



Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
Instituto Federal Catarinense
Campus Rio do Sul

FERNANDO SCHULZ

FRAMEWORK PARA SISTEMAS DE RACIOCÍNIO BASEADO EM CASOS

Rio do Sul
2017

FERNANDO SCHULZ

FRAMEWORK PARA SISTEMAS DE RACIOCÍNIO BASEADO EM CASOS

Projeto de Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia Catarinense – Campus Rio do Sul para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Daniel Gomes Soares, Msc.

Coorientador: Prof. Cristhian Heck, Msc.

Rio do Sul

2017

FERNANDO SCHULZ

FRAMEWORK PARA SISTEMAS DE RACIOCÍNIO BASEADO EM CASOS

Projeto de Trabalho de Conclusão de Curso
apresentado ao curso de graduação em Ciência da
Computação do Instituto Federal de Educação,
Ciência e Tecnologia Catarinense – Campus Rio
do Sul para a obtenção do título de Bacharel em
Ciência da Computação.

Rio do Sul (SC), 07 de dezembro de 2017

Prof. e Orientador Daniel Gomes Soares, Msc.

Instituto Federal de Educação, Ciência e Tecnologia Catarinense – Campus Rio do Sul

Prof. e Coorientador Cristhian Heck, Msc.

Instituto Federal de Educação, Ciência e Tecnologia Catarinense – Campus Rio do Sul

BANCA EXAMINADORA

Prof. Rodrigo Curvello, Msc.

Instituto Federal de Educação, Ciência e Tecnologia Catarinense – Campus Rio do Sul

Prof(a). Marcela Leite, Msc.

Instituto Federal de Educação, Ciência e Tecnologia Catarinense – Campus Rio do Sul

AGRADECIMENTOS

A minha família Edilson Schulz, Anelise Michels Schulz e Djanara Ropelato Martins, pelo incentivo e apoio incondicionais, proporcionados durante os anos letivos.

Aos meus orientadores Daniel Gomes Soares e Cristhian Heck, pela amizade, conselhos e dedicação, sempre prestativos em apoiar as atividades que resultaram nesta monografia.

Ao meu chefe e amigo Fabio Baldo, por permitir horários flexíveis no emprego durante os anos letivos, permitindo assim, minha formação acadêmica no horário matutino.

E a todos os professores e colegas do curso, que direta ou indiretamente contribuíram para minha formação acadêmica e profissional.

RESUMO

A IA abrange uma enorme variedade de subcampos, desde área de uso geral, como aprendizado e percepção, até tarefas específicas como jogos de xadrez, demonstração de teoremas matemáticos, criação de poesia e diagnóstico de doenças.

Uma subárea da IA que se originou a partir de uma abordagem para a solução de problemas e para o aprendizado com base em experiência passada, chama-se Raciocínio Baseado em Casos (RBC). De uma forma simplificada, podemos entender o RBC como a solução de novos problemas por meio da utilização de casos anteriores já conhecidos. Um novo problema que nos é apresentado é resolvido com a reutilização da solução de um problema anterior parecido com o atual.

Diversos artigos, trabalhos de conclusão de curso e livros da área, implementam, utilizam e exemplificam o RBC através de um framework, ou seja, usando-o através de uma aplicação ou bibliotecas, provendo assim, uma funcionalidade genérica para a técnica, porém, atualmente, as soluções existentes para o desenvolvimento de sistema de RBC obrigam o usuário a terem pleno domínio sobre esta técnica. Também não disponibilizam uma interface amigável e consistente para definição da representação do conhecimento e das métricas de similaridade, além disso, outra característica que deve ser levada em consideração na elaboração de sistemas de RBC é a integração desse sistema com diferentes tecnologias, desde a forma como são armazenadas as informações, até as interfaces para apresentação das informações aos usuários.

Sendo assim, esse trabalho abrange o desenvolvimento de um framework na qual permite conexão com diferentes tecnologias de banco de dados, e por tratar-se de uma aplicação desenvolvida em Java, pode também, comunicar-se com diferentes tecnologias de linguagem de programação.

A metodologia utilizada consistiu em pesquisa bibliográfica, levantamento de requisitos funcionais e não funcionais. Com o intuito de validar todas as funcionalidades desenvolvidas no framework, foram elaborados diversos testes distintos. O desenvolvimento deste trabalho resultou em um projeto que atendeu todas as propostas, permitindo assim, conexão com diferentes tecnologias de banco de dados, comunicação com diferentes tecnologias de programação e estruturação de sistemas RBC de qualquer natureza.

Palavras Chave: *Framework*. Inteligência Artificial. Raciocínio Baseado em Casos

ABSTRACT

AI encompasses a wide variety of subfields, from the general use area such as learning and perception, to specific tasks such as chess games, demonstration of mathematical theorems, poetry creation, and disease diagnosis.

An AI sub-area that originated from an approach to problem-solving and learning based on past experience is called Case-Based Reasoning (CBR). In a simplified way, we can understand RBC as the solution of new problems through the use of previous cases already known. A new problem that is presented to us is solved by reusing the solution of a previous problem similar to the current one.

Several articles, course papers and area books, implement, use and exemplify CBR through a framework, ie using it through an application or libraries, thus providing a generic functionality for the technique, however, currently, existing solutions for the development of CBR system require the user to have full control over this technique. They also do not provide a friendly and consistent interface for defining knowledge representation and similarity metrics, and another characteristic that must be taken into account in the elaboration of CBR systems is the integration of this system with different technologies, from the way the information is stored, up to the interfaces for presenting the information to the users.

Therefore, this work covers the development of a framework that allows connection to different database technologies, and how is a Java application, it can also communicate with different programming language technologies.

The methodology used consisted of bibliographical research, survey of functional and non-functional requirements. In order to validate all the functionalities developed in the framework, several different tests were elaborated. The development of this work resulted in a project that met all the proposals, allowing, thus, connection with different technologies of database, communication with different programming technologies and structuring of RBC systems of any nature.

Keywords: Framework. Artificial intelligence. Case-Based Reasoning

.

LISTA DE FIGURAS

Figura 1 - Ciclo RBC	31
Figura 2 - Classificação de Padrões de Projeto	40
Figura 3 - Relacionamentos entre padrões de projeto.....	42
Figura 4 - Arquitetura jColibri 2.....	49
Figura 5 - Interface MyCBR.....	52
Figura 6 - Diagrama de Casos de Uso.....	57
Figura 7 - Diagrama de Classes pacote procbrr	58
Figura 8 - Diagrama de Classes pacote procbrrConnectionBD	59
Figura 9 - Diagrama de Classes de Conexão.....	60
Figura 10 - Diagrama de Classe ClassSimilarity	64
Figura 11 - Diagrama de Classes ClassManipXML	73
Figura 12 - Interface para Conexão com Banco de Dados.....	80
Figura 13 - Interface para Configuração do SQL	81
Figura 14 - Interface para Estruturação do RBC	82
Figura 15 - Conexão com banco de dados Firebird	88
Figura 16 - Estrutura do banco de dados Firebird	89
Figura 17 - Conexão com banco de dados MySQL.....	90
Figura 18 - Estrutura do banco de dados MySQL	91
Figura 19 - Conexão com banco de dados PostgreSQL.....	92
Figura 20 - Estrutura do banco de dados PostgreSQL	93
Figura 21 - Estrutura RBC com base no SQL	94
Figura 22 - Retorno execução do framework.....	94
Figura 23 - Resultado Teste 1.....	96
Figura 24 - Resultado Teste 2.....	97
Figura 25 - Resultado Teste 3.....	98
Figura 26 - Resultado Teste 4.....	99
Figura 27 - Resultado Teste 4.....	99
Figura 28 - Resultado Teste 5.....	101
Figura 29 - Resultado Teste 5.....	101
Figura 30 - Resultado Teste 6.....	102
Figura 31 - Resultado Teste 7.....	103
Figura 32 - Resultado Teste 8.....	104
Figura 33 - Chamada do framework pelo prompt de comando	105
Figura 34 - Interface aplicação externa.....	106
Figura 35 - Teste arquivo de revisão.....	108

