Promotion:

SBC Sociedade Brasileira de Computação

SBSI 2018

TÓPICOS EM SISTEMAS DE INFORMAÇÃO

MINICURSOS DO XIV SIMPÓSIO BRASILEIRO DE SISTEMAS DE INFORMAÇÃO

Organization: UCS Universidade de Caxias do Sul 50 anos; CESI Comissão Especial de Sistemas de Informação

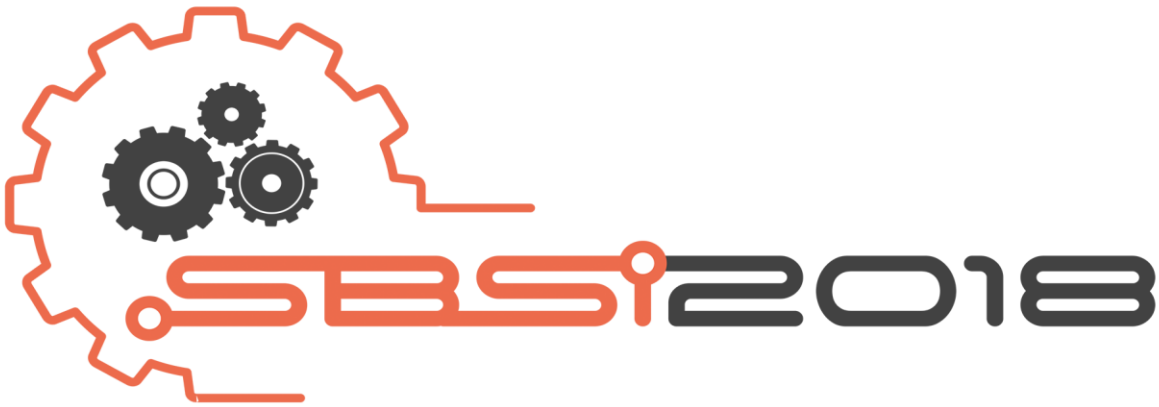
Affiliated to: Affiliated Conference

Cooperation: acm Association for Computing Machinery

Funding: CNPq Conselho Nacional de Desenvolvimento Científico e Tecnológico; CAPES

Support: unioeste Universidade Estadual do Oeste do Paraná; Compasso UMA EMPRESA UOL DIVEO; nic.br Núcleo de Informação e Coordenação do Ponto BR; hos sistemas inteligentes para farmácias; promob software solutions; TECHDEC Sua Infraestrutura Inteligente e Segura; P&F Soluções em Tecnologia; insoft4 Grupo

Promotion:



TÓPICOS EM SISTEMAS DE INFORMAÇÃO
MINICURSOS DO XIV SIMPÓSIO BRASILEIRO DE SISTEMAS DE INFORMAÇÃO

4 a 8 de Junho de 2018 - Caxias do Sul/RS

Sociedade Brasileira de Computação - SBC
CNPJ: 29.532.264/0001-78

Organizadores

Rodrigo Santos (UNIRIO)
Andre Luis Martinotto (UCS)
Scheila de Avila e Silva (UCS)

Dados Internacionais de Catalogação na Publicação (CIP)
Universidade de Caxias do Sul
UCS - BICE - Processamento Técnico

S612t Simpósio Brasileiro de Sistemas de Informação (14. : 2018 jun. 4-8 : Caxias do Sul, RS)

Tópicos em sistemas de informação [recurso eletrônico] : minicursos do XIV Simpósio Brasileiro de Sistemas de Informação = Topics in information systems : short courses of the 14th Brazilian Symposium on Information Systems / Simpósio Brasileiro de Sistemas de Informação ; realização: Sociedade Brasileira de Computação, Universidade de Caxias do Sul ; org. Rodrigo Santos, André Luis Martinotto, Scheila de Avila e Silva. – Caxias do Sul, RS : [s.n.], 2018.

Dados eletrônicos (1 arquivo).

ISBN 978-85-7669-452-6

Apresenta bibliografia.

Modo de acesso: World Wide Web.

1. Sistemas de recuperação da informação - Congressos. I. Sociedade Brasileira de Computação. II. Universidade de Caxias do Sul. III. Santos, Rodrigo. IV. Martinotto, André Luis. V. Silva, Scheila de Ávila e. VI. Título. VII. Título: Topics in information systems : short courses of the 14th Brazilian Symposium on Information Systems.

CDU 2. ed.: 004.775(062.552)

Índice para o catálogo sistemático:

1. Sistemas de recuperação da informação - Congressos 004.775(062.552)

Catalogação na fonte elaborada pela bibliotecária
Ana Guimarães Pereira CRB 10/1460.

Prefácio

Entre as atividades de Simpósio Brasileiro de Sistemas de Informação (SBSI), a discussão de temas atuais sobre pesquisa e ensino bem como a sua relação com a indústria é sempre oportunizada. Neste âmbito, os minicursos representam para a comunidade oportunidades de formação e atualização em determinados tópicos de pesquisa. Na edição 2018, foram selecionados quatro minicursos dentre 24 propostas submetidas por meio de uma chamada pública amplamente divulgada por meio de listas eletrônicas da SBC (Sociedade Brasileira de Computação). Todas as propostas receberam, no mínimo, três avaliações realizadas por um comitê composto por 39 professores doutores, que consideraram critérios como relevância para o evento, expectativa do público, atualidade e conteúdo. O SBSI 2018 mantém a tradição de publicação do conteúdo dos minicursos capítulos deste livro.

O Capítulo 1 apresenta o texto do minicurso intitulado **Construindo Aplicações Distribuídas com Microsserviços**, que visa apresentar como construir aplicações empregando arquitetura de microsserviços. Além disso, soluções para o problema de bancos de dados poliglotas são apresentadas. Como resultado, este trabalho ajuda a identificar situações onde aplicar a arquitetura de microsserviços.

O Capítulo 2 apresenta o texto do minicurso intitulado **Da Pesquisa à Inovação em Sistemas de Informação**, que visa esclarecer sobre os conceitos e processos de inovação, sobre o papel da pesquisa científica nas estratégias de inovação nacionais, em particular a pesquisa em Sistemas de Informação, e apresenta atividades e instrumentos úteis para os pesquisadores identificarem e planejarem a inovação.

O Capítulo 3 apresenta o texto do minicurso intitulado **Governança de Desenvolvedores em Ecossistemas de Software**, que visa apresentar definições e estratégias relevantes para a governança de desenvolvedores em ecossistemas de software. Os fundamentos de ecossistemas de software, relações com desenvolvedores e mineração de repositórios são discutidos com foco na sinergia das relações que envolvem desenvolvedores e organizações.

O Capítulo 4 apresenta o texto do minicurso intitulado **Sistemas de Gerenciamento de Banco de Dados em Memória**, que visa discutir os principais conceitos e técnicas que diferenciam os bancos residentes em memória dos bancos residentes em disco a partir de aspectos como indexação, armazenamento, controle de concorrência, entre outros. Além disso, são apresentados sistemas de gerenciamento de bancos de dados em memória.

Acreditamos que este material poderá ser amplamente utilizado por professores de Sistemas de Informação, para suas aulas; por pesquisadores, para discussão de novas abordagens e como insumos para pesquisas atuais e futuras; e por profissionais, para a sua prática cotidiana. Esperamos que todos que tenham acesso ao conteúdo dos minicursos e façam um ótimo proveito!

Rodrigo Santos (UNIRIO) e André Martinotto (UCS)

Coordenadores da Trilha de Minicursos do SBSI 2018

Comitê de Programa

Alexandre Cidral, Universidade da Região de Joinville (UNIVILLE)
Andre Martinotto, Universidade de Caxias do Sul (UCS)
Carla Merkle Westphall, Universidade Federal Santa Catarina (UFSC)
Carlos Alberto V. Campos, Univ. Federal do Estado do Rio de Janeiro (UNIRIO)
Carlos Eduardo Santos Pires, Univ. Federal de Campina Grande (UFCG)
Claudia Cappelli, Universidade Federal do Estado do Rio de Janeiro (UNIRIO)
Clodis Boscarioli, Universidade Estadual do Oeste do Paraná (UNIOESTE)
Clodoaldo Lima, Universidade de São Paulo (USP)
Daniel Notari, Universidade de Caxias do Sul (UCS)
Daniela Barreiro Claro, Universidade Federal da Bahia (UFBA)
Davi Viana, Universidade Federal do Maranhão (UFMA)
Debora Paiva, Universidade Federal do Mato Grosso do Sul (UFMS)
Denis Silveira, Universidade Federal de Pernambuco (UFPE)
Edmundo Spoto, Universidade Federal de Goiás (UFG)
Elvis Fusco, Centro Universitário Eurípedes de Marília (UNIVEM)
Emanuel Coutinho, Universidade Federal do Ceará (UFC)
Flavia Santoro, Universidade Federal do Estado do Rio de Janeiro (UNIRIO)
Flávio Soares Corrêa da Silva, Universidade de São Paulo (USP)
Geraldo Xexéo, Universidade Federal do Rio de Janeiro (UFRJ)
Glauco Carneiro, Universidade Salvador (UNIFACS)
Jonice Oliveira, Universidade Federal do Rio de Janeiro (UFRJ)
Jorge Barbosa, Universidade do Vale do Rio dos Sinos (UNISINOS)
José Maria David, Universidade Federal de Juiz de Fora (UFJF)
Juliano Lopes de Oliveira, Universidade Federal de Goiás (UFG)
Lais Salvador, Universidade Federal da Bahia (UFBA)
Leonardo Azevedo, Univ. Federal do Estado do Rio de Janeiro (UNIRIO)
Leticia Peres, Universidade Federal do Paraná (UFPR)
Luciano Digiampietri, Universidade de São Paulo (USP)
Lucineia Heloisa Thom, Universidade Federal do Rio Grande do Sul (UFRGS)
Maria Istela Cagnin, Universidade Federal do Mato Grosso do Sul (UFMS)
Márcio Barros, Universidade Federal do Estado do Rio de Janeiro (UNIRIO)
Morganna Diniz, Universidade Federal do Estado do Rio de Janeiro (UNIRIO)
Patricia Vilain, Universidade Federal Santa Catarina (UFSC)
Paulo Sérgio Santos, Universidade Federal do Rio de Janeiro (UFRJ)
Regina Braga, Universidade Federal de Juiz de Fora (UFJF)
Ricardo Choren, Instituto Militar de Engenharia (IME)
Rodolfo Resende, Universidade Federal de Minas Gerais (UFMG)
Rodrigo Santos, Universidade Federal do Estado do Rio de Janeiro (UNIRIO)
Valdemar Vicente Graciano Neto, Universidade Federal de Goiás (UFG)

REVISOR EXTERNO

Awdren Fontão, Universidade Federal do Amazonas (UFAM)

Sumário

Capítulo 1

Construindo Aplicações Distribuídas com Microsserviços

Luís Henrique Neves Villaça, Antônio Francisco Pimenta Jr. e Leonardo Guerreiro Azevedo

Abstract

Solutions in a microservice architecture are developed using software component compositions. This approach is based on a set of services developed and deployed in independent of each other. They can be developed using different programming languages and use the technologies that best fit their needs. One of the main challenge in that approach is to integrate data coming from distinct DBMSs (e.g., relational, spatial, NoSQL), which configures an architecture of polyglot database. This work presents how to create applications using microservices architecture. Besides, it presents solutions that handle the polyglot database problem. As a result, it helps to identify situations where to apply a microservices architecture.

Resumo

Soluções em microsserviços são construídas por meio de composição de componentes de software. Sua abordagem arquitetural baseia-se em um conjunto de serviços que podem ser desenvolvidos e implantados independentemente uns dos outros. Eles podem ser escritos em diferentes linguagens de programação e usar as tecnologias de banco de dados que melhor atendam às suas necessidades. Um dos principais desafios desta abordagem é integrar dados provenientes de SGBDs distintos (por exemplo, relacional, espacial, NoSQL), o que configura uma arquitetura de cando de dados poliglotas. Este trabalho apresenta como construir aplicações empregando arquitetura de microsserviços. Além disso, soluções para o problema de bancos de dados poliglotas são apresentadas. Como resultado, este trabalho ajuda a identificar situações onde aplicar a arquitetura de microsserviços.

1.1. Introdução

A arquitetura de microsserviços é uma abordagem para o desenvolvimento de uma única aplicação formada por um conjunto de serviços aplicados em um domínio limitado (microsserviços), cada um rodando em seu próprio processo (isolamento - contêiner) e se comunicando através de um mecanismo como uma API HTTP. Esses serviços são construídos em torno de uma parte específica do negócio e são implantados de forma completamente automatizada (Automação e Docker) [Fowler and Lewis 2014]. Cada um desses serviços implementa funções específicas, como, por exemplo, recomendações de produtos e gerenciamento de carros de compra, e possui um alto grau de autonomia (seu próprio banco de dados) para permitir sua evolução independente dos demais.

Um desafio dessa abordagem surge a partir da necessidade de integrar dados de origens distintas. Isso é um problema usual em muitas empresas, que já possuem diversas fontes de informação, como bancos de dados relacionais, NoSQL, planilhas, sistemas que gerenciam configurações de artefatos de software, e repositórios de dados não estruturados (configurando Persistência Poliglota). Um único relatório pode correlacionar informações de serviços oriundos de diversas fontes de dados. Nesse contexto, é importante que arquitetos, projetistas e desenvolvedores compreendam os aspectos envolvidos, a fim de avaliarem sua aplicabilidade em um determinado cenário [Sadalage and Fowler 2012].

Este trabalho tem objetivo de capacitar os leitores em pequisa e construção de aplicações empregando arquitetura de microsserviços. Além disso, ele se aprofunda no tema apresentando soluções para integração de dados em SGBDs distintos em uma arquitetura de microsserviços.

Este trabalho está dividido da seguinte forma. A Seção 1.2 apresenta os principais conceitos relacionados a SOA e Microsserviços. A Seção 1.3 apresenta persistência poliglota. A Seção 1.4 apresenta microsserviços e persistência poliglota na prática. Finalmente, a Seção 1.5 apresenta as considerações finais.

1.2. Fundamentação Teórica

Esta seção apresenta a fundamentação teórica sobre SOA e microsserviços.

1.2.1. Arquitetura Orientada a Serviços

Organizações modernas precisam responder de forma efetiva e rápida às oportunidades do mercado. Uma organização de médio e grande porte possui diversos departamentos (ou áreas), os quais em geral utilizam diferentes aplicações para realizar suas atividades. Estas aplicações necessitam se comunicar de forma integrada com o objetivo de atingir agilidade e simplificar processos de negócio, tornando-os mais produtivos, frente à crescente e a intensa competitividade do mercado.

O uso de serviços e de seus padrões de integração automatizada de negócios levou a grandes avanços na integração de aplicações. SOA (Service-Oriented Architecture ou Arquitetura Orientada a Serviços) é um paradigma para a realização e manutenção de processos de negócio em um grande ambiente de sistemas distribuídos que são controlados por diferentes proprietários [Josuttis 2007]. SOA é uma arquitetura conceitual onde funcionalidade do negócio, ou lógica da aplicação, é disponibilizada para usuários

SOA, ou consumidores, como serviços compartilhados e reutilizáveis em uma rede de TI [Marks and Bell 2008].

A nível conceitual, serviços são componentes de software providos através de um *endpoint* (ponto de acesso) acessível na rede [Vinoski 2002]. Serviços são módulos de negócio ou funcionalidades das aplicações que possuem interfaces expostas, e que são invocados via mensagens [Erl 2005]. Por exemplo, em um sistema bancário teríamos os seguintes serviços: serviço de nomes e endereços; serviço de abertura de conta; serviço de balanço de contas; serviço de depósitos. Serviços correspondem a recursos de software bem definidos através de uma linguagem padrão, são auto-contidos, proveem funcionalidades padrões do negócio, independentes do estado ou contexto de outros serviços [Erl 2005]. Serviço corresponde a uma representação lógica de uma atividade do negócio que pode ser mapeada em entrada, processamento e saída. Algumas características (ou princípios) de serviços são [Erl 2005]:

- Deve estar alinhado ao negócio, atendendo a uma necessidade representada em um processo da organização;
- Deve ser operacionalmente independente, garantindo alta coesão e acoplamento fraco;
- Deve fornecer os mesmos resultados para uma mesma entrada (isto é, ser sem estado ou *stateless*);
- Deve permitir composição;
- Deve ser atômico/auto-contido;
- Deve garantir consistência das informações;
- Devem ter pré e pós-condições bem definidas;
- Devem garantir interoperabilidade, ou seja, poder se comunicar com consumidores ou consumir outros serviços desenvolvidos empregando diversificadas tecnologias;
- Devem permitir reuso elevado.

Estes princípios podem ser relaxados de uma forma ou de outra para alcançar determinados objetivos como, por exemplo, serviços compostos os quais não são atômicos, pois utilizam outros serviços para realizar suas tarefas

Um serviço é implementado de acordo com um contrato, é exposto através de um *endpoint*, é governado por políticas, recebe mensagens de requisição e envia mensagens de resposta. Por outro lado, um cliente entende o contrato do serviço e estando aderente às políticas do serviço, liga-se a ele através do *endpoint* do serviço, e envia mensagens de requisição e recebe mensagens de resposta. Estes conceitos são ilustrados na Figura 1.1. A principal tecnologia de implementação serviços é a de web services, apresentada na Seção 1.2.2.

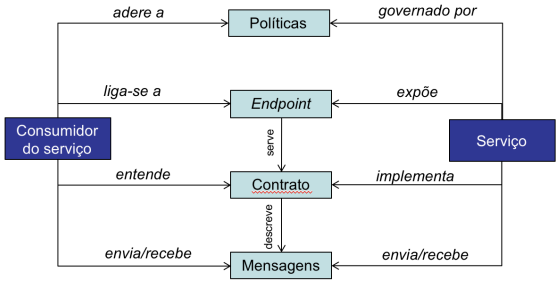


Figura 1.1. Um exemplo de troca de mensagens entre consumidor e provedor

Existe um terceiro elemento (ou parte interessada - *stakeholer*) na arquitetura orientada a serviços - o *service broker* ou registro de serviços. Ele atua como intermediário entre o provedor e o consumidor de serviço. O seu principal papel é prover informações de localização de serviço as quais estão contidas em um registro de serviços. O registro de serviços age como um diretório para serviços publicados, funcionando como um lista de páginas amarelas para números de telefones. Os provedores de serviço utilizam o registro para publicar informações sobre seus serviços e os consumidores de serviço usam o registro para procurarem por serviços. Apesar de *brokers* estarem associados com registros, eles pode realizar tarefas mais avançadas como, por exemplo, provisão de serviços de negociação e distribuição [Gu and Lago 2007].

A Figura 1.2 apresenta os três *stakeholders* de uma arquitetura orientada a serviços.

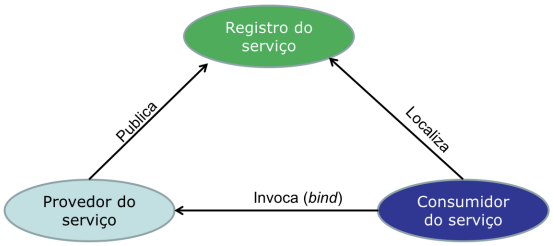


Figura 1.2. SOA *stakeholders*

1.2.2. Web services

Um web service é um sistema de software projetado para apoiar interação interoperável máquina-a-máquina através de uma rede. Ele tem uma interface descrita em um formato processável por máquina. Outros sistemas interagem com o web service de uma maneira prescrita pelo formato de suas mensagens empregando um protocolo de comunicação (tipicamente HTTP) [Booths et al. 2004]. Existem dois tipos de web services: web service SOAP (também conhecidos como WS-* ou “Big” web services [Pautasso et al. 2008]) e web service RESTful.

1.2.2.1. Web Services SOAP

Web Services SOAP empregam um conjunto de tecnologias baseadas em XML[1] (Extensible Markup Language), tais como SOAP, WSDL, XSD e UDDI.

XML[2] descreve uma classe de objetos de dados chamados de documentos XML e parcialmente descreve o comportamento de programas que os processam. Documentos XML são feitos de unidades de armazenamento chamadas de entidades, as quais contem caracteres "parsed" e caracteres não "parsed", alguns dos quais formam dados de caracteres e outros formam *markups*. *Markups* codificam uma descrição do *layout* e da estrutura lógica do armazenamento de documentos, provendo um mecanismo para definir restrições sobre os mesmos.

XSD (XML Schema Definition Language) oferece mecanismos para descrever a estrutura e restrições do conteúdo de documentos XML, incluindo aquele que exploram o mecanismos de Namespace. Namespace é um mecanismo para quebrar esquemas em subconjuntos de forma a obter definições reutilizáveis por mais de um projeto.

SOAP[3] (Simple Object Access Protocol) é um protocolo leve que tem o objetivo de troca de informações estruturadas em um ambiente descentralizado e distribuído. Ele usa as tecnologias XML para definir um *framework* de mensagens extensíveis, provendo um construto de mensagem que pode ser trocado sobre protocolos variados. Este *framework* foi projetado para ser independente de qualquer modelo de programação particular e outas semânticas específicas de implementação.

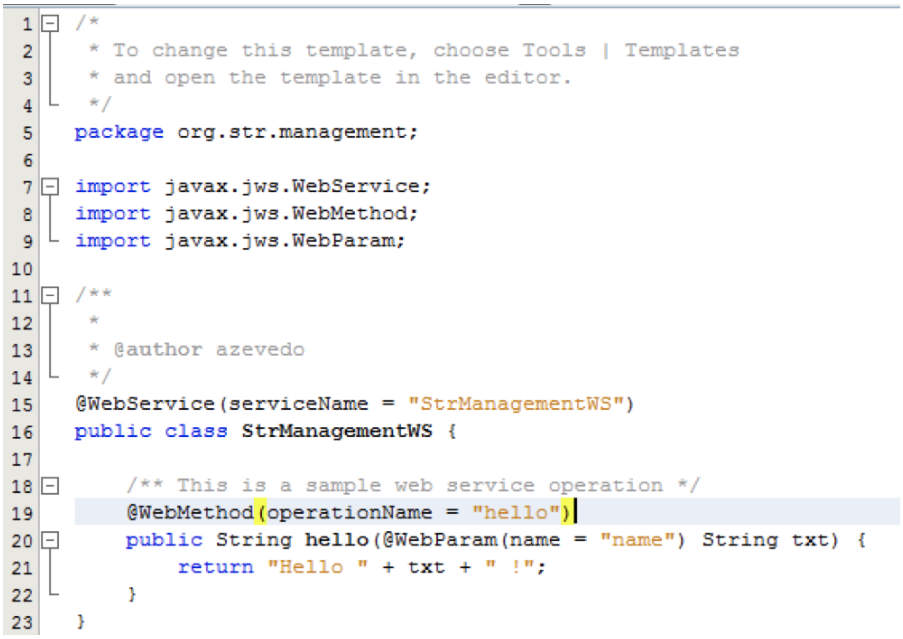
WSDL[4] (Web Service Description Language) define uma gramática XML para descrever serviços de rede como uma coleção de endpoints de comunicação capaz de realizar troca de mensagens. As definições de serviços em WSDL proveem documentação para sistemas distribuídos e serve como uma receita para automatizar os detalhes envolvidos na comunicação de aplicações.

A Figura 1.3 apresenta um exemplo de implementação de web service SOAP em Java. Nas linhas 7, 8 e 9 temos os *imports* necessários para as anotações @WebService, @WebMethod e @WebParam. O primeiro é utilizado (linha 15) para anotar a classe `StrManagementWS` como um web service SOAP disponibilizado como serviço `"StrManagementWS"`. O segundo é utilizado (linha 19) para anotar o método `hello` para ser invocado quando a operação `hello` for invocada. O terceiro é utilizado (linha 20) para definir o nome do parâmetro como sendo `name`.

1.2.2.2. Web Service REST

Web service REST (Representational State Transfer) é uma alternativa mais simples a web services SOAP. Esta tecnologia é mais fácil de utilizar. É um estilo de projeto de aplicações mais coesas e menos acopladas que se baseia em recursos nomeados ao invés

[1] https://www.w3.org/TR/xml/
[2] https://www.w3.org/TR/xml/
[3] https://www.w3.org/TR/soap12-part1/
[4] https://www.w3.org/TR/wsdl/



```java
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package org.str.management;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

/**
 *
 * @author azevedo
 */
@WebService(serviceName = "StrManagementWS")
public class StrManagementWS {

    /** This is a sample web service operation */
    @WebMethod(operationName = "hello")
    public String hello(@WebParam(name = "name") String txt) {
        return "Hello " + txt + " !";
    }
}
```

Figura 1.3. Exemplo de implementação de web service SOAP em Java.

de mensagens. Nesta tecnologia, serviços são vistos como recursos e podem ser identificados unicamente por suas URLs.

REST foi introduzido em 2000 por Roy Fielding em sua tese de doutorado na Universidade de Califórnica, Irvine [Fielding 2000]. Não atraiu muita atenção na época, mas hoje é amplamente utilizado como, por exemplo, pelo Yahoo, Google e Facebook.

REST[5] é apresentado pela W3C como sendo um subconjunto da Web baseado em HTTP nos quais agentes proveem semântica de interface uniforme - essencialmente criar, recuperar, atualizar e apagar - ao invés de interface arbitrárias ou específicas de aplicação, e manipulam recursos apenas pela troca de representações. Além disso, as interações REST são sem estado (*stateless*) no sentido de que o significado da mensagem não depende do estado da conversação.

REST não impõe restrições no formato da mensagem, como SOAP para web services SOAP, mas sim apenas no comportamento dos componentes envolvidos. Dessa forma, o desenvolvedor tem maior flexibilidade em optar pelo formato de mensagem que atenda melhor as suas necessidades. Os formatos mais comuns são JSON[6] (Java Script Object Notation), XML e texto puro, mas em teoria qualquer formato pode ser usado. A principal característica de web services REST é usar explicitamente os métodos HTTP (POST, GET, PUT, DELETE) para denotar a invocação de diferentes operações.

Web services REST se baseiam nos seguintes princípios: (i) Usar explicitamente os métodos HTTP; (ii) Ser sem estado; (iii) Expor URIs como uma estrutura de diretório; (iv) Transferir XML, JSON, ou ambos.

A Figura 1.4 apresenta um exemplo de implementação de web service REST em

[5]https://www.w3.org/TR/ws-arch/\#relwwwrest
[6]https://www.json.org/

Java. Nas linhas 3 e 4 temos os *imports* necessários para as anotações @Path e @GET. O primeiro é utilizado (linha 7) para anotar a classe `HelloResource` como um web service REST disponibilizado no caminho "`\hello`". O segundo é utilizado (linha 10) para anotar o método `getInformation` para ser invocado quando o método HTTP GET for executado neste caminho.

```
1  package com.ibm.cloudoe.samples;
2
3  import javax.ws.rs.GET;
4  import javax.ws.rs.Path;
5
6
7  @Path("/hello")
8  public class HelloResource {
9
10     @GET
11     public String getInformation() {
12
13         return "Hello World!";
14
15     }
16 }
```

Figura 1.4. Exemplo de implementação de web service REST em Java.

1.2.3. GraphQL

A linguagem GraphQL[7] foi elaborada pelo Facebook com o objetivo de viabilizar buscas de dados, com um grau maior de flexibilidade e eficiência que o disponibilizado em serviços REST e SOAP [Ghebremicael 2017]. A especificação GraphQL é de código aberto desde 2015. Ela permite que o usuário solicite informações a partir de um esquema denominado IDL (Interface Definition Language). Este esquema define um formato com os parâmetros de requisição de informações.

A Figura 1.5 apresenta um exemplo de esquema, consulta e resultado em GraphQL. O esquema inclui um tipo *Projeto* com os campos *nome*, *slogan* e uma lista de usuários *colaboradores*. A consulta busca o projeto cujo nome é *GraphQL* e solicita o *slogan* deste projeto como dado de retorno. Como resultado, obtem-se o slogan "Uma linguagem de consultas para APIs".

Esquema	Consulta	Resultado

`type Projeto {`
`  nome: String`
`  slogan: String`
`  colaboradores:`
`[Usuario]`
`}`
	`{`
	`  projeto(`
	`  nome: "GraphQL"`
	`  ) {`
	`  slogan`
	`  }`
	`}`
		`{`
		`  "projeto": {`
		`  "slogan":`
		`"Uma linguagem de`
		`consultas para`
		`APIs"`
		`  }`
		`}`

Figura 1.5. Exemplo de esquema, consulta e resultados em GraphQL.

[7] `http://graphql.org`

Consultas GraphQL não apenas acessam propriedades de um recurso, mas seguem as referências entre eles. Por exemplo, enquanto uma API REST requer carregar dados de múltiplas URLs, uma API GraphQL obtêm todos os dados em apenas uma requisição, resultando em processamento mais eficiente.

GraphQL cria uma API uniforme sem estar atrelada a uma tecnologia de armazenamento específica. Dessa forma, sendo uma tecnologia diretamente aplicável ao acesso em bancos de dados poliglotas.

1.2.4. Arquitetura de Microsservicos

Esta seção apresenta os principais conceitos de microsserviços.

1.2.4.1. Definição

A arquitetura de microsserviços (ou simplesmente microsserviços) surgiu empiricamente a partir de padrões arquiteturais utilizados no mundo real, onde sistemas são compostos por serviços que colaboram entre si para atingir seus objetivos, se comunicando a partir de mecanismos leves (como Web APIs) [Fowler and Lewis 2014, Newman 2015].

Fowler e Lewis definem **Arquitetura de Microserviços** como uma abordagem para o desenvolvimento de uma única aplicação formada por um conjunto de pequenos serviços (microsserviços), cada um rodando em seu próprio processo (isolamento - contêiner) e se comunicando através de algum mecanismo de pequeno porte, geralmente uma API HTTP. Esses serviços são construídos em torno de uma parte específica do negócio (DDD - Domain-Driven Design) e são implantados de forma completamente automatizada (Automação e Docker). Eles podem ser escritos em diferentes linguagens de programação e usar diferentes tecnologias de banco de dados. Existe uma camada mínima centralizada de gerenciamento desses serviços [Fowler and Lewis 2014].

1.2.4.2. Arquitetura Monolítica x Arquitetura de Microsserviços

Microsserviços são uma alternativa às grandes aplicações monolíticas. A Figura 1.6 apresenta um exemplo de arquitetura monolítica [Richardson 2015]. A aplicação, na parte central, contém toda a lógica do negócio, a qual pode ser implementada através de módulos que definem serviços, objetos do negócio, e eventos. Esta aplicação se conecta a um SGBD (Sistema de Gerenciamento de Banco de Dados) para consultas e armazenamento de dados e a outros serviços externos como, por exemplo, serviço de consulta de CEP.

Apesar de ter uma arquitetura interna dividida em módulos, a aplicação é empacotada e disponibilizada como uma aplicação monolítica, por exemplo, um WAR[8] para arquivos Java ou aplicações Java empacotadas como executáveis JAR.

Estes tipos de aplicações são fáceis de desenvolver por IDEs e outras ferramentas focadas em construir uma única aplicação. Elas também são fáceis de serem testadas

[8]Web application ARchive (WAR) é um formato de arquivo para distribuir, por exemplo, uma coleção de JavaServer Pages, Servlets Java, classe Java para ser implantado em servidor de aplicação como o Tomcat ou Glassfish.

e distribuídas. Inclusive elas podem ser escalonadas colocando-se múltiplas cópias em múltiplos empregando balanceamento de carga.

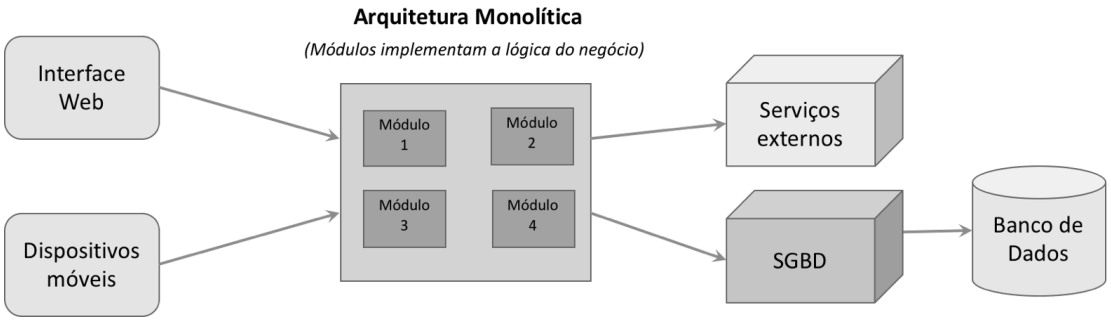


Figura 1.6. Um exemplo de arquitetura monolítica (adaptado de [Richardson 2015])

Problemas começam a surgir quando a aplicação "monolítica" cresce tornando-se extremamente complexa. Como resultado, corrigir erros (*bugs*) e implementar novas funcionalidades corretamente torna-se difícil e consome muito tempo. O problema se agrava se o código é difícil de entender, o que torna ainda mais complicado evoluir corretamente a aplicação . Além disso, aplicações muito grandes podem demorar a iniciarem. Se for necessário implantar a aplicação várias vezes ao longo do dia, será necessário reimplantar toda a aplicação para atualizar uma parte específica do código. Se estiver usando escalonamento em múltiplos servidores, serão várias reimplantações de toda a aplicação. Consequentemente, implantação contínua torna-se muito difícil de ser realizada [Richardson 2015].

Outro problema com aplicações monolíticas é que como módulos estão sendo executados no mesmo processo, um *bug* em um módulo pode derrubar toda a aplicação, ou seja, todos os seus módulos. A adoção de novos *frameworks* também torna-se um problema, pois se há muito código empregando um determinado *framework* e se deseja substituí-lo por outro, provavelmente, será necessário rescrever todo este código para utilizar o novo *framework* [Richardson 2015].

Para solucionar estes problemas, muitas empresas, tais como, Amazon, Netflix, The Guardian e outros estão usando arquitetura de microsserviços [Di Francesco et al. 2017], cuja ideia é dividir a aplicação em um conjunto de serviços menores interconectados. Um microsserviço tipicamente implementa um conjunto de funcionalidades do negócio. Cada microsserviço é uma "mini-aplicação", construída independentemente como ilustrado na Figura 1.7. Microsservicos expõem/consomem funcionalidades para/de outros microsserviços. Alguns microsserviços podem implementar interfaces web, ou seja, mesmo as interfaces web podem ser disponibilizadas como microsserviços independentes. Isto permite implantar experiências distintas para usuários ou dispositivos específicos ou para casos de uso específicos.

Alguns microsserviços são expostos para aplicativos móveis e outras aplicações através de *gateways*. Estes tem o objetivo de intermediar o acesso aos microsserviços, realizando tarefas como balanceamento de carga, *caching*, controle de acesso, medições monitoramento etc.

A fim de reduzir o acoplamento, cada microsserviço tem seu próprio banco de dados. Eventualmente, dados podem ter que ser replicados e tecnologias de bancos de dados distintas serem empregadas, como, por exemplo, um banco de dados que executa consultas espaciais com alto desempenho [Sadalage and Fowler 2012].

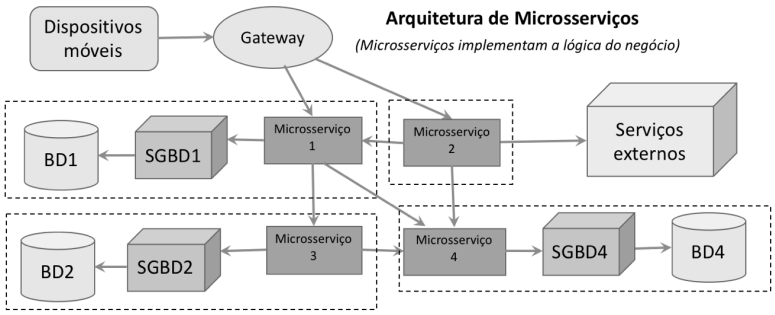


Figura 1.7. Arquitetura empregando microsserviços (adaptado de [Richardson 2015])

A arquitetura de microsserviços trata o problema da complexidade através da decomposição da aplicação monolítica em um conjunto de microsserviços. Cada microsserviço tem uma fronteira muito bem definida o que garante o nível de modularidade na prática, o que é muito difícil de alcançar em uma aplicação monolítica. Consequentemente, serviços individuais são mais fáceis de desenvolver, entender e manter. Times independentes trabalham em cada microsserviço empregando as tecnologias que consideram mais adequada (tanto de software como de hardware), de acordo com as regras da organização em que trabalham. Cada microsserviço é implantado independentemente, facilitando a atualização de novas versões sem impactar a implantação de outros microsserviços [Richardson 2015].

Não existe bala de prata e a arquitetura de microsserviços tem questões importantes a serem consideradas. Apesar de ser indicado que microsserviços sejam pequenos, o objetivo é que microsserviços correspondam a decomposições da aplicação a fim de facilitar desenvolvimento e implantação ágeis. Uma arquitetura de microsserviços corresponde a um sistema distribuído com comunicação interprocesso entre eles, o que é mais complexo do que invocar métodos no nível da linguagem, sendo também mais complexo de testar, por exemplo, necessidade de testes de integração entre os microsserviços. Particionamento do banco de dados entre os microsserviços traz o problema de garantia de integridade dos dados distribuídos entre eles. Existe uma maior quantidade de partes para serem configuradas, implantadas, escalonadas e monitoradas o que requer grande controle e alto nível de automação.

1.2.4.3. Princípios

Há sete princípios atribuídos a microsserviços [Zimmermann 2016, Fowler and Lewis 2014, Newman 2015]:

- **Interfaces com granularidade fina**: Unidades com responsabilidades específicas e independentes [Mehta and Heineman 2002] que encapsulam tanto lógica de pro-

cessamento quanto dados, onde o último é exposto através de APIs da Web ou filas de mensagens assíncronas.

- **Práticas de desenvolvimento orientadas ao negócio**: Técnicas e princípios para identificar e conceituar serviços como, por exemplo, o Projeto Orientado ao Domínio (Domain-Driven Desing - DDD) [Evans 2004]. Contexto bem definido (ou *Bounded Context*) é o padrão central do DDD [Vernon 2013]), o qual divide grandes domínios em contextos menores. Através do DDD equipes de desenvolvimento priorizam de forma sistemática os aspectos mais distintos e valiosos para a organização;

- **Princípios de design baseados em computação em nuvem**: Diretrizes como distribuição, elasticidade e baixo acoplamento[Fehling et al. 2014].

- **Entrega contínua descentralizada**: Promove um alto nível de automação e autonomia, demanda maturidade para construir artefatos de forma automatizada, aliados a uma suíte de testes [Humble and Farley 2010].

- **Contêineres leves**: Uso de metodologias de conteinerização de componentes de software para promover artefatos através de processos de entrega contínua (*e.g.*, Docker [Merkel 2014]).

- ***DevOps***: Uso de técnicas e automatização de configuração, desempenho e gerenciamento de falhas, estendendo práticas ágeis para o monitoramento de serviços [Hüttermann 2012].

- **Múltiplos paradigmas**: Combina ganhos de diferentes abordagens de computação e de tipos distintos de armazenamento[Wampler and Clark 2010].

1.2.4.4. SOA x MSA

Alguns autores entendem a arquitetura de microserviços (Microservices Architecture - MSA) como uma abordagem mais específica de SOA ou como um padrão de SOA (isto é, "padrão de microsserviços") onde SOA preconizaria o uso de serviços grossos (isto é, com granularidade grossa) e microsserviços indica o desenvolvimento de serviços mais finos (isto é, granularidade fina) [Richardson 2016]. Outros apontam a arquitetura de microsserviços como sendo algo completamente novo.

Di Francesco *et al.* apresentam que MSA surge de SOA, mas apresenta diferenças importantes em relação ao segundo. MSA foca em aspectos específicos de SOA, tais como, componentização de pequenos serviços leves, uso de práticas ágeis e DevOps para desenvolvimento, utilização de automação de infraestrutura com entrega contínua, gestão de dados descentralizada e governança descentralizada entre serviços. Dentre as diferenças entre MSA e SOA tem-se: o fato do projeto de serviços em MSA ser direcionado a uma filosofia de compartilhamento de nada (*"share-nothing philosophy"* para o uso de métodos ágeis e promover isolamento e autonomia, enquanto que SOA busca alto grau de reuso (*share-as-much-as-you-can"*); MSA foca em coreografia enquanto SOA foca em orquestração e coreografia [Richards 2015] *appud* [Di Francesco et al. 2017].

1.2.4.5. Vantagens e desvantagens

Resumindo, as vantagens de uma arquitetura de microsserviços são:

- Os microserviços são pequenos: (i) melhoram o isolamento de falhas; (ii) código é facilmente compreendido; (iii) tornam os desenvolvedores mais produtivos; (iv) iniciam muito mais rápido.
- Serviços individuais são mais fáceis de entender e podem ser desenvolvidos e implantados de forma independente. Cada microsserviço pode ser implantado em um hardware mais adequado para as exigências de seus recursos.
- Adotar novas tecnologias e *frameworks* torna-se mais fácil, pois a adoção pode ser aplicada em um microsserviço de cada vez.
- Cada microsserviço pode ser escalonado de forma independente através da duplicação e particionamento.
- A arquitetura de microserviços é adequada para aplicações complexas e de grande porte que estão evoluindo rapidamente como, por exemplo, Netflix e Spotfy.

Como desvantagens da arquitetura de microsserviços temos:

- As complexidades adicionais do desenvolvimento de sistemas distribuídos: (i) Aplicações muito mais complexas e constituídas por mais elementos; (ii) Uso de transações distribuídas ou consistência eventual; (iii) Gerenciamento de um número maior de aplicações em produção.
- Para ser utilizada de forma eficaz, a arquitetura de microsserviços exige um alto nível de automação.

1.2.5. Contêineres

Contêiner é um termo que descreve uma alternativa mais leve às máquinas virtuais. Para isso é feito o encapsulamento da aplicação em um ambiente virtual que contém apenas os ativos necessários para o funcionamento. Os contêineres são isolados a nível de disco, memória, processamento e rede. Essa separação permite uma grande flexibilidade, onde ambientes distintos podem coexistir na mesma máquina hospedeira (*host*), sem causar problemas.

Comparando contêiner com maquina virtual, temos que cada máquina virtual requer um Sistema Operacional próprio além dos softwares e bibliotecas. Além disso uma camada intermediária (chamada *hypervisor*) gerencia a comunicação de cada máquina virtual com o sistema operacional hospedeiro. Já os contêineres acessam diretamente o sistema sistema operacional hospedeiro e seus recursos (por exemplo, disco, memória, rede) para prover um ambiente virtual para as aplicações. Portanto, no ambiente de contêiner, é necessário instalar os requisitos (softwares, arquivos etc.) que o microsserviço precisa sem se preocupar com instalações de outro Sistema Operacional, além de não precisar do *hypervisor*.

1.2.6. Docker

Docker[9] é uma plataforma aberta criada utilizando o modelo de contêiner para "empacotar" a aplicação, que após ser transformada em uma imagem Docker, poderá ser reproduzida em plataforma de qualquer porte. O objetivo é facilitar o desenvolvimento, implantação e execução de aplicações em ambientes isolados da forma mais rápida possível. O Docker permite gerenciar a infraestrutura da aplicação. O que agiliza o processo de criação, manutenção e evolução.

O Dockerfile é um arquivo que descreve as instruções para criar uma imagem de contêiner Docker. Uma imagem sempre deve partir de uma imagem base. Portanto, o Dockerfile descreve de uma forma textual a diferença entre a imagem base e a imagem que se deseja criar, isto é, o Dockerfile contém a sequência de instruções necessárias para modificar a imagem base para que ela fique com as características desejadas.

A Figura 1.8 apresenta um exemplo de Dockerfile. A linha 2 corresponde ao nome e a versão da imagem base. A linha 4 contém informações do mantenedor da imagem base. A linha 6 define variáveis de ambiente. A linha 8 executa uma instrução na linha de comando. Neste caso, é executado o `apt-get`[10] para instalar a biblioteca `openjdk`[11]. A linha 10 executa o comando `ADD` que copia arquivo para diretório de trabalho (`WORKDIR`) do contêiner. Neste caso, está sendo copiado o `arquivo_test` da raiz para o diretório `temp` da raiz do `WORKDIR`.

```
1  # Container Docker
2  FROM ubuntu:14.04
3
4  MAINTAINER nome mantenedor <email@gmail.com>
5
6  ENV DEBIAN_FRONTEND noninteractive
7
8  RUN apt-get install openjdk
9
10 ADD ./arquivo_teste /tmp/
```

Figura 1.8. Dockerfile transformando imagem base em imagem modificada.

Docker Hub[12] provê um recurso centralizado para descoberta, distribuição e gestão de mudanças de imagens de contêineres, colaboração de usuários, e automação de *workflow* através de um *pipeline* de desenvolvimento.

Docker-Compose é a ferramenta do Docker que permite a criação e a execução de aplicações baseadas em múltiplos contêineres. Através dele, é possível construir as imagens dos contêineres e iniciar os serviços da composição. Cada contêiner mantém seu isolamento, mas é possível que um contêiner "identifique" outro através do nome do host usando o comando "link" - sem a necessidade de usar ip fixo. Também é possível criar **volumes**, que são pastas no Sistema Operacional hospedeiro onde podem ser armazenados

[9] `https://www.docker.com/`

[10] O apt-get é um recurso desenvolvido originalmente para a distribuição Debian que permite a instalação e a atualização de pacotes (programas, bibliotecas de funções, etc) no Linux.

[11] OpenJDK ("Open Java Development Kit") é uma implementação livre e gratuita da plataforma Java, Edição Standard ("Java SE").

[12] https://docs.docker.com/docker-hub/

os arquivos persistentes de aplicações executando no contêiner (como os de bancos de dados).

A configuração de uma composição de contêineres no Docker é descrita através do arquivo `docker-compose.yml`. Ele é um arquivo na sintaxe YAML[13] onde é possível descrever serviços, redes e volumes da composição.

A Figura 1.9 apresenta um exemplo de composição utilizando o Docker Composer. A linha 1 define o nome do contêiner. A Seção "links" listam os contêineres que serão compostos, neste caso, *db* e *redis* - linhas 3 e 4 respectivamente. A imagem *redis* tem o nome *redis* e a imagem *db* tem o nome "postgres" - linhas 7 e 9, respectivamente. Finalmente, a Seção "volumes" define os volumes dos contêineres. Neste caso, o contêiner *db* tem, na linha 11, o mapeamento do diretório `/opt/bancodados/` do Sistema Operacional hospedeiro mapeado no diretório `/usr/local/pgsql/data`.

```
1  web:
2    image: httpd
3   links:
4    - db
5    - redis
6  redis:
7    image: redis
8  db:
9    image: postgres
10   volumes:
11       - /opt/bancodados/:/usr/local/pgsql/data/
```

Figura 1.9. Exemplo de composição usando Docker Composer.

Algumas alternativas ao Docker são:

- **LXD ("Lex-Dee")**[14]: proposta de contêineres que operam como máquinas virtuais.
- **Rocket (rkt)**[15]: emprega o conceito de *pod* - uma coleção de uma ou mais aplicações executando em um contexto compartilhado. Configurações podem ser feitas a nível de *pod* ou a nível da aplicação. A arquitetura do rkt preconiza que cada *pod* executa diretamente em um modelo de processo Unix (isto é, não existe *daemon* central), em um ambiente auto-contido e isolado.

1.3. Persistência Poliglota em Microsserviços

Esta seção apresenta os conceitos de persistência poliglota e estratégias para tratar este desafio em microsserviços.

1.3.1. Definição

O termo "Persistência Poliglota" é usado para explicitar uma abordagem de persistência híbrida[Sadalage and Fowler 2012], que propicia a utilização dos melhores mecanismos projetados para armazenamento e recuperação das informações de acordo com a sua natureza. Isso é ilustrado no cenário apresentado na Figura 1.10, no qual uma plataforma de

[13] http://yaml.org/spec/
[14] `https://www.ubuntu.com/containers/lxd`
[15] `https://coreos.com/rkt/`

e-commerce consome quatro microsserviços, cada um com seu próprio SGBD (Sistema de Gerenciamento de Banco de Dados). Dados de um carrinho de compra (por exemplo, id de produto e quantidade) podem ser eficientemente resgatados a partir de um banco chave-valor (*key-value*), e ordens de compra podem ser persistidas de forma agregada e estruturada em um banco NoSQL Document, enquanto que um catálogo de produto pode ser mantido por meio de um banco relacional (SGBDR) e informações cruzadas de compra e clientes podem ser armazenadas em um grafo para inferir recomendações. Sendo assim, para atender diferentes requisitos tecnologias apropriadas de banco de dados são utilizadas.

