância virtual já foi alocada, uma exceção é gerada alertando que a instancia virtual não pode alocar novos contêineres. Em *FogResourceModel* e *CloudResourceModel*, é utilizada uma estratégia de super provisionamento (*overprovisioning*) semelhante ao modelo empregado em nuvem onde, se um contêiner solicita recursos mas toda a capacidade da instância virtual já foi alocada, o novo contêiner é iniciado de qualquer maneira. Entretanto, em *FogResourceModel* o valor de *max_cu* e *max_mu* continuam os mesmos após o instanciamento do novo contêiner, e o limite de tempo de CPU e de memória de cada contêiner é recalculado.

O trecho seguinte do *script* de topologia é semelhante a exemplos de outros emuladores baseados em Mininet e define os outros elementos virtuais da névoa:

```
d1 = c1.addDocker('d1', ip='10.0.0.251', dimage="ubuntu:trusty")
d2 = f1.addDocker('d2', ip='10.0.0.252', dimage="ubuntu:trusty")
d3 = e1.addDocker('d3', ip='10.0.0.253', dimage="ubuntu:trusty")
d4 = topo.addDocker('d4', ip='10.0.0.254', dimage="ubuntu:trusty")
d5 = e1.addDocker('d5', ip='10.0.0.255', dimage="ubuntu:trusty",
    resources=PREDEFINED_RESOURCES['medium'])
d6 = e1.addDocker('d6', ip='10.0.0.256', dimage="ubuntu:trusty",
    resources=PREDEFINED_RESOURCES['large'])
```

Como mostrado acima, cada novo nó virtual é iniciado passando como parâmetro suas configurações (identificador, endereço IP, e a imagem de contêiner Docker utilizada), onde cada um, exceto o nó d4, é iniciado dentro de uma instância virtual. O parâmetro de recursos em d5 e d6 descreve a quantidade de recursos da instância virtual que o contêiner deve receber. Se não for especificado, o recurso predefinido como *small* é definido como *default*. No seguinte trecho do *script* de topologia é encontrada a lista predefinida de recursos:

```
PREDEFINED_RESOURCES = {
    "tiny": {"cu": 0.5, "mu": 32},
    "small": {"cu": 1, "mu": 128},
    "medium": {"cu": 4, "mu": 256},
    "large": {"cu": 8, "mu": 512},
```

```
    "xlarge": {"cu": 16, "mu": 1024},
    "xxlarge": {"cu": 32, "mu": 2048}
}
```

Se nenhum dos valores predefinidos forem adequados para um determinado nó virtual, é possível definir uma nova entrada para o contêiner específico. Após a criação de todos os nós virtuais, dois comutadores virtuais são instanciados, seguidos pelas definições de suas conexões virtuais usando os parâmetros de classe (*cls*), atraso (*delay*) e largura de banda (*bw*) entre os nós virtuais, as instâncias virtuais e os comutadores virtuais:

```
s1 = topo.addSwitch('s1')
s2 = topo.addSwitch('s2')
topo.addLink(d4, s1)
topo.addLink(s1, s2)
topo.addLink(s2, e1)
topo.addLink(c1, f1, cls=TCLink, delay='200ms', bw=1)
topo.addLink(f1, e1, cls=TCLink, delay='350ms', bw=2)
```

6.4.3.3. Executando Experimentos no Fogbed

Uma aplicação de névoa e seus serviços são executados em um ou mais nós virtuais dentro de uma instância virtual. Na arquitetura do Fogbed, existem maneiras diferentes de interagir com o ambiente durante um experimento. A primeira maneira é definindo procedimentos no próprio *script* de topologia. Esses procedimentos serão executados após a inicialização do ambiente de névoa em:

```
exp = FogbedExperiment(topo, switch=OVSSwitch)
exp.start()
```

Onde os procedimentos determinam os passos que o experimento deve realizar:

```
try:
    print exp.get_node("cloud.d1").cmd("ifconfig")
    print exp.get_node(d2).cmd("ifconfig")
    print "aguarde 5 segundos para os algoritmos de roteamento do
           controlador convergirem"
    time.sleep(5)
    print exp.get_node(d1).cmd("ping -c 5 10.0.0.252")
    print exp.get_node("fog.d2").cmd("ping -c 5 10.0.0.251")
finally:
    exp.stop()
```

No exemplo, estamos executando o comando *ifconfig* no nó virtual $d1$ que está sendo executado na instância virtual *cloud*. Em seguida, é executado o mesmo comando dentro do nó virtual $d2$, desta vez usando uma variável de referência em vez passar como parâmetro uma *string* com o identificador do nó virtual. Após 5 segundos de espera, o experimento executa o comando *ping* de $d1$ para $d2$ e vice-versa.