LISTA DE QUADROS

Quadro 1 - Requisitos Funcionais.....	55
Quadro 2 - Requisitos Não-Funcionais.....	56
Quadro 3 – Código classe ClassObjConnection.....	62
Quadro 4 - Código classe PDOConnection.....	62
Quadro 5 - Código função calcSimilarity	65
Quadro 6 - Código função calcSimilarityStringEquals.....	68
Quadro 7 - Código função calcSimilarityStringEqualsIgnoreCase	68
Quadro 8 - Código função calcSimilarityStringDistanceLevenshtein.....	69
Quadro 9 - Código função calcSimilarityStringContagemPalavras	69
Quadro 10 - Código função calcSimilarityBoolean.....	70
Quadro 11 - Código função calcSimilarityEnum	71
Quadro 12 - Código função calcSimilarityNumericEquals.....	71
Quadro 13 - Código função calcSimilarityNumericDifference.....	71
Quadro 14 - Código função calcSimilarityNumericThreshold	72
Quadro 15 - Código função saveXmlConfig.....	73
Quadro 16 - Código função loadXmlConfig	75
Quadro 17 - Código função saveXmlResult.....	77
Quadro 18 - Código função saveXmlRevision.....	78
Quadro 19 - Arquivo XML de Configuração	83
Quadro 20 - Arquivo XML de Resultados	84
Quadro 21 - Arquivo XML de Revisão	85
Quadro 22 - Exemplo chamada do framework via prompt de comando	86
Quadro 23 - SQL Firebird	89
Quadro 24 - SQL MySQL.....	91
Quadro 25 - SQL PostgreSQL.....	93
Quadro 26 - Código chamada do framework em Object Pascal.....	107
Quadro 27 - Código tratamento do XML de retorno	107

LISTA DE TABELAS

Tabela 1 - Configurações da estrutura do RBC utilizadas nos testes	95
Tabela 2 - Atributos de entrada Teste 1	95
Tabela 3 - Atributos de entrada Teste 2	96
Tabela 4 - Atributos de entrada Teste 3	97
Tabela 5 - Atributos de entrada Teste 4	98
Tabela 6 - Atributos de entrada Teste 4	99
Tabela 7 - Atributos de entrada Teste 5	100
Tabela 8 - Atributos de entrada Teste 5	100
Tabela 9 - Atributos de entrada Teste 6	102
Tabela 10 - Atributos de entrada Teste 7	103
Tabela 11 - Atributos de entrada Teste 8	104

LISTA DE SIGLAS

EUA – Estados Unidos da América

IA – Inteligência Artificial

IDE – *Integrated Development Environment*

OOSE – Engenharia de Software Orientada a Objetos

RBC – Raciocínio Baseado em Casos

RF – Requisito Funcional

RNF – Requisito Não Funcional

SQL – *Structured Query Language*

SO – Sistema Operacional

UML – *Unified Modeling Language*

URL – *Uniform Resource Locator*

XML – *Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO.....	14
1.1 PROBLEMATIZAÇÃO	15
1.1.1 Solução Proposta	16
1.1.2 Delimitação do Escopo	16
1.1.3 Justificativa.....	17
1.2 OBJETIVOS	17
1.2.1 Objetivo Geral.....	17
1.2.2 Objetivos Específicos.....	18
1.3 METODOLOGIA	18
2 FUNDAMENTAÇÃO TEÓRICA	20
2.1 RACIOCÍNIO BASEADO EM CASOS	20
2.1.1 Histórico	21
2.1.2 Estrutura de um Sistema RBC.....	22
2.1.3 Casos	24
2.1.3.1 Representação de Casos.....	24
2.1.3.2 Memória de Casos	25
2.1.3.3 Indexação dos Casos.....	26
2.1.4 Similaridade	27
2.1.4.1 Similaridade Global.....	28
2.1.4.2 Similaridades Locais	28
2.1.5 Ciclo RBC	30
2.1.5.1 Recuperação de Casos	31
2.1.5.2 Reutilização de Casos	32
2.1.5.3 Revisão	33
2.1.5.4 Retenção	34
2.1.6 Características e Vantagens de um sistema RBC	35
2.1.7 Aplicações de RBC	36
2.2 FRAMEWORKS	37
2.2.1 Padrões de Projeto	39
2.2.1.1 Padrões de Criação	43
2.2.1.1.1 <i>Abstract Factory</i>	43
2.2.1.1.2 <i>Builder</i>	44
2.2.1.2 Padrões Estruturais	44
2.2.1.2.1 <i>Adapter</i>	44
2.2.1.2.2 <i>Bridge</i>	45

2.2.1.3 Padrões Comportamentais	46
2.2.1.3.1 <i>Strategy</i>	46
2.2.1.3.2 <i>Template Method</i>	47
3 TRABALHOS CORRELATOS	48
3.1 JCOLIBRI: A MULTI-LEVEL PLATFORM FOR BUILDING AND GENERATING CBR SYSTEMS	48
3.2 FERRAMENTA PARA DESENVOLVIMENTO DE SISTEMAS DE RACIOCÍNIO BASEADO EM CASOS	49
3.3 UM FRAMEWORK DE RACIOCÍNIO BASEADO EM CASOS APLICADO PARA ESTRUTURAS A BASE DE CONHECIMENTO EM SISTEMAS TUTORES INTELIGENTES	50
3.4 THE OPEN-SOURCE TOOL MYCBR	51
3.5 FRAMEWORK PARA SISTEMA DE RACIOCÍNIO BASEADO EM CASOS	52
4 DESENVOLVIMENTO	54
4.1 ESPECIFICAÇÕES FORMAIS	54
4.1.1 Requisitos	54
4.1.1.1 Requisitos Funcionais	54
4.1.1.2 Requisitos Não-Funcionais	55
4.1.2 Diagramas de Casos de Uso	56
4.1.3 Diagrama de Classes	57
4.2 IMPLEMENTAÇÃO	59
4.2.1 Técnicas e Ferramentas	59
4.2.2 Codificação	60
4.2.2.1 Conexão com Bancos de Dados	60
4.2.2.2 Cálculo de Similaridade	64
4.2.2.3 Manipulação de XML	72
4.2.3 Interfaces do Sistema	79
4.2.3.1 Configuração de Banco de Dados	79
4.2.3.2 SQL para Consulta do Banco de Dados	80
4.2.3.3 Estrutura do RBC	81
4.2.4 Arquivos XML	82
4.2.4.1 Arquivo de Configuração	82
4.2.4.2 Arquivo de Resultados	83
4.2.4.3 Arquivo de Revisão	84
5 RESULTADOS	86
5.1 TESTES DE CONEXÃO COM DIFERENTES TECNOLOGIAS DE BANCOS DE DADOS	87
5.1.1 Teste 1 – Conexão com banco de dados Firebird	87
5.1.2 Teste 2 – Conexão com banco de dados MySQL	90
5.1.3 Teste 3 – Conexão com banco de dados PostgreSQL	92

5.1.4 Resultados testes de conexão	94
5.2 TESTES DE RESULTADOS DE CÁLCULOS DE SIMILARIDADE	94
5.2.1 Teste 1 – 100% de similaridade	95
5.2.2 Teste 2 – Distância de Levenshtein	96
5.2.3 Teste 3 – Contagem de Palavras	97
5.2.4 Teste 4 – Threshold	98
5.2.5 Teste 5 – Diferença	100
5.2.6 Teste 6 – Filmes do gênero Aventura	101
5.2.7 Teste 7 – Filmes do diretor Peter Jackson	102
5.2.8 Teste 8 – Filmes do ano de 2002	103
5.3 TESTES DE COMUNICAÇÃO COM DIFERENTES TECNOLOGIAS DE PROGRAMAÇÃO	104
5.3.1 Teste 1 – Comunicação com <i>Prompt de Comando</i>	104
5.3.2 Teste 2 – Comunicação com <i>Object Pascal Delphi XE6</i>	105
5.4 TESTES ARQUIVO DE REVISÃO	107
6 CONCLUSÃO	109
REFERÊNCIAS	111

1 INTRODUÇÃO

Denominamos nossa espécie *Homo Sapiens*¹, porque nossas capacidades mentais são muito importantes para nós. Durante milhares de anos, procuramos entender como pensamos, isto é, como um mero punhado de matéria pode perceber, compreender, prever a manipular um mundo muito maior e mais complicado que ela própria. O campo da Inteligência Artificial (IA), vai ainda mais além, ele tenta não apenas compreender, mas também construir entidades inteligentes. A IA sistematiza e automatiza tarefas intelectuais e, portanto, é potencialmente relevante para qualquer esfera da atividade intelectual humana. Nesse sentido, ela é verdadeiramente um campo universal (RUSSELL; NORVIG, 2004, p. 03).

A IA abrange uma enorme variedade de subcampos, desde área de uso geral, como aprendizado e percepção, até tarefas específicas como jogos de xadrez, demonstração de teoremas matemáticos, criação de poesia e diagnóstico de doenças (RUSSELL; NORVIG, 2004, p. 03).

Uma subárea da IA que se originou a partir de uma abordagem para a solução de problemas e para o aprendizado com base em experiência passada, chama-se Raciocínio Baseado em Casos (RBC)², esta técnica se estabeleceu nos últimos anos como uma das tecnologias mais populares e disseminadas para o desenvolvimento de Sistemas Baseados em Conhecimento (WANGENHEIM; WANGENHEIM, 2003).