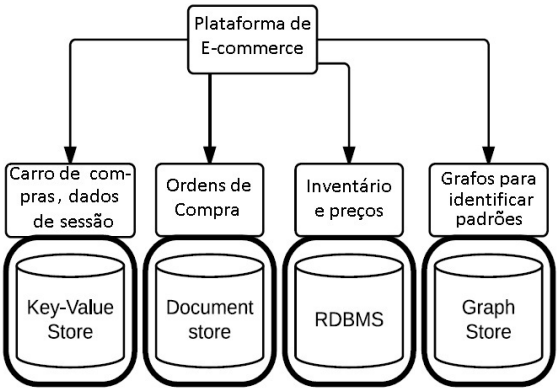


Figura 1.10. Persistência Poliglota - Adaptada de: [Sadalage and Fowler 2012]

1.3.2. Problemas

Descentralizar a responsabilidade pela manutenção de dados em microsserviços tem implicações no gerenciamento de atualizações dos dados e das consultas. Devido à complexidade na implementação de transações distribuídas, as arquiteturas de microsserviço enfatizam a coordenação em atualizações de serviço e delegações em consultas, de modo que a consistência ou a disponibilidade nem sempre serão garantidas. Isso é uma característica inerente a qualquer sistema distribuído que compartilha dados, conforme o estabelecido no teorema CAP [Brewer 2000] (*Consistency, Availability and Partition Tolerance*), pelo qual afirma-se que no máximo dois fatores - dentre tolerância a partição, consistência e disponibilidade - podem ser plenamente e simultaneamente atendidos. Dado que partição de rede é inerante em uma arquitetura de microsserviços, nós temos que balancear entre disponibilidade e consistência, os quais combinados com diferentes tecnologias de persistência trazem novas demandas e requerem integrações complexas [Fowler 2015].

1.3.3. Estratégias de Abordagem dos Problemas

Existem cinco estratégias para consultar e integrar informações distribuídas, em um cenário de persistência poliglota em uma arquitetura de microsserviço, com base em diferentes abordagens encontradas na literatura.

Um conjunto inicial de quatro estratégias foi definido de acordo com abordagens específicas para microservices [Newman 2015] (Seções 1.3.3.1, 1.3.3.2, 1.3.3.3,

1.3.3.4). Esta lista foi complementada por uma estratégia alternativa (Seção 1.3.3.5) baseada em estudos científicos [Ghawi and Cullot 2007, Collins et al. 2002] publicados antes de 2012. Na época, o termo "microservices"[Fowler and Lewis 2014] ainda não havia sido cunhado, e eles foram escolhidos devido à sua proposta e à similaridade de arquitetura. Nesta seção, apresentamos as estratégias bem como a avaliação das mesmas conforme Villaça *et al.* [Villaca et al. 2018], incluindo seus aspectos positivos e negativos mais relevantes. Artigos relacionados, alinhados a cada estratégia, também são apresentados. Villaça *et al.* complementam seu artigo com um quadro comparativo das estratégias conforme o seu grau de adequação à ISO 25010, uma norma de qualidade de software amplamente utilizada como referência.

1.3.3.1. Shared Database

Essa estratégia sugere o aproveitamento de recursos de compartilhamento disponíveis nos próprios bancos de dados de forma que vários serviços possam se beneficiar da mesma fonte de dados [Newman 2015]. Três mecanismos podem ser usados nesta abordagem para gerir os dados [Messina et al. 2016]: (i) Cada serviço tem um conjunto de tabelas em um esquema compartilhado; (ii) Cada serviço tem um esquema em um SGBD compartilhado; (iii) Cada serviço tem seu próprio SGBD.

Essa última opção (Figura 1.11) pode usar diferentes tecnologias para armazenamentos de dados, demandando a utilização de soluções como *gateways*, conectores ou federações de dados, que podem implicar em custo de maior latência para recuperar informações.

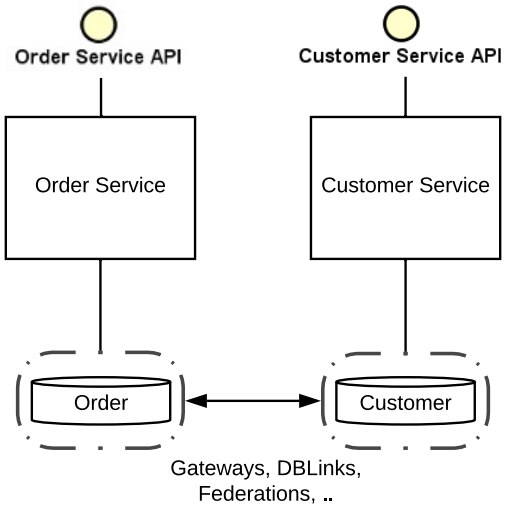


Figura 1.11. Bancos de Dados compartilhados entre microsserviços

Dentre seus aspectos positivos e negativos mais relevantes [Villaca et al. 2018] destacamos:

- **Prós**

1. Desenvolvimento simplificado, uma vez que cada serviço dispõe de acesso direto às fontes de informação;
2. Bons desempenho se o número de nós reduzido, dado que serviços podem efetuar filtros e junções de uma maneira otimizada. O tráfego de rede é reduzido devido à baixa necessidade de colaboração entre os serviços uma vez que o banco de dados é compartilhado.

- **Contras**

1. Alto Acoplamento. Mudanças em esquemas e indisponibilidade de banco de dados impactam todos os componentes que o utilizam. Com o incremento do número de serviços aumenta o risco de impacto das operações transacionais sobre as consultas, e vice-versa. A manutenibilidade dessa infraestrutura também é prejudicada, dado que as entregas (*releases*) dos serviços são vinculadas entre si. Além disso, a dependência a soluções de fornecedores afeta a portabilidade dos componentes envolvidos. Devido a esses aspectos, essa estratégia é vista como um anti-padrão para microsserviços [Richardson 2016].
2. Risco de baixa rastreabilidade. Atividades de teste e análise que dependem de avaliar o código executado demandam a existência de código aberto e compreensível pela equipe de desenvolvimento e operação.

Exemplo prático: Uma abordagem de implementação dessa estratégia considera que um banco de dados pode ser considerado um microsserviço por si só [Messina et al. 2016]. Uma vez que um banco de dados apresente uma arquitetura aberta e forneça mecanismos suficientes para estender seus recursos, ele pode incorporar a lógica de negócios que implementa o serviço desejado, atuando como um serviço de negócios. Para tal, Messina *et al.* usaram OrionDB[16], um banco de dados NoSQL multimodelo que suporta a persistência de modelos em grafo e como documentos, e permite o relacionamento entre os dados de ambos via API. Uma das vantagens dessa implementação é que em casos de muitas instâncias o cluster do banco atua como um registro de serviços (*service registry*).

1.3.3.2. Command Query Responsibility Segregation (CQRS)

CQRS preconiza a utilização de uma visão materializada dos bancos de dados transacionais para fins de recuperação de informações de modo a isolar as operações de carga das operações de consulta [Newman 2015, Richardson 2016]. Nesta estratégia, os modelos devem ser divididos em duas partes (Figura 1.12): (i) *command-side*: responsável pela atualização de informações; (ii) *query-side*: responsável pela execução de consultas.

No *command-side* as operações de alterações (*i.e.*, inserções, atualizações e remoções) são validadas e, se aceitas, são aplicadas ao seu modelo. Seus componentes emitem eventos sempre que os dados são alterados, sinalizando alterações de estado. Esses eventos são publicados em uma fila ou armazenados em um meio como um banco de dados.

[16] `http://orion.cs.purdue.edu/`

O *query-side* consome o fluxo de eventos emitidos para capturar os dados alterados. Este módulo pode criar projeções a partir de eventos armazenados para montar o estado de objetos de domínio (processo conhecido como *event sourcing*), ou simplesmente atualizar um tipo diferente de armazenamento (como um *data warehouse*).

Essa separação permite diferentes tipos de escalonamento. As partes de comando e consulta podem ser implantadas como serviços separados, em infraestruturas de hardware separadas e com diferentes tipos de armazenamentos de dados [Newman 2015].

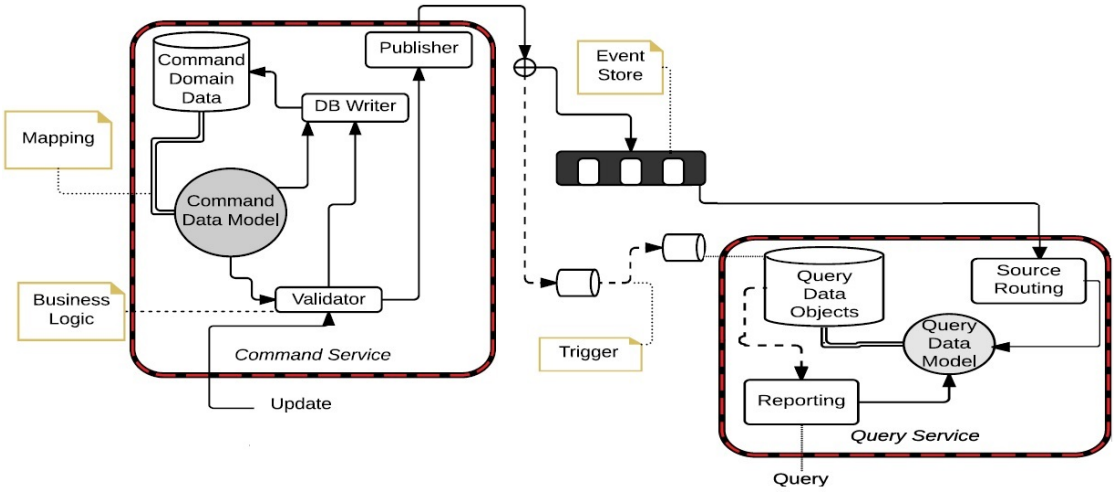


Figura 1.12. *Command-side* e *Query-side* - CQRS

- **Prós**

  1. Otimização de recursos e redução do tempo de resposta, especialmente em cenários em que buscas em grandes volumes de dados impactam operações transacionais;
  2. Serviços apresentam capacidade de evoluir de acordo com sua demanda, sem impactar os demais. Essa abordagem favorece o projeto de componentes mais coesos e fracamente acoplados.

- **Contras**

  1. Eventos de falha na sincronização, ou mesmo de longa duração para sua concretização, podem comprometer a consistência.
  2. Complexidade no desenvolvimento e operacionalização da arquitetura, necessidade de lidar com estratégias de tratamentos de eventos fora da ordem de ocorrência, de idempotência, etc.

Exemplo prático: ***Medical.Net*** [Rajkovic et al. 2013] é um sistema de informação que foi evoluído para seguir esta estratégia. Ele é dedicado para instalações médicas e agrega informações de formulários de admissão de pacientes, exames médicos, análises laboratoriais e tratamentos terapêuticos. A versão inicial deste sistema começou a mostrar

respostas lentas à medida que a quantidade de informações ficou maior, com dados espalhados em tabelas separadas. O desempenho das consultas foi afetado, devido ao aumento no tráfego de dados nas operações de junção (*join*). Para reduzir o impacto nas operações de carga no banco de dados principal, um componente separado foi criado apenas para leitura de dados, juntamente com um mecanismo de sincronização que apresenta o seguinte funcionamento: agentes (microsserviços) capturam alterações de dados específicas dos bancos transacionais (no *command-side*) e as replicam em bancos de dados (somente leitura) desnormalizados para otimizar consultas, usando estratégias para invalidar ou remover registros alterados e migrar novos dados.

1.3.3.3. Event Data Pump

Event Data Pump é muito similar à estratégia CQRS, cuja definição é mais estrutural, essa estratégia apresenta uma definição comportamental. Atualizações de informações de serviços são capturadas e publicadas como uma série de eventos em uma plataforma (Figura 1.13) que permite seu consumo pelos demais serviços interessados [Newman 2015]. Esses serviços (receptores) são responsáveis por manter o estado atualizado de suas fontes de dados, com base nas informações obtidas nos eventos que são do seu interesse.

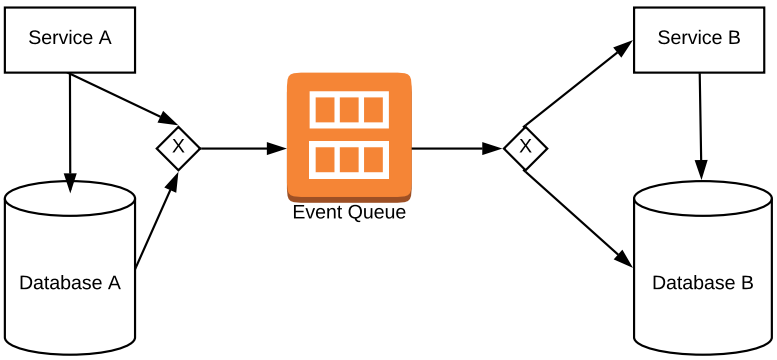


Figura 1.13. Event Data Pump

- **Prós**
  1. Ganho de desempenho, especialmente em cenários de cargas contínuas com um grande volume de dados, e necessidade de processamento e integração com outros dados em tempo próximo ao de ocorrência dos eventos;
  2. Serviços mais coesos e fracamente acoplados.
- **Contras**
  1. Falhas de um publicador ou receptor podem afetar a consistência das informações obtidas, e, devido à natureza distribuída da arquitetura, podem se tornar difíceis de investigar;
  2. Complexidade nas atividades de desenvolvimento e operação.

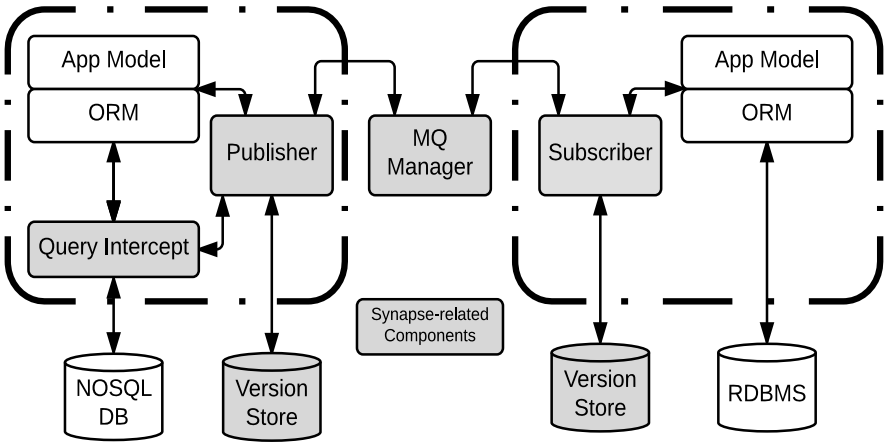


Figura 1.14. Componentes Synapse - adaptado de [Viennot et al. 2015]

Exemplo prático: Synapse [Viennot et al. 2015], um sistema de replicação escalável, permite que diferentes serviços, lidando com diferentes estruturas de dados, sejam desenvolvidos de forma independente. Cada um pode compartilhar subconjuntos de seus dados com os outros, e o Synapse sincroniza esses subconjuntos de dados com base nas informações mapeadas em cada serviço. Nos nós que estão atualizando dados, um agente Synapse é posicionado antes do *driver* de banco de dados, que lhe possibilita interceptar as atualizações de todos os modelos publicados antes de serem persistidos no banco de dados (Figura 1.14). Dessa forma, o sistema identifica quais objetos estão sendo gravados e os transmite para um Publicador, usando todos os atributos de objetos criados ou atualizados para criar uma mensagem de gravação. O Synapse envia a mensagem para um sistema de mensagem confiável, persistente e escalável, que a redistribui para seus serviços consumidores (assinantes).

1.3.3.4. Data Retrieval via Service Call

A ideia dessa estratégia é extrair os dados necessários dos serviços de origem por meio de chamadas API (Figura 1.15). Para produzir relatórios reunindo dados de dois ou mais sistemas, são necessárias várias chamadas. No caso de precisarmos puxar grandes volumes de dados para compor uma visão de diferentes serviços, as consultas podem se tornar muito lentas; portanto, é preciso implementar estratégias para atenuar esse problema [Newman 2015].

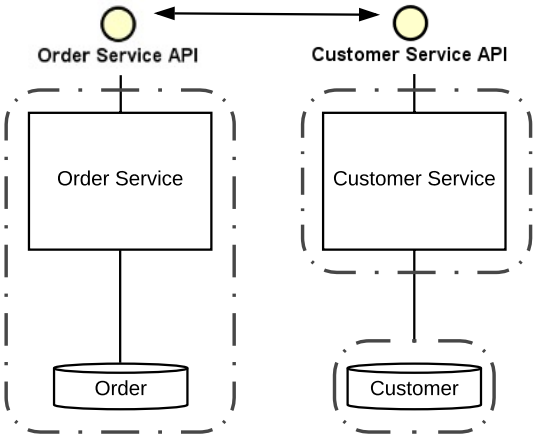


Figura 1.15. Data Retrieval via Service Call

- **Prós**

    1. Aplicações coesas podem ser reutilizadas e reagrupadas na composição de novas soluções;
    2. Novos serviços podem ser facilmente incorporados na infraestrutura.

- **Contras**

    1. APIs expostas podem não prever demandas de uso dos seus relatórios. O desempenho pode ser afetado ao executar operações que coletam dados de vários serviços (com diferentes meios de armazenamentos de dados). Além disso, redundância ou ambiguidade podem ser introduzidas se os serviços não forem cuidadosamente planejados, devido à autonomia.
    2. Cada componente deve ser capaz de tolerar falha, indisponibilidade ou atraso dos serviços que colaboram com o mesmo;
    3. As evoluções devem ser planejadas de maneira coletiva a fim de minimizar os impactos aos demais serviços.

Exemplo prático: Sheibel elaborou um caso de uso para este cenário [Scheibel 2016] no contexto de dados de trajetória de objetos móveis. A quantidade de dados manipulados por soluções nesta área é relevante - o tamanho da maior base de dados utilizada no estudo foi estimado em 820 GB de armazenamento. A fim de abstrair cada aplicação consumidora a partir de detalhes de quais microsserviços e informações do banco de dados devem ser utilizados, um novo serviço (chamado de Catálogo de Trajetórias) foi construído em uma camada acima de todos os microsserviços que lidam com repositórios. O mesmo coleta dados como informações de estado das trajetórias e de seus repositórios, propiciando o roteamento das solicitações para seus serviços específicos. Os resultados são tratados com base nos metadados de serviço mantidos por esse componente.

1.3.3.5. Canonical Data Model

Um modelo de dados canônico (CDM) compreende a semântica e a estrutura da informação, de acordo com regras acordadas por várias partes [Gilpin 2015], simplificando a troca de dados entre os serviços. Essa estratégia é baseada no Serviço Central, que mantém um modelo universal, e de serviços subjacentes (serviços A e B) que mantêm modelos derivados do canônico, além de fontes de dados próprias. Esses serviços são capazes de mapear requisições, que chegam de acordo com seu modelo, aos dados estruturados em suas bases(Figura 1.16).

Para garantir compatibilidade entre todos os componentes, o Serviço Central abstrai os detalhes dos serviços que gerem as fontes de dados das visualizações dos usuários, de modo que traduz suas consultas em subconsultas (com base em modelos centrais e locais). As subconsultas são executadas nos serviços correspondentes, e, a partir disso, seus resultados são integrados para compor a resposta aos usuários.

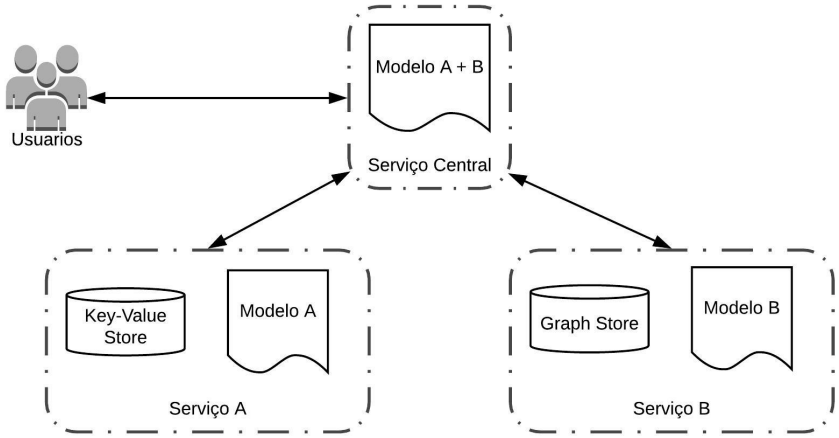


Figura 1.16. Canonical Data Model

- **Prós**

  1. Potencial para maior expressividade em pesquisas, especialmente para dados semânticos;
  2. Modelos canônicos podem auxiliar na validação de todos os modelos de dados dos serviços, uma vez que antecipam problemas de inconsistência e redundância dos dados;
  3. Potencial para composições de vários serviços e inferências de informações com base no modelo conhecido;

- **Contras**

  1. Complexidade na modelagem adequadamente as parcelas que compõem o CDM, na implementação da segregação do processamento, distribuído entre

nós delegados (que podem apresentar falhas), e na composição dos resultados conforme a requisição;

2. Dificuldade em manter um esquema canônico atualizado na medida em que as fontes distribuídas evoluem.

Exemplo prático: A proposta de Ghawi e Cullot [Ghawi and Cullot 2007] utiliza ontologias para a descrição semântica de fontes de informação usando uma ontologia híbrida - em que cada fonte de informação tem sua ontologia local, e uma ontologia de domínio global é usada para representar todo o domínio. Assim, cada ontologia relacionada à fonte de dados pode ser desenvolvida independentemente das ontologias de outras fontes.

1.4. Microsserviços na Prática

Esta seção apresenta práticas com microsserviços e persistência poliglota. A Seção 1.4.1 apresenta o cenário empregado na apresentação prática dos conceitos e também para ser utilizado no exercício prático deste trabalho. A Seção 1.4.2 apresenta a arquitetura tecnológica empregada. A Seção 1.4.3 apresenta os microsserviços de dados implementados. A Seção 1.4.4 apresenta os microsserviços de integração de dados, incluindo o microsserviço mediador implementado e o microsserviço de Processamento de Eventos proposto como exercício.

1.4.1. Cenário de Estudo

Um caso de uso foi elaborado a partir de um cenário de negócio presente em aplicações de *E-Commerce*. Embora bastante simples, contém elementos suficientes para apresentar uma quantidade relevante de desafios, vantagens e desvantagens usualmente observados no contexto de microsserviços. Ele foi construído a partir das estratégias de integração Data Retrieval via Service Call (Seção 1.3.3.4) e Event Data Pump (Seção 1.3.3.3) - neste caso foram propostos exercícios complementares.

O sistema possui como requisito apresentar informações relacionadas a carrinhos de compra dos clientes, tais como: produtos e quantidades, endereço de entrega cadastrado para o cliente e nome do fornecedor de um produto. Também deve apresentar informações isoladas de clientes e produtos para seus usuários. O diagrama ER (Figura 1.17) apresenta uma visão alto de nível do relacionamento entre as entidades envolvidas.

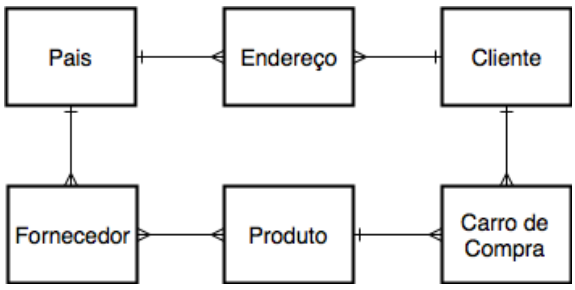


Figura 1.17. Diagrama ER as entidades do negócio (elaborado empregando notação Engenharia de Informações [Heuser 2009])

1.4.2. Arquitetura desenvolvida para o cenário

De acordo com o princípio de responsabilidade única de serviços (Seção 1.2.4.3), realizamos a divisão dos objetos como apresentado na Figura 1.18. Essa segregação proporciona maior grau de autonomia no desenvolvimento de serviços, dado que os mesmos só fornecem visibilidade de seus dados por meio de seu contrato (API) ou de mecanismos de replicação desacoplados de seus modelos de persistência. Cada parte desta divisão será disponibilizada por um microsserviço, o qual utilizará uma fonte de dados com tecnologia distinta dos demais, caracterizando um quadro de persistência poliglota.

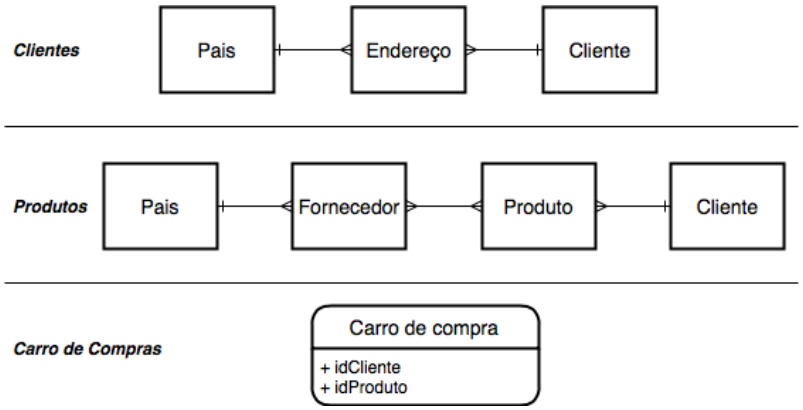


Figura 1.18. Diagrama ER apresentando a divisão das entidades do negócio em domínios

Além disso, outros componentes e serviços também comporão a solução a fim de prover mecanismos de integração, visualização e processamento de relatórios consolidado. A arquitetura dessa solução abrange os componentes apresentados na Figura 1.19. Os estereótipos de dada componente descrevem suas responsabilidades:

- Bancos de Dados: Fontes dos dados relacionados a cada divisão das entidades do negócio.
- Microsservicos de Dados: Responsáveis pelas atividades transacionais sobre os Bancos de Dados. Funcionam como uma camada de acesso a dados via API.
- Microsservicos de Integração: Consumidores dos dados providos via API ou Plataforma de Integração, são responsáveis por integrá-los e processá-los a fim de compor uma visão consolidada (ou mesmo ampliada) a partir das informações existentes. Este estereótipo foi aplicado a três componentes distintos:
  - Mediador: Utiliza estratégia Data Retrieval Via Service Call, consumindo dados via API dos Microsserviços de Dados e Processador de eventos.
  - Processador de eventos: via Plataforma de Integração, usa a estratégia Event Data Pump, por meio de mensagens subscritas a partir dessa plataforma. Ele atualiza uma fonte de dados própria, consolidando e agregando novas informações (como métricas estatísticas) requisitadas em relatórios.

– Plataforma de Integração: Componente que mantém os eventos publicados, a cada atualização das entidades de negócio, pelos Microsserviços de dados.

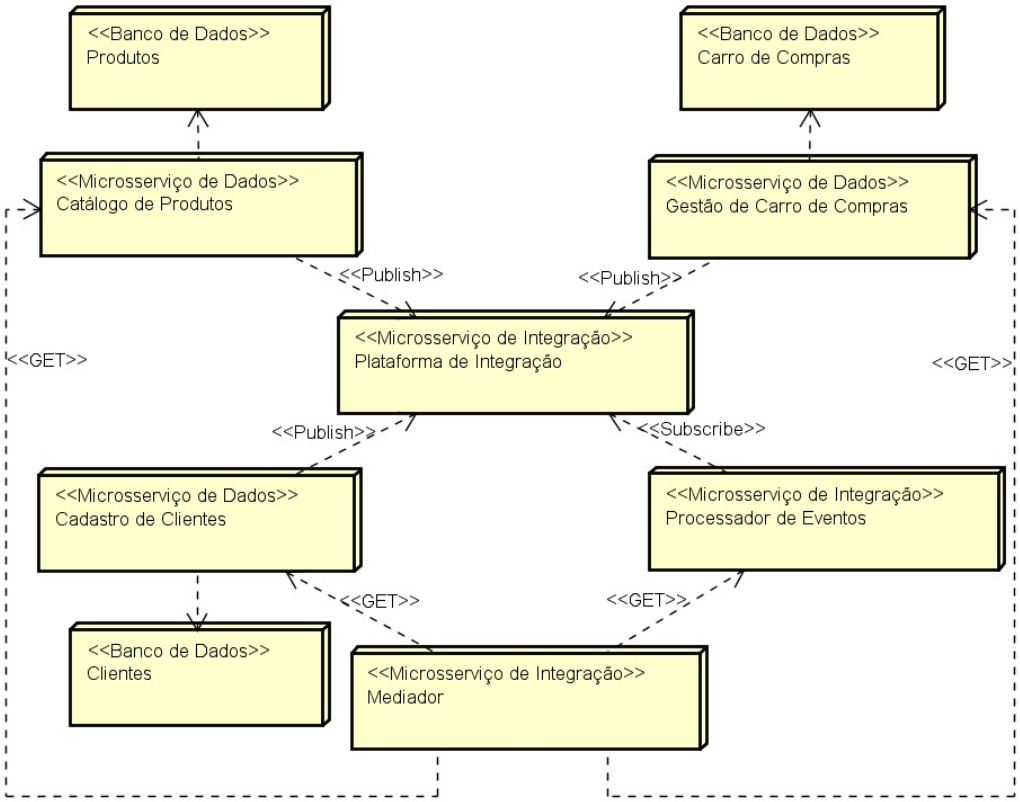


Figura 1.19. Diagrama de Deployment - Arquitetura proposta

1.4.2.1. Repositório Git do trabalho

Um repositório Git[17] público foi criado no Bitbucket[18] para este trabalho disponibilizado na seguinte URL[19], contendo: (i) *readme* explidando o repositório; (ii) A estrutura de arquivos com o código fonte da arquitetura e dos exercício; (iii) Apresentação (*slides*) do trabalho; (iv) Outros links e material de apoio.

1.4.2.2. Componentes tecnológicos comuns aos microsserviços

Três componentes tecnológicos foram utilizados para a criação dos microsserviços desse estudo.

[17] `https://git-scm.com/`
[18] `https://git-scm.com/`
[19] `https://luis-villaca@bitbucket.org/luis-villaca/tutorial_2018_sbsi.git`

Gradle[20]: *script* de *build* utilizado para empacotar aplicações e componentes web, traz uma série de facilidades como definição de atividade e mapeamento de dependências de bibliotecas via Maven[21]. O script Gradle é ainda utilizado para a execução das classes de teste e de eventual geração de métricas (como verificação do percentual de cobertura de código), build do artefato (caso o passo anterior não apresente erros) e posterior criação da imagem e contêiner Docker.

SpringBoot[22]: Permite a criação de aplicações autocontidas (com servidores web, aplicação e mesmo com bancos de dados), propicia integração com ferramentas que provêm métricas de qualidade de software, permite a configuração da aplicação Gradle. Essa tecnologia facilita a automatização na construção de pequenos serviços coesos e desacoplados.

Java8[23]: Dentre outras vantagens para versões anteriores, permite o processamento paralelo ou sequencial de dados em uma coleção (via StreamAPI[24]), com uma sintaxe mais clara e com maior desempenho do que a tradicional.

O site Spring IO provê um passo-a-passo[25] para se criar uma imagem Docker com SpringBoot e Gradle. Similar ao demonstrado no passo-a-passo, utilizaremos a estrutura de pastas indicada na Figura 1.20 para a confecção dos artefatos com SpringBoot (através do Gradle). Em determinados microsserviços essa configuração terá alguns pequenos ajustes.

1.4.2.3. Bancos de Dados

Três tecnologias com modelos distintos de persistência foram utilizadas para os cenários segregados e são apresentadas a seguir. A maior motivação é demonstrar o cenário de persistência poliglota, mas outros fatores, como flexibilidade para a manutenção de estrutura dos dados, desempenho e necessidade de garantia de integridade nas operações transacionais devem ser considerados na escolha das tecnologias.

DB I - Catálogo de Produtos

Para este dominínio foi empregado o MongoDB[26] um banco NoSQL Document, que provê índices, capacidade de consultas simples e uma estrutura de dados mapeada em um esquema de dados flexível. Esta flexibilidade na estrutura de dados é importante porque os atributos de produtos podem mudar ao longo do tempo e se deseja que não haja grandes impactos devido a estas mudanças.

[20] https://gradle.org/
[21] https://maven.apache.org/
[22] https://projects.spring.io/spring-boot/
[23] http://www.oracle.com/technetwork/java/javase/overview/java8-2100321.html
[24] http://www.oracle.com/technetwork/pt/articles/java/streams-api-java-8-3410098-ptb.html
[25] https://spring.io/guides/gs/spring-boot-docker/
[26] https://www.mongodb.com/

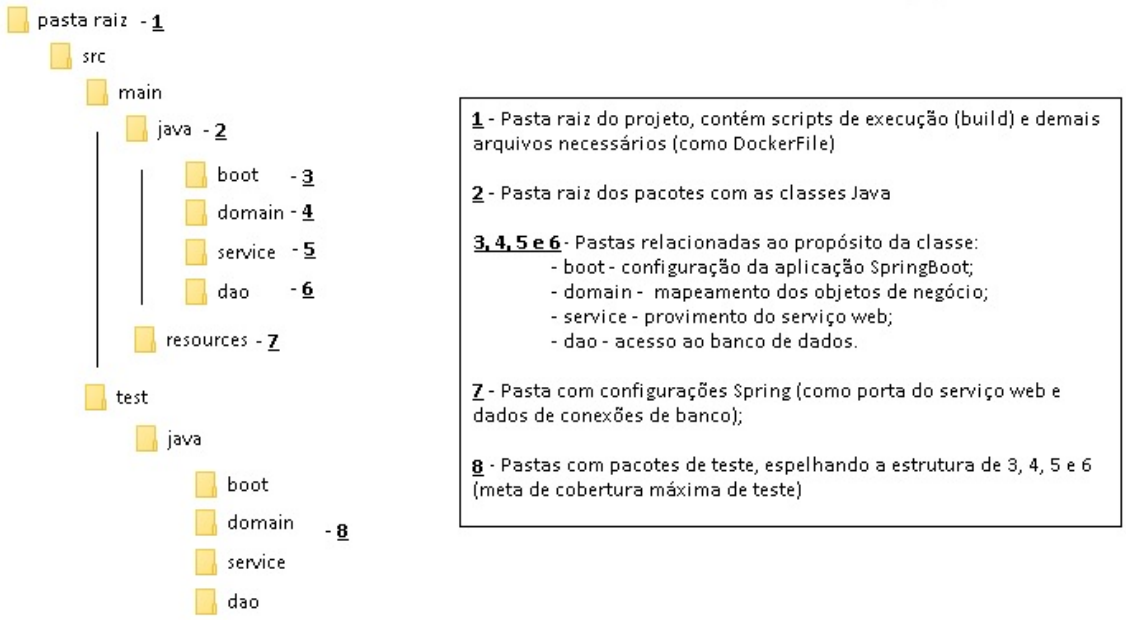


Figura 1.20. Estrutura de pastas (Microsserviços)

Montagem deste ambiente: A estrutura de contêiner (Seção 1.2.5) foi utilizada para facilitar a gestão da infraestrutura de cada componente, bem como prover uma configuração que permitisse que todos os componentes pudessem acessar, a partir de seus ambiente, os demais contêineres pelo atributo *name* registrado no Docker no momento de sua criação. Para tal, foi utilizado o Comando 1 para criar uma rede bridge, a qual será utilizada por este e pelos demais componentes. Nesse caso a execução sem parâmetros já instancia o servidor.

Comando 1 Criando rede bridge

```
sudo docker network create -d bridge grupo_microsservicos
```

Para a criação do ambiente para o Mongo, é necessário baixar uma imagem padrão e executar o comando de criação do contêiner na rede criada (Comando 2).

Para confirmar a execução basta criar uma sessão iterativa rodando o Comando 3. Seguido de *db* (verifica base corrente) ou qualquer outro comando.

DB II - Carro de Compras

Para este domínio, optou-se pelo Redis[27], um banco NoSQL Chave-Valor, opção escolhida devido ao acesso aos dados ser predominantemente via chave primária. A API utilizada para o consumo de dados do Redis também permite a busca em valores em dados serializados.

[27] `https://redis.io/`

Comando 2 Comandos para baixar imagem padrão do MongoDB e criar container na rede

```
docker pull mongo:3.7-jessie
```

```
docker run --name mongodb -p 27017:27017 -d
--network grupo_microsservicos mongo:3.7-jessie
```

Parâmetros indicam:
`name`: *nome da imagem(redisdb)*;
`p <porta externa>:<porta interna>`: *expõe a porta rodando no contêiner na porta da máquina hospedeira*
`d`: *o modo de execução detached*
`network grupo_microsservicos`: *rede da qual o contêiner fará parte*
`mongo:3.7-jessie`: *nome da imagem docker baixada na máquina hospedeira*

Comando 3 Iniciar seção no mongo

```
docker exec -it mongodb mongo
```

Montagem Para a criação do banco executar Comando 4 para baixar uma imagem padrão do redes e executar o comando de criação do contêiner. Para confirmar a execução basta rodar o Comando 5.

Comando 4 Comandos para baixar imagem padrão do redis e criar container na rede

```
docker pull redis:3.2.11
```

```
docker run --name redisdb -p 6379:6379 -d --network
grupo_microsservicos redis:3.2.11 redis-server
--appendonly yes
```

Parâmetros semelhantes ao do Comando 2:
`redis-server -appendonly yes`: *comando para instanciar o servidor*

DB III - Cadastro de Clientes

Para este domínio, optou-se pelo SGBD MySQL[28], que provê garantias de atomicidade, consistência, isolamento e durabilidade (ACID) nas suas transações.

Montagem Para a criação do banco, executar Comando 6 para baixar uma imagem padrão e criar contêiner na rede criada. observe a senha gerada usando `docker logs mysqldb`. Conecte ao MySQL, crie um banco de dados e um usuário para a aplicação, executando Comando 7.

[28] `https://www.mysql.com/`

Comando 5 Iniciar sessão iterativa no redis

```
docker run -it --network grupo_microsservicos
--rm redis:3.2.11 redis-cli -h redisdb -p 6379
```

Parâmetros:
`it`: *roda no modo iterativo*;
`network grupo_microsservicos`: *rede da qual o contêiner fará parte*
`rm`: *máquina "efêmera"- após a execução ou em eventual parada o contêiner é removido*
`redis:3.2.11`: *nome da imagem docker*
`redis-cli -h redisdb -p 6379`: *comando para instanciar cliente*
Seguido de `KEYS *` (verifica chaves no servidor) ou qualquer outro comando

Comando 6 Comandos para baixar imagem padrão do MySQL e criar container na rede

```
docker pull mysql/sqlserver:5.7.22
```

```
docker run --name mysqldb -p 3306:3306 -d
--network grupo_microsservicos mysql/mysql-server:5.7.22
```

1.4.3. Microsserviços de Dados

Os microsserviços de dados formam a base dessa arquitetura: são os gerenciadores dos dados do negócio e proveem uma API Rest (Seção 1.2.2.2) aos demais serviços interessados em consumir informações disponibilizadas por eles. É necessário que seus bancos de dados já estejam ativos e configurados na mesma rede Docker em que serão instanciados os seus contêineres (Seção 1.4.2.3). O repositório indicado na Seção 1.4.2.1 contém todos os arquivos necessários a geração desse componente.

MS I - Catálogo de Produtos

Esse microsserviço é responsável pelo acesso ao banco de dados de Catálogo de Produtos, incluindo informações de Fornecedor. Este serviço foi desenvolvido empregando Spring Boot e acesso ao MongoDB[29].

O script Gradle para este microsserviço inclui[30]:

- Dependências para a execução do próprio script: Inclui a referência para a busca das bibliotecas e são indicadas as dependências para a execução do script.
- Configuração do artefato, da imagem e do contêiner Docker: Indica o nome do artefato (jar) gerado e faz referência ao Docker file (que estará na pasta raiz) a ser utilizado no processo de build. Além disso, inclui instruções para o posicionamento desse arquivo a fim de que seja devidamente referenciado no Dockerfile.

[29] O site Spring IO provê um guia `https://spring.io/guides/gs/accessing-data-mongodb/` para demonstrar a criação de um ambiente Spring Boot com MongoDB

[30] `https://bitbucket.org/luis-villaca/tutorial_2018_sbsi/wiki/ms-produtos-script-gradle`

Comando 7 Comandos para criar banco de dados e usuário no MySQL

```
docker exec -it mysqldb mysql -uroot -p<senha gerada>

create database clientdt
create user 'usuariocli'@'%' identified by 'senhacli';
grant all on *.* to 'usuariocli'@'%';
```

- Configuração das dependências do código da aplicação: Inclui a referência para a busca das bibliotecas dependentes, indicando as versões da JVM e dependências para a execução do script.

Além desse script, o YAML da aplicação[31], localizado na pasta resources (Figura 1.20), indica três configurações: a porta utilizada para o Tomcat embutido no SpringBoot; as configurações de conexão com o banco de dados; e, as configurações do log4j (biblioteca de log) - relacionando os pacotes, o padrão das linhas geradas e a saída padrão do contêiner como destino.

A configuração do Docker para o microsserviço Produtos[32] indica: a imagem a ser utilizada (Linux Alpine com openjdk); a inclusão do arquivo jar (gerado pelo Gradle); e a execução do jar assim que o contêiner for executado.

A partir desta configuração, basta executar os comandos apresentados em Comando 8 para a geração da imagem Docker e execução do contêiner.

Comando 8 Comandos para gerar microsserviço de catálogo de produtos

```
sudo ./gradlew.sh build buildDocker

sudo docker run --name ms01_catalogoprod
--network grupo_microsservicos --rm -p 8081:8080
class_unirio/catalogoprod_microservice:latest
```

O diagrama apresentado na Figura 1.21 indica os principais componentes e suas respectivas dependências, os quais são descritos a seguir de acordo com os estereótipos a eles atribuídos[33].

- **Boot**: Contém a classe principal do SpringBoot, codificada apenas com a instância da aplicação e com referências aos pacotes que auxiliarão no processamento das operações providas pelo serviço.

[31] https://bitbucket.org/luis-villaca/tutorial_2018_sbsi/wiki/ms-produtos-script-application-yml

[32] https://bitbucket.org/luis-villaca/tutorial_2018_sbsi/wiki/ms-produtos-docker-configuration

[33] https://bitbucket.org/luis-villaca/tutorial_2018_sbsi/wiki/ms-produtos-classes

- **Domain**: Contém as classes de domínio. Elas apresentam um quantidade de código reduzida, facilitando o desenvolvimento e a legibilidade do código. O componente Lombok foi utilizado nas anotações a fim de proporcionar esse ganho. Além disso, anotações do Spring Data MongoDB também foram utilizadas para mapear os atributos da classe com a estrutura Document persistida no banco.
- **DAO**: Contém a classe que provê acesso aos dados dos objetos de domínio, a qual também apresenta um quantidade de código reduzida, facilitando o desenvolvimento e a legibilidade.
- **Service**: Contém a classe que implementa as chamadas REST empregando o Spring Data REST.

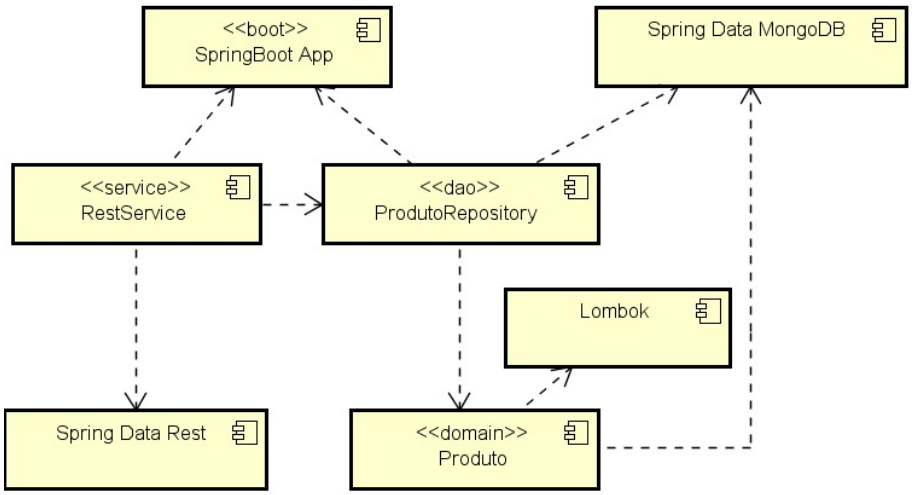


Figura 1.21. Microsserviço de Dados - Catálogo de Produtos (Diagrama de Componentes)

MS II - Carro de Compras

Esse microsserviço é responsável pelo acesso ao banco de dados de Carros de Compra, que contém referências ao campo identificador do Cliente e a campos identificadores dos produtos escolhidos. Em especial, para este serviço, foi necessário configurar um artefato com Spring Data e Redis[34].

Os scripts utilizados para o microsserviço de Carro de Compras[35] são similares aos scripts empregados na construção do microsserviço de Produtos.

O diagrama apresentado na Figura 1.22 indica os principais componentes e suas respectivas dependências, os quais são descritos a seguir de acordo com os estereótipos a eles atribuídos[36].

[34] O site Baeldung provê um guia `http://www.baeldung.com/spring-data-redis-tutorial` para se configurar um artefato com Spring Data e Redis.

[35] `https://bitbucket.org/luis-villaca/tutorial_2018_sbsi/wiki/ms-carrodecompras-scripts`

[36] `https://bitbucket.org/luis-villaca/tutorial_2018_sbsi/wiki/ms-carrosdecompras-classes`

- **Boot**: Contém a classe principal do SpringBoot, codificada apenas com a instância da aplicação e com referências aos pacotes que auxiliarão no processamento das operações providas pelo serviço.
- **Domain**: Assim como no Microsserviço Catálogo de Produtos, as classes de domínio apresentam um quantidade de código reduzida, com anotações Lombok e Spring Data Redis para indicar o Hash a ser utilizado e mapear os atributos do objeto a ser serializado/desserializado.
- **DAO**: A classe CarroDeCompraRepository provê acesso aos dados dos objetos de domínio e apresenta as anotações Spring Data para o provimento das funcionalidades CRUD.
- **Service**: Classe que implementa as chamadas REST (assim como no Microsserviço Catálogo de Produtos). Neste caso, ela provê um serviço de busca sem filtro.

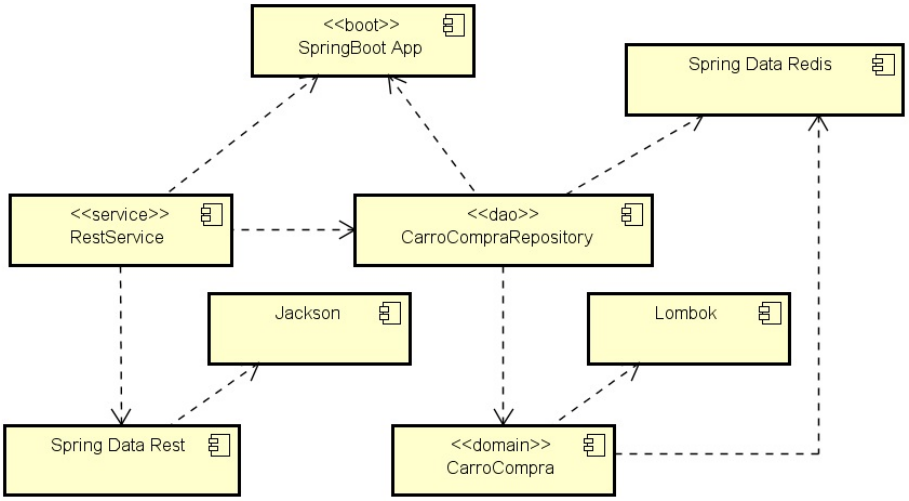


Figura 1.22. Microsserviço de Dados - Carro de Compras (Diagrama de Componentes)

MS III - Cadastro de Clientes

Esse microsserviço é responsável pelo acesso ao banco de dados de Cliente incluindo dados de seu endereço, que são armazenados em uma tabela própria no banco relacional. Este serviço foi desenvolvido empregando Spring Boot e acesso ao MySQL[37].

Os scripts utilizado para o microsserviço de Clientes é similar aos scripts para o microsserviço de Produtos, excetuando-se algumas dependências[38].

O diagrama apresentado na Figura 1.23 indica os principais componentes e suas respectivas dependências, os quais são descritos a seguir de acordo com os estereótipos atribuídos a eles.