Além de permitir procedimentos no próprio *script* de topologia, as imagens de contêineres instanciados como nós virtuais podem incluir *scripts* pré-configurados de forma que, após a sua inicialização, executem serviços, aplicações e procedimentos sobre o ambiente de emulação. Para facilitar o gerenciamento, uma interface de linha de

comando (*Command Line Interface* - CLI) interativa no Fogbed permite que os desenvolvedores interajam com os nós emulados, alterem configurações, visualizem arquivos de log ou executem comandos arbitrários enquanto a plataforma Fogbed executa seus aplicativos e os seus serviços.

Outra forma de interagir com um experimento é implementando um ou mais processos responsáveis pela inicialização e gerenciamento das instâncias virtuais no ambiente de emulação *Fogbed*. O sistema de gerenciamento precisa interagir com o ambiente emulado para controlar os nós virtuais em instâncias virtuais. Neste caso, a comunicação entre uma instância virtual e o sistema de gerenciamento é realizada através de uma API de instância padronizada e extensível. A API de instância fornece uma semântica de infraestrutura como serviço (IaaS) para gerenciar nós virtuais de maneira adaptável. Ela é projetada como uma interface abstrata para permitir a integração e o teste de sistemas de terceiros. Um desenvolvedor pode implementar sua própria interface de gerenciamento sobre uma API de instância virtual. Uma vez implementada, a interação do sistema de gerenciamento com a instância virtual acontece através de uma porta de comunicação definida no *script* de topologia. O exemplo a seguir adiciona uma API de instância específica a cada tipo de instância virtual:

```
api1 = EdgeApi(port=8001)
api1.connectInstance(e1)
api1.start()
api2 = FogApi(port=8002)
api2.connectInstance(f1)
api2.start()
api3 = CloudApi(port=8003)
api3.connectInstance(c1)
api3.start()
```

A implementação de diferentes APIs de instância permite que cada instância virtual use um processo ou sistema diferente de gerenciamento. Com essa concepção flexível, é possível a execução de diferentes estratégias de gerenciamento para cada instância virtual no ambiente emulado.

6.4.3.4. Emulação Distribuída

O exemplo anterior explorou uma emulação em uma única máquina hospedeira. O Fogbed pode ser adaptado, usando uma interface como a fornecida pelo Maxinet, para executar uma topologia distribuída sobre diferentes máquinas em rede local. O MaxiNet é executado em um conjunto de máquinas físicas chamadas trabalhadoras *(*workers). Cada um desses *workers* executa o Fogbed e apenas emula uma parte de toda a rede virtual. Os nós virtuais e comutadores virtuais em diferentes *workers* são interconectados usando túneis GRE. O MaxiNet fornece uma API centralizada para controlar a emulação, onde esta API é invocada em um *worker* especializado chamado de *frontend*. O *frontend* particiona e distribui a rede virtual para os *workers* e mantém uma lista de qual nó virtual reside em qual *worker*. Desta forma, podemos acessar todos os nós através do *frontend*.

Porém, antes de executar a configuração distribuída, é preciso verificar: (*i*) se todas as máquinas hospedeiras que vão executar o ambiente distribuído possuem o Fogbed instalado; e (*ii*) se as máquinas hospedeiras conseguem se comunicar através da rede local.