RBC é uma técnica para resolução de problemas que em muitos aspectos difere de forma fundamental de outros enfoques da Inteligência Artificial (AAMODT e PLAZA, 1994). De uma forma simplificada, podemos entender o RBC como a solução de novos problemas por meio da utilização de casos anteriores já conhecidos. Um novo problema que nos é apresentado é resolvido com a reutilização da solução de um problema anterior parecido com o atual (WANGENHEIM; WANGENHEIM, 2003, p. 01).

Diversos artigos, trabalhos de conclusão de curso e livros da área, implementam, utilizam e exemplificam o RBC através de um *framework* (ERVATI e ANDRADE, 2010; MENDES, 2012; CEAUSU e DESPRÈS, 2007; ABDRABOU e SALEM, 2008), ou seja, usando-o através de uma aplicação ou bibliotecas, provendo assim, uma funcionalidade genérica para a técnica. Dois exemplos de diferentes *frameworks* RBC, são, o *myCBR*, utilizado para exemplificar o desenvolvimento de aplicações usando o RBC no livro Raciocínio Baseado

¹ Em tradução: Homem Sábio.

² Do inglês, *Case-Based Reasoning (CBR)*.

em Casos de Wangenheim e Wangenheim (2003), e a biblioteca Java *Jcolibri2* que de acordo com García *et al.* (2011) foi projetada para ser extensível e reutilizável em diferentes domínios do RBC, podendo ser importada em diferentes projetos.

Problemas comuns no desenvolvimento de sistemas ocorrem independentemente da plataforma, são problemas denominados essenciais, onde, todo software precisa ser compatível com o ambiente no qual será executado tendo em vista que os requisitos mudam constantemente, pois, conforme o software se desenvolve, a complexidade das regras de negócio e das integrações que este precisa fazer evoluem até chegar a um ponto no qual os custos de manutenção se tornam proibitivos (WEISSMANN, 2012, p. 03-04). Dentro deste contexto, *frameworks* vem sendo frequentemente usados para solucionar e diminuir a complexidade dos problemas comuns de desenvolvimento. Assim sendo, este trabalho propõe o desenvolvimento de um *framework* para a técnica de IA conhecida como RBC em formato de uma biblioteca Java, com o objetivo de auxiliar no desenvolvimento de softwares que utilizam RBC em diferentes tipos de áreas e aplicações. Para alcançar esse objetivo, o *framework* necessitará apenas de uma interação inicial do usuário para definir a estrutura do RBC, e após a biblioteca ser acoplada ao projeto, retornará as informações solicitadas a partir dos parâmetros de entrada estruturados na fase inicial.

1.1 PROBLEMATIZAÇÃO

Para o desenvolvimento de sistemas de RBC, as soluções existentes fornecem funções para elaboração do sistema, porém, obrigam os usuários a terem pleno domínio da técnica de RBC. Também não disponibilizam uma interface amigável e consistente para definição da representação do conhecimento e das métricas de similaridade, além disso, outra característica que deve ser levada em consideração na elaboração de sistemas de RBC é o uso de diferentes tecnologias, desde a forma como são armazenadas as informações (sistemas de arquivos ou banco de dados), algoritmos de recuperação e indexação das informações, métodos de adaptação, até as interfaces para apresentação das informações aos usuários (KRAUS, 2009).

Após uma análise realizada em trabalhos acadêmicos correlatos, foi possível constatar a falta de subsídio nessas características, tornando os sistemas de RBC desenvolvidos atualmente dependentes das interfaces das aplicações que deveriam ser a solução, e limitando-as devido ao baixo suporte a tecnologias de armazenamento, impossibilitando, dessa forma, uma integração com diferentes projetos ou aplicações como sistemas comerciais que, por exemplo, já possuem os dados em sua própria base de dados, outra dificuldade também, se dá

na utilização de RBC em trabalhos acadêmicos de áreas diferentes da inteligência artificial aplicada na computação, onde o autor tem a intenção de utilizar o RBC apenas como uma técnica de auxílio, sem a pretensão de obter pleno domínio sobre a técnica para atingir o objetivo do trabalho.

Com isso, tem-se como problema de pesquisa, o desenvolvimento de uma solução que facilite o desenvolvimento de um sistema RBC para, posteriormente, possibilitar a integração desse sistema a diferentes estruturas.

1.1.1 Solução Proposta

Como proposta de solução para os problemas apresentados na problematização, foi desenvolvido um *framework* para RBC chamado ProCBR que, a partir de uma interface amigável e uma documentação sucinta, irá facilitar a estruturação de um sistema RBC, podendo dessa forma, ser utilizado por usuários apenas com conhecimento básico sobre a estruturação de um sistema RBC. Essa interface gera uma biblioteca Java que permite integração com sistemas comerciais como, por exemplo, ERP's e E-Commerces, desenvolvidos em diferentes linguagens desde que as mesmas consigam comunicar-se com uma biblioteca Java e utilizem como tecnologia de armazenamento, os bancos de dados MySQL, Firebird e PostgreSQL na qual o *framework* disponibiliza conexão, também podendo ser utilizado em trabalhos acadêmicos onde o RBC é utilizado apenas como uma técnica que irá auxiliar no objetivo proposto e, dessa forma, o autor não precisará possuir pleno domínio sobre a técnica de RBC.

1.1.2 Delimitação do Escopo

No presente trabalho é descrito o desenvolvimento de um *framework* para RBC. O *framework* foi desenvolvido utilizando a linguagem de programação Java, e limita-se a criação de uma interface que serve para a definição da estrutura de um sistema RBC na área de aplicação definida pelo usuário. Posteriormente a estruturação, será gerada uma biblioteca Java que poderá ser integrada a diferentes projetos acadêmicos ou sistemas, apenas passando os parâmetros definidos na estruturação e o framework irá retornar os resultados obtidos.

A limitação de projetos na qual esse *framework* poderá ser integrado, irá depender de duas características, são elas, a linguagem de programação utilizada pelo usuário que deseja integrar esse *framework*, pois, irá necessitar que a mesma consiga comunicar-se a uma

biblioteca Java, e o Banco de Dados utilizado, pois o *framework* contempla integração apenas com os Bancos de Dados MySQL, Firebird e PostgreSQL.

Por fim, o *framework* contempla ainda, todos os ciclos de um Sistema RBC, os ciclos estão definidos na seção de Fundamentação Teórica.

1.1.3 Justificativa

Os sistemas de RBC possuem muitas características relevantes para sua utilização, porém, um problema importante que deve ser tratado é o armazenamento e o processamento necessário para recuperação dos casos na base de casos. Com isso, desenvolvedores deste tipo de sistema necessitam analisar as diferentes representações de conhecimento e métricas de similaridade para definir qual a mais adequada para cada tipo de domínio de aplicação. A má definição destes dois elementos pode trazer problemas na eficiência dos sistemas de RBC (KRAUS, 2009).

Para facilitar a criação e aplicação de um sistema RBC, os desenvolvedores podem optar pela utilização de ferramentas conhecidas como *frameworks*, que segundo (FAYAD; SCHIMIDT; RALPH, 1999, p. 729), consistem de uma aplicação de software semicompleta e reutilizável que pode ser especializada para produzir aplicações customizadas. *Frameworks* que implementam a técnica de RBC, fornecem funções para elaboração de sistemas que utilizam esta técnica, porém, conforme retratado na seção de Problemática, ferramentas disponíveis atualmente ainda possuem falhas em características consideradas fundamentais para que as mesmas possam ser consideradas soluções consistentes para o desenvolvimento de um sistema RBC.

É neste contexto que a ideia deste trabalho foi desenvolvida, sabendo-se da necessidade de uma solução para os problemas encontrados nas ferramentas atuais, busca-se contribuir com o desenvolvimento de um *framework*, facilitando a criação e aplicação de um sistema RBC.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Desenvolvimento de um *framework* para a criação de sistemas de Raciocínio Baseado em Casos.

1.2.2 Objetivos Específicos

- a) Definição de uma estrutura que contemple a criação de sistemas RBC distintos a partir de uma interface amigável.
- b) Desenvolvimento do framework *ProCBR*.
- c) Realização de testes de integração utilizando sistemas com diferentes linguagens e tecnologias de banco de dados.

1.3 METODOLOGIA

O trabalho foi realizado através das seguintes etapas:

1 – Efetuada revisão bibliográfica em fontes como artigos publicados, livros, monografias e dissertações sobre a utilização de *frameworks* para a criação de sistemas RBC, para que desta forma, sejam identificadas e analisadas as ferramentas de desenvolvimento de sistemas de RBC existentes, visando identificar suas principais características e deficiências para que a solução proposta consiga resolver essas deficiências.

2 – Análise do funcionamento da técnica de RBC: Foram estudados todos os ciclos da técnica, que são: - Recuperação, Reuso, Revisão e Retenção de Casos, e identificado ainda, as formas de representação de casos, métricas de similaridade, os métodos de adaptação e as demais definições pertinentes para elaboração deste tipo de sistema.