[37] O site Spring IO provê um passo-a-passo`https://spring.io/guides/gs/accessing-data-mysql/` para se criar um artefato Spring Boot com MySQL.

[38] `https://bitbucket.org/luis-villaca/tutorial_2018_sbsi/wiki/ms-clientes-scripts`

- **Boot**: Contém a classe principal do SpringBoot, codificada apenas com a instância da aplicação e com referências aos pacotes que auxiliarão no processamento das operações providas pelo serviço.
- **Domain**: Assim como no Microsserviço Catálogo de Produtos, as classes de domínio apresentam um quantidade de código reduzida, com anotações Lombok e nesse caso com Spring Data JPA para mapear os atributos do objeto a ser persistido.
- **DAO**: Contém a classe que provê acesso aos dados dos objetos de domínio. Anotações Spring Data foram usadas para o provimento das funcionalidades CRUD.
- **Service**: Classe que implementa as chamadas REST (assim como no Microsserviço Catálogo de Produtos). Neste caso, ela provê um serviço de busca com filtro customizado.

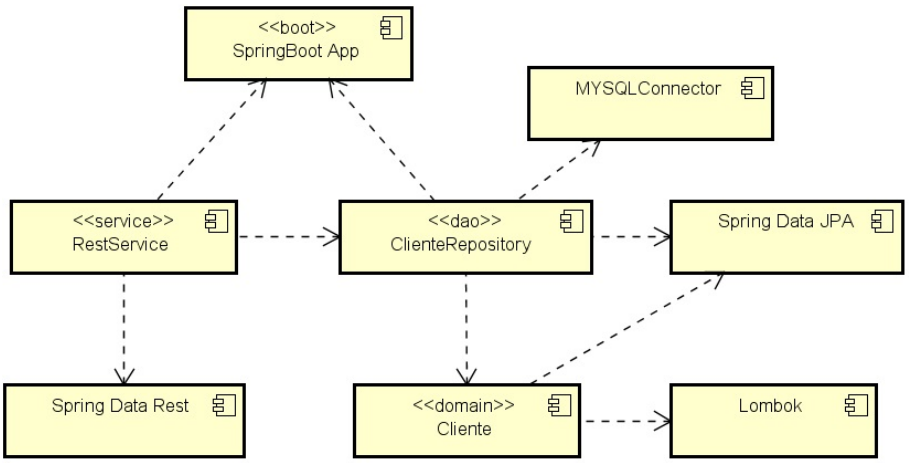


Figura 1.23. Microsserviço de Dados - Cadastro de Clientes (Diagrama de Componentes)

1.4.4. Microsserviços de Integração

Esses serviços atuam como consumidores dos microsserviços de dados, mediante a API disponibilizada ou outro mecanismo. São responsáveis por integrá-los e processá-los a fim de compor uma visão consolidada (no caso do serviço Mediador) ou ampliada (por meio do Processador de Eventos) a partir das informações existentes.

MS Integração I - Mediador

Esse microsserviço é responsável por prover uma visão GraphQL (Sessão 1.2.3) a partir da API web dos Microsserviços de Dados. Nesse caso não há acesso direto a uma base de dados. Os acessos são feitos via requisições REST aos microsserviços de dados, e as instâncias das classes de negócios são mapeadas a partir de suas respostas. Utiliza-se um Schema Graphql[39] para mapear estruturas de dados dessas instâncias e assim compor respostas às requisições.

[39] `https://bitbucket.org/luis-villaca/tutorial_2018_sbsi/wiki/ms-mediador-esquema-graphql`

Os scripts utilizados para o microsserviço Mediador[40] são similares aos scripts empregados na construção do microsserviço de Produtos.

O diagrama apresentado na Figura 1.24 indica os principais componentes e suas respectivas dependências os quais são descritos a seguir de acordo com os estereótipos a eles atribuídos[41].

- **Boot**: Contém a classe principal do SpringBoot, codificada apenas com a instância da aplicação e com referências aos pacotes que auxiliarão no processamento das operações providas pelo serviço.

- **Domain**: As classes de domínio são equivalentes às já apresentadas para os microsserviços de Produtos, Clientes e Carros de Compra, mas sem a necessidade de anotações que mapeiam informações de persistência dos atributos.

- **DAO**: Classes que proveem acesso aos dados dos objetos de domínio atuam, nesse caso, se conectando a um microsserviço de dados via chamadas REST.

- **DataFetcher**: Componentes chamados pelo mediador para a obtenção dos dados.

- **Service**: Corresponde a classe QueryMediator que é a classe mediadora que disponibiliza a API GraphQL.

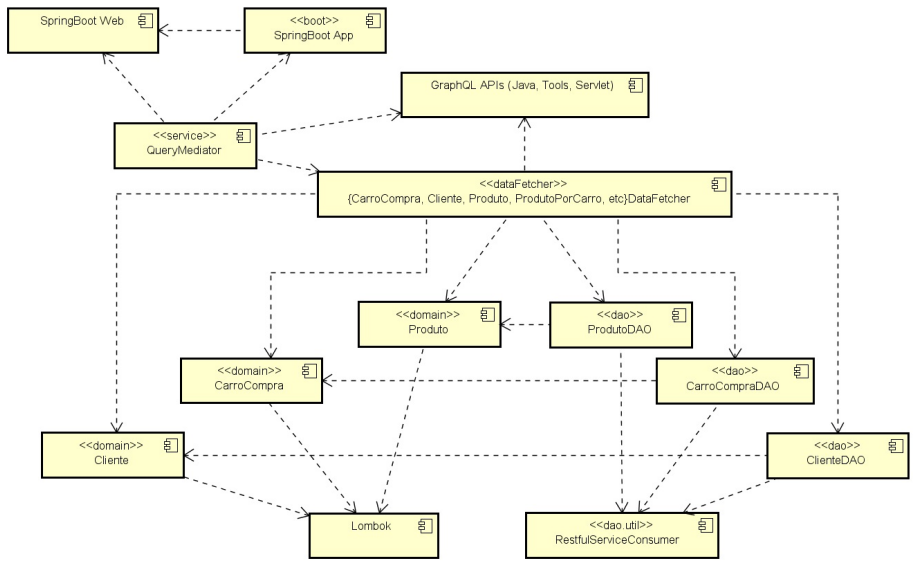


Figura 1.24. Microsserviço de Integração - Mediador (Diagrama de Componentes)

[40] https://bitbucket.org/luis-villaca/tutorial_2018_sbsi/wiki/ms-mediador-scripts

[41] https://bitbucket.org/luis-villaca/tutorial_2018_sbsi/wiki/ms-mediador-classes

MS Integração II - Processador de Eventos

Esse serviço é proposto como um exercício a fim de que sejam praticados os conceitos apresentados. Foi concebido a partir de um cenário em que surge um novo requisito: *trazer a média mensal de compras por país nos últimos 12 meses*.

Na atual arquitetura há algumas limitações que devem ser consideradas:

- O microsserviço de Carro de Compras não provê um mecanismo de busca por data, nem de busca ordenada. Logo, para utilizá-la todos os registros devem ser buscados e suas datas devem ser testadas.
- Alterar o microsserviço de Carro de Compras para prover uma busca por data não é suficiente, uma vez que ele não possui informações de clientes, de modo que uma nova busca seria necessária pelo serviço mediador para complementar o filtro.
- O microsserviço de Cadastro de Clientes possui o método *buscaPorNacionalidade*. No entanto, podem ser retornados muitos clientes. Logo, dependendo do número de clientes, da velocidade de processamento e da latência, pode-se optar por trazer todos as instâncias, manter um cache e sincronizar de tempos em tempos - ou, a cada registro de Carro de Compra aplicável, buscar o cliente (pelo seu id) o que também pode demandar muito tempo.
- Pode-se chegar à conclusão de que o mediador está onerado e que adicionar a estratégia Event Data Pump à arquitetura seja a melhor solução e, dessa forma, ter uma solução que permitirá que novos requisitos de integração para o cenário de Relatório sejam tratados com mais eficiência.

A proposta do exercício é utilizar uma plataforma de fila (sugerimos a utilização da RabbitMQ) para coletar os eventos de alteração dos dados dos microsserviços de dados, e prover um novo serviço de integração que será responsável por manter uma visão materializada de informações de compras e países (ou de mais informações se necessário). Esse serviço tem a atribuição de agregar dados continuamente, a partir dos eventos de alteração, e de gerar as estatísticas de forma contínua[42].

Nesse caso todos os serviços deverão contribuir para essa solução. Os microsserviços de dados deverão publicar suas alterações, e mesmo o nó Mediador pode ajustar o seu schema GraphQL com dados de Relatório (com, por exemplo, as médias mensais de valores de carros de compra por país), devendo para isso consumir também os dados gerados pelo nó Processador de Eventos, por meio de sua API.

1.5. Considerações Finais

O objetivo deste trabalho foi prover uma visão teórica da arquitetura de microsserviços e um guia prático para o desenvolvimento de artefatos nessa abordagem de arquitetura de modo a explicitar benefícios e desafios dessa abordagem em cenários específicos.

[42]Uma proposta de arquitetura para este novo serviço é apresentado em `https://bitbucket.org/luis-villaca/tutorial_2018_sbsi/wiki/ms-processadordeeventos-arquitetura-de-solucao`

Inicialmente foram apresentados os conceitos fundamentais das principais tecnologias relacionadas a microsserviços, seguidos dos desafios observados num contexto em que há necessidade de se relacionar dados a partir de fontes de informação distintas e distribuídas. Foram apresentadas as estratégias atualmente publicadas para a resolução desses desafios, destacando seus aspectos positivos e negativos.

Um cenário para a implementação de duas dessas estratégias foi elaborado para demonstrar suas características:

- Data Retrieval Via Service Call: Foi implementado o microsserviço Mediador, o qual consome informações de outros microsserviços via API Web para prover uma visão consolidada de dados;

- Event Data Pump: A implementação do microsserviço Processador de Eventos foi proposta como exercício a fim de praticar os conceitos apresentados na construção do microsserviço Mediador e de praticar o uso da estratégia Event Dasta Pump. O microsserviço Processador de Eventos subscreve e consome os eventos publicados pelos outros microsserviços e mantém sua própria visão materializada dos dados relevantes para seu contexto de processamento.

Nesse trabalho, foram exercitados cinco dos sete princípios de microsserviços (Seção1.2.4.3):

- Interfaces com granularidade fina: Os microsserviços de dados correspondem a unidades com responsabilidades específicas e independentes encapsulando tanto lógica de processamento como lógica de dados.

- Práticas de desenvolvimento orientadas ao negócio: Uso de DDD na segregação dos domínios para definição dos microsserviços de dados;

- Entrega contínua descentralizada: Várias tecnologias para automação foram empregadas na construção e implantação dos componentes;

- Contêineres leves: Utilização e configuração de imagens e contêineres Docker nas arquiteturas desenvolvidas;

- Múltiplos paradigmas: Foram utilizadas três tecnologias de bancos de dados (isto é, relacional (MySQL), orientada a documentos (MongoDB) e chave-valor (Redis)) para persistência bem como todas as tecnologias necessárias para construir os microsserviços que disponibilizavam acesso aos dados providos pelas mesmas.

O cenário proposto foi composto por nove componentes. A partir deles foi demonstrado algumas características buscadas em microsserviços, como pouco acoplamento e alta coesão, facilitando o desenvolvimento individual dos componentes.

A integração via API, na estratégia Data Retrieval Via Service, apresenta limitações, como a dificuldade de se estimar quão abrangente deve ser a exposição dos serviços.

O tratamento das operações que relacionam informações de diferentes bases de dados, cujos serviços estão distribuídos na rede, pode demandar uma abordagem diferenciada - como o caso da estratégia Event Data Pump.

A integração via Event Data Pump apresenta pontos positivos e negativos, assim como a estratégia Data Retrieval Via Service Call, conforme apontado na Seção1.3.3.

Portanto, por meio da discussão dos conceitos de microsserviços e de sua aplicação num cenário de persistência poliglota foi possível apresentar as alternativas que visam resolver aspectos importantes na construção de sistemas distribuídos e para integração de dados.

Referências

[Booths et al. 2004] Booths, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. (2004). Web services architecture. `http://www.w3.org/TR/ws-arch/`. Acessado em 06/05/2018.

[Brewer 2000] Brewer, E. A. (2000). Towards robust distributed systems. In *PODC*, volume 7.

[Collins et al. 2002] Collins, S. R., Navathe, S., and Mark, L. (2002). Xml schema mappings for heterogeneous database access. *Information and Software Technology*, 44(4):251–257.

[Di Francesco et al. 2017] Di Francesco, P., Malavolta, I., and Lago, P. (2017). Research on architecting microservices: Trends, focus, and potential for industrial adoption. In *Software Architecture (ICSA), 2017 IEEE International Conference on*, pages 21–30. IEEE.

[Erl 2005] Erl, T. (2005). *Service-oriented architecture: concepts, technology, and design*. Pearson Education India.

[Evans 2004] Evans, E. (2004). *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional.

[Fehling et al. 2014] Fehling, C., Leymann, F., Retter, R., Schupeck, W., and Arbitter, P. (2014). *Cloud computing patterns: fundamentals to design, build, and manage cloud applications*. Springer Science & Business Media.

[Fielding 2000] Fielding, R. T. (2000). Rest: architectural styles and the design of network-based software architectures. *Doctoral dissertation, University of California*.

[Fowler 2015] Fowler, M. (2015). Polyglot persistence. 2015. `https://martinfowler.com/bliki/PolyglotPersistence.html`. Acessado em 04/05/2018.

[Fowler and Lewis 2014] Fowler, M. and Lewis, J. (2014). Microservices. https://martinfowler.com/articles/microservices.html. Acessado em 02/02/2018.

[Ghawi and Cullot 2007] Ghawi, R. and Cullot, N. (2007). Database-to-ontology mapping generation for semantic interoperability. In *Third International Workshop on Database Interoperability (InterDB 2007)*, volume 91.

[Ghebremicael 2017] Ghebremicael, E. S. (2017). Transformation of rest api to graphql for opentosca. Master's thesis, Institute of Architecture of Application Systems - University of Stuttgart, Stuttgart, Germany.

[Gilpin 2015] Gilpin, M. (2015). From the field: The first annual canonical model management forum. forrester blogs. `https://go.forrester.com/blogs/10-03-15-from_the_field_the_first_annual_canonical_model_management_forum`. Acessado em 06/05/2018.

[Gu and Lago 2007] Gu, Q. and Lago, P. (2007). A stakeholder-driven service life cycle model for soa. In *2nd international workshop on Service oriented software engineering: in conjunction with the 6th ESEC/FSE joint meeting*, pages 1–7. ACM.

[Heuser 2009] Heuser, C. A. (2009). *Projeto de banco de dados: Volume 4 da Série Livros didáticos informática UFRGS*. Bookman Editora.

[Humble and Farley 2010] Humble, J. and Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader)*. Pearson Education.

[Hüttermann 2012] Hüttermann, M. (2012). *DevOps for Developers*, volume 1. Springer.

[Josuttis 2007] Josuttis, N. M. (2007). *SOA in practice: the art of distributed system design*. "O'Reilly Media, Inc.".

[Marks and Bell 2008] Marks, E. A. and Bell, M. (2008). *Service Oriented Architecture (SOA): a planning and implementation guide for business and technology*. John Wiley & Sons.

[Mehta and Heineman 2002] Mehta, A. and Heineman, G. T. (2002). Evolving legacy system features into fine-grained components. In *Proceedings of the 24th international Conference on Software Engineering*, pages 417–427. ACM.

[Merkel 2014] Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2.

[Messina et al. 2016] Messina, A., Rizzo, R., Storniolo, P., and Urso, A. (2016). A simplified database pattern for the microservice architecture. *The Eighth International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA)*.

[Newman 2015] Newman, S. (2015). *Building microservices: designing fine-grained systems*. "O'Reilly Media, Inc.".

[Pautasso et al. 2008] Pautasso, C., Zimmermann, O., and Leymann, F. (2008). Restful web services vs. big'web services: making the right architectural decision. In *Proceedings of the 17th international conference on World Wide Web*, pages 805–814. ACM.

[Rajkovic et al. 2013] Rajkovic, P., Jankovic, D., and Milenkovic, A. (2013). Using cqrs pattern for improving performances in medical information systems. In *Balkan Conference in Informatics (BCI)*, volume 1036, pages 86–91.

[Richards 2015] Richards, M. (2015). Microservices vs. service-oriented architecture.

[Richardson 2015] Richardson, C. (2015). Introduction to microservices. `https://www.nginx.com/blog/introduction-to-microservices/`. Acessado em 3/5/2018.

[Richardson 2016] Richardson, C. (2016). Microservice architecture patterns and best practices. `http://microservices.io`. Acessado em 3/5/2018.

[Sadalage and Fowler 2012] Sadalage, P. J. and Fowler, M. (2012). *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education.

[Scheibel 2016] Scheibel, G. (2016). A software architecture for storing trajectories using polyglot persistence (in portuguese). *Master dissertation, University of the State of Santa Catarina*.

[Vernon 2013] Vernon, V. (2013). *Implementing domain-driven design*. Addison-Wesley.

[Viennot et al. 2015] Viennot, N., Lécuyer, M., Bell, J., Geambasu, R., and Nieh, J. (2015). Synapse: a microservices architecture for heterogeneous-database web applications. In *Proceedings of the Tenth European Conference on Computer Systems*, page 21. ACM.

[Villaca et al. 2018] Villaca, L. H., Azevedo, L. G., and Baião, F. (2018). Query strategies on polyglot persistence in microservices. In *2018 ACM SIGAPP*. ACM.

[Vinoski 2002] Vinoski, S. (2002). Putting the"web"into web services. web services interaction models, part 1. *IEEE Internet Computing*, 6(3):89–91.

[Wampler and Clark 2010] Wampler, D. and Clark, T. (2010). Multiparadigm programming: guest editors' introduction. *IEEE Software*, 27(5):2–7.

[Zimmermann 2016] Zimmermann, O. (2016). Microservices tenets. *Computer Science-Research and Development*, pages 1–10.

Biografia dos autores

Este trabalho foi escrito pelos seguintes autores.

Luís H. N. Villaça é Bacharel em Informática pela UFRJ (1999) e Tecnólogo em Processamento de Dados pelo Centro de Ensino Superior de Juiz de Fora (1994). Cursa o Mestrado na Universidade Federal do Estado do Rio de Janeiro (PPGI/UNIRIO). Trabalha há 20 anos como como analista/desenvolvedor (1996-2006) (CNPQ, Avanti Prima Engenharia, Procwork, Quality Systems, EDS), líder técnico e arquiteto JEE (IBM-2006-2012), auditor interno e arquiteto de SI (Petrobras Transportes-2012-2018). Desde de Agosto de 2017 trabalha em projeto e desenvolvimento de sistemas em arquitetura de Microsserviços, o qual é o tema principal da sua dissertação de mestrado. Detalhes em `http://lattes.cnpq.br/2362414271105479`.

Antônio Fonseca Pimenta Júnior é Mestre (2017) em Informática pelo Programa de Pós-Graduação em Informática (PPGI) pela Universidade Federal do Estado do Rio de Janeiro (UNIRIO) e Bacharel em Ciência da Computação (2009) pela Universidade Estadual de Santa Cruz (UESC). Trabalha há 8 anos na área de analise de sistemas como analista/desenvolvedor (CPM Braxis 2010-2013); Analista, Líder tecnico e Gerente de projetos na Dataprev desde 2013. Detalhes em `http://lattes.cnpq.br/1614311113198961`.

Leonardo G. Azevedo é pesquisador da IBM Research Brazil desde 2013 e professor da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) desde 2009. Leonardo é Doutor (2005) e Mestre (2001) em Ciências, em Engenharia de Sistemas e Computação pela COPPE(UFRJ) e Bacharel em Informática pela UFRJ (2000). Leonardo tem mais de 20 anos de experiência em pesquisa e desenvolvimento de sistemas tendo atuado em diversas empresas e para o governo. Sua pesquisa concentra-se nos seguintes temas: Sistemas Distribuídos; Arquitetura Orientada a Serviços (SOA); Microsserviços; Gestão de Processos de Negócio (BPM); Computação Cognitiva; e, Bancos de Dados Espaciais. Ele publicou mais de 90 artigos científicos nestes temas em revistas e conferências nacionais e internacionais. Detalhes em `http://researcher.ibm.com/researcher/view.php?person=br-lga` e `http://lattes.cnpq.br/7214791464543522`.

Capítulo 2

Da Pesquisa à Inovação em Sistemas de Informação

Renata Mendes de Araujo, Luciana de Oliveira Vilanova Chueri

Abstract

The purpose of this course is to enable students to identify perspectives and opportunities for innovation from their scientific research, stimulating the generation of intellectual property, innovative products and entrepreneurship. The course clarifies on the concepts and processes of innovation, on the role of scientific research in the innovation and entrepreneurship national strategies, especially in Information Systems research area, and presents useful activities and tools for researchers to identify and plan for innovation.

Resumo

O objetivo deste curso é capacitar os alunos a identificar perspectivas e oportunidades para a inovação a partir de suas pesquisas, estimulando a geração de propriedade intelectual, produtos inovadores e o empreendedorismo. O trabalho esclarece sobre os conceitos e processos de inovação, sobre o papel da pesquisa científica nas estratégias de inovação nacionais, em particular a pesquisa em Sistemas de Informação, e apresenta atividades e instrumentos úteis para os pesquisadores identificarem e planejarem a inovação.

2.1. Introdução

Vemos no cenário nacional e global um crescente movimento no sentido de investir recursos e esforços nos processos de inovação tecnológica como estratégia para solução dos problemas nacionais, globais e de enfrentamento aos desafios de sustentabilidade do mundo. O processo de pesquisa científica possui interseções fundamentais para os processos de inovação e vice-versa, e a distância entre a pesquisa científica e o ambiente produtivo é um dos grandes desafios para a prática sistemática da inovação em larga escala. Uma oportunidade de enfrentar este desafio é a formação de pesquisadores com mentalidade inovadora, capacitados a compreender as particularidades e desafios dos processos de inovação tecnológica, bem como instrumentá-los para complementar seus processos de pesquisa científica com atividades relacionadas à inovação.

O objetivo deste trabalho é estimular nos pesquisadores uma visão empreendedora – não no sentido empresarial, mas no sentido de motivada a gerar conhecimento e soluções de impacto e relevância real na área de Sistemas de Informação – e ao mesmo tempo capacitá-los em conceitos, processos e técnicas úteis para identificar a inovação em suas pesquisas. Ao final deste trabalho, os alunos serão capazes de: i) entender o conceito de inovação exemplificando produtos inovadores e interpretando seus impactos na economia e sociedade; ii) entender o sistema de inovação e difusão tecnológica, em particular, o sistema de inovação brasileiro; iii) entender os conceitos e processos da propriedade intelectual no Brasil explicando sua aplicabilidade em projetos de pesquisa; iv) analisar as relações dos processos de inovação com os processos de pesquisa acadêmica concluindo quanto ao papel da pesquisa no sistema de inovação nacional; v) analisar atividades e projetos de pesquisa em Sistemas de Informação concluindo quanto ao seu potencial de inovação; vi) reconhecer o potencial para inovação a partir de sua pesquisa científica em Sistemas de Informação; vii) aplicar o processo de levantamento de informação tecnológica/propriedade intelectual executando busca em bases de patentes nacionais e internacionais; e viii) elaborar apresentação sobre o potencial de inovação em pesquisa acadêmica.

Este capítulo se organiza da seguinte forma: na Seção 2.2 apresentamos os conceitos, tipos e processos de inovação; na seção 2.3 esclarecemos a respeito dos ecossistemas nacionais de inovação e o papel das universidades neste contexto; na Seção 2.4 traçamos as relações e interseções entre os processo de pesquisa científica e de inovação, apontando questões para reflexão a respeito de uma visão de pesquisa dirigida à inovação; na seção 2.5 apresentamos os conceitos de propriedade intelectual e sua relação com os produtos da pesquisa científica; na Seção 2.6 propomos um processo para levantamento de conhecimento tecnológico sob a forma de patentes; na seção 2.7 refletimos sobre como o exercício de uma mentalidade inovadora pode ampliar o impacto de atuação e das pesquisas; na Seção 2.8 concluímos o capítulo.

2.2. Inovação

Quando ouvimos a palavra "inovação", com muita frequência nossa mente é levada a pensar em algo um tanto quanto futurista, surpreendente, com certo ar high-tech, que vai mudar nossa vida para sempre. A palavra inovação pode também vir à nossa mente invariavelmente associada a pessoas com mentalidade genial, que de alguma forma mágica transformaram uma brilhante ideia em produtos para sempre consumidos e adquiriram enormes fortunas. Inovação também nos remete a empresas que surgiram com força total em nossas vidas, cuja marca torna-se até mais forte do que seus próprios produtos. Pensando assim, a inovação muitas vezes se torna um conceito aparentemente inalcançável para a maioria das pessoas, sendo considerada como o resultado da sorte, ou da genialidade, ou de muito investimento, ou das três coisas juntas.

Talvez pelo excesso de uso do termo ou falta de precisão no seu conceito, a inovação também costuma ser entendida simplesmente como algo novo, inusitado, o resultado da criatividade colocada em ação para a resolução de problemas ou para a exploração de oportunidades. Se por um lado, a inovação, sim, sempre nos surpreende por sua criatividade e capacidade de reconhecer oportunidades, por outro lado ela só pode ser entendida como tal quando colocada em prática e associada ao seu impacto, ou seja, em sua efetividade na solução de problemas e na escala de exploração econômica

ou uso que adquire. Mais recentemente, a inovação também ganha conotações voltadas à solução de problemas sociais [Chueri 2017] e ambientais [Lemme 2017], o que leva a novas definições como – algo novo com ganho econômico, social ou ambiental.

Desta forma, as principais conceituações do termo, historicamente advindas do domínio da Economia, distinguem claramente a invenção - algo novo -, da inovação – *algo novo com ganho econômico ou impacto social*. Por esta razão que, boa parte das pesquisas científicas desenvolvidas em todos os países, embora por natureza apresentem a ideia de "algo novo" e avançado em relação ao estado-da-arte em uma área de pesquisa, não podem ser conceituadas como inovação. Para que o resultado de uma pesquisa se torne uma inovação, há um passo importante que é transformar este algo novo que se mostrou eficiente em um contexto em um produto que possa ser consumido no mercado e que traga impactos econômicos e/ou sociais.

2.2.1 Tipos de Inovação

Uma vez entendida esta distinção importante, vamos aos detalhes. Mesmo que especificamente voltada à exploração econômica ou ao desenvolvimento social, é preciso definir o que pode significar "algo novo", ou seja, quais os tipos de inovação que podem surgir.

Uma primeira forma de compreender a inovação é pelo seu **objeto**. Segundo Schumpeter [apud: Tidd, Bessant e Pavitt 2008], por exemplo, uma inovação pode compreender: i) a introdução de um novo produto no mercado ou a mudança qualitativa em um produto existente; ii) a inovação de um processo que seja novidade para uma indústria; iii) a abertura de um novo mercado; iv) o desenvolvimento de novas fontes de suprimento de matéria-prima ou outros insumos; ou v) mudanças na organização industrial ou em seu paradigma.

Tabela 2.1. Exemplos de Inovação (extraídos de Tidd, Bessant e Pavitt (2008))

Tipo de Inovação	Exemplos
Inovação de produto - mudanças nas coisas (produtos/serviços) que uma empresa oferece	Coca Zero, Smart car
Inovação de processo – mudanças na forma em que os produtos/serviços são criados/entregues	Monte seu carro, self-service starbucks, comida à quilo.
Inovação de posição – mudanças no contexto em que produtos/serviços são introduzidos	Lucozade – produto para convalescentes que se tornou energético popular na Inglaterra
Inovação de paradigma – mudanças nos modelos mentais subjacentes que orientam o que a empresa faz.	Produção artesanal para produção em massa de carros (Ford); café gourmet.

Uma vez entendido o que é o "algo" da inovação, outra dimensão necessária para classificá-la diz respeito ao seu **grau de novidade**, ou seja, o quanto a inovação é "nova". O grau de novidade de uma inovação é um assunto bastante comentado, e significa o quanto o produto existente é novo no mercado ou na sociedade. As

inovações ditas **radicais** são aquelas trazidas por produtos ou serviços antes inexistentes no mercado e que provocam grandes mudanças no mundo, nos mercados e na sociedade. Um exemplo de inovação radical é o advento da WWW, o surgimento dos telefones celulares, a criação das máquinas fotográficas digitais. O desenvolvimento de inovações radicais envolve muitos riscos, e em alguns casos, pode-se nem sequer ter total controle sobre o que se está produzindo e o efeito que causará. Na outra ponta deste espectro relacionado à novidade, estão as inovações **incrementais**, que são aquelas inovações que preenchem continuamente o processo de mudança, quer seja em uma organização, região, país ou no mundo. Por exemplo, as constantes alterações nos dispositivos celulares, são inovações incrementais. Inovações radicais tendem a ter alto impacto e, consequentemente, possuem potencial de resultados comerciais muito altos. Já as inovações incrementais, partem de algo já conhecido que se deseja aprimorar, possuem menor risco em seu desenvolvimento, e embora também apresentem resultados esperados positivos, são em, em geral, em escala menor.

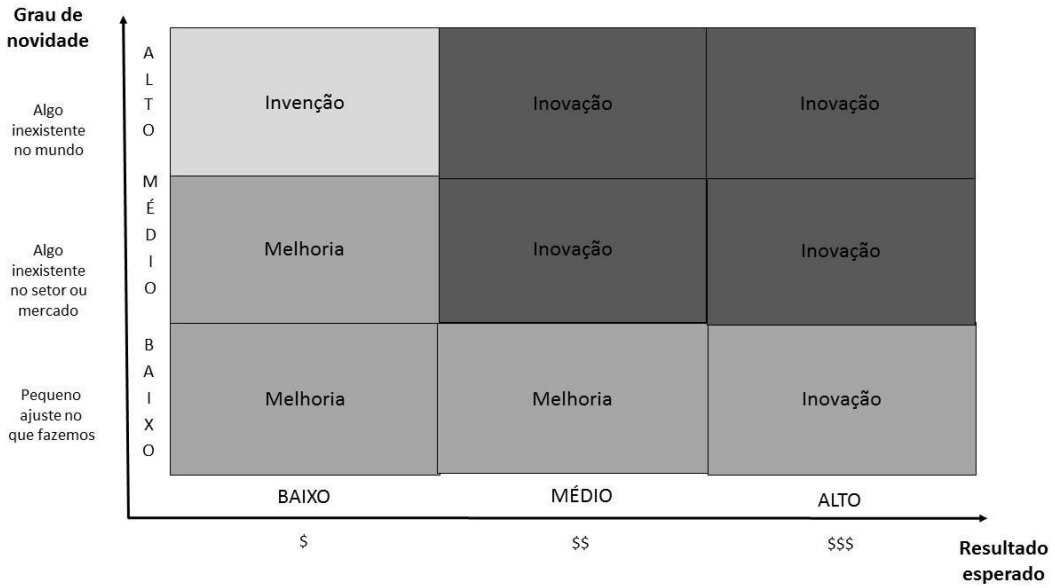


Figura 2.1. Inovação segundo sua intensidade (adaptado de Scherer e Carlomango (2009))

Este quadro nos ajuda muito a pensar a respeito do grau de inovação de nossas ideias, quer sejam produtos ou processos, principalmente quando resultados de nossas pesquisas científicas. Posicionar os produtos de uma pesquisa neste quadro pode nos ajudar a refletir sobre o potencial de inovação do que estamos produzindo e traçar caminhos para explorá-lo durante ou depois de concluídas nossas pesquisas.

2.2.2. Inovação Tecnológica

Dentre os diversos contextos onde as iniciativas de inovação surgem no mundo, o conceito de inovação tecnológica é um dos que mais têm sido explorados. Em grande parte, pelo fato de que a tecnologia, em seu sentido mais amplo, há séculos tem sido o principal agente de avanços científicos, econômicos e sociais. O mercado, as economias globais, as organizações e os indivíduos formam cada vez mais ecossistemas que interagem, são apoiados e sobrevivem com o uso de tecnologia.

Desta forma, o conceito de **inovação tecnológica** tem sido amplamente utilizado para determinar um tipo de inovação voltada a melhorias tecnológicas em produtos e processos, com impactos econômicos, financeiros e comerciais. Para garantir um entendimento global deste conceito e uma avaliação uniforme dos resultados de inovação ao redor do mundo, manuais internacionais foram criados como o Manual de Oslo (vide referências ao final do capítulo), elaborados pela OCDE[1].

O Manual de Oslo define o conceito de Inovações Tecnológicas em Produtos e Processos (TPP) como: *"... implantações de produtos e processos tecnologicamente novos e substanciais melhorias tecnológicas em produtos e processos. Uma inovação TPP é considerada implantada se tiver sido introduzida no mercado (inovação de produto) ou usada no processo de produção (inovação de processo). Uma inovação TPP envolve uma série de atividades científicas, tecnológicas, organizacionais, financeiras e comerciais. Uma empresa inovadora em TPP é uma empresa que tenha implantado produtos ou processos tecnologicamente novos ou com substancial melhoria tecnológica durante o período em análise."*

Devemos reparar que o Manual se preocupa em esclarecer que inovação só existe quando implantada e que deve ser distinguida em relação à sua natureza: inovações de produtos (e aqui incluem-se bens ou serviços) ou processos. Além disso, precisa apresentar características novas ou melhorias substanciais, ou seja, devem caracterizar seu grau de novidade, conforme discutido anteriormente.

Então, **produtos tecnologicamente novos** são aqueles cujas características tecnológicas ou usos pretendidos diferem daqueles dos produtos produzidos anteriormente. Isto pode envolver tanto tecnologias radicalmente novas, como se basear na combinação de tecnologias existentes em novos usos ou mesmo derivadas do uso de novo conhecimento. **Produtos tecnologicamente aprimorados** são produtos existentes cujo desempenho tenha sido significativamente aprimorado ou elevado. Isto inclui tanto um produto simples aprimorado (em termos de melhor desempenho ou menor custo) através de componentes ou materiais de desempenho melhor; como um produto complexo que consista em vários subsistemas técnicos integrados pode ser aprimorado através de modificações parciais em um dos seus subsistemas.

Inovações tecnológicas de processos, por sua vez, incluem a adoção de métodos de produção novos ou significativamente melhorados, incluindo métodos de entrega dos produtos. Estes métodos podem envolver: mudanças em equipamentos, mudanças na organização da produção, combinação das mudanças anteriores ou podem derivar do uso de novo conhecimento. Os métodos adotados podem ter por objetivo produzir ou entregar produtos tecnologicamente novos ou aprimorados, que não possam ser produzidos ou entregues com os métodos convencionais de produção e/ou aumentar a produção ou eficiência na entrega de produtos existentes.

O Manual também classifica as inovações tecnológicas de acordo com sua **cobertura**: i) máxima: algo novo no mundo; ii) intermediária: algo novo em uma região ou país; ou iii) mínima: algo novo na empresa (incluindo também a inovação tecnológica de processo em atividades secundárias da organização). A figura 2.2 resume as características da inovação tecnológica em produtos e processos (TPP) segundo o Manual de Oslo.

[1] Organização para a Cooperação e Desenvolvimento Econômico

Tabela 2.2. Exemplos de inovações tecnológicas de produto e processo (extraído do Manual de Oslo)

Tipo de produto	Exemplos
Tecnologicamente novos	Os primeiros microprocessadores e gravadores de videocassete foram exemplos de produtos tecnologicamente novos, utilizando tecnologias radicalmente novas.
Tecnologicamente aprimorados	O primeiro toca-fitas portátil, que combinava as técnicas existentes de fita e minifones de cabeça, foi um produto tecnologicamente aprimorado, combinando tecnologias existentes em um novo uso.
Processo	No domínio de bancos, a introdução de cartões inteligentes e cartões de múltiplos propósitos em plástico; novas agências bancária sem qualquer pessoal onde os clientes "fazem normalmente seus negócios" através de terminais de computadores à sua disposição; banco via telefone, que permite aos clientes realizar muitas de suas transações bancárias por telefone, no conforto de seus lares.

Uma dúvida que ainda permanece é: o que seria exatamente uma melhoria substancial em um produto ou processo? A melhor maneira de responder a esta pergunta pode ser tentando entender o que não é considerado como uma melhoria substancial. Por exemplo, mudanças organizacionais - alterações de estruturas organizacionais em empresas, implantação de técnicas de gerenciamento avançado, mudanças em orientações estratégias - não podem ser consideradas como inovações TPP, a princípio, a menos que tragam mudanças mensuráveis em resultados – produtividade ou vendas. Em relação aos produtos, mudanças insignificantes, ou "melhorias criativas" onde a novidade não se refere ao uso ou às características objetivas de desempenho dos produtos ou na forma como são produzidos ou entregues, mas em sua estética ou qualidades subjetivas, também não podem ser consideradas inovações TPP.

2.2.3. Processos de Inovação

Uma vez compreendido o conceito da inovação, se motivados ao seu propósito, surge a pergunta: "como tornar a inovação uma realidade?". A boa notícia é que a inovação é uma ação que pode ser, em algum nível, sistematizada. Atualmente já são conhecidas condições básicas e atividades que compõem os processos voltados à produção da inovação. Como a inovação só pode ser considerada como tal quando tem sucesso, até este ponto, o que existe são os processos de inovação.

Algo que é reconhecidamente sabido no processo de inovar é que inovação é questão de conhecimento - tanto científico, como tecnológico, como empírico. Para que a inovação surja, é preciso conhecer **o que já está posto** a respeito da ciência, da tecnologia e da experiência que existe dentro do tema que se deseja inovar. Inovar também envolve informação e observação do status quo – **o que é hoje** – em termos de mercado, sociedade e possibilidades tecnológicas. A partir daí, a inovação diz respeito à criatividade e criação de coisas novas – **o que pode vir a ser**.

			INOVAÇÃO			NÃO INOVAÇÃO
			Máxima	Intermediária	Mínima	
			Novo no mundo	Novo em uma região ou país	Novo na empresa	Já na empresa
INOVAÇÃO TPP	Tecnologicamente novo	Produto				
		Processo de produção				
		Processo de entrega				
	Significativamente aprimorado tecnologicamente	Produto				
		Processo de produção				
		Processo de entrega				
Outras inovações	Novo ou aprimorado	Puramente organizacional				
Não é inovação	Nenhuma mudança significativa, sem novidade ou outras melhorias criativas	Produto				
		Processo de produção				
		Processo de entrega				
		Puramente organizacional				

Figura 2.2. Caracterizando a inovação [Manual de Oslo 1997]

Figura 2.3. Trajetória de inovação tecnológica da iluminação

Um exemplo que ilustra bem este processo é apontado em [Tidd, Bessant e Pavitt 2008], quando analisamos a trajetória de inovação relacionada à tecnologia de iluminação (Figura 2.3). Quando se analisa esta trajetória, é possível perceber as questões que enumeramos acima. As inovações se dão em sequência, avançando a partir de inovações, tecnologias e conhecimento científico, tecnológico ou empírico anteriores. É possível perceber também que a velocidade com a qual as inovações surgem aumenta com o passar do tempo. No passado, o avanço se dava de forma mais

lenta e em situações mais claras de ruptura, associadas à interseção com avanços tecnológicos em outras áreas, principalmente na passagem do uso do óleo para o carvão e para a eletricidade. Nos tempos atuais, dadas as facilidades de compartilhamento de informações, os avanços tecnológicos e a introdução contínua de produtos no mercado, as inovações são muito mais frequentes, incrementais, com impactos mais abrangentes em termos sociais e econômicos tendo em vista as segmentações de mercado e as inúmeras variações de uso de uma mesma tecnologia.

2.3. Ecossistemas de Inovação e o Papel da Universidade

Engana-se quem acredita que inovar seja uma prática exclusiva de uma empresa, em particular, de uma grande empresa. Inovar é um processo cada vez mais disponível a diversos setores, organizações e grupos sociais. Além disso, inovar é um processo colaborativo que exige parcerias e relações entre diversas instituições.

Até o início dos anos 2000, o modelo tradicional de inovação - Inovação Fechada - descrevia que os processos de desenvolvimento de novos produtos e de novos negócios são internos à empresa, ocorrendo dentro do seu funil da inovação, em seus limites organizacionais, conforme apresentado na figura 2.4 [Grizendi 2017]. Este modelo possui um custo mais elevado de pesquisa e desenvolvimento (P&D) que, somado com a mobilidade e disponibilidade de pessoal qualificado, não vinha trazendo o retorno esperado pelas empresas. Como agravante, quando um empregado muda de emprego, ele carrega seu conhecimento com ele, resultando em fluxo de conhecimento entre empresas.

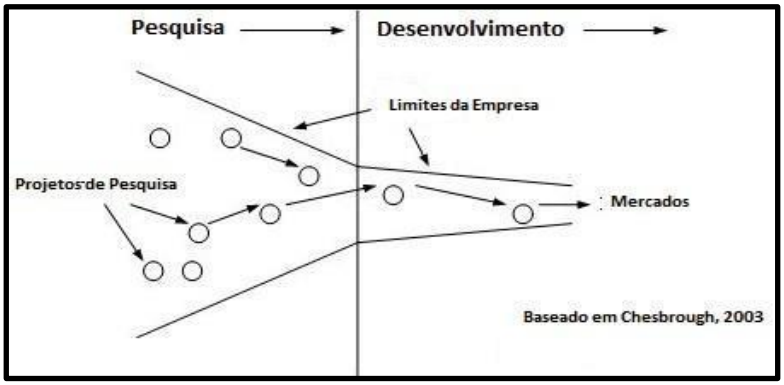


Figura 2.4. Modelo de Inovação Fechada [Grizendi 2017]

Em 2003, Chesbrough introduziu o conceito de Modelo de Inovação Aberta, que preconiza que uma empresa deve operar seu funil de inovação, permeável ao seu ambiente externo e ilustra a ideia mostrando um funil vazado, cheio de furos [Chesbrough 2003] (Figura 2.5), por onde entram e saem resultados e recursos tecnológicos intermediários, além das ideias na boca e o produto final na ponta do funil. A empresa fertiliza seu processo de inovação e aproveita mais as oportunidades que existem, se, de forma aberta, buscar outras bases tecnológicas, além da sua base tecnológica interna, e com isto também alimentar o seu funil da inovação. Operando no modelo aberto, a empresa aproveita mais e melhor os resultados intermediários de P&D, mesmo aqueles que não vão adiante e geram inovações para ela. Neste modelo, um resultado intermediário de P&D pode ser transferido a outra empresa, através de licenciamento ou mesmo através de uma empresa "*spin-off*", para atingir novos mercados, em ambos os casos, gerando receita adicional para a empresa. Naturalmente

que o inverso também deve ser praticado, ou seja, a empresa deve procurar tecnologias para licenciamento, para alimentar o seu funil da inovação.

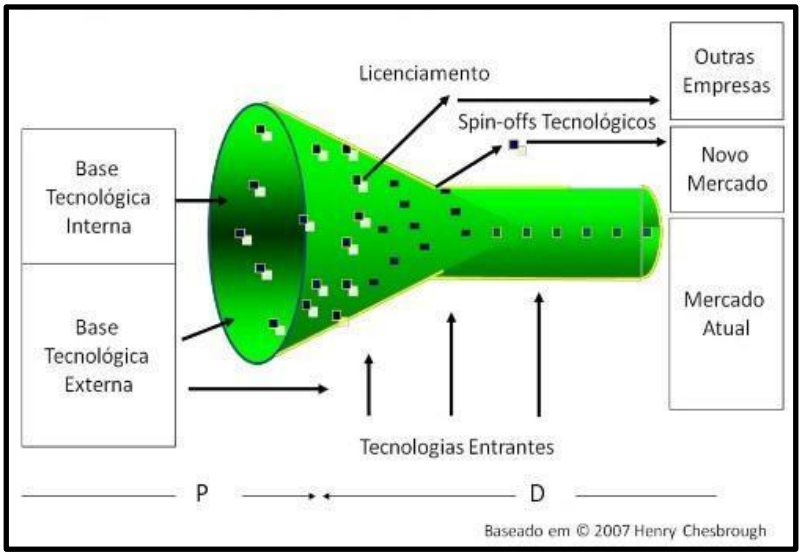


Figura 2.5. O Funil da Inovação no Modelo de Inovação Aberta [Grizendi 2017]

A visão da inovação como elemento fundamental para o desenvolvimento econômico, a competitividade, o empreendedorismo e a sustentabilidade, associada às estratégias de inovação aberta, determinam a configuração de ecossistemas regionais voltados a manter uma rede de atores que se articulem para a promoção da inovação - os ecossistemas de inovação. Wang [2010 apud Kosloski et al. 2016] define um ecossistema de inovação como: "o sistema dinâmico de instituições e pessoas interconectadas que são necessários para impulsionar o desenvolvimento econômico e tecnológico. Este ecossistema inclui uma gama de atores da academia, indústria, fundações, organismos científicos, econômicos, e do governo em todos os níveis. A organização de um ecossistema de inovação não é rigidamente planejada com papéis bem definidos para os diversos atores. Como resultado, as posições relativas de cada ator, bem como as condições para encorajar ou restringir o processo de inovação, podem mudar continuamente."

No Brasil, a Associação Nacional de Pesquisa e Desenvolvimento das Empresas Inovadoras (ANPEI[2]) deu início ao entendimento e representação dos fluxos e interações entre os atores participantes do Sistema Brasileiro de Inovação (Figura 2.6): i) **ICTs**: organizações públicas ou privadas, dedicadas às atividades de pesquisa de caráter científico ou tecnológico, que contribuem para a inovação nas empresas por meio da transferência de conhecimento; ii) **Investidores**: pessoa jurídica (pública ou privada), pessoa física, *angels*, clube de investimentos, *seed capital*, *venture capital*, *private equity*, entre outros tipos que oferece recursos financeiros e que tem como papel analisar e prospectar novas oportunidades, captar recursos e modelar negócios; iii) **Empresas**: organizações que tem como objetivo prover produtos e serviços, geram empregos e tributos; sendo o principal ator responsável por implementar a inovação; iv) **Governo**: liderança dividida em três esferas: Federal, Estadual e Municipal cujas atribuições estão definidas na Constituição Federal e que incluem a arrecadação de tributos, elaboração de políticas públicas, investimentos e o provimento de serviços

[2] http://anpei.org.br/

públicos à população, responsável pela criação do ambiente de inovação, sua regulamentação, fomento e articulação entre os atores; v) **Entidades de Classe**: organizações sem fins lucrativos que tem como papel a representação e articulação de atores internos e externos, contribuindo no fortalecimento destas relações e na proposição de políticas públicas.

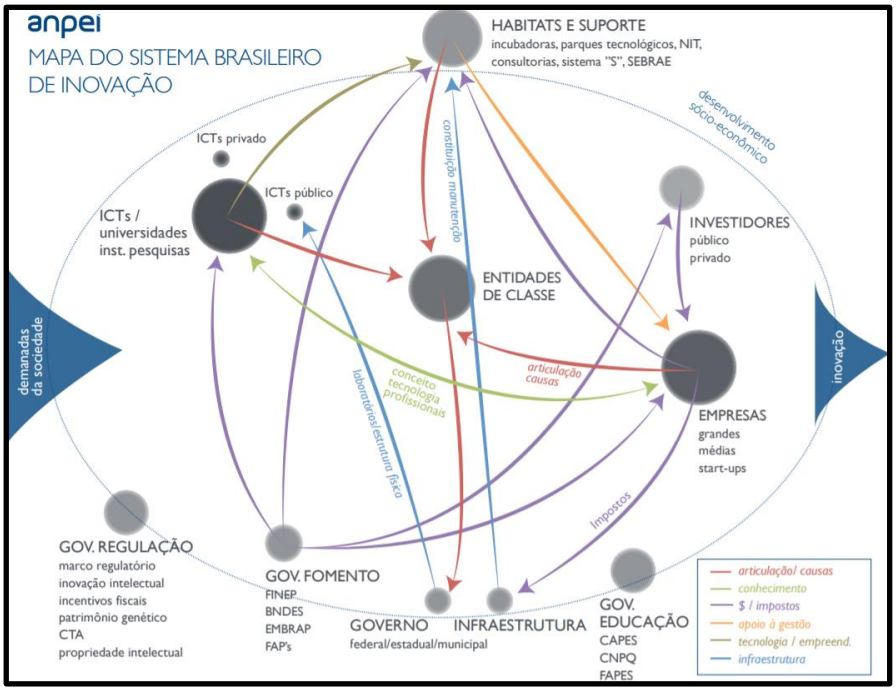


Figura 2.6. Mapa do Sistema Brasileiro de Inovação (Fonte: ANPEI)

As universidades, como instituições de ciência e tecnologia, são atores fundamentais neste ecossistema, não só no Brasil como em todo o mundo, principalmente por seu papel nos processos de transferência de tecnologia. A transferência de tecnologia é "qualquer processo pelo qual o conhecimento básico, a informação e as inovações se movem de uma universidade, instituto ou de um laboratório governamental, para um indivíduo ou para empresas nos setores privados e semi-privados" [Parker e Zilberman 1993, p. 89]. Para esses autores, a transferência inclui a transferência de informação (conferências e publicações), atividades educacionais e de capacitação, consultoria, patenteamento, licenciamento de inovações e criação de empresas start-ups.