Depois é necessário iniciar o controlador POX em uma das máquinas presentes na rede. O controlador POX é um aplicativo em redes definidas por software (*Software-Defined Networking* - SDN) que gerencia o controle de fluxo. Ele é instalado na subpasta *pox* no diretório do Fogbed. No caso da execução do Fogbed como um contêiner Docker privilegiado, o caminho e os comandos para executar o controlador POX são:

```
$ cd /pox
$ ./pox.py forwarding.l2_learning
```

Após executar o controlador SDN em uma maquina na rede, é necessário configurar o Maxinet. Uma máquina da rede local será o *frontend* da emulação distribuída, e as outras máquinas serão os *workers* da rede. No *frontend* da rede copie o conteúdo do arquivo */usr/local/share/maxinet/config.example* para o diretório */etc/MaxiNet.cfg*. No arquivo copiado, substitua o endereço IP do atributo *controlador* pelo endereço IP da máquina que está executando o POX:

```
controller = 172.17.0.2:6633
```

No atributo *ip* do *[FrontendServer]* preencha o endereço da máquina que você pretende executar o servidor *frontend*:

```
[FrontendServer]
ip = 172.17.0.2
```

Abaixo da configuração do *[FrontendServer]*, insira o nome e o endereço IP de cada máquina *worker* da rede. No exemplo a seguir são configuradas duas máquinas:

```
[worker1-hostname]
ip = 172.17.0.2
share = 1
[worker2-hostname]
ip = 172.17.0.3
share = 1
```

Para verificar o nome e o IP de uma máquina, basta executar no terminal:

```
# print hostname
$ hostname

[worker1-hostname]
# print ip
```

Na máquina definida como o *frontend*, execute o comando:

```
$ sudo FogbedFrontendServer
```

E em cada máquina *worker* da rede, execute o comando:

```
$ sudo FogbedWorker
```

É possível verificar o status do cluster de emulação com o comando:

```
$ FogbedStatus
MaxiNet Frontend server running at 172.17.0.2
Number of connected workers: 2
--------------------------------
worker1-hostname            free
worker2-hostname            free
```

Em seguida, é necessário ajustar o *script* de topologia da seção anterior para permitir a execução de maneira distribuída. Para isso, é apenas necessário mudar a classe do experimento no *script* de topologia para *FogbedDistributedExperiment* e salvar o arquivo:

```
exp = FogbedDistributedExperiment(topo, switch=OVSSwitch)
```

Por fim, execute o *script* de topologia na máquina *frontend*:

```
$ python examples/virtual_instance_example.py
```

Após a execução, A partir de uma CLI executada na máquina *frontend* é possível verificar quais nós e comutadores são emulados em qual máquina física. A CLI interativa do *frontend* também ajuda na depuração dos experimentos, sendo possível executar comandos arbitrários e de forma centralizada em qualquer um dos nós virtuais emulados.

6.4.3.5. Coleta de Dados

Assim como em ambientes de névoa IoT do mundo real, a aplicação na coleta de dados no Fogbed depende do experimento executado. Porém, um esquema geral para observar o comportamento do ambiente pode ser implementado através do monitoramento dos fluxos de dados nas interfaces dos nós virtuais e comutadores virtuais. Por exemplo, é possível empregar o NetFlow[B. Claise, Ed. 2004] como tecnologia de monitoramento de fluxo e o NFDUMP[28] como a ferramenta de coleta e análise. Os registros de fluxo são enviados para um *daemon* de captura de fluxo *nfcapd* especificado usando o comando *fprobe* em nós virtuais e o comando *ovs-vsctl* em comutadores virtuais. A ferramenta de análise de fluxo *nfdump* pode ser então usada para estudar o tráfego coletado em cada interface virtual. Neste esquema, os contêineres podem ser configurados para executar as *daemon* de captura e monitoramento durante a sua inicialização.

Em uma emulação distribuída, as funcionalidades do MaxiNet registram e avaliam automaticamente o uso dos recursos físicos ao longo do experimento. Para isso, o MaxiNet monitora a utilização da CPU, o consumo de memória e o uso da rede de todas as máquinas físicas. Após o término de um experimento, esses dados podem ser avaliados para garantir que nenhum recurso físico tenha sido sobrecarregado durante o experimento.