3 – Definição de uma estratégia de desenvolvimento: Analisada a melhor forma de desenvolvimento do *framework*, a fim de obter uma real generalização de linguagens e tecnologias de banco de dados que possam ser utilizados pela aplicação do sistema RBC desenvolvido através deste *framework*, esta etapa visa concluir o primeiro objetivo.

4 – Modelagem: Avaliação dos requisitos, desde as tecnologias utilizadas até as características finais, visando modelar o sistema através de diagramas da linguagem padrão para a elaboração da estrutura de projetos de software conhecida como UML, que são:

- Diagrama de Casos de Uso, que representa os requisitos e funcionalidades.
- Diagrama de Classes, onde será especificada todas as regras de negócio, classes, funções e variáveis necessárias para persistência da aplicação.

Esta etapa tem como objetivo atingir ao terceiro objetivo específico do trabalho.

5 – Desenvolvimento: Implementação do *framework* através da linguagem de programação Java, finalizando o terceiro objetivo.

6 – Testes do sistema: A eficácia do *framework* foi confirmada através de testes integrando o *framework* a sistemas desenvolvidos com diferentes linguagens de programação e que utilizam diferentes tecnologias de banco de dados, finalizando assim, o último objetivo do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são abordados conceitos relacionados à *Framework's* e a técnica Raciocínio Baseado em Casos, os quais são fundamentais para a realização e clareza deste trabalho.

No que diz respeito à técnica de Inteligência Artificial Raciocínio Baseado em Casos, serão abordados seus fundamentos, histórico, estrutura, ciclos e técnicas, características, vantagens, e suas respectivas aplicações, necessários para o conhecimento e posteriormente aplicação da técnica.

Na fundamentação teórica sobre *framework's*, serão abordados conceitos e características de um *framework*, a necessidade de utilização de padrões de projeto no desenvolvimento de um *framework*, e será abordado ainda, as técnicas de padrões de projeto escolhidas para o desenvolvimento do *framework* proposto.

2.1 RACIOCÍNIO BASEADO EM CASOS

A IA é um dos campos mais recentes em ciências e engenharia. O trabalho começou logo após a Segunda Guerra Mundial, e o próprio nome foi cunhado em 1956 (RUSSELL; NORVIG, 2004, p. 03).

Atualmente, a IA abrange uma enorme variedade de subcampos, do geral (aprendizagem e percepção) até tarefas específicas, como jogos de xadrez, demonstração de teoremas matemáticos, criação de poesia, direção de um carro em estrada movimentada e diagnóstico de doenças. A IA é relevante para qualquer tarefa intelectual, é verdadeiramente um campo universal (RUSSELL; NORVIG, 2004, p. 03).

O Raciocínio Baseado em Casos, segundo Wangenheim e Wangenheim (2003), é uma subárea da Inteligência Artificial (IA) que se originou a partir de uma abordagem para a solução de problemas e para o aprendizado com base em experiência passada.

Com seu enfoque na utilização de experiências, o RBC diferencia-se radicalmente de outras metodologias para o desenvolvimento de programas e sistemas da área da IA. Ao contrário de enfoques tradicionais para encontrar soluções para um problema em IA, em que se descreve conhecimento genérico da forma de regras, quadros, roteiros etc., no RBC é o conhecimento específico, na forma de exemplos concretos de casos, que se encontra no centro do processo de solução de um problema (WANGENHEIM; WANGENHEIM, 2003).

Para Carvalho (1996), ao tentar compreender o que está vendo e ouvindo, o ser humano busca em sua memória algo que possa ajudá-lo nesta compreensão, ou seja, ele se recorda de algo que já foi compreendido no passado que lhe é útil para compreender a situação atual. O RBC age de forma semelhante, visando usar os resultados dos casos passados para analisar ou resolver um novo caso. Os problemas a serem resolvidos tendem a ser recorrentes e repetir-se com pequenas alterações em relação a sua versão original. Desta forma, é possível reaplicar soluções anteriores com pequenas modificações (VARELA, 1998).

Conforme Lee (1998), RBC é uma técnica de IA que reproduz aspectos da cognição humana para resolver problemas especialistas. Os sistemas RBC simulam o ato humano de relembrar um episódio prévio para resolver um determinado problema em função da identificação de afinidades entre os mesmos.

De uma forma simplificada, podemos entender o RBC como a solução de novos problemas por meio da utilização de casos anteriores já conhecidos, sendo aplicável de forma simples e direta a um espectro extremamente grande de problemas e situações reais, dessa forma, o RBC se estabeleceu como uma das tecnologias mais populares e disseminadas para o desenvolvimento de Sistemas Baseados em Conhecimento (WANGENHEIM; WANGENHEIM, 2003).

2.1.1 Histórico

De acordo com Watson (1995), a história do RBC começa com a investigação do filósofo Wittgentein em 1953, que posteriormente, leva ao trabalho de Schank e Abelson em Memória Dinâmica e no modelo cognitivo de uma função central de lembrança de situações passadas (casos e memória episódica) e de padrões de situações no começo da década de 80. De acordo com diversos autores (ABEL, 1996; WANGENHEIM E WANGENHEIM, 2003; VITORINO, 2009), esses trabalhos de Schank e Abelson inspiraram o desenvolvimento dos primeiros sistemas RBC.

Schank e Abelson estudaram sobre programas que fossem capazes de compreender o que lessem. Nesses estudos, concluíram que a compreensão da linguagem está diretamente relacionada com as informações em memória. Desenvolveram, então, a teoria da Memória Dinâmica, que foi uma importante contribuição para os estudos na área de RBC. Esta teoria propôs o conceito de pacotes de organização de memória que utilizam a lembrança em experiências passadas associadas a estereótipos de situações para a resolução de problemas e aprendizado (CARVALHO, 1996). Schank e Abelson propuseram ainda, que nosso

conhecimento geral acerca de situações fica gravado na memória como roteiros que permitem que nós construamos expectativas sobre resultados esperados de ações que planejamos e que façamos inferências sobre relacionamentos causais entre ações (WANGENHEIM; WANGENHEIM, 2003).

Segundo Wangenheim e Wangenheim (2003), roteiros foram propostos já no início da década de 80, como uma estrutura de dados para a memória conceitual, descrevendo informação a respeito de eventos estereotipados, como, por exemplo, uma típica ida a um restaurante ou uma típica consulta com um médico, onde são executadas ações previsíveis e esperadas.

Um roteiro auxilia na análise de eventos ou de atividades por meio da previsão das ações específicas que tipicamente serão executadas em determinadas situações. Isto significa, do ponto de vista do processo de reconstrução de eventos, que, em princípio, pode-se partir da premissa de que determinadas coisas ocorrerão sempre conforme o esperado, como acontece com o roteiro de uma peça teatral, em que os atores executam as ações previstas da forma como estão determinadas (WANGENHEIM; WANGENHEIM, 2003).

RBC baseia-se também no estudo da analogia, principalmente no trabalho de Gick e Holyoak, no início dos anos 80. O objetivo do raciocínio analógico é a transformação e extensão do conhecimento proveniente de um domínio conhecido, como, por exemplo, a observação de peixes na lagoa, para dentro de outro domínio com estrutura ainda incompletamente compreendida, como, por exemplo, engenharia naval. No RBC, são resolvidos exclusivamente problemas no âmbito de um mesmo domínio de aplicação (WANGENHEIM; WANGENHEIM, 2003).

O primeiro sistema RBC chama-se CYRUS e foi desenvolvido em 1983 por Janet Kolodner, uma pesquisadora do grupo de Schank. O sistema continha viagens e encontros do ex-secretário de estado dos EUA, Cyrus Vance, descritos na forma de casos e implementado com os modelos de Schank. O modelo de casos do sistema CYRUS, serviu como base para o grupo da universidade de Yale desenvolver diversos outros sistemas de RBC. Esses trabalhos e conceitos evoluíram rapidamente para inúmeras aplicações de sistemas baseados em casos, especialmente nos domínios de direito, medicina e engenharia (ABEL, 1996).

2.1.2 Estrutura de um Sistema RBC

A construção de um sistema RBC, começa pela identificação de índices ou características que representam o problema, após isto, é feita a seleção de um caso que seja

similar a este problema, e finalmente, é feita a adaptação da solução do caso escolhido para que ela seja adequada as necessidades do novo problema. Quando a solução encontrada não é aceita, pode existir a recuperação de casos (HEINRICH, 2000).

Com essa definição, a estrutura de um sistema RBC é composta por três itens principais, citados a seguir:

- a) Memória de casos de domínio;
- b) Mecanismos de pesquisa;
- c) Descrição dos casos com índices para diferenciar os casos.

Segundo Lee (1998), descreve-se o problema através da atribuição de características que descrevem o caso de entrada. As características descritivas podem ter a forma de nomes, números, funções ou textos, e servem para representar características, objetivos, metas, restrições e/ou condições. Servem ainda, para identificar o caso e são estas características que determinam a similaridade com outro caso.