No Brasil, a Lei de Inovação [LEI Nº 10.973] e mais recentemente o Marco Legal da Ciência Tecnologia e Inovação [LEI Nº 13.243] e têm tentado fomentar os processos de inovação e transferência de tecnologia, estimulando, por exemplo, a criação de Núcleos de Inovação Tecnológica (NIT) dentro das universidades e centros de pesquisa nacionais. NITs são escritórios de transferência de tecnologia, conforme definição adotada pela OCDE: "Os Escritórios de Transferência de Tecnologia (ETT) ou de Licenciamento são aquelas organizações ou partes de uma organização que ajudam, nos centros públicos de pesquisa, a identificar e administrar seus ativos intelectuais, incluindo a proteção da propriedade intelectual e transferência ou licenciamento dos direitos a terceiros, orientando a completar um desenvolvimento." Uma instituição pública de pesquisa pode ter um único ETT ou pode ter vários ETT

associados (por exemplo, para unidades diferentes ou departamentos) ou pode recorrer a um ETT externo que possui outros vários clientes" [OCDE 2003, p. 80].

O Marco Legal da Ciência Tecnologia e Inovação [LEI Nº 13.243] têm tentado também flexibilizar a legislação nacional, mais refratária à dinâmica necessária às parcerias, desenvolvimento conjunto e transferência de conhecimento entre instituições de pesquisa, governo, indústria e mercado. O Plano Nacional de Pós-Graduação 2011-2020[3], elaborado pela CAPES-MEC, aponta preocupação a respeito da formação de doutores para atuação na indústria, para a inovação e o empreendedorismo, além da formação acadêmica, tradicionalmente voltada à atuação em universidades e institutos de pesquisa. A despeito do alcance na flexibilização legal, dos estímulos econômicos à inovação e da abertura à inovação por parte das políticas públicas de financiamento à pesquisa, percebe-se um movimento de uma boa parcela da comunidade de pesquisa - um dos atores fundamentais neste sistema – ainda tênue no sentido de responder à dinâmica dos ecossistemas de inovação nacional e mundial. Em parte, pela falta de entendimento do conceito e processos de inovação, em parte pela pouca clareza das políticas públicas nacionais, em parte pelos ainda entraves legais e, por fim, pela cultura acadêmica de produção científica, focada no estímulo à publicação de artigos e na formação de pesquisadores para a pesquisa acadêmica.

2.4. Relações entre os processos de pesquisa e inovação

Segundo Recker (2013), a **pesquisa** é uma atividade fundamental para a ciência e tem como objetivo a produção de conhecimento em uma determinada área, de forma a contribuir para o desenvolvimento científico e para a sociedade em geral. Esse autor também relata que a pesquisa científica tem como missão a busca pela verdade, possibilitando o desenvolvimento do conhecimento de uma determinada área, uma vez que todo o conhecimento científico é um conjunto de explicações sugeridas sobre um fenômeno em particular. Assim, na busca pela verdade, a pesquisa científica requer um formalismo para a sua condução para garantir confiabilidade dos resultados.

A pesquisa na área de Sistemas de Informação (SI) possui características que refletem a própria natureza do artefato pesquisado – um sistema de informação corresponde a um conjunto de componentes inter-relacionados que coletam (ou recuperam), processam, armazenam e distribuem informação [Laudon e Laudon 2016]. Um resumo muito sucinto dos elementos principais da pesquisa na área de SI pode ser observado na figura 2.7. Sistemas de Informação implicam, invariavelmente, em pessoas – tanto as que constroem os sistemas, como as que participam de seu uso. A grosso modo, a pesquisa na área de SI está sempre relacionada a um determinado contexto de aplicação social ou organizacional e a um público-alvo onde oportunidades ou problemas relacionados à coleta, processamento, armazenamento ou distribuição de informação são identificados. Este contexto precisa ser compreendido para que os problemas existentes sejam identificados e devidamente descritos.

A identificação de um problema nos contextos de aplicação pode ser corroborada com evidências tanto da prática como da comunidade científica, onde outros pesquisadores podem ter também identificado e trabalhado em soluções para o mesmo problema ou problemas similares. No entanto, dada sua característica aplicada e a abrangência atual de uso de SIs nos mais variados domínios, os contextos e problemas

[3] http://www.capes.gov.br/plano-nacional-de-pos-graduacao

podem ser estudados sob diferentes perspectivas, disciplinas ou áreas de conhecimento. O entendimento de um problema a ser investigado bem como o delineamento de possíveis soluções é enriquecido com o conhecimento gerado pelo olhar de diferentes domínios de conhecimento sobre ele. A pesquisa em SI se enriquece com, ou até mesmo exige, a abertura à multidisciplinaridade.

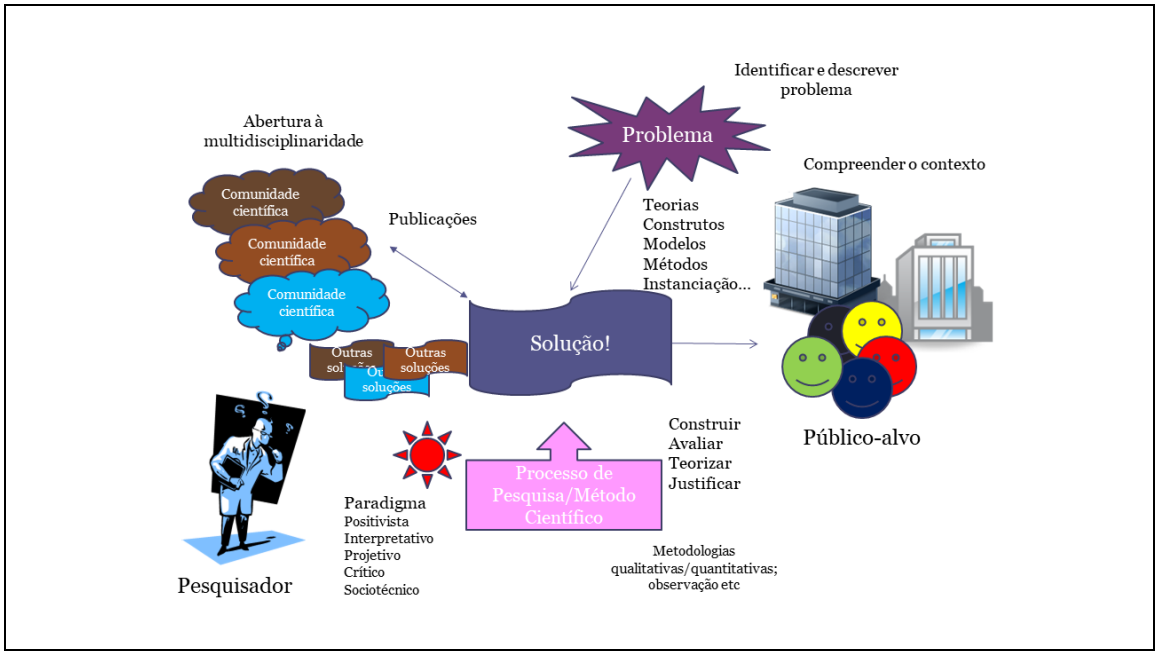


Figura 2.7. Características de Pesquisa Científica em Sistemas de Informação [ARAUJO 2017]

Cabe ao pesquisador, ao compreender de forma aprofundada um cenário problema, preferencialmente sob diferentes perspectivas disciplinares, pensar em uma solução que pode abranger construtos concretos como modelos, métodos, ou experiências de instanciação de uma solução já existente, em uma visão prescritiva (positivista) de solução. O pesquisador pode também optar por aprofundar o contexto problema e suas soluções sob uma perspectiva interpretativa ou crítica, sem necessariamente construir soluções, mas aprofundar o conhecimento geral sobre aquele contexto de forma detalhada. Há também a perspectiva projetiva de pesquisa, como os paradigmas de pesquisa do artificial [Pimentel et al. 2017].

De acordo com a perspectiva (paradigma) utilizada, o pesquisador aplica processos/métodos adequados para construir, avaliar, justificar ou teorizar a respeito do problema ou solução de pesquisa. A aplicação destes métodos, junto com a observação do contexto, a identificação e descrição do problema, a comparação de sua visão com a comunidade científica, sob um paradigma específico resulta na geração de conhecimento científico, usualmente disseminado sob a forma de publicações científicas.

Por outro lado, a **inovação**, segundo o Manual de Oslo (2004), pode ser definida como um processo que se inicia no campo das ideias, se materializa em uma invenção que, posteriormente, será comercializada para gerar uma riqueza econômica e/ou social. Para Tidd, Bessant e Pavitt (2008), a inovação envolve três pilares: conhecimento (científico, tecnológico ou empírico), informação (dados organizados, disponíveis e capturados do ambiente) e criatividade (criação de coisas diferentes e novas).

Os processos de inovação, a grosso modo, surgem de um esforço contínuo de observação do mundo e prospecção de necessidades, problemas e oportunidades, visando a construção de soluções que cheguem efetivamente ao uso de seu público-alvo ou mercado. As soluções representam avanços no estado da técnica vigente e o conhecimento que geram ao serem criadas é valorizado como propriedade, não somente como valor comercial, mas também como valor intelectual. Colocar um produto no mercado não é tarefa simples e exige um ecossistema receptivo, com parcerias constantes entre empresas, governo, financiadores e instituições de conhecimento.

Há uma grande interseção entre estes dois processos – inovação e pesquisa científica - no que se refere a observar o mundo, identificar oportunidades, compreender o estado da arte das soluções existentes, criar novas soluções e avaliar seu uso. No que diferem, é no rigor metodológico para se chegar a resultados (no caso da pesquisa científica) e na preocupação com a colocação rápida no mercado de um produto (no caso da inovação).

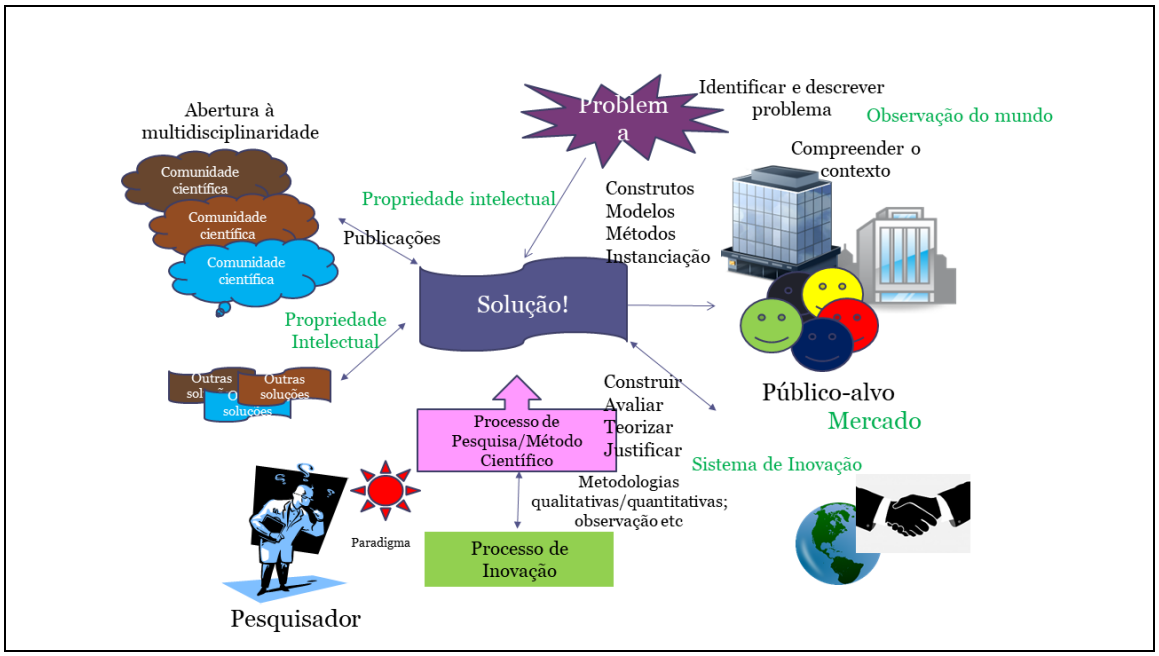


Figura 2.8. Características de Pesquisa Científica em Sistemas de Informação com Inovação [ARAUJO 2018b]

No entanto, os dois processos podem se beneficiar mutuamente (Figura 2.8). Os processos de inovação se beneficiam ao incorporarem métodos para investigação do conhecimento científico, ampliando o espectro de conhecimento para novas soluções e fazendo o uso dos resultados científicos para minimizar riscos de produção de uma invenção. Já os processos de pesquisa científica se beneficiam ao incorporarem em suas estratégias uma visão voltada ao público-alvo, direcionada ao impacto e relevância das soluções sendo criadas, da ampliação do espectro de conhecimento a ser consumido ou gerado ao considerar as bases de propriedade intelectual, e ao identificar parcerias no ecossistema para sustentabilidade de suas ações.

Visando identificar as intersecções entre pesquisa científica e inovação, Araujo et al. (2017), apresentam um quadro-resumo, conforme Tabela 2.3, que registra as principais características entre esses processos.

Tabela 2.3. Características de Pesquisa Científica e Inovação [ARAUJO et al. 2017]

	Pesquisa Científica	**Inovação**
Origem	Questão (relacionada a um tema e problema de pesquisa)	Ideia (relacionada a um problema relevante)
Motivação	Existe um domínio problema importante com uma importante lacuna de conhecimento sobre um importante fenômeno que merece atenção da comunidade de pesquisa.	Geração de riqueza econômica (financeira ou social), pressões adicionais de custo; exigências dos clientes; e busca por maior participação no mercado.
Agente (quem realiza o processo)	Pesquisadores, instituições científicas	Empreendedores, organizações
Contexto	Científico	Econômico e social
Tolerância ao risco	Baixa tolerância ao risco, devido às limitações do cenário de pesquisa científica e ao perfil dos pesquisadores	O processo de combinação dos diferentes conjuntos de conhecimento em uma inovação bem-sucedida ocorre sob condições de alta incerteza. Tanto organizações quanto empreendedores possuem uma tolerância maior ao risco.
Possibilidade de Mobilização de Recursos	Limitado ao cenário econômico e volume de investimentos em pesquisa; restrito a limitações legais.	Maior aproveitamento do sistema de inovação e acesso a financiamentos, tanto para organizações quanto para empreendedores. Recursos são mobilizados no sentido de reduzir a incerteza e risco do desenvolvimento.
Rigor	Se realiza com base num conjunto de procedimentos executados rigorosamente para se produzir o conhecimento científico e para garantir a confiabilidade dos resultados. Pesquisador precisa justificar suas escolhas metodológicas.	Na maior parte das vezes o rigor na execução do processo de inovação não é visível ou disponibilizado ao resultado final. Busca-se produtos viáveis mínimos que possam ser rapidamente colocados em uso/mercado.
Resultado	Artigos, monografias, teses e dissertações, artefatos, conhecimento científico.	Produtos e processos inovadores, economicamente exploráveis. Patentes.
Questão básica	Até que ponto ele terá importância para a comunidade científica?	Até que ponto ele terá importância para o mercado consumidor?

Defendemos a complementariedade dos processos de pesquisa científica e os processos de inovação tecnológica [Araujo 2016][Araujo et al. 2017a]. Enquanto que a pesquisa científica instrumenta o pesquisador com métodos que garantem rigor e visam a geração de conhecimento para comunidades científicas, a inovação confere visão aplicada a problemas reais e a desafios para o desenvolvimento de soluções. A combinação destes processos pode alavancar tanto a geração de conhecimento científico

com visão aplicada como a produção de artefatos inovadores baseados em pesquisa científica.

O exercício de integração entre os processos de pesquisa científica e inovação pode estimular uma visão empreendedora – não no sentido empresarial, mas no sentido de motivada a gerar conhecimento e soluções de impacto e relevância real [COELHO 2017] – ao mesmo tempo que são avaliadas as reais possibilidades e os benefícios desta integração. Sugerimos que os pesquisadores reflitam, a partir dos construtos de suas pesquisas, a respeito das questões relacionadas à inovação. As principais questões para reflexão estão descritas na Tabela 2.4 que reúne os principais aspectos de uma pesquisa científica, incluindo as questões relacionadas à inovação.

Tabela 2.4. Ficha de pesquisa com reflexões sobre inovação

Título do Trabalho	*Título da pesquisa*
Aluno	*Seu nome*
Orientador(es)/Grupo de Pesquisa	*Nomes dos orientadores e suas instituições entre parênteses. Enumere outros potenciais compartilhadores de direitos autorais de tua pesquisa.*
Motivação	*O que o motiva a realizar esta pesquisa? Qual a sua implicação e experiência com o tema?*
Problema de pesquisa	*Defina, em uma frase, qual é o problema que sua pesquisa busca resolver ou reduzir. Caracterize como um problema de pesquisa (para gerar conhecimento) e não como um problema técnico (que pode ser resolvido por meio de técnica ou tecnologia conhecida).*
Comunidades/Áreas de Pesquisa relacionadas	*Quais áreas científicas têm interesse na tua pesquisa e em seus resultados? Há espaço para contribuições de/para outras áreas além da Computação?*
Desafio	*Qual o principal desafio da tua área de pesquisa nos próximos anos? Que terreno foge aos pés de quem atua nesta área? Como sua pesquisa se alinha a este desafio? Há desafios além da tua área de pesquisa que podem ser abordados?*
Contexto	*Em qual contexto a sua pesquisa está sendo desenvolvida? É parte de um projeto maior? É aplicada a uma empresa ou a um contexto social específico?*
Cenário Exemplo	*Descreva um cenário real que exemplifique e evidencie o problema a ser tratado na pesquisa.*

Justificativa e relevância	*Justifique **sucintamente** a importância de se resolver o problema da pesquisa. O problema é relevante? As soluções atuais não resolvem bem o problema? Quais os impactos que o problema traz hoje? Quais os benefícios e impactos se for minimizado/resolvido?*
Objetivo da pesquisa	*Descreva, em uma frase, qual é o objetivo geral da pesquisa. Construir, Avaliar, Teorizar ou Justificar algo?*
Hipótese/Questão	*Indique a hipótese a ser avaliada (pesquisa explanatória) ou a questão de pesquisa (pesquisa exploratória ou descritiva).* *Se hipótese a ser avaliada, então esta deve ser formulada seguindo o modelo: SE (solução proposta) ENTÃO (a observação que indica que o problema foi resolvido).*
Enfoque de Solução	*Quais teorias, conceitos e/ou tecnologias são visualizados como possíveis soluções para o problema?*
Abordagem de Pesquisa	*Teórica ou empírica? Quantitativa ou Qualitativa? Descritiva, exploratória ou explanatória?*
Projeto de Avaliação	*O que será feito para avaliar a hipótese/solução ou investigar a questão de pesquisa? Quais serão os dados coletados nessa pesquisa? Usando que técnicas de coleta e que técnicas de análise de dados?*
Contribuições científicas	*Qual o conhecimento novo que se espera gerar a partir da pesquisa?*
Contribuições tecnológicas	*Produto (ferramenta, técnica, tecnologia, processo, software etc.) gerado a partir da pesquisa.*
Público-Alvo da Pesquisa	*Quem são as pessoas impactadas com o problema de tua pesquisa? Quem são os potenciais consumidores – clientes, usuários ou envolvidos – dos artefatos gerados na tua pesquisa?*
Tipo de inovação (se aplicável)	*Para cada uma das contribuições tecnológicas apontadas acima, avalie: Qual o tipo, grau de novidade e abrangência que representa?*
Risco tecnológico	*Qual o risco do desenvolvimento das soluções previstas– desafio científico (provar que funciona) e desafio tecnológico (produzir em escala)?*

<table>
<tr><td>Ecossistema de Inovação</td><td>Como os produtos inovadores de tua pesquisa se inserem no ecossistema de inovação? Quais poderiam ser seus parceiros para o desenvolvimento dos artefatos como produtos? Quem seriam seus financiadores? Há possibilidades de spin-ins/spin-offs?</td></tr>
<tr><td>Mercado/Sociedade</td><td>Qual o mercado que absorveria produtos advindos de tua pesquisa – nichos, clientes, competidores. Qual o valor de cada uma de suas soluções para este mercado/sociedade? Como você avaliaria sua receptividade? Que implicações este produto traria para o mercado/sociedade? Como seu produto se diferencia dos produtos dos competidores/existentes?</td></tr>
<tr><td>Propriedade Intelectual</td><td>Quais produtos são passíveis de proteção e registro? Sob que forma?</td></tr>
<tr><td>Plano de publicação (artigos científicos)</td><td>Plano de publicação de conhecimento a ser disseminado e explorado em artigos científicos.<table><tr><td>Conferência/Journal</td><td>Previsão de produção</td><td>Conteúdo do trabalho (conteúdo planejado para ser explorado na publicação)</td></tr><tr><td></td><td></td><td></td></tr></table></td></tr>
<tr><td>Plano de proteção (propriedade intelectual)</td><td>Plano de proteção do conhecimento gerado na pesquisa, a ser depositado como item de propriedade intellectual<table><tr><td>Artefato</td><td>Tipo de PI</td><td>Previsão de sua geração no trabalho de pesquisa</td></tr><tr><td></td><td></td><td></td></tr></table></td></tr>
<tr><td>Outros comentários</td><td>Opcional. Por exemplo, indique as principais dificuldades enfrentadas, ou os desafios que ainda precisam ser superados, ou os pontos ainda em aberto na pesquisa.</td></tr>
</table>

2.5. Propriedade intelectual

Nas instituições que adotam um conceito mais estrito de transferência de tecnologia (vide seção 2.3), baseado principalmente na comercialização de ativos intangíveis, como é o caso das instituições públicas de pesquisa da maioria dos países da OCDE, as atividades dos Núcleos de Inovação Tecnológica (NITs) são centradas na comercialização da propriedade intelectual. Por esta razão, julgamos importante que os pesquisadores se apropriem do conceito e possibilidades da propriedade intelectual em suas pesquisas.

No Brasil, a propriedade intelectual é a área do Direito que, por meio de leis [LPI 1996], garante a inventores ou responsáveis por qualquer produção do intelecto - seja nos domínios industrial, científico, literário ou artístico - o direito de obter, por um determinado período de tempo, recompensa pela própria criação. Definição similar também determinada pela Organização Mundial de Propriedade Intelectual (OMPI/WIPO do inglês *World Intellectual Property Organization*[4]).

Nunes e Pinheiro-Machado (2017) mencionam que, no Brasil, a propriedade intelectual está organizada em 3 grandes grupos (Figura 2.9): i) Propriedade Industrial: trata da proteção por patentes, desenho industrial, indicações geográficas, marcas e regula a averbação de contratos e as franquias; ii) Direitos Autorais e Conexos: envolvem a proteção de obras literárias, artísticas, arquitetônicas, musicais, programas de computador, bases de dados entre outras, além de suas interpretações; iii) Proteção *Sui Generis*: protegem variedades de plantas não existentes na natureza, conhecidas como Cultivares, topografia de circuito integrado, e conhecimento tradicional e folclore.

Para as pesquisas na área de Sistemas de Informação e/ou Computação, um subconjunto de itens de propriedade intelectual torna-se mais relevante em relação à organização de conhecimento na área [Nunes e Pinheiro-Machado, 2017]: marcas, desenho industrial, indicação geográfica, direito autoral (principalmente registro de computador e bases de dados), topografia de circuito integrado e patentes.

2.5.1 Patentes

Segundo Alencar (2016), uma patente é um título de propriedade temporária sobre uma invenção ou modelo de utilidade, outorgados pelo Estado aos inventores ou autores ou outras pessoas físicas ou jurídicas detentoras de direitos sobre a criação para fazer uso comercial de suas invenções. Patentes podem estar classificadas em Patentes de Invenção (PI) – descrevem uma tecnologia que solução um problema técnico – e Modelos de Utilidade (MU) – descrevem uma melhoria funcional na forma ou estrutura de um objeto.

Um conhecimento técnico é passível de patenteamento se atender aos critérios de i) **novidade** – algo é considerado novo quando não compreendido no Estado da Técnica[5], ii) **atividade inventiva (PI)/ato inventivo (MU)** – uma invenção é dotada de atividade inventiva sempre que, para um técnico[6] no assunto, a mesma não decorra de maneira evidente ou óbvia do Estado da Técnica; um modelo de utilidade é dotado de ato inventivo sempre que, para um técnico no assunto, não decorra de maneira comum ou vulgar no Estado da Técnica; iii) **melhoria funcional (para MU)** - a introdução em objeto de uma forma ou disposição que acarrete comodidade ou praticidade ou eficiência à sua utilização e/ou obtenção; iv) **aplicação industrial** - se o seu objeto for passível ou capaz de ser fabricado ou utilizado em qualquer tipo/gênero de indústria; e v) **suficiência descritiva** – o texto da patente deve descrever clara e suficientemente o objeto, de modo a possibilitar sua realização por técnico no assunto e indicar, quando

[4] http://www.wipo.int/portal/en/index.html

[5] tudo aquilo tornado acessível ao público antes da data de depósito do pedido de patente, por descrição escrita ou oral, por uso ou qualquer outro meio, no Brasil ou no exterior, ressalvados períodos de graça e prioridades.

[6] Técnico no assunto – pessoa detentora dos conhecimentos medianos sobre a matéria e não um grande especialista ou sumidade na matéria.

for o caso, a melhor forma de execução (um texto bastante distinto da redação científica).

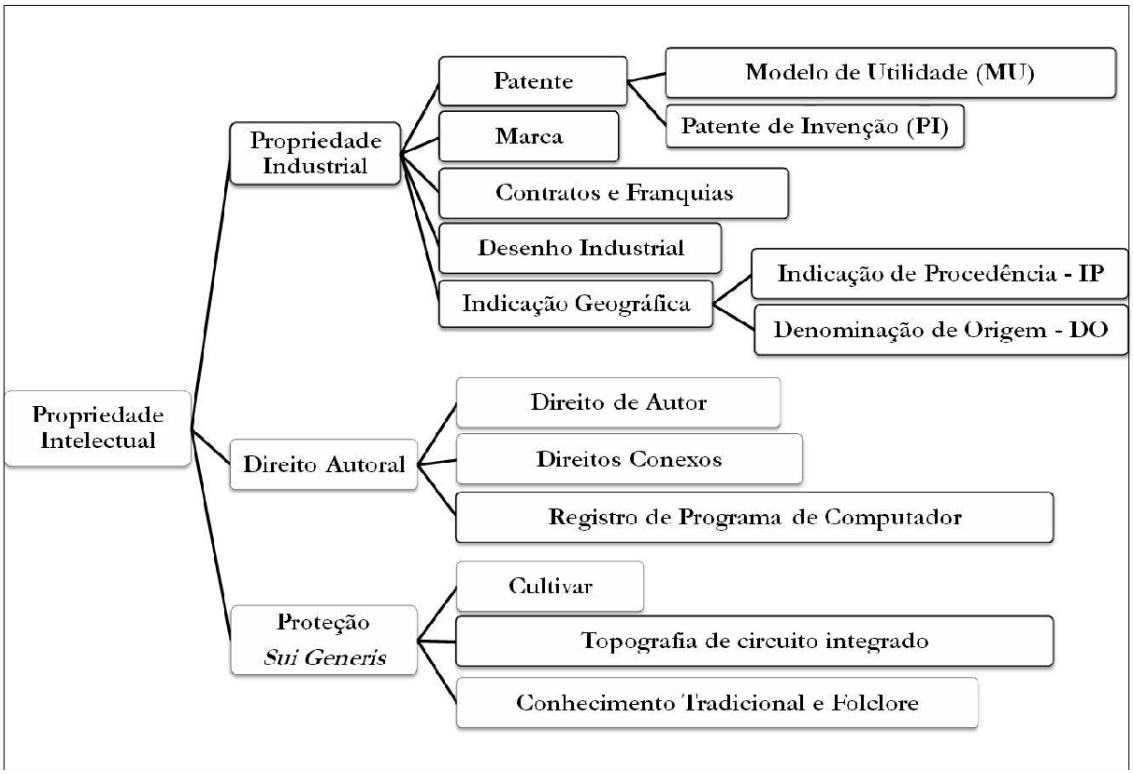


Figura 2.9. A Propriedade Intelectual e suas divisões. [Nunes e Pinheiro-Machado, 2017]

O registro de patentes é realizado mediante depósito de um pedido (patente **depositada**) a instituições responsáveis pela concessão, comumente chamados de escritórios de patentes, como o Instituto Nacional da Propriedade Industrial (INPI), no Brasil. O pedido é analisado por técnicos internamente a estas instituições, de forma a conferir os critérios expostos acima. Uma vez que o pedido atenda aos critérios, a patente passa a ser **concedida**. O registro de uma patente confere benefícios privados ao titular, que detém o monopólio temporário de excluir terceiros de usar sua invenção; bem como benefícios públicos, uma vez que a contrapartida à proteção da invenção é a divulgação da informação tecnológica após o depósito do pedido de patente. Portanto, conforme alertado por Alencar (2016), um documento de patente – independente de ser concedido – contém informação tecnológica.

2.5.2 Estrutura de um Documento de Patente

Alencar (2016) também nos ensina que uma patente é estruturada por: i) folha de rosto com informações bibliográficas; ii) relatório descritivo da tecnologia: detalha o estado da técnica, a invenção, ressaltando os problemas técnicos resolvidos, as vantagens alcançadas e um melhor meio de execução da invenção que permita um técnico no assunto implementá-la.; iii) reivindicações - o escopo legal de proteção de uma patente sendo indiscutivelmente a parte mais importante das patentes uma vez que definem as fronteiras dos direitos protegidos.; iv) desenhos, se for o caso; e v) resumo. As informações bibliográficas disponíveis na folha de rosto são importantes para a busca, conforme veremos, e é importante identificá-las.

A Figura 2.10 apresenta a folha de rosto de uma patente brasileira, registrada no INPI. O conjunto de dados da folha de rosto inclui: as datas de depósito e publicação da patente; sua classificação internacional (já veremos do que se trata adiante); título; resumo; o requerente ou titular da patente; o(s) inventores; e a prioridade para sua análise, se houver. Os códigos entre parênteses (21), (22), (43) etc, representam códigos internacionais de identificação de dados bibliográficos de patentes. Isto significa que, não importa em que idioma esteja a patente, nestes códigos/campos, o leitor encontrará sempre a informação relacionada ao código.

A Classificação Internacional de Patentes (CIP) (veja toda a classificação em: http://ipc.inpi.gov.br) compreende uma estrutura hierárquica de classificação de patentes em 8 grandes assuntos, com subseções e detalhamentos:

Seção A – Necessidades Humanas

Seção B – Operações de Processamento; Transporte

Seção C – Química e Metalurgia

Seção D – Têxteis e Papel

Seção E – Construções Fixas

Seção F – Engenharia Mecânica; Iluminação; Aquecimento; Armas; Explosão

Seção G – Física

Seção H – Eletricidade

República Federativa do Brasil
Ministério da Indústria, Comércio Exterior e Serviços
Instituto Nacional da Propriedade Industrial

(21) BR 102016011501-9 A2

(22) Data do Depósito: 20/05/2016

(43) Data da Publicação: 05/12/2017

(54) Título: MÉTODO DE COMUNICAÇÃO DIRETA ENTRE UMA PÁGINA WEB E UM APLICATIVO LOCAL

(51) Int. Cl.: G06F 17/30

(52) CPC: G06F 17/30861,G06F 17/30058

(73) Titular(es): SCOPUS SOLUÇÕES EM TI LTDA.

(72) Inventor(es): ADREN SASSAKI HIROSE; REGINALDO ARAKAKI; WILSON VICENTE RUGGIERO

(74) Procurador(es): ANTÔNIO MAURÍCIO PEDRAS ARNAUD

(57) Resumo: O método utiliza uma página Web (PW) associada a um servidor (S) de uma instituição a ser acessada a partir de um aplicativo local (AL) carregado em um dispositivo de computação (C). A página Web (PW) já aberta em uma tela (T) do dispositivo de computação (C) solicita, ao aplicativo local (AL), um objeto de imagem que corresponda às informações requeridas pela página Web (PW), gerando, no aplicativo local (AL), uma fila com dados de resposta em dimensões de imagem. Uma imagem, tendo as dimensões correspondentes aos dados de resposta, é retornada à página Web (PW), a cada solicitação de imagem por essa última, até que o aplicativo local (AL) consuma a fila de dados de resposta, permitindo que a página Web (PW) interprete as dimensões de imagem de cada solicitação e as transforme nos dados de comunicação a serem disponibilizados ao servidor (S) associado.

Figura 2.10. Folha de rosto de uma patente. Fonte: INPI

Como exemplo, a classificação da patente apresentada acima (Figura 2.10) – **G06F 17/30** - pode ser entendida conforme apresentado abaixo:

G – Seção G – Física

G06 – Cômputo; Cálculo; Contagem

G06F – Processamento elétrico de dados digitais

G06F 17/00 - Equipamentos ou métodos de computação digital ou de processamento de dados, especialmente adaptados para funções específicas

G06F 17/30 - Recuperação das informações; respectivas estruturas de banco de dados

2.5.3 Bases de Informação sobre Propriedade Intelectual

Jagher (2017), Nunes (2014) e Nunes e Pinheiro-Machado (2017) indicam uma lista de sites e bases de consultas públicas e privadas onde são disponibilizadas informações sobre propriedade intelectual. Estas bases podem conter informações restritas a um país ou ao mundo inteiro. Variam em formas de acesso, disponibilidade de formato dos itens e procedimentos de busca. Neste relatório, listamos as que consideramos mais relevantes para a busca de informação no contexto das pesquisas em Sistemas de Informação Tabela 2.5).

Tabela 2.5. Bases de Informação sobre Propriedade Intelectual [Nunes e Pinheiro-Machado, 2017]

INPI (www.inpi.gov.br)	Disponibiliza documentos nacionais e estrangeiros em todas as áreas de conhecimento, via busca online. Base principal de patentes brasileiras.
PATENTSCOPE (www.wipo.int/patentscope/en)	Base de patentes da OMPI que reúne patentes depositadas através do PCT (*Patent Cooperation Treaty*), oriundas de diversos escritórios de patentes do mundo.
Derwent World Patent Index (DWPI) (www.periodicos.capes.gov.br)	Criada e mantida pela Thomson Reuters, tem cobertura de diversos escritórios de patentes no mundo. Disponibilizada à comunidade brasileira de pesquisa através da Capes e Fapesp, acesso por computadores internos das universidades.
Esp@cenet (ep.espacenet.com)	Base do Escritório Europeu de Patentes (EPO), contendo patentes do mundo todo.
USPTO (www.uspto.gov)	Base de patentes norte-americanas.
Google patent search (www.google.com/patents)	Foco em patentes norte americanas, atualmente permite a busca em bases de patentes públicas de todo o mundo.

2.6. Levantamento de informação tecnológica

Dentre os tipos de propriedade intelectual disponíveis, a principal fonte de informação levantada são as bases de patentes, dado o volume de dados existente e sua abrangência mundial, a organização do conhecimento disponível (uniformizado por padrões internacionais), e o detalhamento da descrição da tecnologia protegida pela patente. Pinheiro-Machado (2017) defende a importância de se ampliar o uso da informação tecnológica contida em documentos de patentes uma vez que observa que a quantidade de documentos de patentes tem tido um crescimento exponencial, e que 70-80% das tecnologias tem divulgação só por patentes. A autora destaca que patentes abrangem todos os campos tecnológicos e o acesso às suas informações pode ser feito por meio eletrônico, conforme mencionado na seção anterior.

A busca de informação sobre propriedade intelectual é uma etapa importante no processo de inovação tecnológica, principalmente quando o objetivo do processo de inovação é a geração de produtos para o mercado e sua comercialização. Tem se tornado também cada vez mais importante no processo de produção científica, haja vista que a compreensão do "estado da técnica" em uma determinada área passa pela identificação do conhecimento disponível nas bases de patentes [Nunes e Pinheiro-Machado 2017][Borschiver e Silva 2016]. Nas bases de patentes encontra-se conhecimento não comumente publicado nas bases científicas, quer seja por razões de estratégias de proteção de conhecimento ou pelo fato de que este conhecimento é gerado por empresas, que não têm como foco a publicação científica. Considerando o aumento do interesse e prática de geração de conhecimento (tanto científico como tecnológico) em todo mundo na forma de patentes, estimulados por políticas públicas internacionais [Araujo e Paula 2017], a busca em bases de patentes torna-se uma atividade importante para o processo de pesquisa científica, cujo conteúdo não pode ser negligenciado.

Para Nunes e Pinheiro-Machado (2017), para oportunizar a produção de um novo produto ou processo advindo de uma pesquisa científica ou da inovação tecnológica, é importante utilizar a busca pelo estado da técnica em bases de patentes para esclarecer questões como: "*i) A tecnologia que será produzida é uma inovação? ii) A tecnologia já foi disponibilizada comercialmente? Ela já foi patenteada? Foi ou está sendo comercializada? iii) Se a tecnologia já foi patenteada, quais as lacunas existentes nessa tecnologia? iv) Se a tecnologia não foi patenteada, será que a mesma tem mercado e é passível de interesse nos processos produtivos locais, regionais ou nacionais?"* Porém, se em um primeiro momento, pensar em propriedade intelectual soa aos nossos ouvidos como uma atividade visando a "proteção" do conhecimento, um aprofundamento do conceito e o exercício da consulta das bases de propriedade intelectual nacionais e internacionais demonstram que, na verdade, planejar a produção de artefatos de pesquisa sob propriedade é, na verdade, disseminar e tornar o conhecimento público. Contrariamente ao que se costuma pensar, o conteúdo das bases de propriedade intelectual está disponível ao acesso pela sociedade, ao mesmo tempo que sob regulamentação adequada para o uso deste conhecimento pelo mercado e indústria nacionais e internacionais. Pensar em produção científica com geração de artefatos de propriedade intelectual é, ao mesmo tempo, publicar e proteger [Carvalho 2017]. Carvalho (2017) destaca o quanto é fundamental a conscientização sobre o que é conhecimento gerado em uma pesquisa e os benefícios que nos traz proteger este conhecimento. Sem um conhecimento e estratégias adequadas, podemos, como país,

estar fadados ao círculo vicioso de importar tecnologias e exportar insumos na maior parte das vezes.

Dada a importância da busca de informações em bases de patentes, ARAUJO et al. (2018a) apresentaram uma proposta de sistematização visando complementar a atividade de levantamento do estado da prática usualmente realizada nos processos de pesquisa científica. Esta proposta de sistematização tem sido aperfeiçoada desde 2015 [Araujo 2016] e, embora ainda em aperfeiçoamento, acreditamos ser já útil como diretrizes para que pesquisadores possam utilizá-la em seus processos de pesquisa.

2.6.1. Processo de Busca de Informação Tecnológica em Bases de Patentes

A proposta de processo para busca prospecção tecnológica feita nas bases de informação sobre propriedade intelectual é equivalente às prospecções acadêmicas feitas nas bases de informação visando mapeamentos e/ou revisões sistemáticas de literatura científica [Kitchenham 2004]. O protocolo compreende as fases e atividades conforme apresentadas na Tabela 2.6.

Tabela 2.6. Fases e atividades para a busca de informação tecnológica em bases de Patentes. Fonte: [Araujo et al. 2018a]

Fase	Atividades
1) Planejamento	● Formular questão para a busca ● Identificar palavras-chave ● Montar string de busca ● Identificar classificação internacional (CIP) ● Decidir quanto ao período de tempo ● Definir critérios de inclusão e de exclusão ● Selecionar base(s) de dados
2) Execução	● Realizar busca nas bases de dados ● Organizar resultado das buscas ● Selecionar patentes
3) Análise	● Analisar patentes selecionadas
4) Relato	● Gerar relatórios/publicações

2.6.1.1 Planejamento

A fase de planejamento tem como objetivo organizar um protocolo para a busca de informação, de forma a sistematizá-la minimamente.

2.6.1.1.1 Formular questão para a busca

O planejamento da busca de patentes se inicia com a formulação da pergunta ou questão que se espera responder com a busca por informação. A questão pode envolver uma combinação de perguntas a respeito do problema, o contexto de aplicação e a solução técnica elaborada. Como exemplo, um pesquisador poderia fazer as perguntas a seguir:

- *Q1: Quais patentes existem que resolvem o problema de transferência de imagens entre servidores web e aplicativos? → foco no problema*

- *Q2: Quais patentes existem que resolvem o problema de transferência de imagens entre servidores web e aplicativos em comércio eletrônico? → foco no problema em seu contexto*
- *Q3: Quais patentes existem que resolvem o problema de transferência de imagens entre servidores web e aplicativos em comércio eletrônico usando métodos de comunicação? → foco no problema, contexto e solução*

Importante considerar que patentes descrevem conhecimento de ampla aplicação. Questões muito fechadas ou muito específicas podem se demonstrar pouco úteis inicialmente. Cabe ao pesquisador calibrar a abrangência de suas questões conforme for aprofundando sua busca. Outro aspecto importante é que como a descrição de patentes não necessariamente apresenta resultados experimentais de uso, questões que envolvam conhecer resultados específicos de aplicação da solução podem não ser efetivas.

2.6.1.1.2 Identificar palavras-chave

A identificação de palavras-chave é outra atividade fundamental da busca de informação sobre patentes. As palavras-chave vêm diretamente das questões de pesquisa, e compreendem termos relacionados ao contexto, ao problema e à solução. Em geral, quando estamos realizando uma pesquisa científica, aos poucos vamos nos familiarizando com os termos chave do referencial teórico e conceitos relacionados ao problema e enfoque de solução da pesquisa comumente utilizados pela comunidade científica. Estes termos também podem ser utilizados aqui, para a busca de patentes.

No entanto, o uso dos mesmos termos identificados na literatura científica pode ser muito restritivo quando se realiza a busca em bases de patentes. A razão é que, no universo da propriedade intelectual, comumente o conhecimento tecnológico de uma solução é generalizada ao máximo, de forma que possa abranger um campo bastante amplo de sua aplicação. Por exemplo, a construção de um novo tipo de *lâmpada*, provavelmente não será registrada com este termo na patente. Muito mais provável será que seja registrado como um *sistema de iluminação*, pois assim a tecnologia produzida e o conhecimento que encerra se tornam mais abrangentes para aplicação em diferentes contextos, aumentando seu espectro de proteção.

Desta forma, caso o pesquisador inicie suas buscas e se frustre com poucos retornos, recomendamos que reflita, mesmo que minimamente, a respeito de como os termos de busca conhecidos em sua pesquisa científica podem ser ampliados em sua forma de aplicação, aumentando a chance de encontrar resultados significativos nas bases de patentes. Por outro lado, este exercício é por si só uma oportunidade de refletir quanto à abrangência do problema e solução de sua pesquisa, identificando oportunidades de aplicação até então não pensadas.

A identificação de palavras-chave é um processo recursivo, que pode ser aprimorado em sua precisão de acordo com o avanço da busca e com o aumento do conhecimento do pesquisador a respeito de como o assunto é organizado na base.

2.6.1.1.3 Montar *string* de busca

As bases de dados variam muito quanto ao mecanismo oferecido para a busca por palavras-chave. Algumas são mais sofisticadas e permitem combinações lógicas para a busca. Outras são mais simples e o processo de busca menos poderoso. A construção de

uma string de busca vai depender muito destes mecanismos, tanto de sua capacidade de combinação de palavras-chave, como nos campos que podem ser pesquisados (título, resumo, etc). O importante aqui é que o pesquisador organize a combinação das palavras-chave de forma a aumentar suas chances de encontrar patentes relevantes.

De maneira lógica, espera-se que as patentes relevantes para a pesquisa contenham as palavras-chave identificadas de forma conjunta (AND). No entanto, sabemos que isto pode tornar a busca muito restritiva. Desta forma, recomendamos que o pesquisador realize diversos ciclos de busca, variando as combinações das palavras-chave, ou mesmo pesquisando por cada uma delas, observando, a cada ciclo o volume e o tipo de patente retornada. Espera-se também que o pesquisador explore os termos sinônimos das palavras-chave identificadas.

2.6.1.1.4 Identificar classificação internacional

Conforme descrito em seção anterior, as patentes são organizadas nas bases segundo uma classificação internacional comum. Recomenda-se que o pesquisador navegue por esta classificação tentando identificar quais categorias estariam as patentes que deseja consultar.

A identificação da CIP das patentes desejadas pode ajudar como critérios de inclusão/exclusão de patentes, quando o pesquisador já possui mais segurança sobre o domínio de conhecimento que deseja consultar. No entanto, a identificação da CIP pode ser útil também para as primeiras buscas de um pesquisador, navegando por patentes dentro de uma determinada classificação, para identificação de termos e palavras-chave relevantes para sua busca.

2.6.1.1.5 Decidir quanto ao período de tempo

As bases de dados de patentes permitem a seleção do período de tempo para o qual se deseja fazer a busca. O pesquisador pode deixar o período de tempo da busca em aberto, de forma a ter uma maior abrangência e oportunidade de encontrar patentes com conhecimento relevante para sua pesquisa, mesmo que antigas – afinal conhecimento não se perde. Determinar um período específico de tempo para a busca pode ser um exercício interessante, pois implica em refletir e buscar informação sobre a evolução de uma determinada tecnologia e sua aplicação industrial/mercado.

2.6.1.1.6 Definir critérios de inclusão e exclusão

Nesta atividade, é importante que o pesquisador identifique critérios simples que o ajudem a determinar se uma patente encontrada deve permanecer ou não em sua lista para investigação. Os critérios de inclusão e exclusão devem estar diretamente relacionados às questões de busca definidas pelo pesquisador, podendo dizer respeito ao conteúdo e da tecnologia em si apresentada pela patente. Os critérios podem dizer respeito também a aspectos adicionais da patente, como por exemplo, o idioma de descrição da patente, a abrangência – país/mundo, tipo de patente (invenção, MU) etc. Não há regras quanto aos critérios a serem utilizados pelo pesquisador, desde que ajudem na determinação da relevância do conhecimento que pretende encontrar.

2.6.1.1.7 Selecionar bases de dados

O planejamento se encerra com a decisão a respeito de quais bases de dados são relevantes para a busca. Esta decisão implica em refletir quanto a qual a abrangência da busca que se deseja realizar – nacional, mundial ou em algum país em específico

(existem também as bases específicas de outros países – Canadá, Japão, América Latina etc, não listadas na seção anterior). Boa parte desta decisão é determinada pelo escopo e contexto da pesquisa que se quer realizar (se o problema ou solução é específico para o Brasil ou para o mundo), bem como na estratégia que o pesquisador possa ter de depósito de patentes ou de comercialização do(s) produto(s) de sua pesquisa no futuro.

2.6.1.2 Execução

A fase de execução tem como objetivo executar a busca nas bases de patentes selecionadas, usando como base o protocolo planejado, refinando-o, conforme as buscas vão sendo realizadas.

2.6.1.2.1. Realizar busca nas bases de dados

Nesta atividade sugere-se que o pesquisador realize diversos ciclos de busca, refinando seu protocolo de acordo com as patentes sendo encontradas. As buscas podem ser feitas combinando o uso da *string* de busca montada a partir das palavras-chave especificadas no planejamento, bem como busca pelas classificações internacionais identificadas como possíveis alvos para a pesquisa.

2.6.1.2.2 Organizar resultado das buscas

Sugere-se que um conjunto de dados sobre as patentes seja organizado conforme as buscas são realizadas, a saber: Nº da patente, País de origem, Classificação Internacional de Patente (CIP), Titular/Inventor(es), Data de depósito, Data de Publicação e resumo.