6.4.3.6. Emulação da Plataforma SOFT-IoT

Neste capítulo, abordamos aspectos básicos de como instalar e configurar o *framework* Fogbed para uma emulação em névoa local e distribuída. No entanto, nos exemplos práticos não foi instanciada nenhuma arquitetura IoT, mas apenas um ambiente ou infraestrutura virtual onde essas arquiteturas podem ser projetadas, executadas e testadas. Embora este *framework* tenha como motivação a plataforma SOFT-IoT, seu projeto foi elaborado de forma aberta, genérica e extensível, podendo ser adaptado para testes de outras arquiteturas de névoa e IoT. Como suplemento a este material, em [Fogbed 2018] é apresentado um estudo de caso utilizando imagens pré-configuradas para demonstrar a emulação de componentes e serviços da plataforma SOFT-IoT. Mais informações sobre o Fogbed e seu código fonte podem ser encontrados na página do projeto no GitHub[29].

[28] https://github.com/phaag/nfdump

[29] https://github.com/fogbed/fogbed

6.5. Considerações Finais

Este capítulo apresentou a plataforma SOFT-IoT, que é uma implementação do modelo *Fog of Things*. A SOFT-IoT é uma plataforma de IoT distribuída que usa a infraestrutura do *Fog of Things* para implantar módulos em FoT-Gateways, FoT-Servers e servidores em nuvem. Além disso, o SOFT-IoT é uma plataforma de IoT agnóstica e que fornece uma infraestrutura flexível e configurável.

A SOFT-IoT busca desenvolver avanços tecnológicos que permitirão promover e viabilizar o paradigma *Fog of Things*. Muitas questões desafiadoras ainda precisam ser abordadas para uma ampla utilização desse paradigma, como visto nos tópicos de pesquisa delineados neste capítulo. Esses desafios vão desde soluções que permitam a interoperabilidade e a integração dos diversos componentes e serviços que compõem os ambientes de IoT, passando pelo processamento de uma grande quantidade de dados, à escalabilidade por conta do grande número de objetos (coisas) envolvidos e pela facilitação no desenvolvimento e no teste de aplicações para esses ambientes.

Nesse contexto, as plataformas de *middleware* para IoT precisam considerar diferentes questões e satisfazer a um conjunto de requisitos a fim de cumprir as demandas dos desafios apresentados. Pesquisas adicionais sobre arquiteturas IoT mais escaláveis e que tirem proveito das características do modelo FoT para tratar aspectos como interoperabilidade, descoberta e gerenciamento de dispositivos, interfaces de comunicação, ciência de contexto, escalabilidade, gerenciamento de dados, segurança e adaptação dinâmica podem permitir o uso da arquitetura SOFT-IoT na construção de ambientes de produção de fácil configuração, eficientes, seguros e confiáveis.

References

[Abadi et al. 2018] Abadi, F. A., Ellul, J., and Azzopardi, G. (2018). The blockchain of things, beyond bitcoin: A systematic review. In *2018 IEEE Cybermatics, 2018 IEEE International Conference on*, pages 1666–1672. IEEE.

[Abdelshkour 2015] Abdelshkour, M. (2015). IoT, from cloud to fog computing. `http://blogs.cisco.com/perspectives/iot-from-cloud-to-fog-computing`. [Online; accessed 26-September-2018].

[Al-Fuqaha et al. 2015] Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials*, 17(4):2347–2376.

[Andrade et al. 2017] Andrade, L., Rios, R., Nogueira, T., and Prazeres, C. (2017). Applying classification methods to model standby power consumption in the internet of things. In *2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC)*, pages 537–542.

[Andrade et al. 2018] Andrade, L., Serrano, M., and Prazeres, C. (2018). The data interplay for the fog of things: A transition to edge computing with iot. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE.

[B. Claise, Ed. 2004] B. Claise, Ed. (2004). Cisco systems netflow services export version 9. Disponível em: https://tools.ietf.org/html/rfc3954. Acesso em: 20 set. 2018.

[Bassi et al. 2013] Bassi, A., Bauer, M., Fiedler, M., Kramp, T., van Kranenburg, R., Lange, S., and Meissner, S., editors (2013). *Enabling Things to Talk: Designing IoT solutions with the IoT Architectural Reference Model*. Springer-Verlag Berlin Heidelberg, 1 edition.

[Batista et al. 2018a] Batista, E., Andrade, L., Dias, R., Andrade, A., Figueiredo, G., and Prazeres, C. (2018a). Characterization and modeling of iot data traffic in the fog of things paradigm. In *2018 17th IEEE International Symposium on Network Computing and Applications TO APPEAR*.