Pode-se descrever a solução através de uma metodologia, que deve descrever a forma com que a solução foi implementada e deve ser acompanhada de uma justificativa para a escolha desta opção (LEE, 1998).

Segundo Vitorino (2009), o funcionamento de um sistema RBC, possui as seguintes características:

- a) A extração do conhecimento a partir de casos ou experiências com que o próprio sistema se depara;
- b) A identificação das características mais significantes dos casos apresentados afim de devolver uma melhor solução (resposta);
- c) O armazenamento do caso e sua respectiva solução.

Sendo assim, o RBC é um enfoque para a solução de problemas e para o aprendizado em experiência passada. RBC resolve problemas ao recuperar e adaptar experiências passadas chamadas casos armazenadas em uma base de casos. Um novo problema é resolvido com base na adaptação de soluções de problemas similares já conhecidos (WANGENHEIM; WANGENHEIM, 2003).

2.1.3 Casos

O conhecimento em um sistema de RBC é principalmente armazenado sob a forma de casos. Um caso é uma peça de conhecimento contextualizado representando uma experiência ou episódio concreto, contém a lição passada, que é o conteúdo do caso e o contexto em que a lição pode ser usada. Casos podem ser encontrados sob muitas formas e tamanhos. Um caso pode, por exemplo, ter diferentes conteúdos e representações dependendo da área de aplicação. (WANGENHEIM; WANGENHEIM, 2003).

De acordo com Weber (1996), a representação dos casos em um sistema RBC, é essencialmente a representação do conhecimento. Há outros momentos em que algum conhecimento especialista é representado no sistema de RBC, entretanto, é nos casos que está representado o conhecimento que servirá para sugerir uma solução para o problema de entrada no sistema.

O conteúdo exato que cada caso deve ter depende do domínio de aplicação específico. Como exemplos, defeitos de impressoras, doenças tropicais, vendas de imóveis, etc. e o objetivo do raciocínio, como exemplos, diagnóstico, planejamento ou atendimento a cliente (WATSON, 1997). Entretanto, de acordo com Watson (1997), duas medidas podem ser tomadas para decidir o que deve ser representado em casos, a funcionalidade da informação e a facilidade de aquisição da informação representada no caso.

2.1.3.1 Representação de Casos

Segundo Lee (1998), determinar o que é um caso é o primeiro problema na modelagem do RBC, pois, são os casos que contém elementos para que a solução para o problema proposto seja alcançada.

Ao desenvolver um sistema utilizando RBC, é necessário estipular como a memória de casos será organizada e indexada para a recuperação efetiva de um novo caso de forma eficiente (AAMODT e PLAZA, 1994).

Casos podem ser representados em uma variedade de formas usando praticamente todos os formalismos de representação de conhecimento da IA incluindo, frames, objetos, predicados, redes semânticas, regras de produção, redes neurais, além de estruturas menos ricas semanticamente como os modelos de dados dos bancos de dados comerciais. O formato mais utilizado é o de frames ou objetos, embora muitos sistemas usem mais de uma das formas acima

combinadas. Além desse formato, faz parte das decisões de representação definir como os casos serão organizados e ligados entre si, compondo a memória de casos (ABEL, 1996).

2.1.3.2 Memória de Casos

A memória de casos é a principal fonte de conhecimento do modelo RBC. Esta memória é formada pelas experiências na resolução de problemas resolvidos pelos especialistas, e cada experiência representa um caso (CARVALHO, 1996).

Quanto maior o número de casos de sucesso armazenados e eficientemente indexados, maior será a chance de que um novo caso possa ser tratado com a mesma solução ou com uma pequena adaptação de uma solução já utilizada. O primeiro passo para utilizar uma solução já aplicada com sucesso anteriormente é determinar qual das experiências passadas mais se assemelha ao problema atual. Para ser possível realizar esta comparação é necessário que as experiências sejam analisadas e armazenadas de forma organizada (ABEL, 1996).

Segundo Abel (1996) dois modelos de memória de casos influenciaram historicamente o modo como os casos são organizados. São o modelo de memória dinâmica de Schank (1982) e o modelo de categoria de exemplares de Porter e Bareiss (1986).

O modelo de memória dinâmica de Schank, é composto principalmente de pacotes de organização de memória (MOPs), que são frames que compõem uma unidade básica da memória dinâmica. Eles representam o conhecimento sobre classes de eventos de duas formas:

- a) Instâncias, que representam casos, eventos ou objetos;
- b) Abstrações, que representam versões generalizadas de instâncias ou de outras abstrações.

O desenvolvimento da teoria mais geral de Schank levou aos pacotes de organização de memória episódicos (E-MOPs), implementados no sistema CYRUS em 1983. A ideia básica é organizar casos específicos que partilham propriedades similares sob uma estrutura mais geral. Uma E-MOP contém os casos, as propriedades comuns entre eles e as feições que os diferenciam. O modelo é dinâmico porque novas E-MOPs são criadas no momento da inserção de novos casos, para discriminá-los em relação aos anteriormente armazenados (ABEL, 1996).

No modelo de categoria de exemplares, os casos no mundo real podem ser vistos como exemplares de acontecimentos. Cada caso é associado a uma categoria e suas feições tem

importância distinta para enquadrá-lo ou não na categoria. Para armazenar um novo caso, é buscado um caso semelhante no banco de casos, se houver pequenas diferenças entre os dois, apenas um deles é retido, ou é armazenada uma única combinação dos dois (ABEL, 1996).

2.1.3.3 Indexação dos Casos

A indexação de casos é uma das tarefas mais importantes em um sistema RBC, pois, a correta recuperação de um caso é o fator chave para a credibilidade de uma aplicação em RBC. Segundo Carvalho (1996), a indexação é encarada como um problema que se resume em escolher que características servirão como índice para os casos colocados na memória, de forma que estes possam ser recuperados quando necessários.

Indexar casos corresponde a atribuir índices aos casos de forma a facilitar sua recuperação. Isso inclui colocar rótulos nos casos, no momento de sua inclusão na base de casos, para que possam ser posteriormente recuperados, organizar os casos para facilitar a busca e recuperação e definir os algoritmos de recuperação mais eficientes (ABEL, 1996).

Índices devem ser escolhidos cuidadosamente. Características superficiais são facilmente retiradas de um caso, mas estas características podem ser menos úteis do que os índices mais complexos obtidos pela combinação e composição de características que distinguem casos de cada lição que ele pode ensinar. A escolha de bons índices pode requerer uma interpretação ou processo de elaboração durante o qual características funcionais podem ser derivadas. (LURREGI, 1999, apud BECKER, 2002, p. 11).

Indexar casos depende da compreensão do conteúdo e finalidade da informação que eles armazenam. Um bom índice permite reconhecer similaridades úteis entre os casos recuperados e essa utilidade só pode ser percebida se os índices forem escolhidos com base em uma boa compreensão do problema (ABEL, 1996).

Kolodner (1993, apud ABEL, 1996, p. 22), aponta como as qualidades necessárias a um bom índice:

- a) Prever a futura utilização da informação para solução de diferentes problemas;
- b) Endereçar as similaridades úteis entre os casos;
- c) Ser abstrato o suficiente para tornar o caso útil em uma variedade de diferentes situações;
- d) Ser concreto o suficiente para ser facilmente reconhecido em futuras situações.

Os índices podem ser selecionados tanto manualmente como automaticamente. A seleção manual analisa caso a caso para determinar quais as características descritas que determinam as variações sobre as conclusões. Os métodos automáticos buscam quantificar as diferenças entre os casos e os relacionamentos entre feições do problema e soluções adotadas (ABEL, 1996).

Conforme Carvalho (1996), a seleção manual é vantajosa quando os casos são bastante complexos e há o conhecimento necessário para a compreensão do caso. Já a seleção via computador demonstra ser mais útil quando a solução e a compreensão do problema já são automatizados.

2.1.4 Similaridade

De acordo com Lee (1998), similaridade é a essência do Raciocínio Baseado em Casos, uma vez que o fundamento do paradigma de RBC, é solucionar um problema atual reutilizando a solução de uma experiência passada similar.

Uma das hipóteses básicas de sistemas de RBC é que problemas similares possuem soluções similares. Esta forma de se proceder é justificada pela premissa de que, em muitas aplicações, a similaridade de definições de problemas implica a aplicabilidade da solução de um sobre outro (WANGENHEIM; WANGENHEIM, 2003).

Uma das maneiras de se fazer a aquisição do conhecimento com objetivo de saber o peso dos índices, é solicitar que o especialista faça uma lista em ordem de importância. A similaridade é o ponto crucial de RBC, pois a partir desta etapa, todo o processo de raciocínio que fundamenta esta técnica torna-se viável (LEE, 1998).