2.6.1.2.3 Selecionar patentes

Por meio da aplicação dos critérios de inclusão e exclusão definidos no planejamento, o pesquisador pode realizar um primeiro filtro sobre as patentes relevantes retornadas na busca. Uma vez selecionada uma lista de patentes potencialmente relevantes, pode-se proceder com a leitura da patente por completo – relatório, reivindicações etc – retirando da lista aquelas que não se demonstrarem pertinentes ou úteis. Importante ressaltar aqui que algumas bases podem não disponibilizar o texto completo das patentes registradas.

2.6.1.3 Análise

Esta etapa compreende a análise das patentes selecionadas. Implica na leitura cuidadosa do texto da patente, tentando responder às questões da pesquisa. As perspectivas pelas quais o pesquisador irá fazer sua análise são, obviamente, livres, mas é importante que seja capaz de tecer conclusões a respeito do estado da técnica representado pelo conhecimento das patentes analisadas e como este estado da técnica traz implicações para sua pesquisa.

Ressaltamos duas informações contidas nas patentes que merecem ser analisadas nesta fase. A primeira se refere às citações feitas à patente e vice-versa, patentes que citam a patente analisada. Navegar pelas citações pode ser uma boa maneira de identificar patentes relevantes relacionadas, eventualmente não identificadas na busca. A segunda se refere a citação no texto da patente de artigos acadêmicos publicados pelos inventores. Convém também buscar artigos publicados pelos inventores nas principais bases científicas.

2.6.1.4 Relato

Recomenda-se a documentação do processo de busca de patentes e a análise resultante no formato de relatórios. Este relatório possui potencial para publicações e disseminação de conhecimento, ressalvadas as estratégias de proteção estipuladas pelo pesquisador, que pode preferir manter a busca para seu uso restrito.

7. Pesquisa com Impacto

A inovação se trata de um conceito sistêmico e abrangente que implica em uma mudança de mentalidade em diversos níveis - nas políticas públicas e no sistema de relações das instituições nacionais, nos processos produtivos das empresas, nos objetivos dos projetos de pesquisa e desenvolvimento tecnológico e na mentalidade e atitudes dos indivíduos.

O processo de inovação, em boa parte compreende os processos usuais da pesquisa científica no que se refere ao entendimento do estado-da-arte (busca), identificação de oportunidades (seleção), desenvolvimento de soluções (implementar) e avaliações com aprendizado (aprender). Portanto, a capacitação em metodologias científicas favorece em muito o processo de inovação e o modelo mental necessário para o entendimento de problemas e a criação de soluções inovadoras. Por outro lado, a pesquisa científica, de uma forma geral, tende a se voltar sobre si mesma, podendo deixar de lado o acompanhamento do estado da prática (mercado). Além disso, a pesquisa científica objetiva a avaliação controlada com o intuito de determinar a viabilidade de suas soluções, não sendo esperados a aplicação de esforço e recursos para transformação destas soluções em produtos utilizáveis em escala.

Os processos de inovação, a grosso modo, surgem de um esforço contínuo de observação do mundo e prospecção de necessidades, problemas e oportunidades, visando a construção de soluções que cheguem efetivamente ao uso de seu público-alvo ou mercado. As soluções representam avanços no estado da técnica vigente e o conhecimento que geram ao serem criadas é valorizado como propriedade, não somente como valor comercial, mas também como valor intelectual. Colocar um produto no mercado não é tarefa simples e exige um ecossistema receptivo, com parcerias constantes entre empresas, governo, financiadores e instituições de conhecimento.

A distância entre a pesquisa científica e o ambiente produtivo torna-se um desafio para a prática sistemática da inovação em larga escala. Este desafio é em parte endereçado pelas políticas públicas de financiamento à pesquisa e inovação, pela atividade de estímulo à inovação realizada pelos Núcleos de Inovação Tecnológica estabelecidos nas instituições de pesquisa. No entanto, a prática da inovação é melhor explorada por pesquisadores que desenvolvem uma mentalidade empreendedora, interessados em direcionar suas pesquisas desde sua concepção em uma estratégia de empreendedorismo e inovação. Mas é preciso compreender o conceito de “empreender” de forma mais ampla. Não falamos aqui de empreendedorismo no sentido estreito de estabelecer novos negócios ou empresas. Falamos de empreendedorismo como enfrentamento de desafios. Grandes desafios.

É fundamental que o pesquisador possa se posicionar em relação ao seu papel e sua contribuição para os processos de inovação, quer seja dentro de uma comunidade acadêmica ou em uma empresa. A visão de que a inovação é um processo sistêmico, que envolve agentes (empresas, instituições de pesquisa, governo) e políticas de indução

e fomento, também é algo a ser apropriado pelos pesquisadores [Araujo e Paula 2017]. Vejamos porquê.

7.1. Contribuindo para resultados globais

Os resultados de inovação de um país são utilizados como indicadores globais de desenvolvimento e competitividade. O Índice Global de Inovação[7] busca capturar facetas multi-dimensionais da inovação e prover ferramentas que permitam costurar políticas para promover o crescimento de resultados de longo prazo, melhor produtividade e crescimento do emprego das diversas nações. Em 2017, o Brasil ocupava o 69º lugar no ranking das 127 nações analisadas pelo estudo, que considera dimensões de avaliação como: instituições, capital humano, infraestrutura, sofisticação do mercado e de negócios, resultados de conhecimento (entre eles, patentes e artigos científicos) e resultados de criatividade. Pensar em pesquisa dirigida à inovação pode ser uma alavanca importante para o desenvolvimento econômico do país, com impactos em seus resultados globais.

7.2. Financiamento à pesquisa

Boa parte dos editais de fomento à pesquisa no país colocam os resultados de inovação como requisito para a avaliação de projetos. Além disso, diversas instituições governamentais nacionais e internacionais, bem como empresas têm lançado editais para a execução de projetos de inovação. Pesquisadores precisam estar atentos a estas possibilidades de parcerias, financiamento e estratégias para suas iniciativas de inovação [Oliveira Jr 2017], bem como desenvolver habilidades para a elaboração de projetos onde o impacto de seus resultados seja claramente compreendido, o que só é possível por meio da compreensão dos conceitos e processos de inovação. Alguns aspectos considerados na avaliação da inovação em projetos podem incluir:

a) a clareza do problema a ser resolvido pela inovação - descrição de cenários reais, existência de evidências sobre o problema (experiências, dados estatísticos, análises qualitativas, etc); público alvo (pessoas, empresas ou instituições) específico ao qual a inovação está direcionada; impacto que se pode esperar com a inovação a ser gerada. Quanto maior os prejuízos relativos trazidos pelo problema, maior o impacto que uma solução pode trazer.

b) o desafio do projeto: relacionado à relevância do problema e o quanto seus resultados trarão de contribuições tanto à ciência como à prática. A clareza do desafio proposto pelo projeto é compreensível a partir da constatação do que existe em termos de conhecimento (científico ou de mercado) como de prática - o chamado estado da arte. Um projeto inovador precisa avançar no estado da arte, demonstrar que será capaz de ultrapassar desafios ainda não superados pela sociedade.

c) sua viabilidade: alguns aspectos de seu planejamento sejam coerentes - objetivos, produtos, cronograma, equipe, recursos, custos, riscos etc - precisam ser equilibrados de forma a garantir que os resultados esperados sejam obtidos. Um entendimento importante aqui está na relação entre o custo do projeto, a relevância do problema e/ou impacto de sua solução, e o desafio tecnológico a ser superado.

d) a competência da equipe desenvolvedora: inovação envolve multidisciplinaridade, visão global e competências complementares. Além disso, ela se

[7] https://www.globalinnovationindex.org/

baseia em conhecimento, experiências e capacidade de gerenciamento, aspectos também observáveis em um bom projeto.

e) exploração: um bom projeto inovador precisa demonstrar como a inovação poderá ser explorada, quer seja comercialmente (visando lucros) ou socialmente (visando seu bem estar). Que processos de exploração serão desencadeados e que parceiros do sistema de inovação serão estimulados para que a inovação ganhe o mercado precisam estar claramente descritos? Qual o negócio que potencialmente advirá desse projeto é uma informação importante para projetos inovadores.

7.3. Atuando em inovação

Em 2015, a Agência Brasileira de Desenvolvimento Industrial (ABDI)[8] mencionava que os percentuais de firmas que empregam doutores, mestres, especialistas e graduados em atividades de P&D tiveram elevação ao longo dos últimos anos [ABDI 2015]. O percentual de empresas que tinham doutores exclusivamente ocupados em P&D no quarto trimestre de 2015 foi de 21,7%, as empresas que possuíam mestres ocupados exclusivamente em P&D alcançaram 44,7%, as com pós-graduados foram 72,3% e as com graduados ocupados exclusivamente em P&D foram 84,9%. Estes resultados demonstram que há espaço para a participação destes profissionais como empregados em áreas de P&D na indústria brasileira e que as empresas têm valorizado estes profissionais, com suas competências e habilidades de pesquisa.

A MIT Technology Review promovê o prêmio "Innovators under 35" (Inovadores abaixo de 35 anos), para eleger e premiar jovens ao redor do mundo por suas ações inventivas, empreendedoras, visionárias, humanistas e pioneiras nas mais diversas áreas, com uso de tecnologia. A instituição classifica seus jovens premiados como persistentes, curiosos, inspirados e inspiradores, e não importa se estão em busca de inovações na área médica, revendo tecnologias energéticas, tornando computadores mais úteis ou criando novos dispositivos; se estão gerenciando startups, se atuando em grandes empresas ou realizando pesquisas em laboratórios acadêmicos, todos estão posicionados a serem líderes em seus campos de atuação. prêmio teve duas edições no Brasil[9], reconhecendo as ações de vários jovens empreendedores brasileiros nas áreas médicas, educação, manufatura, economia coletiva, cidadania e negócios.

À nível profissional e individual, uma mentalidade empreendedora estimula os pesquisadores em sua capacidade de realizar os processos de inovação e no impacto que suas atitudes trazem para a economia e sociedade. Algumas características são recorrentes na avaliação de perfis inovadores: seu histórico de realização de ações empreendedoras (com ou sem sucesso) - criação de novos produtos, serviços e modelos de negócio; as evidências de seu conhecimento e experiência dentro de um tema ou área de negócio; sua capacidade de estabelecer parcerias; o estabelecimento de sua rede de contatos e referências pessoais; sua capacidade de captar recursos e estabelecer estratégias de sustentabilidade para suas ações. Mas há nos perfis inovadores algumas características também muito marcantes, não necessariamente objetivas, que nos saltam aos olhos: uma inquietude perante o status quo; uma curiosidade pelo novo; uma boa dose de ousadia; e um compromisso evidente com a solução de um problema.

[8] http://www.abdi.com.br

[9] http://www.technologyreview.com.br/tr35/

8. Conclusão

O mundo sempre foi um espaço desafiador para se viver, para construir, para compartilhar. Num mundo de desafios, a inovação sempre foi a propulsora de avanços em nossa forma de estar no mundo. Hoje, inovação é sinônimo de desenvolvimento econômico, social e soberania na economia do conhecimento e da colaboração. Inovação é busca, necessidade, curiosidade, inquietude, conhecimento, compartilhamento, interação, reflexão e coragem para mudar. Inovar - um tema espinhoso - mas que tem sido a única certeza neste mundo.

No Brasil, a inovação está ainda muito associada ao estímulo governamental. Inovar, como atividade de risco, se já é temerária para empresários, para os pesquisadores é uma ação ainda carregada de preconceitos e desconhecida. Mas, sem volta. É preciso pensar nela. Pensar em inovação requer uma visão de mundo além do contexto restrito de nossas atuações. Significa olhar para as implicações do que fazemos para nós, os coletivos que nos cercam, para a sociedade e para o futuro. Quais as implicações de nossas pesquisas? Que utilidades e valores trazem para o mundo?

Esperamos este curso estimule a curiosidade e a motivação para empreender, com foco na inovação. Inovar métodos, práticas de pesquisa e desenvolvimento de ideias. Inovar no relacionamento com as instituições e parceiros. Inovar nos objetivos do que você produz e traz de contribuição para a sua área de atuação, seu universo de conhecimento, a sociedade onde vive. Inovar seu olhar para encontrar oportunidades. Inovar sua atitude de observação do mundo onde nada pode ser considerado a verdade absoluta ou a solução definitiva.

Embora organizada em legislação, regras, e orientações criadas para dar organicidade à sua realização, para a inovação não há limites. Inovação não se limita a indicadores, a investimentos, a retorno econômico ou a benefícios sociais diretos e indiretos. Inovar significa crescer. Significa tornar realidade aquilo ao que o ser humano foi dado a possibilidade de fazer – moldar o mundo à sua vontade.

Referências Bibliográficas

ABDI (2015) “Sondagem de Inovação da Agência Brasileira de Desenvolvimento Industrial”, 4o Trimestre de 2015, Disponível em: http://www.abdi.com.br/Paginas/estudo_detalhe.aspx?e=Boletim+de+Acompanhamento+Setorial&f=Intelig%C3%AAncia%20industrial&n=2.

Alencar, M.S. (2016) “Patentes: Fonte de informação fundamental para inovação”. Palestra proferida na disciplina Estudos Dirigidos à Inovação PPGI-UNIRIO em novembro/2016.

ANPEI - Associação Nacional de Pesquisa e Desenvolvimento das Empresas Inovadoras 2014 “Mapa do Sistema Brasileiro de Inovação”. http://www.anpei.org.br/download/Mapa_SBI_Comite_ANPEI_2014_v2.pdf. 2014.

Araujo, R.M., Alves, A., Gouvea, M.T., Gomes, S.B., Frattini, V.C.M.S. (2018a) "Levantamento de Informação Tecnológica para Pesquisa: Uma Proposta de Sistematização". RelaTe-DIA – Relatórios Técnicos do DIA/UNIRIO. n° 0001/2018.

Araujo, R.M., Alves, A., Gouvea, M.T., Gomes, S.B., Frattini, V.C.M.S., Bessa, A.T., Reis, L.C.D., Dutra, E., Estruc, M.A. (2018b) "Onde está a Inovação? Perspectivas de inovação para pesquisas na área de Sistemas de Informação", RelaTe-DIA – Relatórios Técnicos do DIA/UNIRIO. n° 0002/2018.

Araujo, R.M., Procaci, T., Classe, T.M., Chueri, L.O.V. (2017) "Da Pesquisa Científica à Inovação". Araujo, R.M. e Chueri, L.O.V. (eds) Pesquisa & Inovação: Visões e Interseções, PUBL!T Soluções Editoriais. p. 22-46.

Araujo, R.M., Paula, L.G. (2017) "Avaliação da Inovação". Araujo, R.M. e Chueri, L.O.V. (eds) Pesquisa & Inovação: Visões e Interseções, PUBL!T Soluções Editoriais, p. 203-215.

Araujo, R. M. (2016) "Estudos Dirigidos à Inovação: Uma experiência na formação de pesquisadores-inovadores em Sistemas de Informação". In: III Encontro de Inovação em sistemas de Informação, Florianópolis. Sociedade Brasileira de Computação.

Borschiver, S. e Silva, A.L.R. (2016) "Technology Roadmap. Planejamento Estratégico para alinhar Mercado-Produto-Tecnologia". Editora Interciência. 120 p.

Carvalho, M. B. (2017) "Publicar e Proteger, um desafio possível". Palestra proferida na disciplina Estudos Dirigidos à Inovação PPGI-UNIRIO em 19/10/2017.

Chesbrough, H. (2003) "Open Innovation: The New Imperative for Creating and Profiting from Technology". Boston, Massachussets: Harvard Business School Press.

Chueri, L.O.V. (2017) "Inovação Social". In: Araujo, R.M., Chueri, L.O.V. eds. Pesquisa e Inovação: Visões e Interseções, Rio de Janeiro: PUBL!T Soluções Editoriais, p. 266-281.

Coelho, D. (2017) "Empreendedorismo". In: Araujo, R.M., Chueri, L.O.V. eds. Pesquisa e Inovação: Visões e Interseções, Rio de Janeiro: PUBL!T Soluções Editoriais, p. 216-242.

Grizendi, E. (2017) "Estratégias para inovação e maximização dos resultados tecnológicos". In: Araujo, R.M., Chueri, L.O.V. eds. Pesquisa e Inovação: Visões e Interseções, Rio de Janeiro: PUBL!T Soluções Editoriais, p. 139-162

Kitchenham, B. (2004) "Procedures for Performing Systematic Reviews". Joint Technical Report Software Engineering Group, Department of Computer Science Keele University, United King and Empirical Software Enginneering, National ICT Australia Ltd, Australia, 2004.

Koslosky M.A.N.; Speroni, R.M.; Gauthier, O. (2015) "Ecossistemas de inovação – Uma revisão sistemática da literatura". Espacios. Vol. 36 (Nº 03). Pág. 13

Jagher, T. (2017) “Busca em Banco de Dados de Patentes”. Agência de Inovação/ UTFPR. Disponível em: http://www.utfpr.edu.br/medianeira/estrutura/diretorias/direc/downloads/PROCEDIMENTOPARAPESQUISAUMAPATENTE.pdf

Laudon, K.C., Laudon, J.P (2016) “Management Information Systems. Pearson Education”.

Lei Nº 10.973, DE 2 DE DEZEMBRO DE 2004. “Lei de Inovação” http://www.planalto.gov.br/ccivil_03/_ato2004-2006/2004/lei/l10.973.htm.

LEI Nº 13.243, DE 11 DE JANEIRO DE 2016. “Marco Legal da Ciência, Tecnologia e Inovação”. Disponível em: http://www.planalto.gov.br/ccivil_03/_Ato2015-2018/2016/Lei/L13243.htm

Lemme, C.F. (2017) “Sustentabilidade, Inovação e Liderança”. In: Araujo, R.M., Chueri, L.O.V. eds. Pesquisa e Inovação: Visões e Interseções, Rio de Janeiro: PUBL!T Soluções Editoriais, p. 243-265.

LPI, Lei de Propriedade Industrial (LPI) no 9.279 de 14 de maio de 1996. Disponível , em: http://www.planalto.gov.br/ccivil_03/leis/l9279.htm Acesso em: 22/12/2017.

Manual de Oslo. (1997) “The Organisation for Economic Co-operation and Development – OECD”.

Manual de Oslo. (2004). Disponível em: http://download.finep.gov.br/imprensa/manual_de_oslo.pdf.

Nunes, M. A. S. N., Pinheiro-Machado, R. (2017) “Propriedade Intelectual e Busca de Informação Tecnológica na área da Computação”. Araujo, R. M. e Chueri, L.O V.(eds) Pesquisa & Inovação: Visões e Interseções, PUBL!T Soluções Editoriais. p. 67-92.

OCDE (OECD). (2003) “Turning Science into Business – Patenting and Licensing at Public Research Organizations”. Paris.

Oliveira Jr, J.A. (2017) “Estratégias de Financiamento para Projetos Inovadores”. Araujo, R. M. e Chueri, L.O.V. (eds) Pesquisa & Inovação: Visões e Interseções, PUBL!T Soluções Editoriais. p. 187-202.

Parker, D. P.; Zilberman, D. (1993) “University Technology Transfers: Impacts on Local and U.S. Economies”, Contemporary Policy Issues, Vol. XI, Abril, pp.87-99.

Pimentel, M., Filippo, D., Calvão, L.D., Silva, A.R. (2017) “Design Science Research: pesquisa científica para o desenvolvimento de artefatos inovadores”. In: Araujo, R.M., Chueri, L.O.V. eds. Pesquisa e Inovação: Visões e Interseções, Rio de Janeiro: PUBL!T Soluções Editoriais, p. 47-66.

Pinheiro-Machado, R. (2017) “Prospecção Tecnológica”, Palestra proferida na disciplina Estudos Dirigidos à Inovação PPGI-UNIRIO em 26/10/2017.

Recker, J. (2013) “Scientific Research in Information System: A Beginner’s Guide”, Springer.

Scherer, F.O, Carlomagno, M.S. (2009) “Gestão da Inovação na Prática”, Editora Atlas.

Tidd, J., Bessant, J., Pavitt, K. (2008) “Gestão da Inovação”, Bookman.

Wang, J. F. (2010); “Framework for university-industry cooperation innovation ecosystem: Factors and countermeasure”, Wuhan, p. 303-306.

Biografia das Autoras

Renata Araujo - http://lattes.cnpq.br/3589012014320121

Doutora em Engenharia de Sistemas e Computação, Renata é professora associada do Departamento de Informática Aplicada e pesquisadora do Programa de Pós-Graduação em Informática da UNIRIO. Seus tópicos de pesquisa são: Sistemas de Informação, Democracia e Governança Digital, Gestão de Processos de Negócio e Gestão da Inovação. Coordena o Núcleo de Pesquisa e Inovação em Ciberdemocracia, e o grupo de pesquisa em Sistemas de Informação de Governo Abertos e Colaborativos (https://sites.google.com/site/ciberdem/). Atualmente ocupa a Diretoria de Educação da SBC (2018-2019). Renata é responsável pela disciplina de Estudos Dirigidos à Inovação, do Programa de Pós-Graduação em Informática da UNIRIO e é uma das editoras do livro Pesquisa e Inovação, da editora PUBL!T Soluções Editoriais. Renata é bolsista de Produtividade em Desenvolvimento Tecnológico e Extensão Inovadora do CNPq, Brasil processo no 305060/2016-3.

Luciana de Oliveira Vilanova Chueri - http://lattes.cnpq.br/0586189515626396

Mestre em Engenharia de Sistemas e Computação, Luciana é doutoranda no Programa de Pós-Graduação em Informática da UNIRIO. Seus tópicos de pesquisa são: Sistemas de Informação, Inovação Social e Gestão de Conhecimento. Possui vinte e sete anos de atuação em projetos nas áreas: Óleo & Gás, Telecomunicações, Informática e Seguros, com sólida experiência em gerenciamento de projetos de grande porte e Comunidades de Práticas. Coordenadora e revisora do livro "Metodologia de Gerenciamento de Projetos do Terceiro Setor". Luciana é co-editora do livro Pesquisa e Inovação da editora PUBL!T Soluções Editoriais.

Capítulo 3

Governança de Desenvolvedores em Ecossistemas de Software

Awdren de Lima Fontão, Igor Scaliante Wiese, Rodrigo Pereira dos Santos e Arilo Claudio Dias-Neto

Abstract

In a Software Ecosystem (SECO), software organizations have started to open their internal structure to external developers to reach goals, such as increasing the number of mobile applications (apps). In this way, the organization needs to develop and evolve the mechanisms of developer governance (i.e., concepts, values and practices) as a way to maintain ecosystem sustainability and diversity. The purpose of this paper is to present definitions and strategies relevant to SECO developer governance. This is a work with an introductory scope that presents the fundamentals of the following topics: SECO, Developer Relationships and Mining Software Repositories.

Resumo

Em um Ecossistema de Software (ECOS), as organizações de software passaram a abrir a sua estrutura para desenvolvedores externos visando atingir metas, como o aumento do número de aplicações móveis (apps). Desta forma, a organização precisa elaborar e evoluir os mecanismos (i.e., conceitos, valores e práticas) de governança de desenvolvedores como forma de manter a sustentabilidade e a diversidade do ecossistema. O objetivo deste trabalho é apresentar definições e estratégias relevantes para a governança de desenvolvedores em ECOS. Este é um trabalho com escopo introdutório que apresenta os fundamentos dos seguintes tópicos: ECOS, Relações com Desenvolvedores e Mineração de Repositórios.

3.1. Introdução

As organizações que mantêm plataformas de software, como Apple e Google, têm investido em uma infraestrutura para recrutar e engajar desenvolvedores externos como forma de contribuir para a expansão da plataforma. Este cenário que envolve o

relacionamento dos desenvolvedores com uma organização central (*keystone*) por meio de uma plataforma tecnológica central tem sido estudado na Engenharia de Software como Ecossistema de Software (ECOS) [Jansen e Brinkkemper 2009].

Um ecossistema envolve as dimensões técnica (o processo de desenvolvimento), social (a interação entre organização e desenvolvedores) e negócio (balanço entre os objetivos da organização e as expectativas do desenvolvedor). Neste contexto, o desenvolvedor é um ator essencial, pois é ele quem utiliza os recursos disponibilizados pela organização central para gerar contribuições para o ecossistema, que serão consumidas e avaliadas pelos usuários [Fontão *et al.* 2016]. O desempenho do desenvolvedor precisa ser acompanhado com o objetivo de identificar e prever áreas de melhoria. Este cenário na literatura técnica se refere ao conceito de saúde, que Manikas e Hansen (2013) definem como "*a habilidade do ecossistema em suportar perturbações e permanecer variável e produtivo durante o tempo*".

As organizações utilizam modelos de governança para atingir suas metas, melhorar os recursos disponíveis e, em última instância, aumentar seus lucros e reduzir eventuais riscos [Jansen e Cusumano 2012]. A governança consiste em um modelo que reúne um conjunto de premissas, conceitos, valores e práticas relativos à organização, ao relacionamento entre as partes envolvidas e a como os recursos são administrados e monitorados para que atinjam as metas estabelecidas [Alves *et al.* 2017].

O bem-estar econômico e social dos desenvolvedores assim como relacionamentos sinérgicos entre as expectativas do desenvolvedor e os objetivos da organização central devem ser garantidos por um grupo de relacionamentos com os desenvolvedores (DevRel, do inglês *Developer Relations*). Neste cenário, o ecossistema emerge como forma de apoiar a diversidade e sustentabilidade dos negócios da organização que mantém a plataforma.

Como a organização mantenedora do ECOS não mantém uma relação direta com um conjunto de desenvolvedores, estes usam repositórios como: portais de perguntas e respostas (StackOverflow) e repositórios de código (Github), entre outros, como uma forma de avançar dentro do ecossistema. Logo, os repositórios são fonte de informação relevante para adaptação de estratégias de governança de desenvolvedores.

Nesse contexto, o objetivo deste trabalho é apresentar definições e estratégias relevantes para a governança de desenvolvedores em ECOS. Este é um trabalho com escopo introdutório que envolve os seguintes tópicos: ECOS, Relações com Desenvolvedores e Mineração de Repositórios. A governança de desenvolvedores cobre três dimensões: social, técnica e negócio. Além disso, busca promover a sinergia nas relações que envolvem as expectativas dos desenvolvedores e os objetivos da organização que mantém um ecossistema.

Em uma edição em que o SBSI tem como tema "Sistemas de Informação: uma perspectiva social, sustentável e de negócios", discutindo uma prática gerencial que aplica as características das redes sociais e ferramentas sociais na administração de uma organização como forma de criar valor para os envolvidos, este trabalho vem então a contribuir com conceitos e aplicações em um cenário que envolve novos tipos e interações com/de sistemas de informação. No nosso caso, demonstramos as estratégias

relevantes para governar desenvolvedores e abordamos o uso de repositórios como fonte de informações técnicas, sociais e de negócios relativas aos desenvolvedores.

3.2. Ecossistemas de Software

Os relacionamentos que passaram a existir com a evolução do desenvolvimento de software, envolvendo componentes, infraestrutura e serviços de outras empresas, direcionaram o cenário do produto de software para um ecossistema Desta forma, fornecedores e consumidores de produtos de software começaram a criar tecnologia de forma colaborativa para gerar valor [Jansen e Bloemendal 2013].

Em [Jansen *et al.* 2009], os autores se referem a ECOS como um conjunto de negócios funcionando em unidade e interagindo com um mercado compartilhado de software e serviços, junto com suas relações apoiadas por uma plataforma tecnológica central e realizadas por meio de troca de informação, recursos e artefatos. Segundo Bosch (2009), ECOS consiste de uma plataforma de software, um conjunto de desenvolvedores internos e externos e uma comunidade de especialistas a serviço de uma comunidade de usuários que constroem soluções relevantes para satisfazer as suas necessidades.

3.2.1 Elementos de ECOS

O cenário de ecossistemas (Figura 3.1) fez novos modelos de negócios surgirem na Engenharia de Software, redefinindo os papéis e padrões de colaboração e inovação, criando complexas redes de organizações ou comunidades de contribuidores. Este cenário torna o ECOS um tema de pesquisa novo e importante na área de Engenharia de Software [Manikas 2016]. Para entender o tema "ECOS", é importante ter uma visão sobre os tipos de elementos que os compõem, seus papéis e atividades descritos a seguir [Jansen *et al.* 2013]:

- **Plataforma:** termo genérico que se refere a padrão de arquitetura, protocolo de comunicação ou qualquer conhecimento fundamental e compartilhado. Uma plataforma é a base na qual elementos técnicos de um ecossistema são construídos. Provê o suporte para a customização em larga escala;
- **Organização Central:** provê padrões e tecnologias que são o fundamento, ou parte, do ECOS. Cria e compartilha valor dentro do ECOS. Precisa ter visão geral do ECOS e saber qual o foco do contribuidor para assim identificar oportunidades. Influencia, define padrões e práticas, acelera a especialização e aumenta o valor em número de contribuidores. Tem como missão melhorar a saúde do ECOS pela disponibilização de um conjunto estável e previsível de artefatos comuns que outros elementos podem utilizar para construir, por meio do reúso, suas próprias contribuições dentro do ECOS. A organização central pode monitorar a saúde do ECOS a tomar medidas para promover a saúde do ECOS, se necessário. Para isso, a organização central deve ter visão geral do ECOS e estar consultando as suas medidas da saúde;
- **Desenvolvedor:** requer que os padrões ou tecnologias que sejam fornecidos pela organização central e que possam gerar valor de negócio estejam bem definidos e divulgados. Existem alguns perfis de contribuidores: os *evasivos*,

que participam em dois ECOS para minimizar riscos; os *discípulos*, que adotaram recentemente a plataforma e a divulgam para outros; e os *influenciadores*, que requisitam características da plataforma, organizam conferências e formam comunidades. Existe ainda o *agente intermediário* que serve como uma interface entre dois contribuidores. O desenvolvedor é responsável por novas ideias e pelo desenvolvimento de aplicações móveis no ECOS, respondendo aos requisitos dos usuários [Fontão *et al.* 2014]. Quando um desenvolvedor adota uma API para o desenvolvimento de aplicações, ele pode ser considerado um cliente [Kim *et al.* 2002]. Desenvolvedores podem ser classificados em individual ou organizacional [Hyrynsalmi *et al.* 2014];

- **Usuário:** é a pessoa, companhia ou entidade que pode comprar ou obter um produto parcial ou completo de um ECOS ou de um contribuidor;
- **Comunidade:** estrutura de colaboração e coordenação de atividades de um ECOS, composta por contribuidores internos e externos [Miranda *et al.* 2014]. Existem comunidades específicas, tais como: comunidades de usuários, comunidades de desenvolvedores e comunidades de especialistas, sendo que esta última realiza treinamento ou serviços de suporte [Taylor 2013];
- **Evangelista:** equipe de profissionais que mantém a relação com os desenvolvedores da organização que participa em treinamentos, palestras e competições de desenvolvimento com o objetivo de ajudar na expansão do ECOS e na formação de novos desenvolvedores [Fontão *et al.* 2014]. O evangelista faz parte da comunidade de especialistas e é um funcionário interno da organização do ECOS. É um especialista em um campo específico e tem conhecimento das atividades dentro do ECOS [Taylor 2013].

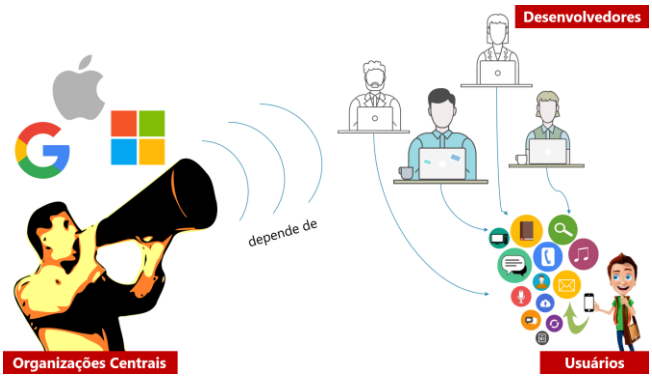


Figura 3.1. Um cenário em ECOS

3.2.2 Tipos e Dimensões de ECOS

Segundo Manikas (2016), pode-se classificar os ECOS por uma perspectiva de meios de criação de valor no ecossistema, como segue:

- **Proprietário:** a criação de valor no ecossistema é baseada em contribuições proprietárias. Normalmente, é protegido por processos de gerenciamento de propriedade intelectual e o valor se refere à compensação monetária. Industrial, plataforma como serviço e ecossistemas de comércio eletrônico são incluídos nesta categoria;

- **Código aberto:** as contribuições estão abertas aos participantes do ecossistema. O valor se refere a compensações não monetárias, e.g., conhecimento e experiência, ou pesquisa de satisfação. A Fundação Eclipse, o Gnome, a Fundação Apache e os ecossistemas do governo são incluídos nesta categoria;
- **Híbrido:** apoia contribuições proprietárias e de código aberto. Os seguintes ecossistemas foram adicionados a esta categoria: Android, iOS, Windows Phone e ecossistema de software móvel.

Além do contato com outras áreas, podem ser observadas oportunidades de pesquisa em dimensões dentro de ECOS. Santos e Werner (2012) apresentam uma visão "3+1" das dimensões de ECOS, como exibido na Figura 3.2:

- ***Dimensão Técnica:*** foca na plataforma de um ECOS. Isto engloba o mercado, a tecnologia, infraestrutura ou organização, definindo assim o seu ciclo de vida e as características da plataforma;
- ***Dimensão de Negócios:*** o fluxo do conhecimento dentro de um ECOS é o objetivo da análise a partir desta dimensão;
- ***Dimensão Social:*** todas as relações entre partes envolvidas e interessadas dentro de um ECOS, os *stakeholders*, fazem parte desta dimensão. Tem como objetivo ainda entender qual o motivo dos envolvidos integrar, estender e modificar o conhecimento em um ECOS e a interação entre os envolvidos;
- ***Dimensão de Gerenciamento e Engenharia:*** junção das três dimensões anteriores, por meio de três relacionamentos:
    - *Motivando o desenvolvimento e a evolução da plataforma*: entendimento dos relacionamentos e modelos do sistema, mas não de forma técnica;
    - *Contribuindo para o estabelecimento da plataforma:* envolvimento e atenção para a comunidade;
    - *Mapeamento de proposições valorosas e realizações:* conceito de valor da plataforma do ponto de vista de todos os *stakeholders* e qual o sentido deste valor.

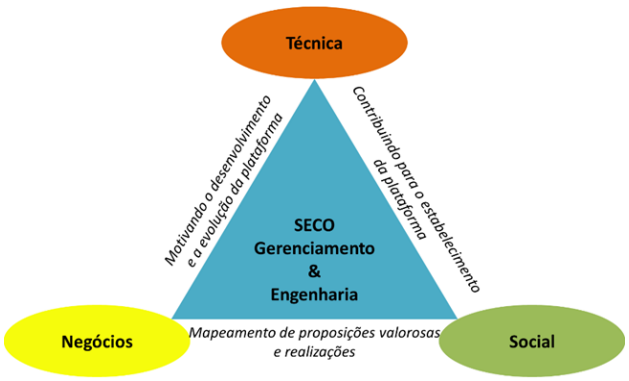


Figura 3.2. Visão "3+1" das dimensões de um ECOS
Fonte: [Santos e Werner, 2012]

3.3. Governança em ECOS

Governança é definida como a forma que uma organização é gerenciada, incluindo suas responsabilidades e processos de apoio à decisão [Dubinsky e Kruchten 2009]. A governança envolve a atribuição de responsabilidades e direitos de decisões, assim como indicadores e políticas que permitem a avaliação contínua. O gerenciamento e monitoramento bem-sucedidos ainda são grandes desafios para profissionais de ecossistemas [Dhungana *et al.* 2010]. Isto acontece porque a comunidade de pesquisa em ECOS ainda carece de teorias específicas de gerenciamento, ferramentas de suporte e experiência consolidada no tema [Manikas 2016].

Alves *et al.* (2017) definem governança de ecossistemas como ferramentas gerenciais para os atores do ecossistema que têm o objetivo de influenciar a saúde do ECOS. Saúde, neste contexto, se referee à garantia de que o ecossistema está funcionando como esperado. Indicadores específicos podem ser utilizados para prover uma visão geral do estado do ecossistema e ao mesmo tempo direcionam para ações e permitem a comparação com outros ecossistemas. Para medir o quanto um ecossistema é saudável, Iansiti e Richards (2006) definem as métricas:

- **Produtividade:** é a habilidade para criar energia. Pode ser medida por meio do fator total de produtividade, melhoria de produtividade por meio do tempo e entrega de inovações (i.e., habilidade do ecossistema de se adaptar e entregar aos seus membros novas tecnologias, processos e até mesmo ideias);
- **Robustez:** é a habilidade do ecossistema de sustentar perturbações e desligamentos. É medida por meio da persistência da estrutura do ecossistema (capacidade de manter os relacionamentos), previsibilidade (capacidade do núcleo de um ecossistema permanecer sólido mesmo acontecendo ruído entre os envolvidos), obsolescência limitada (capacidade de controlar a utilidade de componentes e tecnologia), e continuidade da experiência de uso e casos de uso (capacidade dos produtos evoluírem em resposta a mudança em tecnologias);
- **Criação de Nicho ou Inovação:** é a habilidade do ecossistema aumentar significativamente a diversidade de envolvidos ao longo do tempo. É medida por meio do crescimento/decrescimento na variedade de produtos da companhia e crescimento/decrescimento na variedade técnica (conhecimento dos desenvolvedores) e de produto (criação de valor).

3.3.1 Sáude de ECOS

O conceito de saúde de ecossistemas começou a ser abordado por Iansiti e Levien (2004) como um modo de medir o desempenho de um ecossistema de negócios. A literatura técnica faz analogias com Ecossistemas Naturais e aproveita termos dos Ecossistemas de Negócios. Em ecossistemas naturais, um ecossistema saudável é definido como estável e sustentável, mantendo sua organização e autonomia em relação ao tempo e à sua capacidade de resistir a situações de estresse [Schaeffer *et al.* 1988].

Do ponto de vista do desenvolvedor nas comunidades que atuam com software de código aberto, o conceito de saúde de um projeto é definido como a capacidade de sobrevivência, ou seja, a capacidade do projeto sobreviver com o passar do tempo

[Manikas 2016]. Em [Mustofa *et al.* 2007], os autores identificam três métricas que afetam a saúde de um projeto de código aberto:

- **Ânimo da comunidade de desenvolvedores:** o projeto deve ser atrativo para novos desenvolvedores e deve manter os desenvolvedores atuais, melhorando a sua motivação (estimulação intelectual, melhoria de habilidades, acesso ao código fonte e necessidades dos usuários);
- **Ânimo da comunidade de usuários:** os usuários têm um importante papel na evolução do projeto ao reportar defeitos e ao requisitar novas funcionalidades. Uma comunidade grande e ativa de usuários indica que o software produzido é de qualidade;
- **Qualidade do produto:** um produto competitivo com outros produtos comerciais, tanto no uso quanto na qualidade, quando atrai usuários e desenvolvedores, aumenta a atividade no projeto e melhora a capacidade de sobrevivência.

Em uma analogia com a Ecologia, especificamente no estudo da ciência dos ecossistemas, Begon *et al.* (2007) apontam que um dos motivos pelos quais é necessário estudar ECOS é para entender os processos que dão suporte aos produtos que são construídos e consumidos e, essencialmente, aumentam a produção. Além de disseminar boas práticas e padrões arquiteturais para garantir a integração de produtos no ECOS, a organização central deve rever seus processos com o objetivo de melhorar a experiência dos contribuidores do ECOS e estabelecer controles adequados aos diferentes tipos de contribuidores. Wazlawick (2013) discorre sobre as vantagens de analisar o desenvolvimento de algum software a partir da perspectiva de processos:

- **Reduzir o tempo de treinamento:** é mais fácil encaixar novos indivíduos em uma equipe quando os processos são bem definidos e documentados;
- **Uniformizar a construção de produtos:** uma equipe com um processo bem definido tende a construir um produto mais bem definido se comparada a uma equipe sem processo;
- **Capitalizar experiências:** se um desenvolvedor faz algo de forma diferente a partir de sua criatividade, isto pode ser incorporado nos processos como uma melhoria.

3.3.2 Processos em ECOS

A partir da análise das fases e dos elementos que compõem um ECOS, Fontão (2016) identificou três processos (Figura 3.3): (1) *Orquestração* – que surge da interação entre a organização central e o evangelista, uma vez que este se baseia nas diretrizes do ECOS para executar atividades de suporte; (2) *Suporte* – que estrutura o fluxo de trabalho entre o evangelista e o desenvolvedor que é orientado para seguir as diretrizes do ecossistema na produção de aplicações móveis; e (3) *Desenvolvimento* – atividades para a construção da aplicação móvel que será adquirida e avaliada pelos usuários do ECOS.

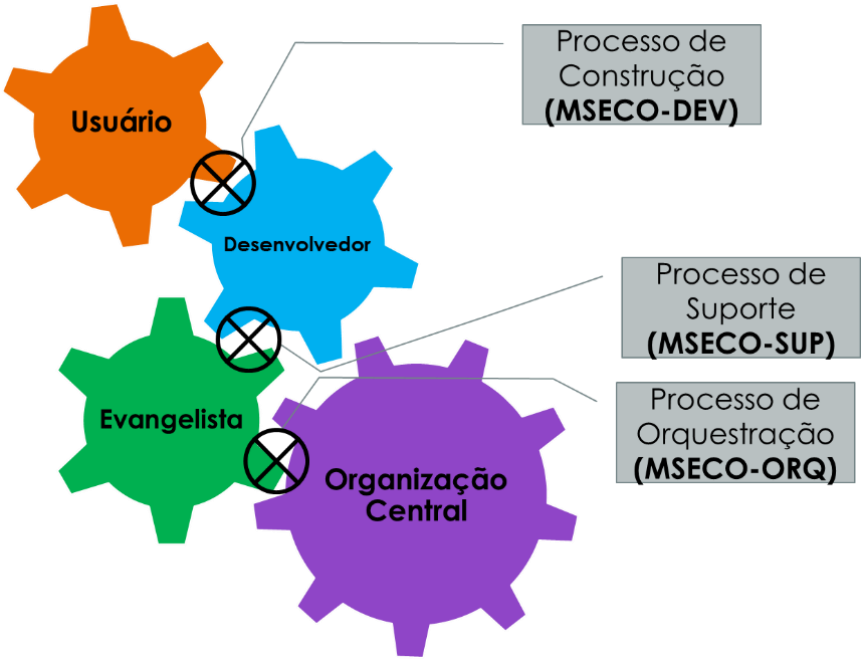


Figura 3.3. Relação entre os processos de Orquestração, Suporte e Construção

O objetivo do processo de orquestração é preparar, gerenciar e coordenar alguns elementos (e.g., organização central, desenvolvedor, evangelista e a contribuição, no caso, a aplicação móvel) [Jansen *et al.* 2009] e alguns de seus relacionamentos em ECOS. Além disso, visa fornecer diretrizes e guias necessários às contribuições para o ecossistema com o objetivo de manter os indicadores de saúde de ECOS: robustez, produtividade e criação de nicho.

O objetivo do processo de suporte é prover o relacionamento entre a organização central e os desenvolvedores. Para isso, o ECOS dispõe de um elemento responsável por esta ligação: o evangelista. Por fim, o objetivo do processo de construção é que o desenvolvedor idealize, planeje e construa uma aplicação móvel que será adicionada à loja de aplicações. Para isso, o desenvolvedor poderá utilizar artefatos gerados tanto no processo de orquestração como no processo de suporte. Dessa forma, ele pode contribuir com a produtividade e a criação de nicho do ECOS.

O evangelista precisa motivar e engajar outros desenvolvedores externos e apoiá-los na expansão da comunidade e, consequentemente, na disponibilização de uma maior quantidade de aplicações móveis para os usuários. A organização central precisa empreender esforços na disponibilização de uma estrutura de base para que o ECOS possa se expandir por meio de diretrizes para atrair novos desenvolvedores e criar aplicações móveis, além do fornecimento de ferramentas e suporte à comunidade. O desenvolvedor precisa criar e disponibilizar contribuições que atendam da melhor forma os nichos existentes de usuários e também obter visibilidade dentro do ECOS.

Na Figura 3.4, apresentamos um processo que a organização central utiliza para realizar a governança dos desenvolvedores dentro de um ECOS. A sua descrição é realizada a seguir:

O papel responsável por este processo é:

Papel:	**Proprietário da Plataforma (ou *Organização Central*)**
Descrição:	Responsável pelo provimento de padrões e práticas e pela identificação de oportunidades.

Os papéis participantes neste processo são:

Papel:	**Equipe de Desenvolvimento da Organização Central**
Descrição:	Responsável pelo desenvolvimento da plataforma que faz parte do ECOS.
Papel:	**Equipe de Design e UX (Experiência de Usuário) da Organização Central**
Descrição:	Responsável pela definição de padrões de interface da plataforma.
Papel:	**Equipe de Marketing de Produto**
Descrição:	Responsável pela divulgação da plataforma e dos produtos que são construídos a partir dela.
Papel:	**Equipe de Marketing de Desenvolvedores**
Descrição:	Responsável pela divulgação dos desenvolvedores e das contribuições deles dentro do ECOS.
Papel:	**Equipe de Estratégia de Tecnologia**
Descrição:	Responsável pela proposta, definição e análise das tecnologias pertencentes à organização central.
Papel:	**Equipe de Validação de Produto da Organização Central**
Descrição:	Responsável por definir requisitos de qualidade para aceitação de contribuições para o ECOS.
Papel:	**Equipe Jurídica da Organização Central**
Descrição:	Responsável pela parte legal que envolvem os elementos e a organização central no ECOS.
Papel:	**Equipe de Evangelismo**
Descrição:	Responsável pela ligação entre a organização central e o desenvolvedor. Faz parte de uma comunidade de especialistas da organização.

Os artefatos do processo de orquestração são:

Artefato:	**Especificação da Plataforma**
Descrição:	Descreve como a plataforma está organizada internamente e seu nível de abertura para os desenvolvedores externos. No caso de ECOS, deve descrever os dispositivos móveis envolvidos e suas características, linguagem de programação que pode ser utilizada, tecnologias suportadas, APIs, SDKs e o posicionamento no mercado e com os usuários.

Artefato:	**Guias de Design e Interface de Usuário**
Descrição:	Descreve quais são os padrões de interface de usuário dos dispositivos móveis e da interação entre a aplicação e o usuário (padrões de tela e elementos visuais, componentes, animações, gestos de interação, mensagens).

Artefato:	**Aplicações Móveis de Referência**
Descrição:	Um conjunto de aplicações móveis com os padrões de interface, componentes e APIs e SDKs, que podem servir como modelo para o desenvolvedor iniciar um novo projeto.

Artefato:	**Guias e Ferramentas de Marketing**
Descrição:	Descreve como otimizar o acesso às aplicações dentro da loja e lista ferramentas e orientações para divulgação logo após a publicação.

Artefato:	**Ferramenta de Desenvolvimento**
Descrição:	Permite a criação da aplicação utilizando as linguagens, APIs e SDKs disponíveis para a plataforma. Gera ainda o binário da aplicação, pacote que poderá ser disponibilizado na loja.

Artefato:	**Central do Desenvolvedor**
Descrição:	Permite o acesso a *links* relacionados a: plataforma, ferramentas, documentos, suporte, fóruns, *wikis* e controle de publicação de aplicações da loja.

Artefato:	**Loja de Aplicações Móveis**
Descrição:	É um ambiente democrático para acesso às aplicações desenvolvidas e que poderão ser adquiridas pelos usuários.

Artefato:	**Critérios da Loja**
Descrição:	Descrevem os critérios de aceitação para qualidade de aplicações que poderão compor a loja. Esses critérios se baseiam em requisitos funcionais.

Artefato:	**Diretrizes do ECOS**
Descrição:	Sintetizar e alinhar todos os outros artefatos gerados nesse processo para um conjunto coerente de diretrizes para atuar dentro de um ECOS.

Artefato:	**Pacote de Políticas de Incentivos**
Descrição:	Descreve um conjunto de políticas para executar as estratégias definidas pela organização central com o objetivo de motivar e engajar os desenvolvedores a partir das contribuições (aplicação ou conteúdo para compartilhar conhecimento) para o ECOS, por meio de metas estabelecidas pela própria organização central.

Nesse processo, as atividades têm como objetivo principal gerar a base para o funcionamento do ECOS. Por isso, o responsável por todas as atividades é a organização central. As atividades contam ainda com a participação de equipes que fazem parte da organização (e que podem contribuir na geração dos artefatos): de desenvolvimento de software, de design, de marketing de produto, de marketing de desenvolvedores, jurídica, de validação de produto, de evangelistas e de desenvolvimento do sistema operacional da plataforma do ecossistema.