[Batista et al. 2018b] Batista, E., Figueiredo, G., Peixoto, M., Serrano, M., and Prazeres, C. (2018b). Load balancing in the fog of things platforms through software-defined networking. In *Proceedings of the IEEE International Conference on Computer and Information Technology (CIT)*, pages 1785–1791. IEEE.

[Bonomi et al. 2014] Bonomi, F., Milito, R., Natarajan, P., and Zhu, J. (2014). Fog computing: A platform for internet of things and analytics. In Bessis, N. and Dobre, C., editors, *Big Data and Internet of Things: A Roadmap for Smart Environments*, volume 546 of *Studies in Computational Intelligence*, pages 169–186. Springer International Publishing.

[Bonomi et al. 2012] Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA. ACM.

[Christidis and Devetsikiotis 2016] Christidis, K. and Devetsikiotis, M. (2016). Blockchains and smart contracts for the internet of things. *IEEE Access*, 4:2292–2303.

[Coutinho et al. 2018a] Coutinho, A., Greve, F., Prazeres, C., and Cardoso, J. (2018a). Fogbed: A rapid-prototyping emulation environment for fog computing. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE.

[Coutinho et al. 2018b] Coutinho, A., Rodrigues, H., Greve, F., and Prazeres, C. (2018b). Scalable fogbed for fog computing emulation. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–7. IEEE.

[Coutinho et al. 2016] Coutinho, A. A. T. R., Greve, F. G. P., and Carneiro, E. O. (2016). Computação em névoa: conceitos, aplicações e desafios. In *Minicursos - XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 266–315. SBC.

[de Oliveira et al. 2014] de Oliveira, R. L. S., Shinoda, A. A., Schweitzer, C. M., and Prete, L. R. (2014). Using mininet for emulation and prototyping software-defined networks. In *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*, pages 1–6. IEEE.

[Fogbed 2018] Fogbed (2018). How to setup a soft-iot testbed environment. Disponível em: https://github.com/fogbed/softiot. Último acesso: 20 jul. 2018.

[Greve et al. 2018] Greve, F., Sampaio, L., Abijaude, J., Coutinho, A., Valcy, I., and Queiroz, S. (2018). Blockchain e a revoluçao do consenso sob demanda. In *Minicursos do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos*, pages 1–52. SBC.

[Khan et al. 2012] Khan, R., Khan, S. U., Zaheer, R., and Khan, S. (2012). Future internet: The internet of things architecture, possible applications and key challenges. In *2012 10th International Conference on Frontiers of Information Technology*, pages 257–260.

[Lewis and Fowler 2014] Lewis, J. and Fowler, M. (2014). Microservices: a definition of this new architectural term. *Mars*.

[Mouradian et al. 2017] Mouradian, C., Naboulsi, D., Yangui, S., Glitho, R. H., Morrow, M. J., and Polakos, P. A. (2017). A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*.

[Pöhls et al. 2014] Pöhls, H. C., Angelakis, V., Suppan, S., Fischer, K., Oikonomou, G., Tragos, E. Z., Rodriguez, R. D., and Mouroutis, T. (2014). Rerum: Building a reliable iot upon privacy-and security-enabled smart objects. In *Wireless Communications and Networking Conference Workshops (WCNCW), 2014 IEEE*, pages 122–127. IEEE.

[Prazeres et al. 2017] Prazeres, C., Barbosa, J., Andrade, L., and Serrano, M. (2017). Design and implementation of a message-service oriented middleware for fog of things platforms. In *Proceedings of the Symposium on Applied Computing*, SAC '17, pages 1814–1819, New York, NY, USA. ACM.

[Prazeres and Serrano 2016] Prazeres, C. and Serrano, M. (2016). Soft-iot: Self-organizing fog of things. In *Advanced Information Networking and Applications Workshops (WAINA), 2016 30th International Conference on*, pages 803–808. IEEE.

[Puschmann et al. 2017] Puschmann, D., Barnaghi, P., and Tafazolli, R. (2017). Adaptive clustering for dynamic iot data streams. *IEEE Internet of Things Journal*, 4(1):64–74.

[Serrano et al. 2015a] Serrano, M., Barnaghi, P., Carrez, F., Cousin, P., Vermesan, O., and Friess, P. (2015a). IoT Semantic Interoperability: Research Challenges, Best Practices, Recommendations and Next Steps. Technical report, IERC: European Research Cluster on the Internet of Things.