De acordo com Wangenheim e Wangenheim (2003), a utilidade de um caso é medida pela similaridade, ou seja, o caso mais útil para a solução de um problema, é o que necessita do menor número de modificações para adaptá-lo ao problema atual. Atrás desta afirmativa não se encontra apenas a meta de uma solução de problemas eficiente, mas ela reflete também o fato de que, quanto menos se modificar a descrição de problema de um caso, menor é a incerteza da utilidade de sua solução para o problema atual.

De acordo com Burkhard (1998, apud Wangenheim e Wangenheim, 2003), podemos afirmar que, para a determinação da similaridade em um sistema de RBC, as seguintes premissas têm de ser satisfeitas:

- a) A similaridade entre a questão atual e o caso implica utilidade;

- b) A similaridade é baseada em fatos *a priori*³;
- c) Como casos podem ser mais ou menos úteis em relação a uma questão, a similaridade precisa prover uma medida.

2.1.4.1 Similaridade Global

De acordo com Wangenheim e Wangenheim (2003), uma medida de similaridade frequentemente utilizada é a técnica chamada “vizinho mais próximo”, bastante simples para a determinação de similaridade entre dois casos.

Conforme Abel (1996), nesta técnica de recuperação utiliza-se uma soma ponderada das características entre um novo caso e um armazenado no banco de dados, sendo que cada um dos atributos que compõem o caso possui um peso, de acordo com sua relevância. Abaixo listamos a função de cálculo do vizinho mais próximo segundo Watson (1995):

$$Similaridade(T, S) = \sum_{i=1}^n f(T_i, S_i) * W_i$$

Onde:

- a) T é o caso de entrada;
- b) S é o caso da base;
- c) n é o número de atributos de cada uso;
- d) i é um atributo individual;
- e) f é a função de similaridade para o atributo i nos casos T e S;
- f) W é o peso dado ao atributo i.

A maioria das ferramentas RBC utilizam algoritmos como este. Normalmente o resultado deve ser entre zero (0) e um (1), onde zero não tem nenhuma similaridade e um é exatamente similar.

2.1.4.2 Similaridades Locais

³ Relativo a ou que resulta de raciocínio cujas definições foram dadas inicialmente.

De acordo com Wangenheim e Wangenheim (2003), para o exame compreensivo da similaridade entre uma questão dada e os casos na base de casos, a similaridade local entre atributos específicos pode ser considerada ao se computar a similaridade global. Como consequência, um caso com valores de atributo diferentes, mas que podem ainda ser similares ao procurado, não é passível de ser distinguido de outro, cujos valores são completamente diferentes. A consideração de similaridades locais permite a integração das similaridades entre atributos isolados ao cálculo da similaridade global, tornando-a muito mais sensitiva.

Estas medidas de similaridade local devem ser definidas em relação ao tipo específico de um atributo, por exemplo, haverá medidas de similaridade local diferentes para valores numéricos e valores simbólicos. Além disso, a medida de similaridade local tem de ser definida no contexto de aplicação específico, pois a similaridade entre dois valores pode variar entre diferentes sistemas de RBC (WANGENHEIM; WANGENHEIM, 2003).

Na implementação deste trabalho, foram utilizados quatro tipos diferentes de atributos, sendo eles, atributos textuais (caracteres e palavras), numérico, booleanos e valores *Enum*, onde os valores já estão pré-definidos no banco de dados. Todos esses tipos possuem o cálculo de similaridade de igualdade, onde o caso retorna 1, ou seja, totalmente similar apenas em casos onde os atributos comparados são totalmente iguais, caso contrário, irá retornar 0, ou seja, nada similar. Para os tipos textuais, existe ainda a opção de comparar ignorando letras maiúsculas ou minúsculas.

Além das comparações de igualdade, neste projeto foram utilizadas também, métricas que necessitam de algum tipo de cálculo, que serão demonstrados a seguir.

Para atributos numéricos de acordo com Wangenheim e Wangenheim (2003) foram utilizadas as seguintes medidas:

- Função degrau (*Threshold*): É utilizada quando um atributo de um caso é totalmente útil ou totalmente inútil somente com relação ao valor da consulta. A função degrau calcula uma similaridade de 1 se a distância entre dois valores for menor que o limiar definido para a função, senão, a função retorna similaridade 0.
- Diferença entre valores: Atributos podem ser descritos por números e o módulo da diferença destes valores pode ser considerado uma medida de similaridade.

Para atributos textuais de acordo com Wangenheim e Wangenheim (2003); Heeringa (2004), foram utilizadas as seguintes medidas:

- Distância de Levenshtein: É dada pelo número mínimo de operações necessárias para transformar um texto em outro. As operações são inserção, remoção ou substituição de um caracter.
- Contagem de Palavras: Contando o número de palavras idênticas entre dois casos.

2.1.5 Ciclo RBC

De acordo com Vitorino (2009) o ciclo de funcionamento de um sistema de RBC é composto por quatro etapas de execução, conhecida como 4R's, conforme definido por Aamodt & Plaza (1994), explicados abaixo e ilustrado pela Figura 1 posteriormente:

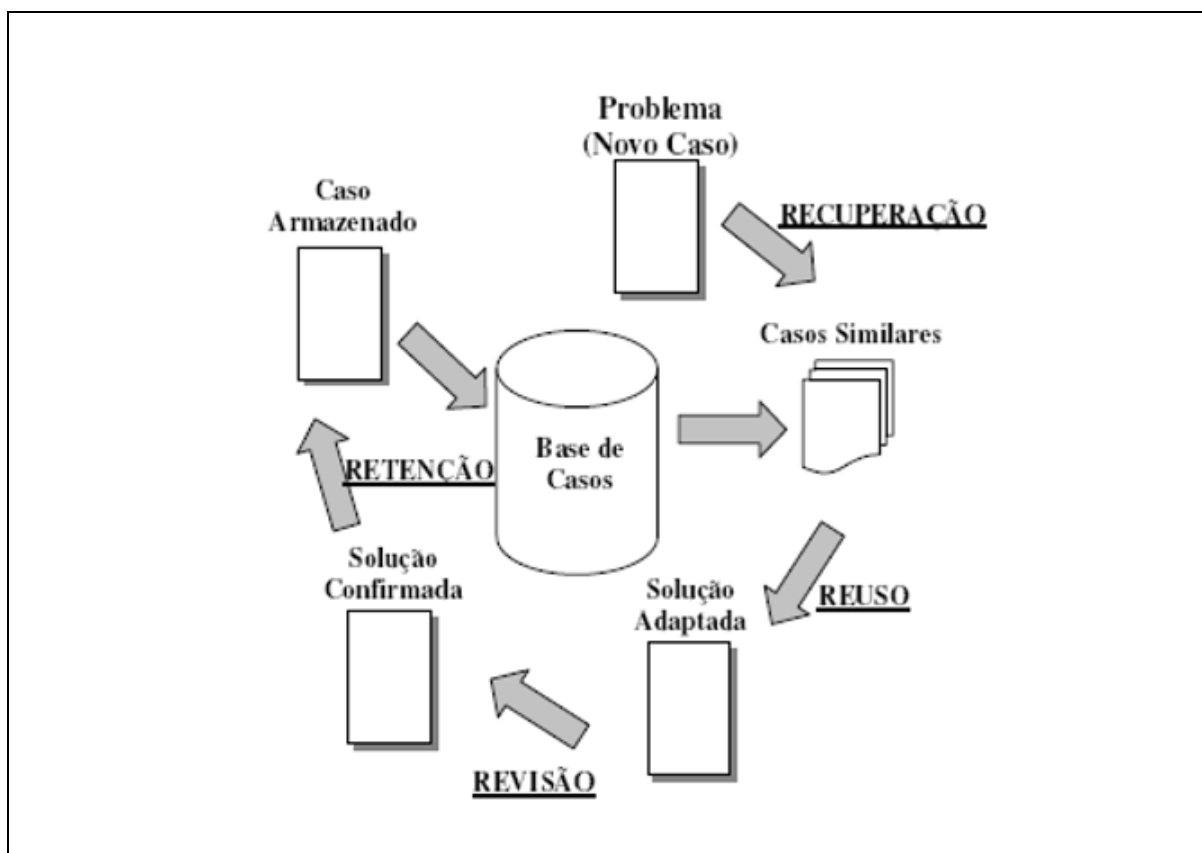
Recuperação: A partir da apresentação ao sistema de um novo problema é feita a recuperação na base de casos daquele mais parecido com o problema em questão. Isto é feito a partir da identificação das características mais significantes em comum entre os casos.

Reuso: A partir do caso recuperado é feita a reutilização da solução associada aquele caso. Geralmente a solução do caso recuperado é transferida ao novo problema diretamente como sua solução.