Os documentos gerados ao final deste processo formam a base do ecossistema, que compõem, ao final do processo, o artefato *Diretrizes do ECOS*, que servirá para outros processos. Esses documentos são: *Especificação da Plataforma*, *Guias de Design e Interface*, *Guias e Ferramentas de Marketing*, *Ferramentas de Desenvolvimento*, *Central do Desenvolvedor*, *Loja de Aplicações* e *Critérios da Loja.* Na Figura 3.4, o Processo de Orquestração é apresentado.

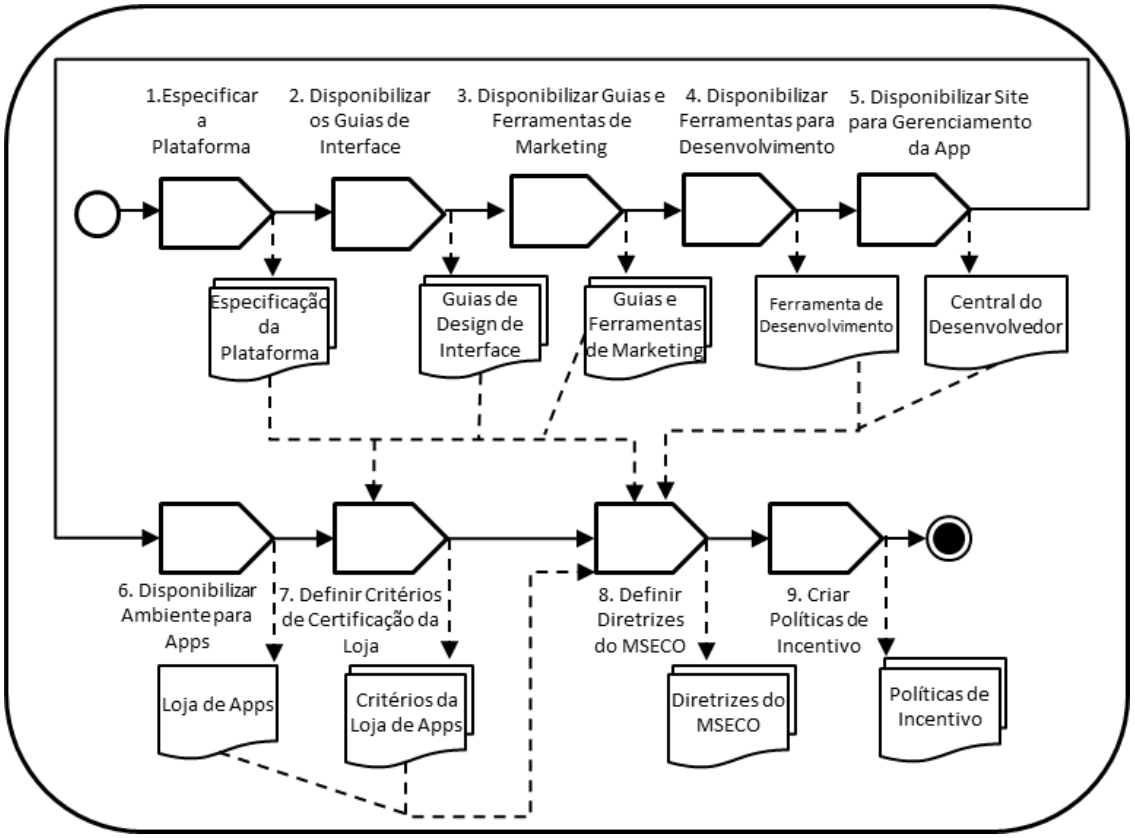


Figura 3.4. Processo de orquestração de desenvolvedores

As atividades que estão relacionadas à definição de estratégias têm como objetivo fornecer o suporte para funcionamento do ecossistema, inclusive as ameaças ou oportunidades que possam afetar de alguma forma o ecossistema. Essa atividade é de responsabilidade da organização central, podendo contar com a participação de evangelistas e de outros setores da organização. Abaixo, são apresentadas as atividades que compõem o processo de governança:

Atividade:	**1 Especificar a Plataforma**
Descrição:	Especificar detalhes das plataformas apoiadas pelo ecossistema é uma atividade de base para o bom funcionamento do ECOS. Seu objetivo é descrever dispositivos suportados e suas características, tecnologias, linguagens de desenvolvimento, APIs, SDKs, posicionamento no mercado e com os usuários.
Critérios de Entrada:	Não se aplica.
Critérios de Saída:	Especificação da plataforma consolidada e aprovada.

Responsável:	*Organização Central*.
Participantes:	Equipe de Desenvolvimento da Organização Central e Equipe de Estratégia de Tecnologia.
Artefatos Requeridos:	Não se aplica.
Artefatos Produzidos:	Especificação da plataforma (dispositivos suportados e todas suas características, tecnologias, linguagens de desenvolvimento, APIs, SDKs, posicionamento no mercado e com os usuários).
Recomendação:	A especificação deve ter versões tanto para acesso interno da organização (com informações mais restritas), como para acesso dos desenvolvedores externos, de forma a não expor informações confidenciais.

Atividade:	**2 Disponibilizar Guias de Design e Interface de Usuário da Plataforma**
Descrição:	A interface de usuário de uma plataforma é sua identidade. É importante divulgar guias que ajudem os desenvolvedores a criar e desenvolver soluções que respeitem os padrões de design e interface da plataforma do ECOS. Essa atividade tem como objetivo disponibilizar a base necessária para manter a boa interação com o usuário e o padrão da plataforma do ECOS.
Critérios de Entrada:	Ter-se a Especificação da Plataforma pronta ou atualizada.
Critérios de Saída:	Documentos com especificação de interface e interação da plataforma criados e aprovados.
Responsável:	*Organização Central*.
Participantes:	Equipe de Design e Equipe de UX (Experiência do Usuário) da Organização Central.
Artefatos Requeridos:	Especificação da Plataforma.
Artefatos Produzidos:	Guias de Design e Interface de Usuário, e Aplicações Móveis de Referência.
Recomendação:	Deve-se levar em consideração as limitações e potencialidades da plataforma na execução desta atividade.

Atividade:	**3 Disponibilizar Guias e Ferramentas de Marketing**
Descrição:	Toda aplicação precisa de um trabalho para ganhar exposição e visibilidade. É necessário que seja criado um conjunto de guias para rapidamente entregar a aplicação ao mercado e dar o reconhecimento que ela necessita, assim como ao seu desenvolvedor. É necessário que sejam disponibilizadas ferramentas para ajudar no reconhecimento da aplicação e desenvolvedor e também na entrega da aplicação ao mercado.
Critérios de Entrada:	Para essa atividade iniciar, a especificação da plataforma deve ter sido concluída, assim como os guias de design de interface da plataforma devem estar prontos.
Critérios de Saída:	Ferramentas e documentos que ajudem na otimização da aplicação quanto a marketing devem estar concluídos e aprovados.
Responsável:	*Organização Central*.
Participantes:	Equipe de Marketing de Produto, Equipe de Marketing de Desenvolvedores.
Artefatos Requeridos:	Especificação da Plataforma e Guias de Design e Interface de Usuário.
Artefatos Produzidos:	Guias e Ferramentas de Marketing.

Recomendação:	É importante que os guias de marketing levem consideração: •Marketing dentro da Loja de Aplicações •Marketing fora da Loja de Aplicações •Marketing dentro do dispositivo

Atividade:	**4 Disponibilizar Ferramenta para Desenvolvimento**
Descrição:	Essa atividade tem como objetivo disponibilizar as ferramentas principais que são necessárias para a construção de aplicações para a plataforma.
Critérios de Entrada:	Para essa atividade iniciar é necessário que a plataforma tenha sido especificada, os guias de design e interface de usuário estejam prontos e aprovados.
Critérios de Saída:	Ferramenta deve permitir a geração de um pacote binário da aplicação, criação de projetos, disponibilização de modelos de projetos de aplicações e depuração de aplicações.
Responsável:	*Organização Central*.
Participantes:	Equipe de Desenvolvimento da Organização Central e Equipe de Experiência do Usuário.
Artefatos Requeridos:	Aplicações Móveis de Referência, Especificação da Plataforma, Guias de Design e Interface de Usuário.
Artefatos Produzidos:	Ferramenta de Desenvolvimento.
Recomendação:	Uma ferramenta de desenvolvimento deve permitir: •O acesso a guias da plataforma; •Desenvolvimento de uma aplicação completa; •Desenvolvimento de testes, no mínimo em nível de unidade; •Depuração do código da aplicação; •Geração de pacote tanto em modo de depuração como em modo de entrega; •Interagir com emuladores de dispositivos da plataforma.

Atividade:	**5 Disponibilizar Site para Gerenciamento da Aplicação Móvel**
Descrição:	Com esta atividade, pretende-se disponibilizar um site que permita o envio e o gerenciamento das aplicações criadas pelo desenvolvedor.
Critérios de Entrada:	Para essa atividade iniciar, é necessário que a plataforma tenha sido especificada, os guias de design de interface estejam prontos e aprovados e que, além disso, exista uma ferramenta que possa gerar um pacote com o conteúdo da aplicação.
Critérios de Saída:	Site que permita a submissão de uma nova aplicação, edição, remoção e atualização. Deve-se ainda permitir o acesso a relatórios de *downloads* (total e por cada aplicação) e de revisões de usuários (estrelas e comentários).
Responsável:	*Organização Central*.
Participantes:	Equipe de Desenvolvimento da Organização Central e Equipe de Marketing de Produto.
Artefatos Requeridos:	Especificação da Plataforma.
Artefatos Produzidos:	Central do Desenvolvedor.
Recomendação:	Deve-se levar em consideração que o desenvolvedor, nesse momento, é um publicador de conteúdo (a aplicação), logo, o site deve conter todas as informações e/ou *links* que possam juda-lo durante o processo de submissão

	e gerenciamento de aplicações.

Atividade:	**6 Disponibilizar Ambiente para Aplicações Móveis**
Descrição:	Atividade que tem como objetivo a disponibilização de um ambiente que represente: para o desenvolvedor, um local de distribuição de aplicações; para o usuário, um local de aquisição de aplicações; e para o ecossistema, um local democrático de acesso a aplicações pelos elementos do ECOS.
Critérios de Entrada:	Deve existir um portal de gerenciamento da aplicação, que foi desenvolvido sobre a especificação da plataforma do ECOS e dos guias de design de interface.
Critérios de Saída:	Este ambiente deve estar disponível como um *site* e também estar embarcado nos dispositivos da plataforma do ECOS. Deve permitir que usuários possam adquirir e revisar aplicações.
Responsável:	*Organização Central*.
Participantes:	Equipe de Desenvolvimento da Organização Central.
Artefatos Requeridos:	Não se aplica.
Artefatos Produzidos:	Loja de aplicações.
Recomendação:	O ambiente deve ser democrático dos seguintes pontos de vista: •*Desenvolvedor/publicador:* a partir da participação de tipos variados de desenvolvedores (individuais ou companhias), oferecer opções para disponibilização de uma aplicação gratuitamente, de forma paga ou com pagamento dentro da aplicação; •*Usuário:* ambiente embarcado no celular como uma loja que permita a busca, indicações de aplicações, aquisição de aplicação e revisão da aplicação. O usuário pode adquirir aplicações gratuitas ou pagas e aplicações em testes.

Atividade:	**7 Definir Critérios de Certificação de Qualidade para Aceitação da Loja**
Descrição:	Os critérios de certificação de qualidade de uma loja de aplicações têm como objetivo garantir que requisitos em conformidade com a plataforma e com a interface de usuário sejam atendidos. Essa atividade tem como objetivo consolidar um guia com os critérios para a loja do ECOS que devem ser atendidos pela aplicação construída por um desenvolvedor.
Critérios de Entrada:	Especificação da plataforma e guias de design de interface devem estar prontos e aprovados.
Critérios de Saída:	Os critérios devem considerar especificação da plataforma, guias de marketing, legislação referente a conteúdo local do país e guias de interface da plataforma.
Responsável:	*Organização Central*.
Participantes:	Equipe de Validação de Produto da Organização Central e Equipe de Desenvolvimento da Organização Central.
Artefatos Requeridos:	Especificação da Plataforma, Guias e Ferramentas de Marketing e Guias de Design e Interface de Usuário.
Artefatos Produzidos:	Critérios da Loja.

Recomendação:	Os critérios devem levar em consideração os dispositivos que compõem a plataforma, a especificação tanto de hardware quanto de software, os guias de design de interface e legislação própria de cada país para questões de acesso a conteúdo digital.

Atividade:	**8 Definir Diretrizes do ECOS**
Descrição:	O objetivo desta atividade é condensar todos os artefatos gerados nas atividades anteriores de forma a criar um documento e/ou local para acesso a todos os artefatos. A Equipe de Marketing de Desenvolvedores reunirá em um documento, que pode ser um *website*, todos os artefatos gerados nas fases anteriores provendo rápido e fácil acesso aos artefatos.
Critérios de Entrada:	Todos os artefatos devem estar consolidados e atualizados.
Critérios de Saída:	Documento que possa ser acessado tanto por desenvolvedores internos quanto externos do ECOS.
Responsável:	*Organização Central*.
Participantes:	Equipe de Marketing de Desenvolvedores e Equipe de Estratégia de Tecnologia.
Artefatos Requeridos:	Especificação da Plataforma, Guias de Design e Interface de Usuário, Guias e Ferramentas de Marketing, Ferramenta de Desenvolvimento, Central do desenvolvedor.
Artefatos Produzidos:	Documento com diretrizes do ECOS contendo as ligações entre os artefatos gerados em atividades anteriores.
Recomendação:	As diretrizes devem fazer uma ligação coerente entre todos os artefatos gerados nas atividades anteriores, deve ainda estar em um local de fácil acesso para qualquer contribuidor que desejar visualizar.

Atividade:	**9 Criar Políticas de Incentivo**
Descrição:	Definir, planejar e executar estratégias que envolvam a motivação e reconhecimento dos desenvolvedores e de suas contribuições ajudam a manter um ecossistema produtivo e a sua arquitetura estável. Essa atividade define políticas para incentivar a participação e engajamento de desenvolvedores dentro de ECOS.
Critérios de Entrada:	Deve-se utilizar os guias de marketing e os direcionamentos do ECOS presentes no documento de diretrizes.
Critérios de Saída:	Políticas detalhadas (público, metas e formas de reconhecimento) e validadas.
Responsável:	*Organização Central*.
Participantes:	Equipe de Evangelistas, Equipe Jurídica da Organização Central e Equipe de Marketing de Desenvolvedores.
Artefatos Requeridos:	Diretrizes do ECOS.
Artefatos Produzidos:	Pacote de Políticas de Incentivo.
Recomendação:	As políticas devem: •Reconhecer a contribuição; •Reconhecer o desenvolvedor;

	•Facilitar o engajamento do desenvolvedor; •Definir regras e envolvidos, inclusive, os responsáveis pelo suporte ao desenvolvedor, como os evangelistas; •Validadas pelo jurídico da organização central a partir da definição de um regulamento.

Neste cenário, há três principais categorias de mecanismos de governança: 1) Criação de valor – gerar e distribuir valor; 2) Coordenação – manter a consistência e integração de atividades, relacionamentos e estruturas do ecossistema; e 3) Controle e abertura organizacional – capturar a atenção entre modelos abertos e fechados.

Baars e Jansen (2012) afirmam que a governança pode ajudar uma empresa a atingir seus objetivos, fazer melhor uso dos recursos disponíveis e direcionar a um aumento na renda e na redução de riscos. A governança de ecossistema requer um equilíbrio de controle pela organização central e de autonomia entre os desenvolvedores que são externos à organização [Tiwana *et al.* 2010]. Entretanto, uma vez que este é um campo relativamente novo, muitas organizações podem não saber como gerenciar efetivamente seus ecossistemas, ou como iniciar um ecossistema. Não existe uma formalização adequada para a governança do ecossistema e há muitos desafios a serem superados pelas organizações [Jansen e Finkelstein 2009], por exemplo, a atração e o envolvimento dos desenvolvedores. Existe também a necessidade de: um vocabulário comum na governança dos ECOS, orientação prática e o entendimento da governança de desenvolvedores.

Jansen *et al.* (2012) ainda indicam que é necessário buscar a clareza da governança dentro do ecossistema. Para isso, os autores apontam partes da governança que ajudam a estabelecer esta clareza, que são:

- **Clareza do ecossistema:** há questões essenciais para o sucesso de um ecossistema, se um ecossistema não possuir uma estrutura que seja clara para os elementos que o compõe não é possível ter uma estratégia de governança clara;
- **Clareza da estratégia de governança:** as organizações podem estabelecer regras, procedimentos, protocolos e processos formalizados, o que pode ajudar no controle do ecossistema. Isto ajuda a estabelecer uma estratégia de governança clara;
- **Responsabilidade:** os elementos que compõem o ecossistema são responsáveis por ele. Se não houver execução correta das responsabilidades de cada elemento, uma estratégia de governança não poderá ser garantida;
- **Medição:** acontece com o objetivo de determinar o benefício para o ecossistema a partir da utilização de uma estratégia de governança, utilizando-se de indicadores para analisar o estado atual do ecossistema e prospectar o estado futuro;
- **Compartilhamento de conhecimento:** é um aspecto importante do núcleo de negócios de um ecossistema como parte de uma estratégia de governança.

3.4. Governança de Desenvolvedores em ECOS

A governança de desenvolvedores, no contexto de ECOS, pode ser definida como um conjunto de mecanismos (e.g., abordagens, estratégias, práticas, algoritmos, ferramentas de análise) para apoiar relações sinérgicas (“ganha-ganha”) entre uma comunidade próspera de desenvolvedores e uma organização central com o objetivo de garantir e monitorar o bem-estar social e econômico dos desenvolvedores. Neste cenário, as comunidades de desenvolvedores emergem para apoiar a sustentabilidade do ecossistema, ou seja, a capacidade de um ecossistema aumentar ou manter sua comunidade de desenvolvedores durante o tempo e sobreviver a mudanças.

A perspectiva da análise das expectativas e experiências do desenvolvedor durante a entrada, engajamento e abandono do ecossistema pode ser estudada por meio de métodos e indicadores que capturem a Experiência do Desenvolvedor (DX, do inglês *Developer eXperience*). A DX ajuda na análise do impacto no desenvolvimento de aplicações (popularmente conhecidas como *apps*) a partir das expectativas, emoções e percepções do desenvolvedor relacionadas a todos os tipos de artefatos que ele pode encontrar como parte do seu envolvimento no ecossistema [Fagerholm 2015] visando um ecossistema que mantenha a sustentabilidade e diversidade [Dhungana *et al.* 2010]. Neste cenário, a governança consiste na busca pelo equilíbrio entre os objetivos da organização central e as expectativas do desenvolvedor (Figura 3.5).

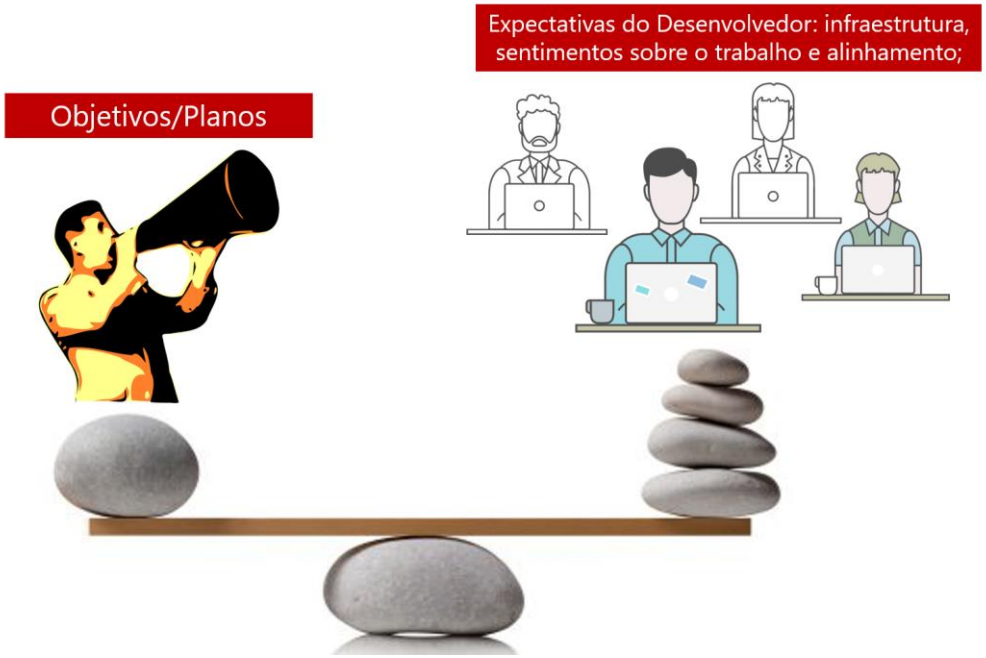


Figura 3.5. Equilíbrio entre objetivos da organização central e a experiência do desenvolvedor

Segundo Falgerholm e Münch (2012), a DX consiste nas experiências relativas a todos os tipos de artefatos e atividades que um desenvolvedor possa encontrar como parte de sua participação no desenvolvimento de software. Pode ser analisada a partir de três fontes:

1. **Infraestrutura de Desenvolvimento:** relacionada à forma como os desenvolvedores percebem a infraestrutura de desenvolvimento. Alguns tópicos

precisam ser estudados, tais como: ferramentas de gerenciamento, linguagens de programação, bibliotecas, plataformas, métodos, técnicas, habilidades e procedimentos;

2. **Percepções sobre o Trabalho:** analisa como os desenvolvedores se sentem sobre o seu trabalho, ou seja, sobre o que produzem. Nesta fonte, a análise se volta para o respeito, reconhecimento, interação social, time e afeto;
3. **Senso de Contribuição:** como os desenvolvedores veem o valor de suas contribuições. Para entender esta fonte de DX, analisa-se os seguintes fatores: planos, objetivos, intenções, alinhamento e engajamento.

Quando passamos a analisar DX a partir do desenvolvedor e o alinhamento com a organização central de um ECOS e suas respectivas funções, temos os pontos:

- Em relação à fonte *infraestrutura de desenvolvimento*, a organização central precisa investir esforços para prover aos desenvolvedores um *framework* com o objetivo de expandir as fronteiras do ECOS. Isso pode ser alcançado com diretrizes para atrair novos desenvolvedores e criar *apps*, provendo ferramentas e suporte ao desenvolvedor;
- Do ponto de vista da fonte *percepções sobre o trabalho*, o desenvolvedor precisa criar e entregar *apps* que alcancem os melhores nichos de usuários e que ganhem visibilidade com base em sua qualidade. Uma *app* poderá ser adquirida por usuários e contribuir para as metas da organização central. Neste contexto, número de *apps*, médias de avaliação das *apps* por usuários e número de desenvolvedores que publicam *apps* alimentam as métricas de ECOS.
- Sobre a fonte *senso de contribuição*, a organização central tem metas que podem ser atingidas e potencializadas por desenvolvedores externos: (i) o número de *apps*, (ii) a quantidade de renda gerada pela venda de *apps*, e (iii) o número de *apps* publicadas na loja. O alinhamento dos planos da organização central com os planos dos desenvolvedores pode ser utilizado para atender a demanda por *apps* da sociedade uma vez que a organização central, somente com sua estrutura interna, pode dificilmente responder a esta demanda [Fontão *et al.* 2016].

Em uma analogia com a ecologia, os indivíduos (desenvolvedores externos) possuem características que são únicas. Uma comunidade de desenvolvedores externos é formada por indivíduos com características diferentes [Begon *et al.* 2007]. Explicamos o comportamento de uma comunidade a partir do comportamento dos indivíduos que a compõem. Uma forma de analisar o comportamento e interação dos desenvolvedores em ECOS é minerando os dados disponíveis em repositórios de software. Isto permite "escutar" a voz do desenvolvedor.

3.5. Mineração de Repositórios

Não há comunicação direta entre grandes organizações que mantêm plataformas móveis (e.g., Apple, Google e Microsoft) e desenvolvedores terceirizados para resolver questões técnicas que surgem no design e desenvolvimento de contribuições de desenvolvedores em ECOS. Neste cenário, as organizações podem não saber como definir e evoluir

estratégias para governar seus desenvolvedores no sentido de atingir suas metas organizacionais. Essas organizações usam uma infraestrutura para dar suporte a desenvolvedores, por exemplo, um repositório de perguntas e respostas (Q&A, do inglês *Questions and Answers*). As interações entre os desenvolvedores nesses portais alimentam um repositório de Q&A que pode servir como um mecanismo para entender e definir estratégias para dar suporte aos desenvolvedores.

O desenvolvedor é um ator essencial para sustentar as contribuições do ECOS, como aplicativos e documentação técnica [Fontão *et al.* 2016]. Tais contribuições são normalmente armazenadas em um repositório oficial ECOS interno, como repositórios Android e Apple Developers. Além disso, existem links entre ECOS e repositórios externos, como código (e.g., GitHub) [Casalnuovo *et al.* 2015] e repositórios de Q&A. Como exemplo de um repositório de perguntas e respostas externo, o Stack Overflow (SO) tem um conjunto de perguntas/respostas técnicas que surgem do uso de APIs, SDKs e ferramentas de desenvolvimento [Lin e Serebrenik 2016].

Esses repositórios externos ajudam a manter a interação entre os desenvolvedores em uma plataforma comum, resultando em um conjunto de contribuições e influenciando direta ou indiretamente o ecossistema como um todo. [Santos e Werner 2012]. Portanto, um repositório de perguntas e respostas como o SO tem comunicações arquivadas entre os desenvolvedores de ecossistema e pode ser usado para investigar alguns aspectos do ECOS, por exemplo, engajamento do desenvolvedor e fragmentos de código.

A partir de uma dimensão técnica, uma grande quantidade de dados está disponível em repositórios de software. Esses dados são estáveis e não influenciados por pesquisadores [Shull *et al.* 2008]. Neste cenário, um método usado para conduzir estudos experimentais no campo de ecossistemas é a mineração de repositórios de software [Farias *et al.* 2016]. Este método pode ajudar na definição e evolução de estratégias para governar desenvolvedores em ECOS.

Repositórios de software podem ser fontes valiosas de informação uma vez que eles contêm (ou podem permitir a extração) de informação sobre aspectos sociais, técnicos e de negócios de um projeto, como acontece nas fontes da comunicação entre desenvolvedores [Genc-Nayebi e Abran 2016]. A área de Mineração de Repositórios de Software (MRS, do inglês *Mining Software Repositories*) foca em descobrir informação útil sobre o software por meio da extração e análise de dados de diferentes repositórios de software [Hassan 2008]. Abordagens de MRS têm sido usadas para diferentes objetivos, por exemplo, análises de contribuições e comportamento do desenvolvedor. Neste cenário, podemos dar como exemplos de tipos de repositórios software: Q&A, Código e Projeto, e Desafios.

Repositórios de Q&A são plataformas colaborativas web que têm como objetivo apoiar o compartilhamento de conhecimento ao permitir que usuários postem e respondam questões. Esses repositórios não são apenas uma plataforma para especialistas compartilharem o conhecimento e serem identificados como membros do ecossistema, mas também para ajudarem desenvolvedores iniciantes na resolução de seus problemas eficientemente [Bhat 2014]. O SO é um exemplo deste tipo de repositório [Shah *et al.* 2014].

SO é um portal de Q&A direcionado para a comunidade de desenvolvedores que postam questões e respostas (aproximadamente 14 milhões de questões e 19 milhões de respostas) relacionadas à programação de computadores. Tanto as questões quanto as respostas podem receber votos dos usuários (contra/a favor de). Tais votos se tornam pontos de reputação que possibilitam aos desenvolvedores conseguirem alguns privilégios como a liberação de restrições na criação de uma publicação e edição de perguntas e respostas de outros usuários. Outro mecanismo de privilégio envolve a atribuição de distintivos, ou seja, reconhecimento de desenvolvedores ao usarem o SO. Os desenvolvedores podem obter distintivos de várias atividades, por exemplo, um desenvolvedor pode receber um distintivo se ele fez uma pergunta que atingiu mais de mil visitas.

Zagalsky *et al.* (2016) identificaram que os desenvolvedores usam o SO por vários motivos: (a) a capacidade de obter reconhecimento de pares; (b) interface rica e amigável; (c) as respostas são diretas ao ponto; (d) as perguntas geralmente são respondidas mais rapidamente do que outros fóruns; e (e) é fácil procurar por perguntas e respostas anteriores. Neste trabalho, como os estudos anteriores baseados na mineração do SO, usamos uma abordagem (Figura 3.6) não supervisionada para extrair tópicos de perguntas de SO. Nossa metodologia é composta por quatro etapas descritas a seguir.

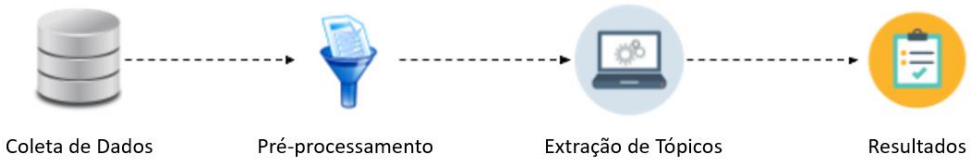


Figura 3.6. Etapas para Mineração e Tratamento dos Dados

Coleta de Dados: para este trabalho, utilizamos a base de dados fornecida pelo SO que está disponível publicamente em formato XML licenciado sobre a licença CC BY-as, que é uma licença pública que permite a distribuição gratuita de uma obra protegida por direitos autorais. Para os propósitos deste trabalho, nós utilizamos o arquivo posts.xml que contêm: o texto das postagens, quantidade de visualizações e respostas, *tags*, contagem de favoritos e data de criação, por exemplo.

Pré-processamento: pré-processamos o conteúdo textual (*Body*) dos *posts* extraídos em três etapas. Primeiro, descartamos qualquer fragmento de código presente nos *posts* (ou seja, entre *tags* HTML <code>), porque a sintaxe do código fonte (por exemplo, instruções “if” e *loops* “for”) introduz ruído na fase de análise. Em seguida, removemos todas as *tags* HTML (por exemplo, <p> e <a href="...">), pois esse não é o foco de nossa análise. Terceiro, removemos as palavras comuns em inglês, como "a", “the” e “is”, que não ajudam a criar tópicos significativos. Usamos o Spark como uma estrutura que suporta a análise de *big data*. O Spark[1] é um *framework* que ajuda no processamento de grande volume de dados com uma variedade de conjuntos de dados diversos (e.g., texto, grafos etc.) e de diferentes origens (*batch* ou *streaming* de dados

[1] http://spark.apache.org/

em tempo real) e foi desenvolvido em 2009 pelo AMPLab da Universidade da Califórnia com foco em velocidade e análises sofisticadas.

O procedimento de mineração de dados foi automatizado desde a construção do conjunto de dados até a análise do tópico. O pré-processamento é o passo responsável pela eliminação de termos não representativos (e.g., palavras irrelevantes, URLs, *emoticons* e *hashtags*) na coleta e o processo de extração de recursos. Neste contexto, usamos a ferramenta NTLTK para eliminar termos não representativos e, para fazer o processo de extração de características, usamos a abordagem *bag-of-words* com o TF-IDF (do inglês, *Term Frequency-Inverse Document Frequency*), onde eliminamos termos com frequência menor que cinco.

A abordagem *bag-of-words* consiste na transformação dos dados que não estão estruturados em um formato estruturado, especificamente uma tabela atributo-valor. O TF-IDF [Forman 2008] consiste em uma medida estatística para indicar a importância de uma palavra dentro de um documento em relação ao corpo do documento. O valor de TF-IDF aumenta à medida que cresce o número de ocorrências de uma palavra dentro do documento.

Extração de Tópicos: utilizamos *Latent Dirichlet allocation* (LDA) [Krestel *et al.* 2009], um modelo de tópico estatístico usado para recuperar automaticamente tópicos em vários domínios de um corpus de documentos de texto. Escolhemos o LDA porque ele é capaz de modelar tópicos em grandes volumes; no nosso caso, o corpo de questões dos desenvolvedores relacionadas aos ecossistemas.

Para a extração das informações do SO (Figura 3.7), podemos usar dados como: o título, corpo, pontuação da questão, número de visualizações, quantidade de respostas e *tags* utilizadas para categorizar a questão.

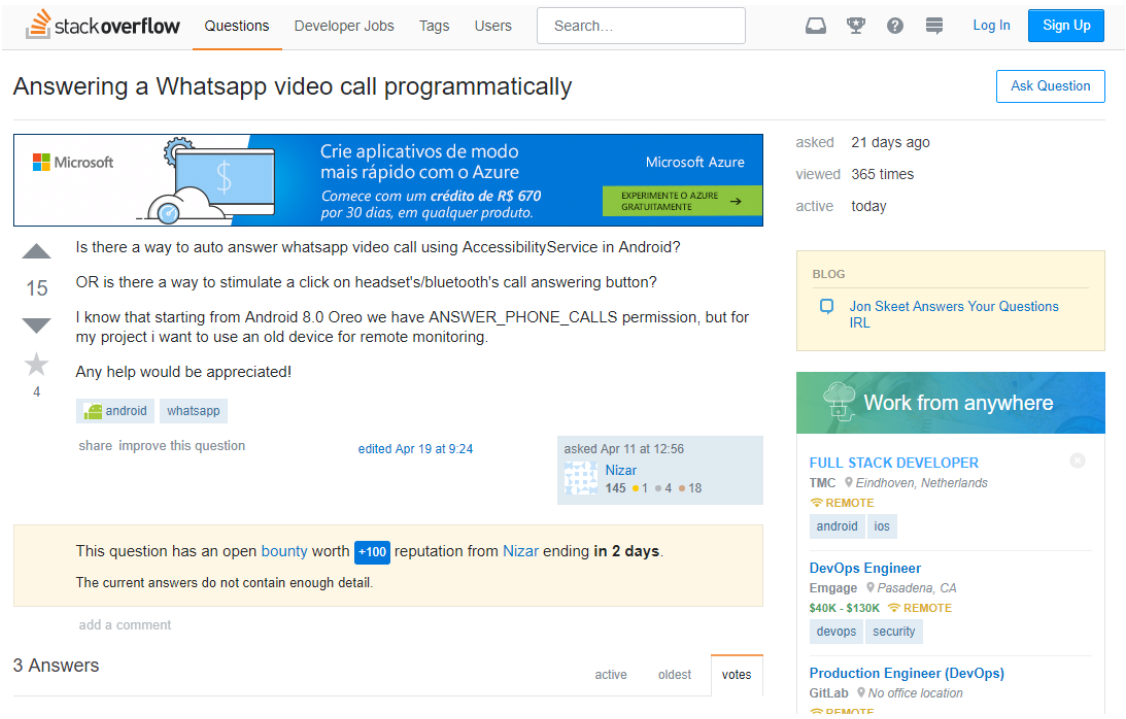


Figura 3.7. Estrutura de uma pergunta no Stack Overflow

Além disso, com o objetivo de identificar o número apropriado de tópicos (e não uma escolha aleatória), aplicamos uma técnica de particionamento chamada *Silhouette*. Esta técnica retorna um número apropriado (silhueta) de agrupamentos em relação às características dos documentos que estão sendo analisados [Rousseeuw 1987]. Esse número, fornecido pela silhueta, foi aplicado como entrada para o LDA.

Cada cluster é representado por uma silhueta chamada, que é baseada na comparação de sua rigidez e separação. Essa silhueta mostra quais objetos estão bem dentro de seu cluster e quais estão meramente em algum lugar entre esses clusters. A largura média da silhueta fornece uma avaliação da validade do agrupamento. Pode ser usado para selecionar um número apropriado de clusters. A *Silhouette* fornece valores no intervalo de -1 a 1, em que 1 significa que as amostras pertencentes ao cluster estão distantes dos outros clusters, 0 significa que a divisão entre os clusters já está na borda da separação e -1 significa que algumas amostras têm uma chance de serem atribuídas ao cluster errado. Os resultados são analisados pelos pesquisadores. Por exemplo, os tópicos mais frequentes, as questões ou respostas que são relacionadas a estes tópicos e também as séries temporais que podem indicar o comportamento da comunidade a partir da interação dentro do repositório.

Nesse cenário, podemos perceber que a área de MSR se concentra na descoberta de informações úteis sobre software, extraindo e analisando dados de diferentes repositórios de software [Ahasanuzzaman *et al.* 2016]. As abordagens de MSR foram usadas para diferentes objetivos como, por exemplo, análises de contribuição e comportamento do desenvolvedor.

O comportamento do desenvolvedor pode ser analisado a partir de emoções. Ele pode fornecer uma perspectiva diferente para interpretar a satisfação do desenvolvedor sobre o seu envolvimento no processo de desenvolvimento [Murgia *et al.* 2014]. Emoção é um “estado psicológico que surge espontaneamente (não por meio de esforço consciente) e é por vezes acompanhado de alterações fisiológicas” [Novielli *et al.* 2014]. Tipos gerais de emoções são: alegria, tristeza, raiva, desgosto e medo. A alegria está normalmente associada a conquistas positivas. A raiva geralmente está relacionada a um forte sentimento de desprazer e beligerância despertado por um erro. A tristeza é geralmente expressa por infelicidade. Repulsa é uma repugnância causada por algo ofensivo. Finalmente, o medo é uma emoção angustiante despertada pelo perigo iminente, mal, dor etc.

3.6. *Big Data* e Ferramentas

No cenário discutido na seção anterior, verificamos que existe um grande volume de informações geradas a partir das interações dos desenvolvedores dentro de um ecossistema. Por exemplo, somente no SO já são mais de 14 milhões de questões que podem ser mineradas para fornecer *insights* para as organizações que mantêm ecossistemas. Esse grande volume de dados tem sido estudado dentro da área de *Big Data*. Para processar esses dados, há diversos *frameworks*; neste trabalho, vamos utilizar o Spark.

O Spark estende o MapReduce, que é um modelo de programação desenhado para processar grandes volumes de dados em paralelo, dividindo o trabalho em um conjunto de tarefas independentes e evitando mover os dados durante o seu

processamento. Isso é feito por meio de recursos como armazenamento de dados em memória e processamento próximo ao tempo real. Neste contexto, o desempenho pode ser várias vezes mais rápido do que outras tecnologias de Big Data. A seguir, apresentamos o passo-a-passo do funcionamento do MapReduce (Figura 3.8):

1. Divide os dados de entrada em M pedaços. Em seguida, inicia várias cópias do programa em um cluster de computadores. Há uma cópia principal: *master*. As outras cópias são denominadas *workers* e recebem trabalho do *master*. Existem M tarefas de *map* e R tarefas de *reduce* para serem assinaladas. O *master* seleciona *workers* ociosos e assinala a eles uma tarefa de *map* ou de *reduce*;
2. Há um *worker* dedicado à tarefa de mapear o conteúdo correspondente ao pedaço da entrada. Ele interpreta as tuplas de chave/valor a partir dos dados de entrada e passa como parâmetro para a função de *map* do usuário. As tuplas chave/valor intermediárias produzidas pela função de *map* são armazenadas em memória;
3. Frequentemente, as tuplas de dados dos *buffers* são escritas em disco, particionados em R regiões pela função de particionamento. A localização dessas tuplas de dados no disco é informada ao *master*, que irá repassar essa localização para os *workers* com tarefas de *reduce*;
4. A partir do momento que um *worker* de *reduce* é notificado da localização dos dados pelo *master*, este faz uma chamada de procedimento remota para buscar os dados do disco local dos *workers* de *map*. Quando os dados já foram lidos, ele ordena os dados pelas chaves intermediárias, para que todas as ocorrências de uma mesma chave sejam agrupadas em conjunto. É realizada uma ordenação, pois muitas chaves diferentes são mapeadas para a mesma tarefa de *reduce*. Se a quantidade de dados intermediários é muito grande para caber em memória, uma ordenação externa em disco é executada;
5. O *worker* de *reduce* vasculha os dados intermediários já ordenados e, para cada chave encontrada, ele passa como conteúdo para a chave os valores intermediários para a função de *reduce* definida pelo usuário. A saída de cada função de *reduce* é adicionada ao final de um arquivo de saída para aquela partição de *reduce*;
6. Quando todas as tarefas de *map* e *reduce* foram terminadas, o *master* retorna o programa do usuário;
7. Ao final da execução do programa, o resultado está disponível em R arquivos (um para cada operação de *reduce*). É comum que os arquivos de resultados de uma operação de MapReduce sejam entradas para outra chamada de MapReduce.

O Spark armazena resultados intermediários na memória, em vez de escrevê-los no disco, o que é muito útil quando se precisa processar o mesmo conjunto de dados mais de uma vez. Logo, ele é um mecanismo de execução que funciona tanto na memória como em disco. Assim, é possível usá-lo para o processamento de conjuntos de dados maiores que a memória agregada em um cluster. O Spark armazenará a maior quantidade possível de dados na memória e, em seguida, irá persisti-los em disco. Com

esse mecanismo de armazenamento de dados em memória, o uso do Spark traz vantagens de desempenho. Outras características são:

- Suporta mais do que apenas as funções de *map* e *reduce*;
- Otimiza o uso de operadores de grafos arbitrários;
- Avaliação sob demanda de consultas de *Big Data* contribui com a otimização do fluxo global do processamento de dados;
- Fornece APIs concisas e consistentes em Scala, Java e Python;
- Oferece *shell* interativo para Scala e Python.

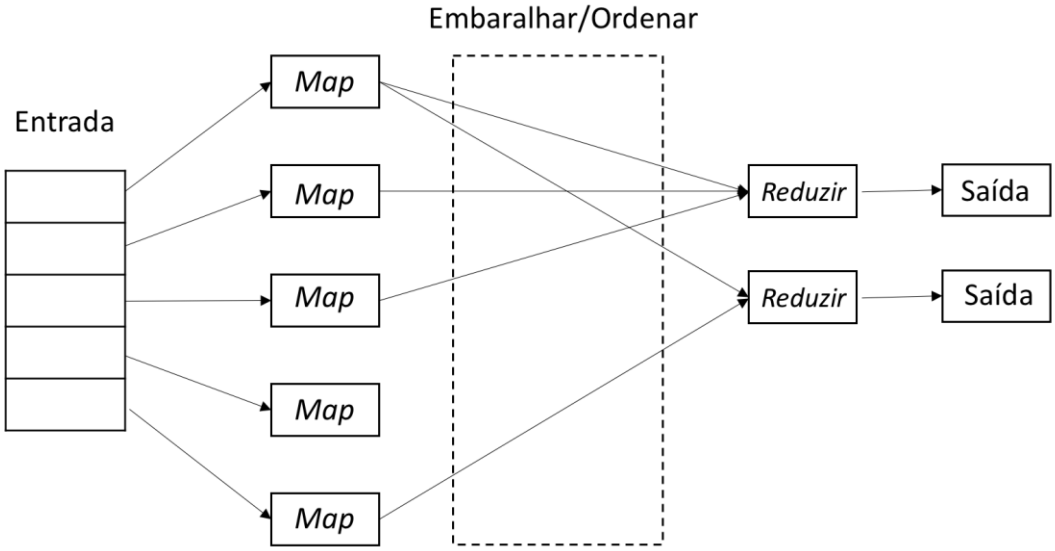


Figura 3.8. Passo-a-passo da execução do MapReduce

O Spark é escrito na linguagem Scala e executa em uma máquina virtual Java. Atualmente, suporta as seguintes linguagens para o desenvolvimento de aplicativos: Scala, Java, Python, Clojure e R.

Outro *framework* que envolve documentação e códigos para mineração de dados é o ScikitLearn[2]. O ScikitLearn é baseado na linguagem de programação Python e é um *framework open-source*. Este *framework* aborda as categorias de: classificação (identificação de que categoria um objeto pertence), regressão (prevê um valor contínuo de um atributo associado um objeto), agrupamento (agrupamento automático de objetos similares em conjuntos), redução de dimensionalidade (redução do número de variáveis randômicas a serem consideradas), seleção de modelos (comparação, validação e escolha de parâmetros e modelos) e pré-processamento (extração de características e normalização). Neste trabalho, abordamos a parte de pré-processamento, redução de dimensionalidade e agrupamentos.

[2] http://scikit-learn.org/stable/

3.7. Considerações Finais

Para o estudo da governança de desenvolvedores em ECOS, é necessário buscar o equilíbrio entre os objetivos da organização que mantém o ECOS e as expectativas dos desenvolvedores. Desta forma, a governança de desenvolvedores considera a disponibilização de processos e de um ambiente que favoreçam o bem-estar econômico e social dos desenvolvedores. Nesse cenário, como forma de obter informações sobre o engajamento e interações dos desenvolvedores, é necessária a utilização de técnicas de mineração de repositórios de software. Repositórios de software guardam informações sobre, por exemplo, Questões e Perguntas técnicas (e.g., Stack Overflow) e projetos de software (e.g., Github). Neste trabalho, compartilhamos conceitos de ECOS, governança e mineração de repositórios de software. Ainda sobre mineração de repositórios de software compartilhamos uma abordagem para o pré-processamento, extração de tópicos e análise de resultados.

Agradecimentos

O autor Rodrigo Pereira dos Santos agradece ao DPq/PROPGPI/UNIRIO pelo apoio para realização desse trabalho.

Referências

A.E., H. (2008). The road ahead for mining software repositories. In Proceedings of the Frontiers of Software Maintenance, FoSM, p. 48-57.

Ahasanuzzaman, M., Asaduzzaman, M., Roy, C. K. and Schneider, K. A. (2016). Mining duplicate questions in stack overflow. Proceedings of the 13th International Workshop on Mining Software Repositories - MSR ’16, p. 402–412.

Alves, C., Oliveira, J. and Jansen, S. (2017). Software Ecosystems Governance - A Systematic Literature Review and Research Agenda. Proceedings of the 19th International Conference on Enterprise Information Systems, n. January, p. 215–226.

Baars, A. and Jansen, S. (2012). A Framework for Software Ecosystem Governance. Proceedings of the International Conference of Software Business. p. 168–180.

Begon, M., Townsend, C. R. and Harper, J. L. (2007). Ecologia - De Indivíduos a Ecossistemas. 4. ed. Porto Alegre: Artmed.

Bhat, V. (2014). Min (e) d Your Tags : Analysis of Question Response Time in StackOverflow. Proceedings of the , Advances in Social Networks Analysis and Mining (ASONAM). p. 328–335.

Bosch, J. (2009). From Software Product Lines to Software Ecosystems. In SPLC ’09 Proceedings of the 13th International Software Product Line Conference. p. 111-119.

Casalnuovo, C., Vasilescu, B., Devanbu, P. and Filkov, V. (2015). Developer Onboarding in GitHub: The Role of Prior Social Links and Language Experience. ESEC/FSE conf., p. 1–12.

De Fontão, A. L., Dos Santos, R. P., Filho, J. F. and Dias-Neto, A. C. (2016). MSECO-DEV: Application development process in mobile software ecosystems. In Proceedings of the International Conference on Software Engineering and Knowledge Engineering. P. 317-322.

Dhungana, D., Groher, I., Schludermann, E. and Biffl, S. (2010). Software Ecosystems vs . Natural Ecosystems : Learning from the Ingenious Mind of Nature. Proceedings of the 4th European Conference on Software Architecture Companion Volume (ECSA '10), p. 96–102.

Dubinsky, Y. and Kruchten, P. (2009). Software development governance (SDG): report on 2nd workshop. ACM SIGSOFT Software Engineering Notes, v. 34, n. 5, p. 46–47.

Fagerholm, F. (2015). Software Developer Experience : Case Studies in Lean-Agile and Open Source Environments.

Fagerholm, F. and Münch, J. (2012). Developer experience: Concept and definition. In 2012 International Conference on Software and System Process, ICSSP 2012 - Proceedings.

Farias, M. A. de F., Novais, R., Colaço, M., *et al.* (2016). A Systematic Mapping Study on Mining Software Repositories. In 31st ACM/SIGAPP Symposium on Applied Computing.

Fontão, A., Bonifácio, B., Dias-Neto, A., Bezerra, A. and Santos, R. (2014). MSECO skill: Construction skills developer ecosystem in mobile software | MSECO skill: Construção de competências de desenvolvedores em ecossistemas de software móvel. In CIBSE 2014: Proceedings of the 17th Ibero-American Conference Software Engineering.

Forman, G. (2008). BNS feature scaling: an improved representation over tf-idf for svm text classification. Proceedings of the 17th ACM conference on Information and knowledge management, p. 263–270.