[Serrano et al. 2015b] Serrano, M., Quoc, H., Le Phuoc, D., Hauswirth, M., Soldatos, J., Kefalakis, N., Jayaraman, P., and Zaslavsky, A. (2015b). Defining the Stack for Service Delivery Models and Interoperability in the Internet of Things: A Practical Case With OpenIoT-VDK. *IEEE Journal on Selected Areas in Communications*, 33(4):676–689.

[Soldatos 2016] Soldatos, J. (2016). *Building Blocks for IoT Analytics:*. River Publishers Series in Signal, Image and Speech Processing. River Publishers.

[Sousa and Prazeres 2018] Sousa, N. R. and Prazeres, C. V. S. (2018). M2-fot: a proposal for monitoring and management of fog of things platforms. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6. IEEE.

[Sundmaeker et al. 2010] Sundmaeker, H., Guillemin, P., Friess, P., and Woelffle, S., editors (2010). *Vision and Challenges for Realising the Internet of Things*. European Commission.

[Wette et al. 2014] Wette, P., Draxler, M., Schwabe, A., Wallaschek, F., Zahraee, M. H., and Karl, H. (2014). Maxinet: Distributed emulation of software-defined networks. In *Networking Conference, 2014 IFIP*, pages 1–9. IEEE.

[Zhu and Badr 2018] Zhu, X. and Badr, Y. (2018). A survey on blockchain-based identity management systems for the internet of things. In *2018 IEEE Confs on Internet of Things, Green Computing and Communications, Cyber, Physical and Social Computing, Smart Data, Blockchain, Computer and Information Technology, Congress on Cybermatics*, number 6, pages 1568–1573.

Biografia Resumida dos Autores

Leandro Andrade, é doutorando em Ciência da Computação pela Universidade Federal da Bahia (UFBA). Obteve o título de Mestre em Ciência da Computação (2014) e também o de Bacharel em Ciência da Computação (2012) pela UFBA. É pesquisador do grupo WISER-UFBA, atuando em projetos relacionados a Internet das Coisas, Computação em Névoa e Big Data. Realizou doutorado sanduíche no The Insight Centre for Data Analytics (NUI Galway - Irlanda). Leandro é membro da Sociedade Brasileira de Computação (SBC), da Communications Society e possui publicações em conferências nacionais e internacionais. É professor substituto da UFBA, vinculado ao Departamento de Ciência da Computação. Possui interesse em pesquisa nas áreas de Internet das Coisas, Computação em Névoa, Serviços Web, Web Semântica, Big Data, Aprendizado de Máquina, Informática na Educação e Software Livre.

Cleber Santana é doutorando em Ciência da Computação pela Universidade Federal da Bahia (UFBA). Obteve o título de Mestre em Sistemas e Computação pela Universidade Salvador (UNIFACS) em 2015. É pesquisador do grupo WISER e NUMAC, atuando em projetos relacionados a Microserviços, Internet das Coisas, Serviços Web Semânticos e Educação. Santana é membro da IEEE Communications Society e possui publicações em conferências nacionais e internacionais. Desde 2013 é professor do Instituto Federal da Bahia e possui interesse em pesquisa nas áreas de Web Semântica, Serviços Web, Microserviços, Internet das Coisas, Computação em Névoa, Inteligência Artificial e Informática na Educação.

Brenno de Mello é mestrando em Ciência da Computação da Universidade Federal da Bahia (UFBA). Atualmente, é participante do grupo de pesquisa WISER, pesquisando principalmente os seguintes temas: Internet das Coisas, Mineração de Fluxo de Dados, Fog Computing, Smart Water. Mello possui graduação em Análise e Desenvolvimento de Sistemas pelo Instituto Federal da Bahia (2016). Tem experiência na área de Ciência da Computação, com ênfase em Sistemas de Informação, atuando como Analista de Sistemas.