Revisão: É feita quando a solução não pode ser aplicada diretamente ao novo problema. O sistema avalia as diferenças entre os problemas (o novo e o recuperado), quais as partes do caso recuperado são semelhantes ao novo caso e podem ser transferidas adaptando assim a solução do caso recuperado da base à solução do novo caso.

Retenção: É o processo de armazenar o novo caso e sua respectiva solução para futuras recuperações. O sistema irá decidir qual informação armazenar e de que forma.

Figura 1 - Ciclo RBC



Fonte: Vitorino, 2009, p. 20.

2.1.5.1 Recuperação de Casos

O objetivo da recuperação de casos é encontrar um caso ou um pequeno conjunto de casos na base de casos que contenha uma solução útil para o problema ou situação atual. Para realizar essa recuperação, é necessário casar a descrição do problema atual com os problemas armazenados na base de casos (WANGENHEIM; WANGENHEIM, 2003).

Para julgar qual o caso armazenado na base é similar ou igual ao novo problema, é preciso medir a similaridade (semelhança) entre eles, ou seja, realizar uma indexação dos casos. Desta forma, a definição dos métodos de recuperação de casos está fortemente conectada a verificação da similaridade entre os casos contidos na base e os novos problemas (VITORINO, 2009).

De acordo com Wangenheim e Wangenheim (2003), o processo de recuperação de casos pode ser formalmente descrito por meio de um conjunto de subtarefas que devem ser realizadas pelo sistema de RBC:

- a) **Assessoramento da situação:** Objetivando a formulação de uma consulta representada por um conjunto de descritores relevantes da situação ou problemas atuais;
- b) **Casamento:** Objetivando a identificação de um conjunto de casos suficientemente similares à consulta;
- c) **Seleção:** Que escolhe o melhor casamento ou casamentos com base no conjunto de casos selecionado.

Ainda segundo Wangenheim e Wangenheim (2003), para a recuperação de casos, uma medida de similaridade frequentemente utilizada é a técnica do vizinho mais próximo.

Conforme fundamentado na seção de similaridade, a técnica de vizinho mais próximo será utilizada para a recuperação de casos neste trabalho.

2.1.5.2 Reutilização de Casos

O objetivo da reutilização de casos, é adaptar a solução associada a um caso recuperado para as necessidades do problema corrente. Normalmente o caso selecionado não casa perfeitamente com a descrição do problema do usuário. Existem diferenças entre o problema do usuário e o caso contido na memória de casos que devem ser levadas em conta. O processo de adaptação procura por diferenças salientes entre as duas descrições e aplica regras de forma a compensá-las (ABEL, 1996).

A reutilização consiste principalmente da adaptação da solução do caso anterior ao caso atual (WANGENHEIM; WANGENHEIM, 2003). Segundo Abel (1996), existem dois tipos de adaptação:

- a) **Adaptação Estrutural:** As regras de adaptação são aplicadas sobre a solução armazenada junto aos casos. É o mecanismo possível de ser utilizado quando as soluções associadas aos casos não são bem compreendidas.
- b) **Adaptação Derivacional:** O algoritmo reusa os algoritmos, métodos ou regras que geraram a solução que consta no banco de casos para gerar uma nova solução para o problema corrente. Neste método a sequência que contruiu a solução original deve ser armazenada juntamente com o caso. Esta adaptação somente poderá ser utilizada para domínios que são bem compreendidos.

Ainda de acordo com Abel (1996), definir a melhor forma de adaptação para alguns problemas pode ser uma tarefa extremamente complexa que pode comprometer a confiabilidade do sistema. Em muitas aplicações, a solução é simplesmente apresentar o melhor caso e deixar o problema de adaptação para o usuário do sistema.

Segundo Wangenheim e Wangenheim (2003), a necessidade de recuperar um caso, acontece uma vez que o caso recuperado da base de casos pode não satisfazer completamente os requisitos dados pela nova situação descrita no caso recuperado antes de aplicá-la ao caso atual. Na maioria das circunstâncias e domínios de aplicação de RBC, no entanto, geralmente será suficiente que se copie a solução do caso encontrado para o caso atual e se aplique esta solução, ou então que se adapte o caso manualmente.

Uma estratégia frequentemente utilizada para construir sistemas de RBC é a de se inflar a base de casos ao máximo, para garantir que todo problema possível possua na base um caso cuja solução seja suficientemente similar, de maneira a não necessitar ser adaptada (WANGENHEIM; WANGENHEIM, 2003).

2.1.5.3 Revisão

Segundo Wangenheim e Wangenheim (2003), a etapa de revisão ocorre quando uma solução para um caso gerada na fase de reuso não é correta, desta forma, surge uma oportunidade para o aprendizado a partir desta falha.

Caso a solução proposta não venha produzir um resultado satisfatório, então esta solução deve ser reparada para que uma nova solução seja gerada. Após encontrar a solução correta para o caso, a experiência obtida deve ser aprendida, sendo armazenada para uso futuro. Estes processos podem ser vistos como obtenção da experiência e os processos de recuperação e adaptação podem ser vistos como a aplicação da experiência adquirida (KRAUS, 2009).

De acordo com Wangenheim e Wangenheim (2003), a revisão consiste de duas tarefas:

1. Avaliar criteriosamente a solução gerada pelo reuso. Se for considerada como correta, aprenda com o sucesso e continue com a retenção.
2. Caso contrário, reparar a solução para o caso, utilizando conhecimento específico sobre o domínio de aplicação ou informações fornecidas pelo usuário.

O primeiro passo da revisão, trata-se de avaliar a solução fornecida pelo sistema de RBC. A tarefa de avaliação toma o resultado da aplicação da solução no ambiente de aplicação real por meio de monitoração automática de resultados, ou pela interação com o usuário. Este é um passo executado externamente ao sistema de RBC, uma vez que implica a interação com o mundo real (WANGENHEIM; WANGENHEIM, 2003).

O segundo passo da fase de revisão focaliza na eliminação de falhas, ou seja, detectar quais as partes da solução proposta contêm falhas e a recuperação ou geração de explicações para estas falhas. Quando um sistema inteligente necessita explicar uma falha, ele em geral já resolveu o problema, tentou aplicar a solução ao mundo real e obteve algum tipo de *feedback* contando-lhe a respeito da falha. As explicações das falhas são então utilizadas para modificar a solução ou a forma como o sistema chegou à solução, de maneira que o caso presente possa ser resolvido com sucesso e a falha não torne a ocorrer (WANGENHEIM; WANGENHEIM, 2003).

2.1.5.4 Retenção

A retenção do caso é o processo de reter o que é útil do problema resolvido. O aprendizado do sucesso ou das falhas da solução proposta é efetuado depois da avaliação e possíveis reparos. Isto envolve selecionar que informação do caso deve-se reter, a forma de retê-las, como indexar o caso para posterior recuperação de problemas similares e como integrar o novo caso na estrutura da memória (SILVA, 2002).

De acordo com Wangenheim e Wangenheim (2003), podemos distinguir basicamente três tipos de retenção em sistemas de RBC:

1. **Sem retenção de casos:** Os sistemas de RBC mais simples, desconsideram a inclusão automática de novo conhecimento na base de casos. Este tipo de sistema é aplicado principalmente em domínios de aplicação bem compreendidos, que podem ser modelados de forma satisfatória já durante o desenvolvimento do sistema de RBC, fazendo desnecessária a inclusão de novos casos para que o refinamento da performance do sistema ocorra.
2. **Retenção de solução de problemas:** Uma forma de aprendizado típica de RBC é aquela integrada ao processo de solução de novos problemas. Assim que um novo problema é resolvido, a experiência é retida de forma a auxiliar na solução de problemas similares no futuro.

3. **Retenção de documentos:** Este processo é praticado em sistemas de RBC em que o novo conhecimento é adquirido de forma assíncrona ao processo de solução de problemas, ou seja, é adquirido de forma independente da operação do sistema. A fase de retenção de casos é separada do processo de solução de problemas, estando a retenção dependente da disponibilidade de novos conhecimentos sobre o domínio da aplicação, que são adicionados quando disponíveis.

Segundo Abel (1996), os casos registram experiências concretas que podem auxiliar a alcançar um determinado objetivo, porém, nem todos os casos devem ser selecionados para serem incluídos no sistema. Apenas aqueles que contêm uma lição útil em relação aos demais devem ser armazenados, ou seja, os casos que ampliam a capacidade de solução de problema do sistema.

2.1.6 Características e Vantagens de um sistema RBC

A técnica de Raciocínio Baseado em Casos diferencia-se das outras técnicas de IA por uma série de características.