Genc-Nayebi, N. and Abran, A. (2016). A Systematic Literature Review: Opinion Mining Studies from Mobile App Store User Reviews. Journal of Systems and Software,

Hyrynsalmi, S., Seppänen, M. and Suominen, A. (2014). Sources of value in application ecosystems. Journal of Systems and Software, v. 96, p. 61–72.

Iansiti, M. and Levien, R. (2004). Strategy as ecology. Harvard business review, v. 82, p. 68–78, 126.

Iansiti, M. and Richards, G. L. (2006). The information technology ecosystem: Structure, health, and performance. Antitrust Bulletin, v. 51, p. 77–110.

Jansen, S. and Bloemendal, E. (2013). Defining App Stores: The Role of Curated Marketplaces in Software Ecosystems. Software Business. From Physical Products to Software Services and Solutions. p. 195–206.

Jansen, S., Brinkkemper, S. and Cusumano, M. (2013). Software Ecosystems.

Jansen, S., Brinkkemper, S. and Finkelstein, A. (2009). Business Network Management as a Survival Strategy : A Tale of Two Software Ecosystems. n. 2, p. 34–48.

Jansen, S., Brinkkemper, S., Souer, J. and Luinenburg, L. (2012). Shades of gray: Opening up a software producing organization with the open software enterprise model. Journal of Systems and Software, v. 85, n. 7, p. 1495–1510.

Jansen, S. and Cusumano, M. A. (2012). Defining software ecosystems : A survey of software platforms and business network governance. In Proceedings of the international Workshop on Software Ecosystems 2012.

Jansen, S., Finkelstein, A. and Brinkkemper, S. (2009). A sense of community: A research agenda for software ecosystems. 2009 31st International Conference on Software Engineering - Companion Volume, p. 187–190.

Kim, H., Kim, J., Lee, Y., Chae, M. and Choi, Y. (2002). An Empirical Study of the Use Contexts and Usability Problems in Mobile Internet. In HICSS "02 Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS"02). . IEEE Computer Society Washington.

Krestel, R., Fankhauser, P. and Nejdl, W. (2009). Latent dirichlet allocation for tag recommendation. In Proceedings of the third ACM conference on Recommender systems. . http://portal.acm.org/citation.cfm?doid=1639714.1639726.

Lin, B. and Serebrenik, A. (2016). Recognizing Gender of Stack Overflow Users. MSR conf, p. to appear.

Manikas, K. (2016). Revisiting software ecosystems Research: A longitudinal literature study. Journal of Systems and Software, v. 117, p. 84–103.

Manikas, K. and Hansen, K. M. (2013). Reviewing the Health of Software Ecosystems – A Conceptual Framework Proposal. In Proceedings of the 5th International Workshop on Software Ecosystems.

Miranda, M., Ferreira, R., De Souza, C. R. B., Figueira Filho, F. and Singer, L. (2014). An exploratory study of the adoption of mobile development platforms by software engineers. Proceedings of the 1st International Conference on Mobile Software Engineering and Systems - MOBILESoft 2014, p. 50–53.

Murgia, A., Tourani, P., Adams, B. and Ortu, M. (2014). Do developers feel emotions? an exploratory analysis of emotions in software artifacts. Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014, p. 262–271.

Mustofa, K., Biffl, S., Schatten, A., Tjoa, A. M. and Wahyudin, D. (2007). Monitoring the "health" status of open source web-engineering projects. International Journal of Web Information Systems

Novielli, N., Calefato, F. and Lanubile, F. (2014). Towards discovering the role of emotions in stack overflow. Proceedings of the 6th International Workshop on Social Software Engineering - SSE 2014, p. 33–36.

Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis.

Santos, R. P. and Werner, C. (2011). Treating business dimension in Software Ecosystems. In Proceedings of the International Conference on Management of Emergent Digital EcoSystems, MEDES'11.

Santos, R., Werner, C., Barbosa, O. and Alves, C. (2012). Software ecosystems: Trends and impacts on software engineering. Proceedings - 2012 Brazilian Symposium on Software Engineering, SBES 2012, p. 206–210.

Santos, R. P. Dos and Werner, C. M. L. (aug 2012). ReuseECOS: An Approach to Support Global Software Development through Software Ecosystems. In 2012 IEEE Seventh International Conference on Global Software Engineering Workshops. . IEEE.

Schaeffer, D. J., Herricks, E. E. and Kerster, H. W. (1988). Ecosystem health: I. Measuring ecosystem health. Environmental Management

Shah, C., Kitzie, V. and Choi, E. (2014). Questioning the question - Addressing the answerability of questions in community question-answering. In Proceedings of the Annual Hawaii International Conference on System Sciences.

Shull, F., Singer, J. and Sjøberg, D. I. K. (2008). Guide to advanced empirical software engineering.

Taylor, R. N. (2013). The role of architectural styles in successful software ecosystems. In ACM International Conference Proceeding Series.

Tiwana, A., Konsynski, B. and Bush, A. A. (dec 2010). Platform Evolution: Coevolution of Platform Architecture, Governance, and Environmental Dynamics. Information Systems Research, v. 21, n. 4, p. 675–687.

Wazlawick, R. S. (2013). Engenharia de softtware: conceitos e práticas. 1. ed. Rio de Janeiro: Elsevier Ltd.

Zagalsky, A., Teshima, C. G., German, D. M., Storey, M. and Poo-caamaño, G. (2016). How the R Community Creates and Curates Knowledge : A Comparative Study of Stack Overflow and Mailing Lists. p. 441–451.

Biografia dos Autores

Awdren Fontão é doutorando do Programa de Pós-Graduação em Informática do Instituto de Computação (ICOMP) na Universidade Federal do Amazonas (UFAM). Suas áreas de pesquisa envolvem: Ecossistemas de Software, Aplicações Móveis, Relações com Desenvolvedores. Tem experiência com comunidades de desenvolvedores, onde atuou por seis anos na área de *Developer Relations* como evangelista de desenvolvedores da Nokia e Microsoft (Brasil/América Latina).

Igor Scaliante Wiese é professor do Departamento de Computação na Universidade Federal Tecnológica do Paraná (UTFPR) – Campo Mourão, onde ele pesquisa técnicas de mineração de repositórios, aspectos humanos na engenharia de software e tópicos relacionados. Wiese fez doutorado em Ciência da Computação na Universidade de São Paulo e foi pesquisador visitante na University of California - Irvine.

Rodrigo Pereira dos Santos é professor do Departamento de Informática Aplicada e membro efetivo do Programa de Pós-Graduação em Informática da Universidade Federal do Estado do Rio de Janeiro (UNIRIO). Atuou como pesquisador visitante na University College London (UCL, 2014-2015) e como consultor em projetos de P&D em engenharia de sistemas na indústria nacional pela Fundação Coppetec (2008-2017). É editor-chefe da iSys: Revista Brasileira de Sistemas de Informação. É membro do Comitê Gestor da Comissão Especial de Sistemas de Informação da SBC. Seus temas de pesquisa envolvem Ecossistemas de Software e Engenharia de Requisitos.

Arilo Claudio Dias Neto é professor do Instituto de Computação (ICOMP) na Universidade Federal do Amazonas (UFAM). Ele lidera o grupo de pesquisa em Experimentação e Teste de Software e participa de pesquisa na indústria e projetos de desenvolvimento. Revisor de periódicos, como: Information and Software Technology, Journal of Information Processing Systems, Journal of The Brazilian Computer Society (Online) e Software Quality Journal. CTO da Méliuz.

Capítulo 4

Sistemas de Gerenciamento de Banco de Dados em Memória

Arlino Magalhães, José Maria Monteiro e Ângelo Brayner

Abstract

The in-memory databases store their data directly in physical memory. This feature makes these databases provide a response time of orders of magnitude less than traditional disk-based systems. However, memory systems are more sensitive to failures, since main memory is volatile. Rather than attempting to optimize disk access, as in traditional databases, in-memory databases focus on optimizing CPU cycles and multicore processing. Differences between in-memory and disk databases affect almost every aspect of the database management, such us: data storage, indexing, concurrency control, durability and recovery techniques, and query processing and compilation. This work will discuss the main concepts and techniques that differentiate in-memory and disk resident databases. In addition, some in-memory database management systems will be introduced.

Resumo

Os bancos de dados em memória armazenam seus dados diretamente na memória física. Essa característica faz esses bancos possuírem um tempo de resposta de ordens de magnitude muito menor do que os sistemas tradicionais baseados em disco. Entretanto, os sistemas em memória são mais sensíveis a falhas, visto que, a memória principal é volátil. Ao invés de tentar otimizar acesso ao disco, como nos bancos de dados tradicionais, os bancos de dados em memória focam em otimizar ciclos de CPU e processamento multicore. As diferenças entre os bancos de dados em memória e os em disco afetam quase todos os aspectos de gerenciamento desses bancos, como: armazenamento, indexação, controle de concorrência, durabilidade e técnicas de recuperação, e processamento e compilação de consultas. Nesse trabalho serão discutidas os principais conceitos e técnicas que diferenciam os bancos residentes em memória dos bancos residentes em disco. Além disso, serão apresentados alguns sistemas de gerenciamento de bancos de dados em memória.

4.1. Introdução e motivação

Bancos de Dados em Memória (*Main Memory Databases - MMDBs* ou *In-Memory Databases - IMDBs*) são Sistemas de Gerenciamento de Bancos de Dados - SGBDs onde os dados estão armazenados diretamente na memória RAM em vez de permanecer em discos rígidos, como ocorre nos bancos de dados convencionais. Essa abordagem faz os bancos de dados em memória possuírem um tempo de resposta muito mais rápido do que os bancos de dados em disco. Apesar dos bancos de dados em memória serem apontados como uma tendência recente para solução de problemas, o desenvolvimento e pesquisa de bancos em memória começaram no início da década de 80. Entretanto, apenas na década de 90, com os avanças tecnológicos em *hardware*, os bancos de dados em memória começaram a refletir as tendências e arquiteturas utilizadas nos bancos de dados em memória mais modernos [Garcia-Molina and Salem 1992, Faerber et al. 2017].

Os SGBDs, tradicionalmente, são baseados em disco, ou seja, os dados são armazenados em disco rígido. Esses sistemas precisam copiar seus dados para memória RAM para que esses dados possam ser processados pela CPU e, se necessário, atualizações nos dados devem ser copiadas de volta para o disco. Nos bancos de dados em memória, os dados residem permanentemente na memória. Essa característica tem importantes implicações em como os bancos devem ser estruturados e acessados. A Figura 4.1 ilustra de forma básica as arquiteturas dos bancos de dados baseados em disco e dos baseados em memória. Os dados nos bancos em memória são acessados diretamente da memória RAM, eliminando o gargalho de I/O de acesso a disco. Entretanto, devido a volatilidade da memória RAM, os bancos em memoria são mais sensíveis a falhas (como falta de energia, por exemplo). Por prover tolerância a falhas, devem ser feitos *backups* (*logs* e *checkpoints*) periódicos dos dados para o disco. Os I/Os ao disco podem levar a possíveis *overheads* no sistema, mas para evitar isso os bancos de dados em memória utilizam estratégias para otimizar o acesso ao disco [Faerber et al. 2017, Zhang et al. 2015]. Na Seção 4.3.4 será brevemente discutido como implementar durabilidade e tolerância a falhas em bancos de dados.

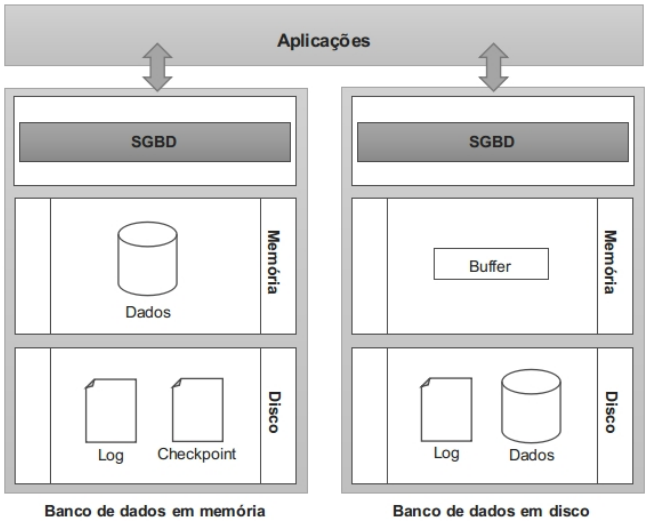


Figura 4.1. Arquitetura dos bancos de dados em memória *vesus* Arquitetura dos bacos de dados em discos.

A principal motivação para uso de bancos de dados em memória é sua baixa latência de acesso aos dados e possibilidades de análise em tempo real. Nas últimas décadas os *chips* de memória RAM tem dobrado em capacidade de armazenamento a cada 3 anos, enquanto que, seus preços tem caindo em um fator de 10 a cada 5 anos. Adicionalmente, já é comum as CPUs possuírem vários *cores* (até dezenas ou centenas). O processamento *multicore* tem sido largamente usado pelos bancos de dados em memória para ganhos de desempenho no processamento e acesso a dados. Como consequência, usar sistemas em memória principal para aplicações que tradicionalmente usam sistemas em disco tornou-se tecnicamente possível e economicamente viável [Zhang et al. 2015, Hazenberg and Hemminga 2011]. A memória principal claramente possui propriedades muito diferentes da memória secundária. Embora essas diferenças sejam bem conhecidas, segue um resumo delas [Garcia-Molina and Salem 1992]:

- O acesso à memória é de ordens de magnitude menor do que ao disco.
- A memória principal é volátil, enquanto que, o disco não é. Isso torna a memória mais vulnerável a fontes de *overhead* não existentes no disco, como durabilidade e tolerância a falhas, visto que, são necessários *backups* periódicos para o disco.
- Os discos são dispositivos de armazenamento orientados a blocos, equanto que, a memória não é. Essa característica permite uma representação mais flexível dos dados na memória e da forma mais adequada a melhorar o desempenho do sistema.
- O organização dos dados no disco é muito mais crítica do que na memória. Por exemplo, o acesso sequencial nos discos é mais rápido do que o acesso aleatório.

As diferenças entre as memórias principal e secundária se refletem nas diferenças entre os bancos de dados baseados em memória e os em disco. Os sistemas baseados em disco tentam otimizar o I/O, visto que, o acesso ao disco é um gargalho. Por esse motivo, esses sistemas utilizam a memória como uma *cache*. Além disso, caso a memória disponível não seja suficiente, técnicas de paginação são utilizadas para mover os dados em processamento na memória para o disco (e vice-versa), dando a impressão de que a memória utilizada é maior do que a disponível [Ramakrishnan and Gehrke 2002]. Os bancos de dados em memória acessam os dados diretamente da memória e os dados são organizados de forma a minimizar o *footprint* (quantidade de memória de trabalho necessária). Os sistemas em memória focam em otimizar o acesso à memória e ciclos de CPU [Hazenberg and Hemminga 2011]. A Tabela 4.1 sumariza as diferenças entre os bancos de dados em disco e os em memória.

Tabela 4.1. Diferenças entre os bancos de dados em disco e os em memória.

Banco de dados em disco	**Banco de dados em memória**
armazena dados em disco	armazena dados em memória
usa memória como *cache*	acessa a memória diretamente
assume que a memória é abundante	usa a memória disponível mais eficientemente
otimiza acesso ao disco	otimiza acesso à memória
mais ciclos de CPU	menos ciclos de CPU

As diferenças entre bancos em disco e os em memória afetam quase todos os aspectos de gerenciamento desse dois tipos de bancos de dados. Na Seção 4.3 serão descritos brevemente as principais detalhes relacionadas a *design* e escolhas arquitetônicas. Entretanto, antes disso, é importante salientar algumas questões que são comumente perguntadas sobre bancos de dados em memória:

1. **Um banco de dados inteiro pode ser armazenado na memória?**
   Sim! Alguns bancos de dados tem um tamanho limitado e/ou sua taxa de crescimento é muito menor do que a taxa de crescimento atual de armazenamento das memórias RAMs. Por exemplo: o tamanho de um banco de dados pode ser proporcional a quantidade de empregados ou clientes de uma empresa e essas informações podem não precisar de muitos *bytes* para serem armazenadas. Além disso, existem dados raramente acessados. De acordo com a regra 80-20, oitenta por cento dos acessos a bancos de dados são a apenas vinte porcento dos dados. Assim, os dados podem ser divididos de acordo com a frequência de acesso, como por exemplo em: frequentemente acessados (*hot*) e raramente acessados (*cold*). Apenas os dados *hot* precisam ser carregados na memória, enquanto que, os *cold* podem permanecer em disco até serem necessários [Garcia-Molina and Salem 1992, Gruenwald and Eich 1994]. Adicionalmente, caso realmente a base de dados seja muito maior do que a quantidade de memória disponível, a base pode ser distribuída entre as memórias dos vários nós de um banco de dados em memória distribuído [Kemper and Neumann 2011, Färber et al. 2012].

2. **Utilizar um banco de dados baseado em disco com memória RAM suficiente para armazenar a base de dados inteira é equivalente a utilizar um banco de dados baseado em memória?**
   Não! Possuir memória RAM suficiente para armazenar toda a base de um banco de dados baseado em disco pode trazer ganhos de desempenho, visto que, minimiza a quantidade de acessos ao disco. Entretanto, eventuais atualizações nos dados devem ser descarregadas para o disco, o que leva a I/Os. Além disso, mesmo com um *buffer* grande, as estruturas de dados ainda continuam por tentar otimizar o acesso ao disco. Nos bancos de dados em memória, os dados residem permanentemente na memória e as estratégias de acesso focam em otimizar a memória [Hazenberg and Hemminga 2011, Faerber et al. 2017]. A Seção 4.3.1 trata melhor da organização e *layout* dos dados.

3. **O bancos de dados em memória podem perder todos os seus dados após um desligamento de energia?**
   Não! Os bancos de dados em memória implementam estratégias de durabilidade e recuperação após falhas. Como a memória principal é volátil, as informações contidas no banco de dados devem ser movidas periodicamente para a memória secundária. I/Os podem implicar em *overheads* no sistema. Entretanto, os bancos de dados em memória utilizam estratégias para otimizar os *commit*, *log* e *checkpoint* com o objetivo de prover durabilidade e recuperação após falhas. O *commit* é a efetivação de atualizações de uma transação no bancos de dados. O *log* contêm informações de atualizações feitas no banco de dados. O *checkpoint*, em bancos de dados em memória, é considerado como o *backup* mais recentemente atualizado da

base de dados. *Commit*, *log* e *checkpoint* precisam ser descarregados para o disco e, por isso, devem ser realizados de maneira a não comprometer o funcionamento normal do sistema [Mohan and Levine 1992, DeWitt et al. 1984]. Durabilidade e tolerância a falhas serão melhor explicadas na Seção 4.3.4.

4. **Memória RAM não volátil pode ser utilizada em bancos de dados em memória no lugar da memória RAM convencional?**
   Sim! Memória RAM não volátil (NVRAM) é uma tecnologia que captura as principais vantagens da memória RAM (alto desempenho) e do disco rígido (persistência). Entretanto, o uso de NVRAM para armazenar grandes quantidades de dados ainda é economicamente proibitivo. Mas essa tecnologia promete ser promissora em um futuro próximo [Zhang et al. 2015]. Segundo Jim Gray *"Memory is the new disk, disk is the new tape"*, visionando que a memória principal iria substituir o disco rígido e este último seria utilizado apenas para *backup* [Robbins 2008]. Na Seção 4.2.4, a memória NVRAM será abordada com mais detalhes.

5. **Que tipo de aplicações tipicamente empregam bancos de dados em memória?**
   Bancos de dados em memória são utilizados comumente em aplicações que demanda altas velocidades de acesso, armazenamento e manipulação de dados. Alguns exemplos são sistemas de roteamento de IPs em redes, controle industrial, roteamento de ligações telefônicas, mercados financeiros, *e-comerce*, redes sociais e etc. Vários sistemas embarcados, como *MP3 players*, utilizam bancos de dados em memória. Recentemente, o fenômeno *Big Data* tem impulsionado pesquisas no desenvolvimento de sistemas com serviços de latência muito baixa (milissegundos) e análise de dados em tempo real. Os sistemas baseados em disco existentes não conseguem atingir esse tempo de reposta devido a alta latência dos discos rígidos. Para atingir estritamente requisitos de análise de grandes quantidades de dados em tempo real e atender requisições em milissegundos, os sistemas de bancos de dados em memória tem sido utilizados [Zhang et al. 2015].

4.2. Tecnologias *core* para bancos de dados em memória

Nessa seção serão introduzidos alguns conceitos e técnicas que são a base para desempenho de bancos de dados em memória, incluindo hierarquia da memória, acesso não uniforme a memória, memória transacional e memória RAM não volátil.

4.2.1. Hierarquia da memória

A hierarquia da memória é definida em termos de latência com os seguintes componentes: registrador, *cache* (L1, L2 e L3), memória principal e disco (disco rígido, memória *flash* e SSD). A Figura 4.2 ilustra a hierarquia da memória mostrando do componente mais rápido para o mais lento, seguindo a ordem de cima para baixo. Nas arquiteturas modernas, a CPU não pode processar dados que não estejam nos registradores. Assim, o dado deve ser transmitido através de cada uma das camadas de memória até atingir o registrador. Consequentemente, cada camada funciona como uma *cache* da camada adjacente inferior para reduzir a latência de acessos repetidos. O desempenho de sistemas de uso intensivo de dados depende muito da hierarquia da memória. Entretanto, esses sistemas também

precisam de otimizações para atingir localização espacial e temporal. Localização espacial assume que dados adjacentes são mais comumente acessados juntos. A localização temporal refere-se a observação de que um dado acessado será novamente acessado em um futuro próximo [Zhang et al. 2015].

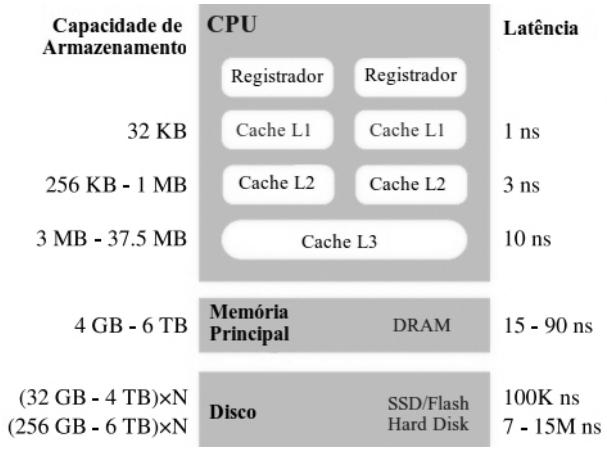


Figura 4.2. Hierarquia da memória
Fonte: [Zhang et al. 2015]

4.2.2. Acesso não uniforme a memória

O Acesso não Uniforme a Memória (*Non-Uniform Memory Access* - NUMA) é uma arquitetura para projeto de memória principal de computadores multiprocessados. Nessa arquitetura, o acesso à memória executado pelos processadores é não uniforme, ou seja, a latência de uma operação na memória depende da localização relativa do processador que está executando essa operação. Cada processador, em um sistema NUMA, possui uma memória local cuja latência é mínima, mas o processador também pode fazer acessos remotos a outras memórias cuja latência é maior [Li et al. 2013, Zhang et al. 2015]. A Figura 4.3 esboça uma arquitetura NUMA com N nós.

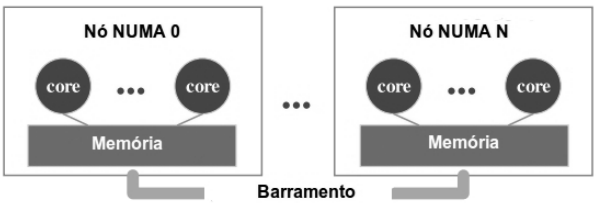


Figura 4.3. Tecnologia NUMA
Fonte: [Zhang et al. 2015]

A motivação para empregar a arquitetura NUMA é melhorar a largura de banda da memória e o tamanho total de memória que pode ser implementado em um nó do servidor. No contexto de bancos de dados, as pesquisas em NUMA podem ser divididas em (1) particionamento dos dados de maneira que o acesso remoto em NUMA seja minimizado [Maas et al. 2013, Leis et al. 2014a]; (2) gerenciamento dos efeitos de NUMA em cargas de trabalho sensíveis a latência, como cargas de transações OLTP

[Porobic et al. 2012, Porobic et al. 2014]; e (3) eficiência de transferência de dados através de domínios NUMA [Li et al. 2013].

4.2.3. Memória transacional

Memória Transacional (*Transactional Memory* - TM) é um mecanismo para controle de concorrência a dados compartilhados em sistemas concorrentes, análogo ao controle de concorrência em transações de banco de dados. Existem dois tipos de memória transacional: Memória Transacional em *Software* (*Software Transactional Memory* - STM) [Saha et al. 2006, Shavit and Touitou 1997], que implementa o sistema de execução transacional em *software*, utilizando bibliotecas e compiladores, por exemplo; e Memória Transacional em *Hardware* (*Hardware Transactional Memory* - HTM) [Knight 1986, Zilles and Rajwar 2007], que implementa o sistema de execução transacional por meio de suporte arquitetural, modificando *cache*, barramento ou protocolos de coerência, por exemplo. Uma terceira possibilidade é chamada de Memória Transacional Híbrida (*Hybrid Transactional Memory* - HyTM), que visa obter melhor desempenho utilizando características de STM e HTM [Lintzmayer et al.].

HTMs e STMs são responsáveis por: identificar acessos a memória dentro de transações, gerenciar os conjuntos de leitura e escrita da transação, detectar e resolver conflitos, gerenciar o estado dos registradores da arquitetura, efetivar e abortar transações. STM oferece as vantagens de (1) permitir a implementação de uma grande variedade de algoritmos sofisticados; (2) possuir modificação de *software* fácil e barata; e (3) poder ser integradas mais facilmente com sistemas existentes. Embora STMs possam ser bem flexíveis, elas causam um *overhead* significante ao sistema, inviabilizando sua aplicação prática. HTM tem atraído mais atenção por (1) oferecer quase nenhum *overhead* a execução das transações; (2) e por ser menos invasiva nos sistemas atuais. Como desvantagem, HTM não suporta transações grandes [Lintzmayer et al. , Cascaval et al. 2008]. Atualmente já existem implementações de HTM nos processadores modernos que estão sendo utilizados nos bancos de dados em memória, como o Haswell da Intel [Leis et al. 2014b].

4.2.4. Memória RAM não volátil

A Memória RAM não Volátil (*Non-Volatile Random Access Memory* - NVRAM) é uma tecnologia emergente com perspectivas de persistência com alta velocidade e grande capacidade de armazenamento. Existem algumas implementações de NVRAM; como *Phase-Change Memory* - PCM [Raoux et al. 2008], Memristors [Strukov et al. 2008] e *Spin-Transfer Torque Magnetic RAM* - STT-MRAM [Driskill-Smith 2010]; que podem prover um desempenho semelhante a DRAM, mas com persistência. PCM é apenas duas vezes mais lenta que a DRAM, e STT-MRAM e Memristor podem atingir latência maior que a DRAM. Embora NVRAM esteja atualmente disponível apenas em pequenas quantidades, especula-se que na próxima década haverão memórias PCM de 1TB e Memristors de 30TB ao preço dos discos rígidos atuais [Zhang et al. 2015]. Também existem vários trabalhos explorando o uso de NVRAM para armazenamento em bancos de dados, como: [Arulraj et al. 2015], [Chatzistergiou et al. 2015] e [Oukid et al. 2014]. Esses trabalhos utilizam simuladores de memória NVRAM para fazer seus experimentos, como o Intel Lab's NVM Hardware Emulator [Dulloor et al. 2014].

4.3. *Design* e escolhas arquitetônicas

Nessa seção serão discutidos brevemente aspectos chaves que fazem os bandos de dados em memória possuírem alto desempenho. Esses aspectos influenciam a arquitetura e técnicas de implementação do gerenciamento do bancos de dado. Para cada questão de *desing* e escolha arquitetônica, primeiro será mostrado como elas são implementadas em bancos de dados em disco para, em seguida, ser explicado como elas são implementadas nos bancos de dados em memória.

4.3.1. Organização dos dados e *layout*

Quando uma requisição é recebida por um banco de dados em disco, o gerenciador de *buffer* deve checar se a página do dado requisitado está no *buffer* (porção da memória destinada aos dados) ou não. Se essa página não estiver na memória, ela deve ser carregada do bloco no disco para a memória. Uma página pode conter vários registros de dados e possui uma representação semelhante ao bloco do disco para evitar traduções entre as duas representações. Se a página já estiver carregada na memória, muitos gerenciadores de *buffer* implementam acesso indireto ao dado: (1) encontrar a página do dado requisitado no *buffer*, através de uma tabela *hash*, por exemplo; e (2) calcular o descolamento (*offset*) necessário dentro da página para acessar o registro do dado [Ramakrishnan and Gehrke 2002]. A Figura 4.4 esquematiza o funcionamento do gerenciador de *buffer*.

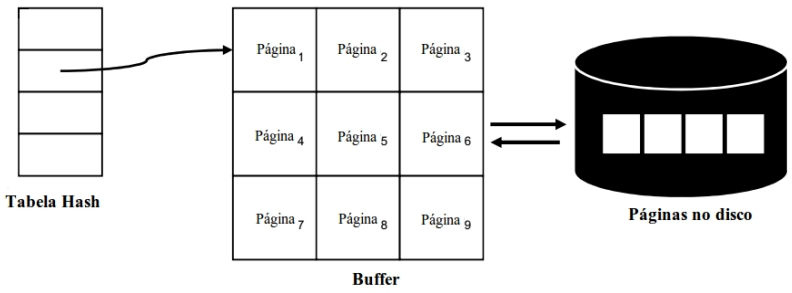


Figura 4.4. Gerenciador de *buffer* em um SGBD baseado em disco.
Fonte: [Faerber et al. 2017]

Os bancos de dados em memória não são orientados a blocos e suas representações são implementadas de forma a levar ao melhor desempenho no sistema. São exemplos de *layouts* de dados usados nos sistemas em memória modernos: particionado, multi-versionado e linha/coluna. Como os dados residem permanentemente na memória, não há a necessidade carregá-los do disco para o *buffer*. Comumente, os registros são acessados diretamente através de ponteiros. Acessar os dados diretamente na memória leva a menos ciclos de CPU, evitando o *overhead* do acesso indireto. As estratégias de acesso focam em otimizar ciclos de CPU e paralelismo *multi-core* [Hazenberg and Hemminga 2011, Faerber et al. 2017].

4.3.2. Indexação

Índices são estruturas de dados que permitem rápido acesso a dados, sem a necessidade de percorrer uma tabela inteira. A principal meta dos índices, nos sistemas baseados em disco, é minimizar o acesso ao disco, mapeando dados do banco de dados. Para

isso, tipicamente, os bancos de dados utilizam índices em árvores B+. Uma alternativa é o índice com tabela *hash*, muito eficiente em buscas com um valor específico. Sistemas em disco também utilizam índices clusterizados, que ordenam os registros em uma ordem específica no disco. Esse tipo de índice é muito eficiente para buscas usando faixas de valores. Entretanto, cada tabela só pode ter um índice clusterizado (índice primário), enquanto que, pode ter vários dos outros tipos de índices (índices secundários) [Ramakrishnan and Gehrke 2002, Elmasri and Navathe 2010].

Nos sistemas em memória, a principal meta dos índices é reduzir o tempo computacional e o uso da memória ao máximo possível. Os índices armazenam ponteiros para os registros em vez de *IDs* ou chaves primárias, como é feito nos índices para bancos em disco. A Figura 4.5 ilustra as diferenças entre os índices de sistemas em discos e em memória. Como a velocidade das CPUs modernas é superior a da memória RAM, a memória torna-se um gargalho em sistemas em memória. Assim, para resolver esse problema, técnicas de otimização do uso da *cache* são implementadas em bancos de dados em memória, como índices de árvores CSB+ [Rao and Ross 2000] e Pb+ [Chen et al. 2001], por exemplo. Os índices são estrutura frequentemente acessadas e atualizadas, o que pode comprometer a concorrência e o paralelismo. Os sistemas em memória utilizam técnicas, como PLP [Pandis et al. 2011] e árvores Mass [Mao et al. 2012], para atingir altos níveis de concorrência e paralelismo.

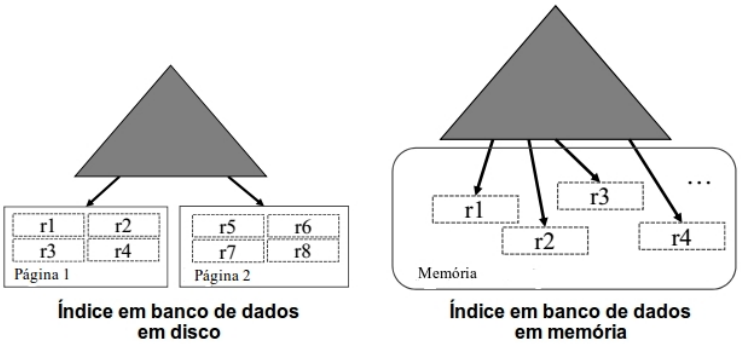


Figura 4.5. Estruturas de índices nos bancos de dados em disco e em memória. Fonte: [Faerber et al. 2017]

4.3.3. Controle de concorrência

Controle de concorrência é um método usado para impedir que transações acessem o mesmo dado simultaneamente e levem o banco de dados a uma inconsistência. Os bancos de dados relacionais em disco, tipicamente, utilizam alguma forma de bloqueio em duas fazes (*Two-Phase Locking* - 2PL) para garantir o controle de concorrência. No protocolo 2PL, antes de começar a executar, uma transação deve adquirir bloqueios a todos os dados que necessita. Em uma segunda faze, a transação deve liberar os bloqueios adquiridos, após não precisar mais dos dados. Uma transações que necessite alterar dados bloqueados por uma outra transação deve esperar até a liberação desses bloqueios [Ramakrishnan and Gehrke 2002, Elmasri and Navathe 2010].

O gerenciador de bloqueio realiza as operações de bloqueio e desbloqueio atomicamente, ou seja, nenhuma outra operação no banco de dados é permitida enquanto

uma delas está em execução. Além disso, sempre que uma transação precisa ler/escrever em um dado, o gerenciador de bloqueio deve verificar se o dado já está bloqueado por outra transação. Essas operações realizadas pelo gerenciador de bloqueio podem ser toleradas em bancos de dados em disco devido ao gargalho de acesso ao disco, mas como o acesso à memória é muito mais rápido do que ao disco, em sistemas em memória, bloqueios tornam-se muito custosos e acrescentam mais ciclos de CPU. Por esse motivo, os sistemas em memória evitam protocolos de bloqueio e preferem executar as transações serialmente ou adotar um modelo de controle de concorrência multi-versionado (*Multi-Version Concurrency Control* - MVCC) otimista [Ramakrishnan and Gehrke 2002, Hazenberg and Hemminga 2011].

Para executar uma transação serialmente, o banco é divido em partições. E quando uma transação for executada, ela deve adquirir acesso exclusivo a todas as partições que precisa. Nessa estratégia, as transações podem ser executadas em paralelo em partições diferentes com *cores* diferentes [Stonebraker and Weisberg 2013, Malviya et al. 2014]. No MVCC, uma transação faz suas alterações em uma nova versão do dado original. Apenas a transação pode visualizar a versão criada por ela. Ao realizar *commit*, a transação não sobrescreve o dado original, ela apenas marca a nova versão do dado como atual e a original como obsoleta. Nessa abordagem não há partições e as transações podem executar concorrentemente [Diaconu et al. 2013, Freedman et al. 2014].

4.3.4. Durabilidade e recuperação após falhas

A maioria dos bancos de dados relacionais baseados em disco utilizam alguma forma de recuperação após falhas baseada no protocolo ARIES. O protocolo ARIES utiliza o esquema *write-ahead logging* (WAL) que escreve para um arquivo de *log* informações sobre atualizações feitas em uma página antes da confirmação de sua atualização. Cada registro no *log* possui um número sequencial (LSN) que serve para determinar a ordem em que cada ação foi executada no banco. ARIES utiliza um *log* físico onde são armazenadas as imagens da página antes e depois de sua atualização. Assim, o *log* possui informação suficiente para desfazer (UNDO) ou refazer (REDO) uma atualização. Após uma falha, ARIES executa os seguintes passos: (1) analisa o *log* para determinar o que deve ser recuperado; (2) refaz as transações que não tiveram suas atualizações descarregadas para o disco até o momento da falha; e (3) desfaz as transações que não terminaram. Após esses passos, o banco volta a um estado consistente igual a antes da falha e pode voltar a funcionar normalmente. Além disso, periodicamente, *checkpoints* são realizados para identificar transações que já foram descarregas para o disco e, consequentemente, diminuir a quantidade de *log* a ser analisada na recuperação [Mohan and Levine 1992].

Os bancos de dados em memória também devem guardar informações de *log* no disco para prover recuperação após falhas. Como o I/O é um gargalho, o *log* deve ser otimizado para não interferir no processamento normal das transações no sistema. Para diminuir o volume de *log*, apenas informações de REDO são gravadas em *group commit*, ou seja, as informações são acumuladas para envio em lote (até o tamanho de uma página, por exemplo) em um único I/O. Os sistemas em memória adotam um *log* lógico onde são gravadas descrições em alto nível de como as atualizações são feitas, como a inserção de um registro em uma tabela, por exemplo. O *log* lógico grava menos informações do que o físico evitando *overheads* ao sistema [Garcia-Molina and Salem 1992, Kim et al. 2012].

Para reduzir ainda mais o tráfico do *log*, alguns sistemas adotam um *log* chamado de *Command logging* (ou *Transaction logging*). *Command loggin* armazena apenas o nome da *store procedure* de uma transação e seus parâmetros. Assim, todas as ações de uma transação podem ser gravadas em um único registro. Como desvantagem do *Command loggin*, todas as transações devem ser definidas em forma de *store procedure*, ou seja, precisam ser definidas previamente [Malviya et al. 2014, Wu et al. 2017].

Os *checkpoints*, nos bancos em memória, propagam assincronamente as ações dos registros contidas no *log* para um arquivo no disco [Diaconu et al. 2013]. Em vez de propagar registros de *log*, alguns sistemas produzem arquivos persistentes (chamados de *snapshots*) também de forma assíncrona, que são equivalentes a um estado do banco materializado, através de uma técnica chamada de *copy-on-update*. O *checkpoint* pode ser considerado como o *backup* mais recente do banco de dados. Após uma falha, o gerenciador de recuperação carrega o *checkpoint* mais recente para a memória e, em seguida, executa os registros de *log* gravados logo após o último *checkpoint*. Produzir *checkpoints* diminui o tempo de recuperação do banco de dados, visto que, carregar para a memória as informações físicas do arquivo de *checkpoint* é mais rápido do que executar novamente as informações lógicas do *log* [Diaconu et al. 2013, DeWitt et al. 1984, Funke et al. 2014].

4.3.5. Processamento de consultas e compilação

O processamento de consultas é a atividade responsável por extrair informações de um banco de dados da forma mais eficiente possível. A Figura 4.6 ilustra as etapas do processamento de consultas. Quando uma consulta é submetida ao sistema, a primeira ação é traduzir a consulta em alto nível (SQL, por exemplo) para uma representação interna do banco (álgebra relacional, por exemplo) melhor entendida pelo sistema. Ao gerar a forma interna da consulta, são verificados a sintaxe da consulta e se os nomes das relações, atributos e etc. pertencem ao banco. Em seguida, o otimizador escolhe um plano de execução de consulta (sequência de operações) que tenha menos custo para o banco de dados. Diferentes planos de execução podem ser gerados e espera-se que o otimizador escolha o que execute mais rápido a consulta [Silberschatz et al. 2015].

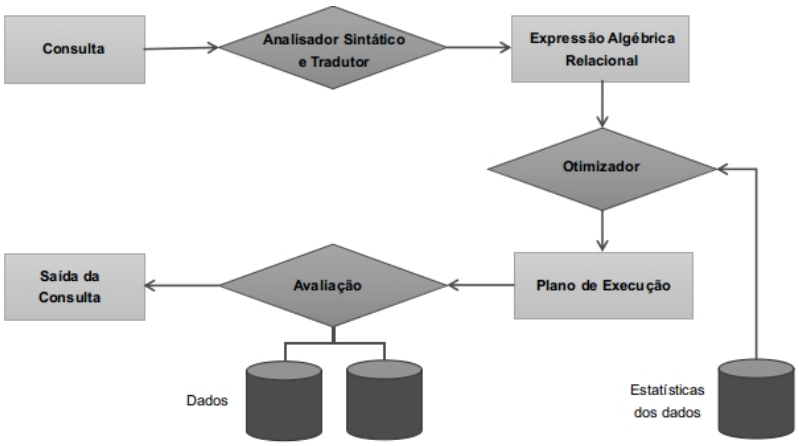


Figura 4.6. Etapas do processamento de consultas.
Fonte: [Silberschatz et al. 2015]

Os bancos de dados em disco e em memória possuem implementações semelhan-

tes nas etapas de análise e otimização, entretanto, diferem-se na execução da consulta. Os bancos em disco utilizam um modelo de iteração e interpretação em tempo de execução para executar as consultas. Essas técnicas servem para bancos de dados em discos, em que o I/O é o *overhead* dominante. Entretanto, elas são bastante custosas em bancos de dados em memória, em que ciclos de CPU a mais podem causar *overheads*. Os bancos em memória, em geral, compilam as consulta diretamente em código de máquina. Alguns bancos até restringem o uso de transações a *store procedures* para reduzir o processamento de consulta em tempo de execução [Faerber et al. 2017].

4.4. Bancos de dados em Memória

Com a tendência de residir os dados na memória, foram propostos vários tipos de bancos de dados em memória, desde os tradicionais bancos relacionais até os bancos NoSQL. Os bancos de dados relacionais utilizam o modelo relacional em que os itens de dados são organizados em tuplas de tabelas/relações com garantias ACID. São exemplos de Bancos de Dados em Memória Relacionais: SAP HANA [Färber et al. 2012], IBM DB2 with BLU Acceleration [Raman et al. 2013], HStore/VoltDB [Malviya et al. 2014], Oracle TimesTen [TimesTen Team 1999], P*Time [Cha and Song 2004], Microsoft Hekaton [Diaconu et al. 2013], ClustRa [Hvasshovd et al. 1995], solidDB [Lindström et al. 2013], NuoDB [Brynko 2012], eXtremeDB [McObject 2010], Pivotal SQLFire [Pivotal 2013], MemSQL [Memsql 2012], HyPer [Kemper and Neumann 2011], Silo [Tu et al. 2013], HYRISE [Grund et al. 2010], Dalí [Bohannon et al. 1997], DataBlitz [Baulier et al. 1999], Crescando [Unterbrunner et al. 2009], System K [Whitney et al. 1997] e MySQL Cluster NDB [Oracle 2004].

Os bancos de dados NoSQL geralmente estruturam seus dados como árvores, grafos ou chave/valor; e a linguagem de consulta, em geral, não é SQL. NoSQL é motivado pela sua simplicidade, escalabilidade horizontal, alta disponibilidade e tolerância a falhas; em detrimento da consistência. Alguns Sistemas de Bancos de Dados NoSQL são parcialmente em memória, pois usam arquivos de memória mapeada para armazenar dados de tal forma que o dado possa ser acessado como se estivesse na memória. Seguem alguns exemplos de bancos de dados NoSQL em memória: MongoDB [MongoDB 2009], MonetDB [Boncz et al. 2005], MDB [Chu 2011], MemepiC [Cai et al. 2014], RAMCloud [Ousterhout et al. 2010], Trinity [Shao et al. 2013], Redis [Sanfilippo and Noordhuis 2009] e Bitsy [Brown 2012].

Existe a categoria de Sistemas de *Cache* em Memória que são usados como uma *chace* entre o servidor de aplicação e o banco de dados adjacente. Esses sistemas são utilizados comumente por empresas como Facebook, Twitter e Wikipedia para prover bons serviços [Zhang et al. 2015]. Muitos sistema em *cache* tem sido desenvolvidos para vários propósitos. Por exemplo, existem sistemas em *cache* para uso geral, como Memcached [Fitzpatrick and Vorobey 2003] e BigTable Cache [Chang et al. 2008]; sistemas destinados a acelerar serviços de análise, como PACMan [Ananthanarayanan et al. 2012] e GridGain [Team 2007]; sistemas projetados para suportar *frameworks* específicos, como NCache [alachisoft 2005] para .NET e Velocity/AppFabric [Sampathkumar et al. 2009] para servidores Windows; sistemas que suportam semântica transacional rígida, como TxCache [Ports et al. 2010]; e *cache* de rede, como HashCache [Badam et al. 2009].

O advento de *Big Data* impulsionou o desenvolvimento de Sistemas de Processamento/Análise de Dados em Memória cujo objetivo principal é analisar grandes quantidades de dados em um pequeno intervalo de tempo [Zhang et al. 2015]. Existem basicamente dois tipos de sistemas de processamento de dados em memória: sistemas de análise de dados (por exemplo, Spark [Zaharia et al. 2012], Piccolo [Power and Li 2010], SINGA [Felber et al. 2008], Pregel [Malewicz et al. 2010], GraphLab [Malewicz et al. 2010], Mammoth [Shi et al. 2015], Phoenix [Yoo et al. 2009], GridGain [Team 2007]); e sistemas de processamento de dados em tempo real (por exemplo: Storm [BackType and Twitter 2011], Yahoo! S4 [Neumeyer et al. 2010], Spark Streaming [Zaharia et al. 2013], MapReduce Online [Condie et al. 2010]).

No restante dessa sessão serão abordados alguns sistemas de bancos de dados em memória relacionais, mas, antes disso, será dado um breve histórico do desenvolvimento de bancos de dados em memória.

4.4.1. Histórico

O desenvolvimento e pesquisa em bancos de dados em memória não é recente, começaram ainda no início da década de 80, como em IMS/Fast Path [Gawlick and Kinkade 1985, Strickland et al. 1982], MM-DBMS [Lehman and Carey 1986, Lehman and Carey 1987], MARS [Eich 1987, Gruenwald and Eich 1991], System M [Gruenwald and Eich 1991, Salem and Garcia-Molina 1990], TPK [Li and Naughton 2000], OBE [Bitton et al. 1987, Whang and Krishnamurthy 1990] e HALO [Garcia-Molina and Salem 1992, Eich 1987]. Muitas dessas pesquisas tentavam otimizar o uso da memória em bancos de dados em disco. Entretanto, a pouca capacidade de armazenamento e o custo elevando das memórias nessa época inviabilizaram a ampla adoção dos sistemas em memória.

A partir da década de 90, os avanços tecnológicos no *hardware* (principalmente na memória RAM e processamento *multicore*) e o seu barateamento impulsionaram o desenvolvimento e a pesquisa em sistemas de bancos de dados em memória. O bancos de dados desenvolvidos na década de 90 (por exemplo: ClustRa, P*Time, Dalí, DataBlitz, System K, TimesTen) já começaram a refletir as tendências e escolhas arquiteturas utilizadas nos bancos de dados em memória mais modernos, como: Hekaton, VoltDB, HyPer e SAP HANA [Garcia-Molina and Salem 1992, Eich 1987].

4.4.2. TimesTen

O TimesTen da Oracle é um banco de dados relacional em memória capaz de prover baixa latência e alto taxa de *throughput* requeridos pela indústria e empresas contemporâneas, como telecomunicações, mercado de capital e defesa, por exemplo. São exemplos de empresas que utilizam TimesTen: Ericsson, Dell, Bank of America Merrill Lynch, United States Postal Service, entre outras. Aplicativos podem acessar o TimesTen usando SQL padrão via *drivers* JDBC, ODBC e ODP.NET (*Oracle Data Provider for .NET*), além das *interfaces* Oracle PL/SQL, OCI (*Oracle Call Interface*), Pré-Compilador Pro*C/C++, e TTClasses (*TimesTen C++ Interface Classes*). A Figura 4.7 esquematiza a arquitetura de TimesTen. TimesTen pode ser instalado separadamente do banco de dados Oracle (chamado de TimesTen In-Memory Database ou simplesmente TimesTen) ou como uma *cache* para um bancos de dados Oracle (chamado de TimesTen Application-Tier Database

ou simplesmente TimesTen Cache). Implantado na camada de aplicação como um sistema embarcado, TimesTen opera em um banco de dados que cabe inteiramente na memória. TimesTen Cache é uma opção para melhora de desempenho de subconjuntos de uma banco de dados Oracle. Atualizações nas tabelas TimesTen Cache são sincronizados para tabelas Oracle automaticamente [TimesTen 2018]. No restante do texto vamos utilizar apenas TimesTen para nos referir tanto a TimesTen como a TimesTen Cache.