Andressa Andrade é graduanda em Engenharia de Computação na Universidade Federal da Bahia (UFBA) e bolsista de iniciação científica pela CNPq. Atuou como monitora na disciplina de Robótica Inteligente na UFBA. Desde 2015, é membro do grupo de pesquisa WISER (Web, Internet and Intelligent Systems Research). Sua área de interesse são tecnologias da camada de percepção, trabalhando principalmente no desenvolvimento de dispositivos físicos (Nó de Sensores/Atuadores) baseados na arquitetura da Internet das Coisas.

Antonio Coutinho é doutorando na UFBA. Ele recebeu o título de Mestre em Informática (2000) e Bacharel em Ciência da Computação (1998) pela Universidade Federal de Campina Grande (UFCG), Paraíba. Desde 2004, é professor assistente no Departamento de Tecnologia (DTEC) da Universidade Estadual de Feira de Santana (UEFS), Brasil. Desde 2016, é membro dos grupos de pesquisa WISER e GAUDI da UFBA. Seus principais interesses de pesquisa envolvem a aplicação da tecnologia blockchain em sistemas de Computação em Névoa e IoT. No SBRC 2016, ele foi autor e ministrou o minicurso "*Computação em Névoa: Conceitos, Aplicações e Desafios*". No SBRC 2018, ele foi autor do minicurso "*Blockchain e a Revolução do Consenso sob Demanda*". Atualmente, participa do projeto FOGGY, cujo objetivo é desenvolver soluções confiáveis para o ambiente de névoa.

Fabíola Greve é pesquisadora e professora Associada do Departamento de Ciência da Computação (DCC) da Universidade Federal da Bahia (UFBA). Ela é membro permanente do Programa de Pós-Graduação em Ciência da Computação (PGCOMP), onde atua como o líder do grupo de computação distribuída GAUDI. Realizou seu doutorado em Ciência da Computação pela Université Rennes I e Laboratórios IRISA-INRIA, França e pós-doutorado no LIP6, Université Pierre et Marie Curie (Paris-Sorbonnes Universités), França. Ao longo de sua carreia científica tem contribuído nas áreas de algoritmos e sistemas distribuídos, tolerância a falhas, desenvolvimento de sistemas confiáveis e blockchain, sendo especialista em problemas de concordância e consenso distribuído, onde tem diversas publicações; particularmente, no SBRC 2005, ministrou o minicurso "*Protocolos Fundamentais para o Desenvolvi-*

mento de Aplicações Robustas". No SBRC 2018, ela é autora e ministrou o minicurso ""*Blockchain e a Revolução do Consenso sob Demanda*". Profa. Fabíola coordena atualmente a Comissão Especial de Redes e Sistemas Distribuídos da SBC (Sociedade Brasileira de Computação), é membro e presidente do Comitê Consultivo da Conferência SBRC e membro do Conselho Administrativo da Rede Nacional de Pesquisa (RNP), representando a SBC. Realizou visitas, como pesquisadora convidada, a laboratórios de pesquisa na França: LIP6 - Laboratoire de Recherche en Informatique de l'Université Paris 6 (2009, 2010); LRI - Laboratoire de Recherche en Informatique, Université Paris-Sud, Orsay (2007); IRISA - INRIA, Université de Rennes I (2004, 2005).

Cássio Prazeres é Doutor em Ciências - Área de Ciências de Computação e Matemática Computacional - pela Universidade de São Paulo (2009), é professor Associado I na Universidade Federal da Bahia (UFBA) nas áreas Internet/Web e é orientador permanente no Programa de Pós-graduação em Ciência da Computação (PGCOMP-UFBA). Prazeres é membro: da Sociedade Brasileira de Computação (SBC); do ACM SIGWEB (Special Interest Group on Hypertext the Web); do IEEE Computer Society Technical Committee on Services Computing; do IEEE Smart Cities Technical Community; do IEEE Internet of Things Technical Community; e do W3C Web of Things Community Group. É co-fundador e líder do Laboratório e Grupo de Pesquisa CNPq WISER (Web, Internet and Intelligent Systems Research Group). Tem interesse em pesquisas envolvendo tópicos de: Internet of Things, Web of Things, Web Services, Semantic Web, Microservices, Fog Computing, Fog of Things, Web of Data. Em 2015, Prazeres realizou estágio pós-doutoral como professor visitante no DERI (Digital Enterprise Research Institute) na National University of Ireland (Galway) nas áreas de Internet das Coisas e Web Semântica.

Índice de Autores