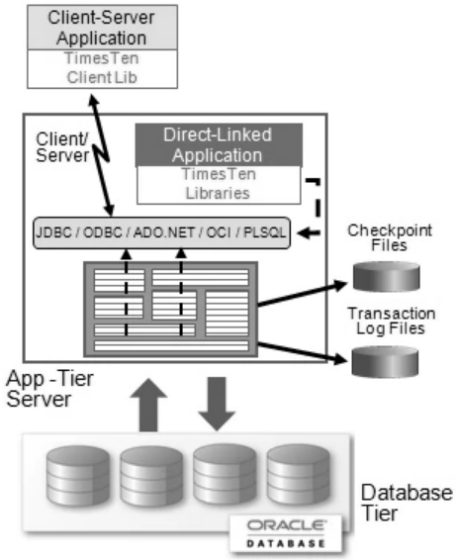


Figura 4.7. Arquitetura do banco de dados TimesTen.
Fonte: [TimesTen 2018]

TimesTen provê durabilidade e recuperação após falhas através de *log* de transações e *checkpoints*. Registros de *log* são escritos assincronamente ou sincronamente para o disco. Log assíncrono é aplicado em sistemas que exigem o máximo de *throughput* possível. Em casos onde a integridade dos dados deve ser preservada, o *log* síncrono deve ser utilizado. TimesTen executa periodicamente uma operação chamada de Fuzzy Checkpoint para gravar uma imagem do banco no disco. Essa operação é feita em *background* e causa poucos impactos a funcionamento normal do banco de dados. TimesTen também possui um *checkpoint* (chamado de Transaction-Consistent Checkpoint) que bloqueia exclusivamente o banco de dados para gravar uma imagem do banco para o disco. Esse *checkpoint* deve ser iniciado a partir de uma aplicação. Para recuperar o banco após uma falha, é necessário carregar o último *checkpoint* para memória e, após isso, executar os registros de *log* gravados após esse *checkpoint*. Adicionalmente, TimesTen pode replicar seus dados para prover alta-disponibilidade em caso de falhas [TimesTen 2018].

TimesTen possui um otimizador de consulta baseado em custos que tenta escolher o melhor plano de execução da consulta baseado em fatores como:

- presença de índices: *hash*, *bitmap* e *range*
- estatísticas sobre os dados: número de linhas e colunas de uma tabela; existência de chave primária em uma tabela; e tamanho e configuração de índices existentes.

- métodos para percorrer tabelas: *full table scan*, *rowid lookup*, *range index scan*, *bitmap index lookup*, e *hash index lookup*.
- algoritmos de junção: *nested loop joins*, *nested loop joins with indexes*, e *merge join*.

Além de tentar escolher a melhor estratégia para executar uma consulta com base nos fatores existentes, o otimizador também pode criar índices temporariamente. O otimizador também aceita *hints* que permitem decidir quais fatores podem influenciar na escolha do plano de execução da consulta [TimesTen 2018].

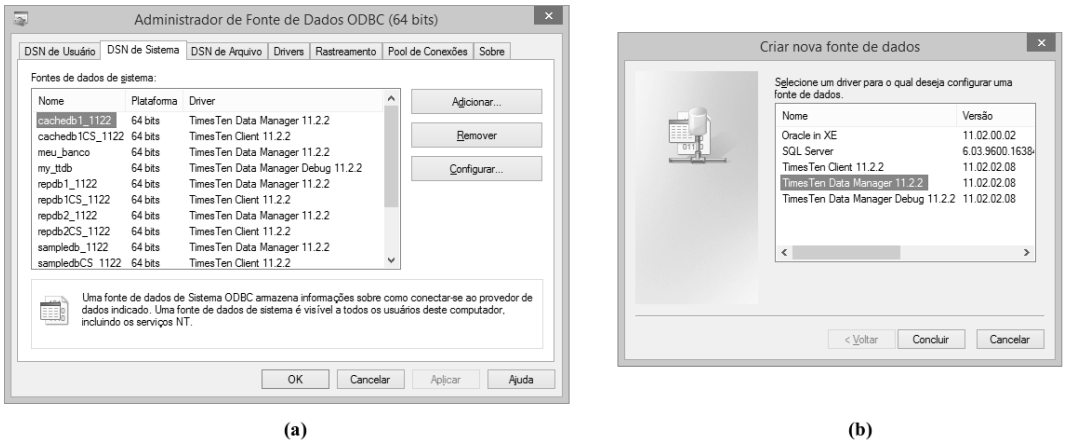
TimesTen dá suporte a bancos de dados compartilhados e coordena o acesso concorrente aos dados através de isolamento de transações e bloqueios. Esse banco provê opções para o usuário escolher um balanceamento entre tempo de resposta, *throughput* e semântica da transação. O isolamento dá a transação a impressão de estar executando sozinha, apesar de estar executando concorrentemente com outras transações no banco de dados. TimesTen suporta dois níveis de isolamento de transação: *read committed* e *serializable*. O isolamento *read committed* não utiliza bloqueios e é o modo de isolamento padrão. Nesse isolamento, as leituras são feitas em uma versão diferente dos dados das escritas, assim, não é necessário realizar bloqueios. No isolamento *serializable*, uma transação adquire bloqueios para ler/alterar dados e, consequentemente, outra transação devem esperar o desbloqueio desses dados para lê-los e/ou alterá-los. Os bloqueios realizados podem ser compartilhado, que permite leituras simultâneas de várias transações ao mesmo dado, mas não permite escritas; ou exclusivo, que permite a apenas uma transação alterar o dado durante o bloqueio. As aplicações podem selecionar o nível do bloqueio que desejam fazer: linha, tabela ou banco de dados. *Read committed* é útil para aplicações cujas transações executam longas varreduras de dados que podem conflitar com outras operações que precisam acessar uma linha percorrida. *Serializable* é útil para transações que requerem um nível mais forte de isolamento. [TimesTen 2018].

TimesTen tem distribuições para Windows e Linus e pode ser facilmente instalado, após ser baixado no site da Oracle através do link abaixo:
http://www.oracle.com/technetwork/database/database-technologies/timesten/downloads/.
Nas seções a seguir será dado um breve guia de como operar TimesTen.

4.4.2.1. Criar e manipular banco de dados TimesTen

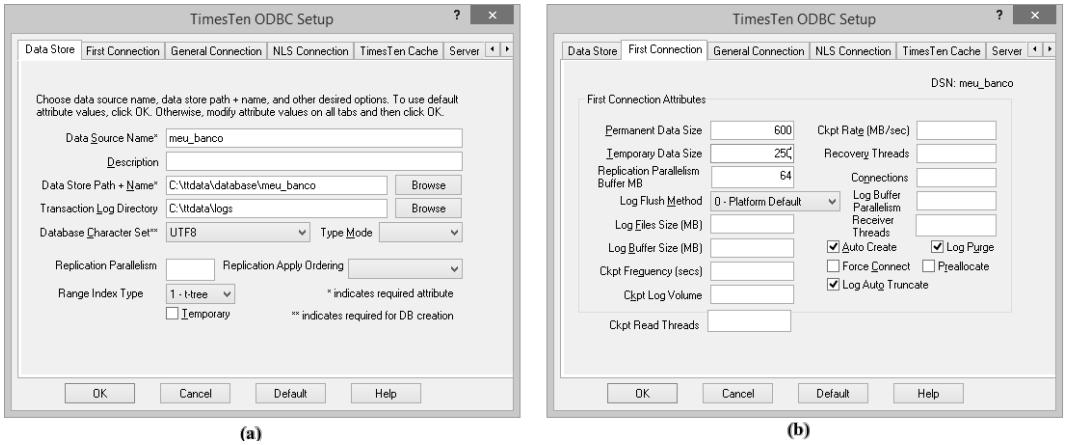
Bancos de dados TimesTen são acessados através de DSN (*Data Source Name*), que é um nome lógico para identificar um banco de dados. Cada DSN contem atributos que especificam as propriedades do banco de dados a ser criado ou acessado, tais como: nome, *path*, tamanho e etc.. Nessa seção será mostrado como configurar DSN na plataforma Windows. Um DSN pode ser criado e mantido através do "Administrador de Fonte de Dados ODBC"(Figura 4.8 (a)). Para criar uma nova fonte de dados deve ser clicado no botão "Adicionar..."da aba "DNS do Sistema"do "Administrador de Fonte de Dados ODBC". Na tela "Criar nova fonte de dados"(Figura 4.8 (b)) deve ser selecionado "TimesTen Data Manager 11.2.2"e, em seguida, clicado no botão "Concluir". Na aba "Data Source"da janela de configuração ODBC do TimesTen (Figura 4.9 (a)), devem ser setadas

as informações sobre a fonte de dados do banco: o nome do banco de dados em "Data Souce Name"; o diretório e o nome do banco de dados em "Data Source Path + Name"; o diretório dos arquivos de *log* do banco de dados em "Transaction Log Directory"; e o padrão de caracteres em "Database Character Set". No exemplo da Figura 4.9 (a) o nome do banco de dados é *meu_banco*; o *path* do banco é *C://ttdata/database/meu_banco*; o diretório de *logs* é *C://ttdata/database/logs*; e o padrão de caracteres segue o UTF-8. Na aba "First Connection"(Figura 4.9 (b)) devem ser setados, em MB, o tamanho máximo da base de dados em "Permanent Data Size"; e o tamanho máximo reservado a dados temporário em "Temporary Data Size". No exemplo da Figura 4.9 (b) o espaço reservado ao banco foi 600MB e o aos dados temporários foi 250MB. Após todos esses dados serem digitados, deve ser clicado no botão "Ok"para finalizar a configuração ODBC do banco.



(a) (b)

Figura 4.8. (a) Administrador de Fonte de Dados ODBC do Windows; e (b) Criar nova fonte de dados.



(a) (b)

Figura 4.9. Configuração ODBC de um banco de dados TimesTen.

Um banco de dados TimesTen é criando em sua primeira conexão. A primeira conexão a um banco já existente carrega-o na memória e, dependendo do tamanho da base de dados, esse carregamento pode demorar. Como exemplo, para criar e manipular um banco TimesTen é utilizada a ferramenta ttIsql, que é instalada juntamente com TimesTen

e pode ser acessada no Prompt de Comando através do comando *ttisql*. Outras ferramentas, como o SQLDeveloper, também podem ser utilizadas. No ttIsql, o comando *connect "dsn=<nome_do_banco>"* deve ser digitado para se conectar a um banco de dados. A Figura 4.10 mostra um exemplo de como se conectar ao banco chamado de *meu_banco* utilizando a ferramenta ttIsql. TimesTen utiliza o SQL padrão e , consequentemente, utiliza as instruções SQL habituais para manipular bancos de dados. A Figura 4.11 mostra instruções SQL para criar uma tabela e inserir e selecionar registros.

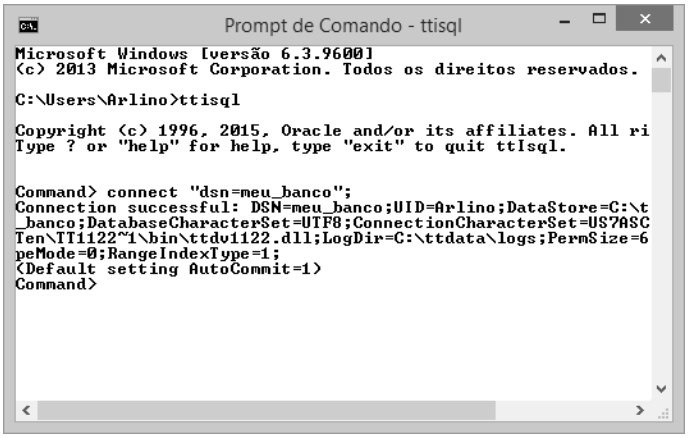


Figura 4.10. Conexão a um banco de dados TimesTen através da ferramenta ttIsql.

```
CREATE TABLE CUSTOMER (
   C_CUSTKEY NUMBER primary key,
   C_NAME VARCHAR2(25 BYTE),
   C_ADDRESS VARCHAR2(40 BYTE),
   C_PHONE VARCHAR2(15 BYTE) );

INSERT INTO CUSTOMER VALUES (1, 'Cliente', 'Rua ...', '9999-9999');

SELECT * FROM CUSTOMER;
```

Figura 4.11. Instruções SQL para criar uma tabela e inserir e selecionar registros.

4.4.3. Hekaton

Hekaton é uma *engine* do banco de dados Microsoft SQL Server que trabalho com cargas de trabalho OLTP residentes na memória. O nome oficial de Hekaton é *OLTP in-memory*. Uma tabela no Haketon (também chamada de tabela otimizada na memória) é armazenada inteiramente na memória e é durável. Um banco de dados pode conter tabelas Haketon (na memória) e tabelas SQL Server (no disco). Uma transação é acessada usando T-SQL (*Transact-SQL*) e pode ler/escrever em ambas, tabelas em memória e tabelas em disco. *Store procedures* que referenciam apenas tabelas Haketon podem ser compilados em código nativo de máquina. Os dados em Hekaton são livres de bloqueios e são implementados em multi-versionamento. Os índices também não utilizam bloqueios e podem ser

índices *hash*, *Bw-Tree* e *columnstore*. Haketon utiliza controle de concorrência MVCC otimista. O protocolo de controle de concorrência otimista não se preocupa em prevenir conflitos entre transações, ao invés disso, detecta conflitos em um processo de validação das atualizações de uma transação antes do *commit*. Se a validação falhar, a transação é abortada [Diaconu et al. 2013, Freedman et al. 2014, Larson et al. 2013].

Para atingir durabilidade e recuperação após falhas, Haketon utiliza *logs* e *checkpoints*. Apenas *logs* de REDO são produzidos, assim, quando uma transação é efetivada, informações sobre versões criadas e deletadas são enviados para o *log* em um *group commit*. Periodicamente, *checkpoints* são realizados durante a execução normal das transações no sistema. O *checkpoint* é armazenado em dois tipos de arquivos: (1) arquivos *data* que contem versões inseridas (geradas por *inserts* e *updates*); e (2) arquivos *delta* que armazenam informações sobre quais versões contidas em um arquivo *data* foram deletadas. Após uma falha, a recuperação de Haketon procura o arquivo de inventário do *checkpoint* que contem referências sobre todos os arquivos *data* e *delta* e, em seguida, carrega as versões contidas nos arquivos *data* utilizando os arquivos *delta* como filtro para versões deletadas. O par de arquivos *data*/*delta* é considerado a unidade de recuperação e permite uma recuperação altamente paralelizada. Após todos os arquivos *data* serem carregados na memória, os registros de *log* são executados a partir do *timestamp* do último *checkpoint* realizado [Diaconu et al. 2013, Faerber et al. 2017].

Haketon está disponível nas plataformas Windows 8 ou superiores e é instalado juntamente com o SQL Server a partir da versão 2014. Há uma versão *trial* do SQL Server disponível para *download* e instalação no *link*:
https://www.microsoft.com/pt-br/sql-server/sql-server-downloads
Nas seções a seguir será dado um breve guia de como operar Haketon.

4.4.3.1. Criar e manipular banco de dados Hekaton

O primeiro passo para utilizar tabelas Haketon é criar grupos de arquivos otimizados para memória. Esses arquivos retem um ou mais contêineres que contem arquivos *data* e/ou *delta*. Os grupos de arquivos podem ser criados durante a criação de um banco de dados ou posteriormente. Como exemplo, para criar e manipular um banco SQL Server Haketon será utilizada a ferramenta SQLCMD (Figura 4.12), que é instalada junta o SQL Server, e pode ser acessada através do Prompt de Comando. Digite o comando "*SQLCMD -E -S <endereço_do_servidor>\<nome_da_instância>*"no Prompt de Comando para conectar-se ao SQL Server [Haketon 2018]. No exemplo da Figura 4.12 é feita uma conexão ao servidor *localhost* na instância *MSSQLSERVER01*.

Depois de conectar-se ao SQL Server, na Figura 4.12, é criado um banco de dados chamado de *TPCH* com suporte a tabelas em memória utilizando o comando da Figura 4.13; em seguida, o banco *TPCH* é acessao através do comando "*use TPCH*"; e, finalmente, é criada uma tabela em memória chamada de *CUSTOMER* utilizando o comando da Figura 4.14. No comando da Figura 4.13 é criado o banco de dados *TPCH* com o arquivo *TPCH_data*, para armazenar os dados; e com o grupo de arquivos *TPCH_mod*, para os contêineres de otimização de memória. Os arquivos são armazenados no diretório *C:\HKData\TPCH*. No comando da Figura 4.14 é criada *CUSTOMER* que será uma ta-

bela em memória (através da expressão *MEMORY_OPTIMIZED = ON*) com durabilidade em disco (através da expressão *DURABILITY = SCHEMA_AND_DATA*). Os comandos para inserir, alterar e excluir registros de uma tabela utilizam o SQL padrão. Adicionalmente, SQLCMD exige que cada comando seja finalizado e executado através da palavra *GO* [Haketon 2018].

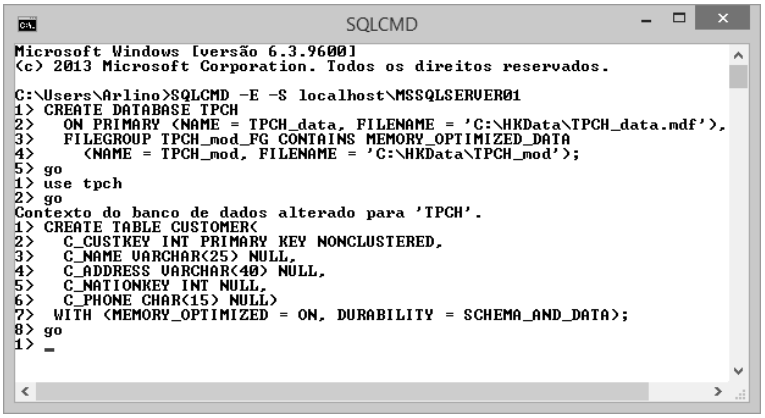


Figura 4.12. Conexão, criação de banco de dados, e criação de tabela em memória no SQL Server Haketon através da ferramenta SQLCMD.

```
CREATE DATABASE TPCH
  ON PRIMARY (NAME = TPCH_data, FILENAME = 'C:\HKData\TPCH_data.mdf'),
  FILEGROUP TPCH_mod_FG CONTAINS MEMORY_OPTIMIZED_DATA
  (NAME = TPCH_mod, FILENAME = 'C:\HKData\TPCH_mod');
```

Figura 4.13. Instrução SQL para criar um banco de dado com suporte a tabelas em memória no SQL Server Haketon.

```
CREATE TABLE CUSTOMER
  C_CUSTKEY INT PRIMARY KEY NONCLUSTERED,
  C_NAME VARCHAR(25) NULL,
  C_ADDRESS VARCHAR(40) NULL,
  C_NATIONKEY INT NULL,
  C_PHONE CHAR(15) NULL)
  WITH (MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_AND_DATA);
```

Figura 4.14. Instrução SQL para criar uma tabela durável em memória no SQL Server Haketon.

4.4.4. VoltDB

VoltDB é um banco de dados em memória projetado para cargas de trabalho OLTP que pode ser implantado em um *cluster* e cujo *design* é baseado no H-Store [Kallman et al. 2008] (versão acadêmica). VoltDB particiona seus dados (Figura 4.15 (a)) e, consequentemente,

pode suportar bases de dados maiores do que a memória disponível em um único nó. Além disso, os nós podem ser replicados para implementar alta disponibilidade e tolerância a falhas. As transações são executadas serialmente (Figura 4.15 (b)), ou seja, uma transação deve ter acesso exclusivo a todas as partições que precisa antes de começar a executar. Cada transação deve ser um *store procedure* pré definida [Malviya et al. 2014, Faerber et al. 2017, Stonebraker and Weisberg 2013].

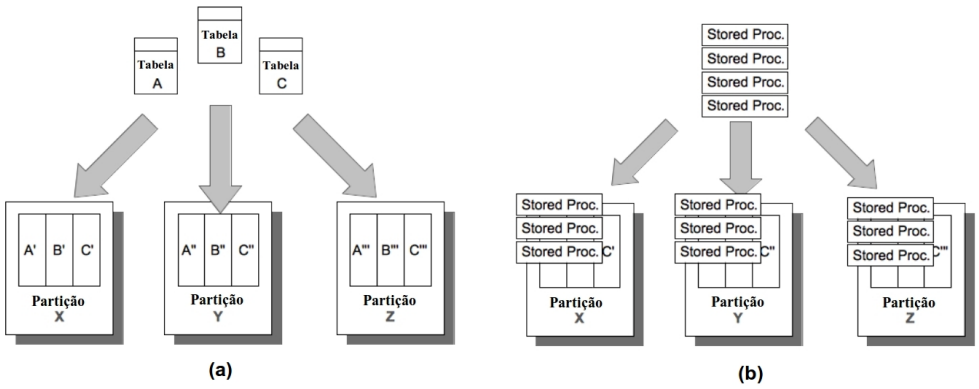


Figura 4.15. (a) particionamento de dados e (b) execução serial no VoltDB.

VoltDB utiliza *command loggin*, descrito na Seção 4.3.4, e um *checkpoint* sem bloqueios consistente a nível de transação para prover tolerância a falhas. Uma transação executada em uma única partição grava suas ações em seu arquivo de *log*. Em uma transação distribuída (executadas em várias partições), apenas o nó que coordenada a execução da transação grava os registros de *log*. Os *logs* são escritos também para as replicas de um nó, se elas existirem. Checkpoints são executados periodicamente durante o funcionamento normal das transações no sistema. O *checkpoint* produz uma imagem do banco de dados para o disco chamada de *snapshot*. Após uma falha, o processo de recuperação carrega para a memória o último *snapshot* e, em seguida, os registros de *log* gravados depois do *snapshot* são executados [Malviya et al. 2014, Faerber et al. 2017].

VoltDB está disponível nas plataforma Linux. Há uma versão *trial* do VoltDB disponível para *download* no *link*:
https://www.voltdb.com/try-voltdb/
Na seção a seguir será dado um breve guia de como operar VoltDB em um único nó.

4.4.4.1. Criar e manipular banco de dados VoltDB

A instalação de VoltDB é simples e requer apenas descompactar o arquivo de instalação. A descompactação pode ser feita utilizando as ferramentas gráficas de compactação/descompactação de arquivos disponíveis no Linux ou através de linha de comando em Terminal, como o *bash*, por exmeplo. Por exemplo, o comando "*tar -zxvf voltdb-ent-8.0.tar.gz -C $HOME/*"faz a descompactação do arquivo *voltdb-ent-8.0.tar.gz* para o diretório pessoal do usuário. É recomendado que o nome do diretório descompactado do VoltDB seja mudado para um nome mais simples como simplesmente *voltdb* [VoltDB 2018]. A Figura 4.16 sumariza a sequência de comandos a serem digitados em

um Terminal para instalação do VoltDB.

```
$ tar -zxvf voltdb-ent-8.0.tar.gz -C $HOME/
$ sudo mv voltdb-ent-8.0 voltdb
```

Figura 4.16. Passos para instalação do VoltDB no Linux via Terminal.

VoltDB possui *scripts* que facilitam o processo de desenvolvimento e implantação de aplicações. Esses *scripts* estão na pasta */bin* que está no diretório onde o VoltDB foi instalado. Para tornar esses *scripts* disponíveis em forma de comandos no Terminal do Linux é necessário adicionar o *path* do diretório */bin* às variáveis de ambiente do sistema utilizando, por exemplo, o comando "*export PATH="$PATH:$HOME/voltdb/bin*"no Terminal. A Figura 4.17 ilustra o comando para adicionar o *path* "*$HOME/voltdb/bin*"às variáveis de ambiente. Para não precisar adicionar o *path* a cada vez que uma sessão for criada, o comando da Figura 4.17 pode ser adicionado no final do arquivo "*$HOME/.bashrc*".

```
$ export PATH="$PATH:$HOME/voltdb/bin
```

Figura 4.17. Comando para adicionar o *path* do diretório */bin* do VoltDB às variáveis de ambiente no Linux via Terminal.

Antes de iniciar o VoltDB, o diretório *root* do sistema deve ser inicializado. Essa diretório armazena informações de configuração de dados, *logs* e outras informações relacionadas ao disco. Após o diretório *root* ser inicializado, o VoltDB pode ser iniciado. Os comandos "*voltdb init*"e "*voltdb start*"devem ser digitados no Terminal para inicializar o diretório *root* e inciar o VoltDB, respectivamente. Adicionalmente, para parar o VoltDB, o comando "*voltadmin shutdown*"deve ser utilizado. A Figura 4.18 mostra a execução dos comandos para inicializar o diretório *root* e para iniciar o VoltDB em um Terminal mensagens retornadas por esses comandos.

```
$ voltdb init
$ voltdb start
$ voltadmin shutdown
```

Figura 4.18. Comandos para inicializar o diretório *root* do VoltDB e para iniciá-lo e pará-lo.

Após iniciado o VoltDB, podem ser executados comandos SQL, como: criação de tabelas e inserção, alteração e remoção de linhas de tabelas. VoltDB utiliza o SQL padrão, como exemplificado na Figura 4.11. A ferramenta SQLCMD pode ser utilizada para executar comandos SQL no VoltDB e pode ser acessada através do comando *sqlcmd* no

Terminal. Adicionalmente, VoltDB possui a ferramenta de gerenciamento VoltDB Management Center que pode ser acessada digitando a *url http://<nome_do_servidor>:8080* no *browser*, onde *<nome_do_servidor>* é o nome do servidor Web do VoltDB, como *localhost*, por exemplo.

4.5. Conclusões

Muitas aplicações contemporâneas tem requerido serviços de latência muito baixa e com possibilidades de análise em tempo real. Os sistemas convencionais baseados não tem conseguido atender aos requerimentos das aplicações contemporâneas. Assim, sistemas em memória tem sido utilizados como alternativa para atender aos requerimentos dessas aplicações, visto que, os bancos de dados em memória armazenam seus dados diretamente na memória proporcionando altas velocidades de acesso. Entretanto, devido a grande diferença entre o disco e a memória, todos os aspectos de gerenciamento em banco de dados em memória devem ser repensado. Este trabalho discutiu os principais conceitos e técnicas que diferenciam os bancos residentes em memória dos bancos residentes em disco, auxiliando na tomada de decisão de tipo de banco é mais adequado a um determinado sistema.

Referências

[alachisoft 2005] alachisoft (2005). Ncache: In-memory distributed cache for .net. *http://www.alachisoft.com/ncache/*. Acessado em: 14/04/2018.

[Ananthanarayanan et al. 2012] Ananthanarayanan, G., Ghodsi, A., Wang, A., Borthakur, D., Kandula, S., Shenker, S., and Stoica, I. (2012). Pacman: Coordinated memory caching for parallel jobs. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 20–20. USENIX Association.

[Arulraj et al. 2015] Arulraj, J., Pavlo, A., and Dulloor, S. R. (2015). Let's talk about storage & recovery methods for non-volatile memory database systems. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 707–722. ACM.

[BackType and Twitter 2011] BackType and Twitter (2011). Storm. *https://storm.incubator.apache.org*. Acessado em: 18/04/2018.

[Badam et al. 2009] Badam, A., Park, K., Pai, V. S., and Peterson, L. L. (2009). Hashcache: Cache storage for the next billion. In *NSDI*, volume 9, pages 123–136.

[Baulier et al. 1999] Baulier, J., Bohannon, P., Gogate, S., Gupta, C., and Haldar, S. (1999). Datablitz storage manager: main-memory database performance for critical applications. In *ACM SIGMOD Record*, volume 28, pages 519–520. ACM.

[Bitton et al. 1987] Bitton, D., Hanrahan, M. B., and Turbyfill, C. (1987). Performance of complex queries in main memory database systems. In *Data Engineering, 1987 IEEE Third International Conference on*, pages 72–81. IEEE.

[Bohannon et al. 1997] Bohannon, P., Lieuwen, D., Rastogi, R., Silberschatz, A., Seshadri, S., and Sudarshan, S. (1997). The architecture of the dali main-memory storage manager. In *Multimedia Database Management Systems*, pages 23–59. Springer.

[Boncz et al. 2005] Boncz, P. A., Zukowski, M., and Nes, N. (2005). Monetdb/x100: Hyper-pipelining query execution. In *Cidr*, volume 5, pages 225–237.

[Brown 2012] Brown, M. C. (2012). *Getting Started with Couchbase Server: Extreme Scalability at Your Fingertips*. "O'Reilly Media, Inc.".

[Brynko 2012] Brynko, B. (2012). Nuodb: Reinventing the database. *Information Today*, 29(9):9–9.

[Cai et al. 2014] Cai, Q., Zhang, H., Chen, G., Ooi, B. C., and Tan, K.-L. (2014). Memepic: Towards a database system architecture without system calls.

[Cascaval et al. 2008] Cascaval, C., Blundell, C., Michael, M., Cain, H. W., Wu, P., Chiras, S., and Chatterjee, S. (2008). Software transactional memory: Why is it only a research toy? *Queue*, 6(5):40.

[Cha and Song 2004] Cha, S. K. and Song, C. (2004). P* time: Highly scalable oltp dbms for managing update-intensive stream workload. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 1033–1044. VLDB Endowment.

[Chang et al. 2008] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4.

[Chatzistergiou et al. 2015] Chatzistergiou, A., Cintra, M., and Viglas, S. D. (2015). Rewind: Recovery write-ahead system for in-memory non-volatile data-structures. *Proceedings of the VLDB Endowment*, 8(5):497–508.

[Chen et al. 2001] Chen, S., Gibbons, P. B., and Mowry, T. C. (2001). *Improving index performance through prefetching*, volume 30. ACM.

[Chu 2011] Chu, H. (2011). Mdb: A memory-mapped database and backend for openldap. In *Proceedings of the 3rd International Conference on LDAP, Heidelberg, Germany*, page 35.

[Condie et al. 2010] Condie, T., Conway, N., Alvaro, P., Hellerstein, J. M., Elmeleegy, K., and Sears, R. (2010). Mapreduce online. In *Nsdi*, volume 10, page 20.

[DeWitt et al. 1984] DeWitt, D. J., Katz, R. H., Olken, F., Shapiro, L. D., Stonebraker, M. R., and Wood, D. A. (1984). *Implementation techniques for main memory database systems*, volume 14. ACM.

[Diaconu et al. 2013] Diaconu, C., Freedman, C., Ismert, E., Larson, P.-A., Mittal, P., Stonecipher, R., Verma, N., and Zwilling, M. (2013). Hekaton: Sql server's memory-optimized oltp engine. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1243–1254. ACM.

[Driskill-Smith 2010] Driskill-Smith, A. (2010). Latest advances and future prospects of stt-ram. In *Non-Volatile Memories Workshop*, pages 11–13.

[Dulloor et al. 2014] Dulloor, S. R., Kumar, S., Keshavamurthy, A., Lantz, P., Reddy, D., Sankaran, R., and Jackson, J. (2014). System software for persistent memory. In *Proceedings of the Ninth European Conference on Computer Systems*, page 15. ACM.

[Eich 1987] Eich, M. H. (1987). A classification and comparison of main memory database recovery techniques. In *Data Engineering, 1987 IEEE Third International Conference on*, pages 332–339. IEEE.

[Elmasri and Navathe 2010] Elmasri, R. and Navathe, S. (2010). *Fundamentals of database systems*. Addison-Wesley Publishing Company.

[Faerber et al. 2017] Faerber, F., Kemper, A., Larson, P.-Å., Levandoski, J., Neumann, T., Pavlo, A., et al. (2017). Main memory database systems. *Foundations and Trends® in Databases*, 8(1-2):1–130.

[Färber et al. 2012] Färber, F., May, N., Lehner, W., Große, P., Müller, I., Rauhe, H., and Dees, J. (2012). The sap hana database–an architecture overview. *IEEE Data Eng. Bull.*, 35(1):28–33.

[Felber et al. 2008] Felber, P., Fetzer, C., and Riegel, T. (2008). Dynamic performance tuning of word-based software transactional memory. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pages 237–246. ACM.

[Fitzpatrick and Vorobey 2003] Fitzpatrick, B. and Vorobey, A. (2003). Memcached: A distributed memory object caching system. *http://memcached.org/*. Acessado em: 14/04/2018.

[Freedman et al. 2014] Freedman, C., Ismert, E., and Larson, P.-Å. (2014). Compilation in the microsoft sql server hekaton engine. *IEEE Data Eng. Bull.*, 37(1):22–30.

[Funke et al. 2014] Funke, F., Kemper, A., Mühlbauer, T., Neumann, T., and Leis, V. (2014). Hyper beyond software: Exploiting modern hardware for main-memory database systems. *Datenbank-Spektrum*, 14(3):173–181.

[Garcia-Molina and Salem 1992] Garcia-Molina, H. and Salem, K. (1992). Main memory database systems: An overview. *IEEE Transactions on knowledge and data engineering*, 4(6):509–516.

[Gawlick and Kinkade 1985] Gawlick, D. and Kinkade, D. (1985). Varieties of concurrency control in ims/vs fast path. *IEEE Database Eng. Bull.*, 8(2):3–10.

[Gruenwald and Eich 1991] Gruenwald, L. and Eich, M. H. (1991). *MMDB reload algorithms*, volume 20. ACM.

[Gruenwald and Eich 1994] Gruenwald, L. and Eich, M. H. (1994). Mmdb reload concerns. *Information sciences*, 76(1):151–176.

[Grund et al. 2010] Grund, M., Krüger, J., Plattner, H., Zeier, A., Cudre-Mauroux, P., and Madden, S. (2010). Hyrise: a main memory hybrid storage engine. *Proceedings of the VLDB Endowment*, 4(2):105–116.

[Haketon 2018] Haketon (2018). Otimizar o desempenho usando tecnologias in-memory no banco de dados sql. *https://docs.microsoft.com/pt-br/azure/sql-database/sql-database-in-memory*. Acessado em: 30/04/2018.

[Hazenberg and Hemminga 2011] Hazenberg, W. and Hemminga, S. (2011). Main memory database systems. *SC@ RUG 2011 proceedings*, page 113.

[Hvasshovd et al. 1995] Hvasshovd, S.-O., Torbjørnsen, Ø., Bratsberg, S. E., and Holager, P. (1995). The clustra telecom database: High availability, high throughput, and real-time response. In *Proceedings of the 21th International Conference on Very Large Data Bases*, pages 469–477. Morgan Kaufmann Publishers Inc.

[Kallman et al. 2008] Kallman, R., Kimura, H., Natkins, J., Pavlo, A., Rasin, A., Zdonik, S., Jones, E. P., Madden, S., Stonebraker, M., Zhang, Y., et al. (2008). H-store: a high-performance, distributed main memory transaction processing system. *Proceedings of the VLDB Endowment*, 1(2):1496–1499.

[Kemper and Neumann 2011] Kemper, A. and Neumann, T. (2011). Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 195–206. IEEE.

[Kim et al. 2012] Kim, J.-J., Kang, J.-J., and Lee, K.-Y. (2012). Recovery methods in main memory dbms. *International journal of advanced smart convergence*, 1(2):26–29.

[Knight 1986] Knight, T. (1986). An architecture for mostly functional languages. In *Proceedings of the 1986 ACM conference on LISP and functional programming*, pages 105–112. ACM.

[Larson et al. 2013] Larson, P.-Å., Zwilling, M., and Farlee, K. (2013). The hekaton memory-optimized oltp engine. *IEEE Data Eng. Bull.*, 36(2):34–40.

[Lehman and Carey 1986] Lehman, T. J. and Carey, M. J. (1986). *Query processing in main memory database management systems*, volume 15. ACM.

[Lehman and Carey 1987] Lehman, T. J. and Carey, M. J. (1987). *A recovery algorithm for a high-performance memory-resident database system*, volume 16. ACM.

[Leis et al. 2014a] Leis, V., Boncz, P., Kemper, A., and Neumann, T. (2014a). Morsel-driven parallelism: a numa-aware query evaluation framework for the many-core age. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 743–754. ACM.

[Leis et al. 2014b] Leis, V., Kemper, A., and Neumann, T. (2014b). Exploiting hardware transactional memory in main-memory databases. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 580–591. IEEE.

[Li and Naughton 2000] Li, K. and Naughton, J. F. (2000). Multiprocessor main memory transaction processing. In *Proceedings of the first international symposium on Databases in parallel and distributed systems*, pages 177–187. IEEE Computer Society Press.

[Li et al. 2013] Li, Y., Pandis, I., Mueller, R., Raman, V., and Lohman, G. M. (2013). Numa-aware algorithms: the case of data shuffling. In *CIDR*.

[Lindström et al. 2013] Lindström, J., Raatikka, V., Ruuth, J., Soini, P., and Vakkila, K. (2013). Ibm soliddb: In-memory database optimized for extreme speed and availability. *IEEE Data Eng. Bull.*, 36(2):14–20.

[Lintzmayer et al.] Lintzmayer, C. N., Theodoro, E., and Sambinelli, M. Memória transacional.

[Maas et al. 2013] Maas, L. M., Kissinger, T., Habich, D., and Lehner, W. (2013). Buzzard: A numa-aware in-memory indexing system. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1285–1286. ACM.

[Malewicz et al. 2010] Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. (2010). Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM.

[Malviya et al. 2014] Malviya, N., Weisberg, A., Madden, S., and Stonebraker, M. (2014). Rethinking main memory oltp recovery. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 604–615. IEEE.

[Mao et al. 2012] Mao, Y., Kohler, E., and Morris, R. T. (2012). Cache craftiness for fast multicore key-value storage. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 183–196. ACM.

[McObject 2010] McObject (2010). extremedb database system. *http://www.mcobject.com/extremedbfamily.shtml*. Acessado: 14/04/2018.

[Memsql 2012] Memsql (2012). Memsql inc. *http://www.memsql.com/*. Acessado em: 14/04/2018.

[Mohan and Levine 1992] Mohan, C. and Levine, F. (1992). *ARIES/IM: an efficient and high concurrency index management method using write-ahead logging*, volume 21. ACM.

[MongoDB 2009] MongoDB (2009). Mongodb. *http://www.mongodb.org/*. Acessado em: 14/04/2018.

[Neumeyer et al. 2010] Neumeyer, L., Robbins, B., Nair, A., and Kesari, A. (2010). S4: Distributed stream computing platform. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pages 170–177. IEEE.

[Oracle 2004] Oracle (2004). Mysql cluster ndb. *http://www.mysql.com/*. Acessado em: 14/04/2018.

[Oukid et al. 2014] Oukid, I., Booss, D., Lehner, W., Bumbulis, P., and Willhalm, T. (2014). Sofort: A hybrid scm-dram storage engine for fast data recovery. In *Proceedings of the Tenth International Workshop on Data Management on New Hardware*, page 8. ACM.

[Ousterhout et al. 2010] Ousterhout, J., Agrawal, P., Erickson, D., Kozyrakis, C., Leverich, J., Mazières, D., Mitra, S., Narayanan, A., Parulkar, G., Rosenblum, M., et al. (2010). The case for ramclouds: scalable high-performance storage entirely in dram. *ACM SIGOPS Operating Systems Review*, 43(4):92–105.

[Pandis et al. 2011] Pandis, I., Tözün, P., Johnson, R., and Ailamaki, A. (2011). Plp: page latch-free shared-everything oltp. *Proceedings of the VLDB Endowment*, 4(10):610–621.

[Pivotal 2013] Pivotal (2013). Pivotal sqlfire. *http://www.vmware.com/products/vfabric-sqlfire/overview.html*. Acessado em: 14/04/2018.

[Porobic et al. 2014] Porobic, D., Liarou, E., Tozun, P., and Ailamaki, A. (2014). Atrapos: Adaptive transaction processing on hardware islands. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 688–699. IEEE.

[Porobic et al. 2012] Porobic, D., Pandis, I., Branco, M., Tözün, P., and Ailamaki, A. (2012). Oltp on hardware islands. *Proceedings of the VLDB Endowment*, 5(11):1447–1458.

[Ports et al. 2010] Ports, D. R., Clements, A. T., Zhang, I., Madden, S., and Liskov, B. (2010). Transactional consistency and automatic management in an application data cache.

[Power and Li 2010] Power, R. and Li, J. (2010). Piccolo: Building fast, distributed programs with partitioned tables. In *OSDI*, volume 10, pages 1–14.

[Ramakrishnan and Gehrke 2002] Ramakrishnan, R. and Gehrke, J. (2002). *Database management systems*. McGraw Hill.

[Raman et al. 2013] Raman, V., Attaluri, G., Barber, R., Chainani, N., Kalmuk, D., KulandaiSamy, V., Leenstra, J., Lightstone, S., Liu, S., Lohman, G. M., et al. (2013). Db2 with blu acceleration: So much more than just a column store. *Proceedings of the VLDB Endowment*, 6(11):1080–1091.

[Rao and Ross 2000] Rao, J. and Ross, K. A. (2000). Making b+-trees cache conscious in main memory. In *ACM SIGMOD Record*, volume 29, pages 475–486. ACM.

[Raoux et al. 2008] Raoux, S., Burr, G. W., Breitwisch, M. J., Rettner, C. T., Chen, Y.-C., Shelby, R. M., Salinga, M., Krebs, D., Chen, S.-H., Lung, H.-L., et al. (2008). Phase-change random access memory: A scalable technology. *IBM Journal of Research and Development*, 52(4.5):465–479.

[Robbins 2008] Robbins, S. (2008). Ram is the new disk... *https://www.infoq.com/news/2008/06/ram-is-disk/*. Acessado em: 14/04/2018.

[Saha et al. 2006] Saha, B., Adl-Tabatabai, A.-R., Hudson, R. L., Minh, C. C., and Hertzberg, B. (2006). Mcrt-stm: a high performance software transactional memory system for a multi-core runtime. In *Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 187–197. ACM.

[Salem and Garcia-Molina 1990] Salem, K. and Garcia-Molina, H. (1990). System m: A transaction processing testbed for memory resident data. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):161–172.

[Sampathkumar et al. 2009] Sampathkumar, N., Krishnaprasad, M., and Nori, A. (2009). Introduction to caching with windows server appfabric. *Microsoft Corporation, Albuquerque, NM, USA, Tech. Rep*.

[Sanfilippo and Noordhuis 2009] Sanfilippo, S. and Noordhuis, P. (2009). Redis. *http://redis.io*. Acessado em: 14/04/2018.

[Shao et al. 2013] Shao, B., Wang, H., and Li, Y. (2013). Trinity: A distributed graph engine on a memory cloud. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 505–516. ACM.

[Shavit and Touitou 1997] Shavit, N. and Touitou, D. (1997). Software transactional memory. *Distributed Computing*, 10(2):99–116.

[Shi et al. 2015] Shi, X., Chen, M., He, L., Xie, X., Lu, L., Jin, H., Chen, Y., and Wu, S. (2015). Mammoth: Gearing hadoop towards memory-intensive mapreduce applications. *IEEE Transactions on Parallel and Distributed Systems*, 26(8):2300–2315.

[Silberschatz et al. 2015] Silberschatz, A., Korth, H. F., Sudarshan, S., et al. (2015). *Database system concepts*, volume 4. McGraw-Hill New York.

[Stonebraker and Weisberg 2013] Stonebraker, M. and Weisberg, A. (2013). The voltdb main memory dbms. *IEEE Data Eng. Bull.*, 36(2):21–27.

[Strickland et al. 1982] Strickland, J. P., Uhrowczik, P. P., and Watts, V. L. (1982). Ims/vs: An evolving system. *IBM Systems Journal*, 21(4):490–510.

[Strukov et al. 2008] Strukov, D. B., Snider, G. S., Stewart, D. R., and Williams, R. S. (2008). The missing memristor found. *nature*, 453(7191):80.

[Team 2007] Team, G. (2007). Gridgain: In-memory computing platform. *http://gridgain.com/*. Acessado em: 14/04/2018.

[TimesTen 2018] TimesTen (2018). Oracle timesten in-memory database documentation. *https://docs.oracle.com/cd/E21901_01/index.html*. Acessado em: 24/04/2018.

[TimesTen Team 1999] TimesTen Team, C. (1999). In-memory data management for consumer transactions the timesten approach. In *ACM SIGMOD Record*, volume 28, pages 528–529. ACM.

[Tu et al. 2013] Tu, S., Zheng, W., Kohler, E., Liskov, B., and Madden, S. (2013). Speedy transactions in multicore in-memory databases. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 18–32. ACM.

[Unterbrunner et al. 2009] Unterbrunner, P., Giannikis, G., Alonso, G., Fauser, D., and Kossmann, D. (2009). Predictable performance for unpredictable workloads. *Proceedings of the VLDB Endowment*, 2(1):706–717.

[VoltDB 2018] VoltDB (2018). Voltdb documentation - using voltdb. *https://docs.voltdb.com/UsingVoltDB/*. Acessado em: 04/05/2018.

[Whang and Krishnamurthy 1990] Whang, K.-Y. and Krishnamurthy, R. (1990). Query optimization in a memory-resident domain relational calculus database system. *ACM Transactions on Database Systems (TODS)*, 15(1):67–95.

[Whitney et al. 1997] Whitney, A., Shasha, D., and Apter, S. (1997). High volume transaction processing without concurrency control, two phase commit, sql or c++.

[Wu et al. 2017] Wu, Y., Guo, W., Chan, C.-Y., and Tan, K.-L. (2017). Fast failure recovery for main-memory dbmss on multicores. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 267–281. ACM.

[Yoo et al. 2009] Yoo, R. M., Romano, A., and Kozyrakis, C. (2009). Phoenix rebirth: Scalable mapreduce on a large-scale shared-memory system. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 198–207. IEEE.

[Zaharia et al. 2012] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., and Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association.

[Zaharia et al. 2013] Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., and Stoica, I. (2013). Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 423–438. ACM.

[Zhang et al. 2015] Zhang, H., Chen, G., Ooi, B. C., Tan, K.-L., and Zhang, M. (2015). In-memory big data management and processing: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1920–1948.

[Zilles and Rajwar 2007] Zilles, C. and Rajwar, R. (2007). Implications of false conflict rate trends for robust software transactional memory. In *Workload Characterization, 2007. IISWC 2007. IEEE 10th International Symposium on*, pages 15–24. IEEE.

Sobre os autores

Arlino Henrique Magalhães de Araújo

Possui mestrado em Ciência da Computação pela Universidade Federal do Ceará - UFC (2013) e graduação em Bacharelado em Ciência da Computação pela Universidade Federal do Piauí - UFPI (2004). Atualmente, trabalha como professor no curso de Sistemas de Informação no Centro de Educação Aberta à Distância da UFPI e está cursando doutorado na UFC. Tem como áreas de interesse Banco de Dados e Engenharia de Software atuando principalmente nos seguintes temas: sintonia automática de bancos de dados e bancos de dados em memória. Detalhes em: http://lattes.cnpq.br/6618696720418338.

José Maria da Silva Monteiro Filho

Possui graduação em Bacharelado em Computação pela Universidade Federal do Ceará - UFC (1998), mestrado em Ciência da Computação pela UFC (2001) e doutorado em Informática pela Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio (2008). Atualmente é professor na UFC. Tem experiência na área de Ciência da Computação, com ênfase em Banco de Dados e Engenharia de Software, atuando principalmente nos seguintes temas: sintonia automática de bancos de dados, bancos de dados em nuvem, dados ligados na Web e qualidade de software. Detalhes em: http://lattes.cnpq.br/9790693300026949.

Ângelo Roncalli Alencar Brayner

Ângelo Brayner concluiu sua graduação em Ciência da Computação em 1988 pela Universidade Federal do Ceará - UFC. Em 1994, obteve o título de Mestre em Ciência da Computação pela Universidade Estadual de Campinas. Em 1999, concluiu o doutorado em Ciência da Computação pela Universität Kaiserslautern, Alemanha, sob a orientação do Prof. Dr.-Ing. Theo Härder. Atualmente é professor titular na UFC. Em 2014, ministrou uma disciplina na Universidade de Stuttgart (programa de mestrado - IMSE), Alemanha, como professor visitante. É autor do livro Transaction Management in Multidatabase Systems, publicado pela Shaker-Verlag, Alemanha, em 1999. Prof. Ângelo Brayner é autor de mais de 50 artigos publicados em periódicos e conferências internacionais e nacionais. Detalhes em: http://lattes.cnpq.br/3895469714548887.