Segundo Vitorino (2009), as principais características do uso de RBC como suporte a implementação de ambientes de aprendizagem são:

- a) Não requer uma modelagem explícita do domínio;
- b) Sua implementação é reduzida a identificar as características significantes que descrevem um caso;
- c) Grandes volumes de informação podem ser gerenciados;
- d) A atualização do conhecimento pode ser feita automaticamente, na medida que as experiências são utilizadas, assim, o sistema pode crescer e incrementar sua robustez e eficiência;
- e) As justificativas são sempre consistentes com as soluções por serem as próprias experiências, representando mais um aspecto de proximidade ao comportamento humano do paradigma;
- f) Não é necessário que o sistema entenda o processo RBC de forma com que os usuários sejam os executores do ciclo, com o auxílio de uma ferramenta computacional adequada.

Segundo Abel (2002 apud Silva, 2002), tais características resultam em vantagens da técnica de RBC sobre outros métodos equivalentes de resolução de problemas. Abaixo são listadas algumas vantagens:

- a) Fornecem uma amostragem significativa do tipo de problemas que o sistema deve resolver;
- b) Facilita a aquisição de conhecimento, especialmente em domínios pouco estruturados e muito complexos, mesmo antes de uma perfeita compreensão do domínio;
- c) Permitem o reuso de conhecimento armazenado em bancos de dados e outras fontes;
- d) Permite o encapsulamento da descrição do conhecimento com a solução aplicada;
- e) Realizam a manutenção do banco de conhecimento por aprendizagem automática de novos casos com pouca ou nenhuma interferência de um engenheiro de conhecimento;
- f) A reutilização de soluções implementadas geralmente exige menos processamento do que a geração de uma solução através de um modelo.

2.1.7 Aplicações de RBC

Segundo Vitorino (2009), RBC deve ser aplicado quando:

- a) Especialistas falam sobre o seu domínio dando exemplos;
- b) Experiência tem o mesmo valor que conhecimentos em livros;
- c) Problemas não são completamente entendidos (modelos ruins, pouca disponibilidade de conhecimento do domínio);
- d) Existem muitas exceções para as regras;
- e) Consideração de conhecimento incompleto;
- f) Conhecimento idêntico a situação atual não existe.

De acordo com Abel (2002 apud Silva, 2002), as áreas potenciais para aplicação de sistemas de RBC podem ser classificadas em duas grandes classes: tarefas de classificação e tarefas de síntese.

As tarefas de classificação buscam verificar qual a classe do problema em questão para reutilizar uma solução já aplicada, incluem:

- a) Diagnósticos de falhas em equipamentos ou diagnósticos de doenças;
- b) Previsão de falhas em equipamentos ou de renovação de estoques;
- c) Avaliação, como análise de investimentos de risco, de seguros ou estimativa de custos de projeto;
- d) Controle de processos, como controle de processos industriais.

As tarefas de síntese criam uma nova solução por combinar partes de soluções aplicadas anteriormente, são elas:

- a) Projeto, um novo produto é criado por adaptar elementos de outros;
- b) Planejamento, onde os novos planos são criados a partir da combinação de elementos de planos anteriores;
- c) Configuração, novas configurações são propostas a partir de modificações de anteriores.

A utilização dessa técnica fica limitada apenas ao acesso às bases de dados completas, corretas e confiáveis que contenham entre as informações armazenadas, a descrição completa de problemas e das soluções que foram aplicadas em algum momento, pois, esta é a matéria prima inicial e básica para a construção de sistemas baseados em casos (VITORINO, 2009).

2.2 FRAMEWORKS

Define-se *frameworks* como um conjunto de classes-base (esqueleto) sobre o qual uma ferramenta é construída. Toda a ferramenta criada a partir de um *framework* tem a responsabilidade de interpretar a necessidade do usuário (REINALDO, 2003). Modelos de *framework* aumentam o nível de abstração do projeto, tentando identificar padrões de projeto de arquitetura reutilizáveis, encontrados em tipos de aplicações similares (PRESSMAN, 2016).

Segundo Larman (2007) *framework* é um conjunto de objetos extensível para funções relacionadas. A qualidade distintiva de um *framework*, é que ele fornece uma implementação para as funções básicas e invariantes e inclui um mecanismo para permitir que o desenvolvedor se conecte às diversas funções ou as estenda. Larman (2007) complementa ainda descrevendo o *framework* como um subsistema extensível para um conjunto de serviços relacionados. Também é possível ser classificado como um conjunto coeso de classes que colaboram para fornecer serviços para o núcleo invariante de um sistema lógico. Contém classes concretas e abstratas que definem interfaces a serem seguidas, interações entre objetos das quais a aplicação deve participar como também outros invariantes. (SILVA, 2000) define frameworks como um conjunto de estruturas de classes que constituem implementações incompletas que quando estendidas, permitem produzir diferentes artefatos de um sistema.

Um *framework* captura as decisões de projeto que são comuns ao seu domínio de aplicação. Assim, *frameworks* enfatizam reutilização de projetos em relação a reutilização de código, embora um *framework* geralmente, inclua subclasses concretas que podemos utilizar imediatamente (MONTEIRO, 2002).

De acordo com Larman (2007), em geral, um *framework*:

- a) É um conjunto coeso de interfaces e classes que colaboram para fornecer serviços para a parte básica e constante de um subsistema lógico;
- b) Contém classes concretas e abstratas, que definem interfaces a serem seguidas, interações entre objetos das quais participar e outros invariantes;
- c) Em geral, (mas não necessariamente) exige que o usuário do framework defina subclasses das classes existentes do *framework*, para fazer uso, personalizar e estender os serviços do *framework*;
- d) Possui classes abstratas que podem conter métodos abstratos e concretos;
- e) Classes definidas pelo usuário (por exemplo, novas subclasses) receberão mensagens das classes predefinidas do *framework*. Normalmente, elas são tratadas usando a implementação de métodos abstratos da superclasse.

Embora tanto um *framework* quanto um padrão de projeto possam ser definidos como conjuntos de classes cooperantes, um *framework* se mostra muito mais amplo, contando com uma quantidade maior de componentes (BEZERRA, 2015). Um *framework* montado através do uso de padrões tem maior possibilidade de atingir altos níveis de reusabilidade de projeto e código, comparado a um que não usa padrões (GAMMA et al, 2008).

Frameworks maduros comumente incorporam vários padrões de projeto. Os padrões ajudam a tornar a arquitetura do *framework* adequada a muitas aplicações diferentes, sem necessidade de reformulação (GAMMA et al, 2008). Horstmann (2007) complementa que o desenvolvimento de um *framework* não constitui de uma regra geral de projeto, empregando geralmente múltiplos padrões de projeto no seu desenvolvimento.

2.2.1 Padrões de Projeto

Um padrão de projeto pode ser caracterizado como uma regra de três partes que expressa uma relação entre um contexto, um problema e uma solução. Tratam de problemas associados ao desenvolvimento de subsistemas e componentes, da maneira pela qual eles se comunicam entre si e de seu posicionamento em uma arquitetura maior (PRESSMAN, 2016). Um padrão de projeto nomeia, abstrai e identifica os aspectos-chave de uma estrutura de projeto comum para torná-la útil para a criação de um projeto orientado a objetos reutilizável. O padrão de projeto identifica as classes e instâncias participantes, seus papéis, colaborações e a distribuição de responsabilidades. Cada padrão de projeto focaliza um problema ou tópico particular de projeto orientado a objetos. Ele descreve em que situação pode ser aplicado, se ele pode ser aplicado em função de outras restrições de projeto e as consequências, custos e benefícios de sua utilização (GAMMA et al, 2008).

De acordo com Pressman (2016), caracteriza-se um padrão de projeto eficaz da seguinte maneira:

- a) Ele soluciona um problema: Os padrões apreendem soluções, não apenas estratégias ou princípio abstratos;
- b) Ele é um conceito comprovado: Os padrões apreendem soluções com um histórico, não teorias ou especulação;
- c) Uma solução não é óbvia: Muitas técnicas para resolução de problemas tentam obter soluções com base nos primeiros princípios. Os melhores padrões geram uma solução para um problema indiretamente, uma abordagem necessária para os problemas de projeto mais difíceis;
- d) Ele descreve uma relação: Os padrões não apenas descrevem módulos, como também estruturas e mecanismos de sistema mais profundos;

- e) O padrão possui um componente humano significativo: Todo software visa a atender o conforto humano ou a qualidade de vida, os melhores padrões apelam explicitamente à estética e à utilidade.

Um padrão de projeto evita que tenhamos que “reinventar a roda” ou, pior ainda, inventar uma “nova roda” que não será perfeitamente redonda, será muito pequena para o uso pretendido e muito estreita para o terreno onde irá rodar. Os padrões de projeto, se usados de maneira eficiente, invariavelmente o tornarão um melhor projetista de software (PRESSMAN, 2016).

De acordo com Gamma et al (2008), a utilização de padrões de projeto no desenvolvimento de um *framework* garante altos níveis de reusabilidade de projeto e código, quando comparado a um projeto que não incorpora padrões. Além disso, torna o aprendizado mais suave, fazendo com que os elementos-chave do projeto se mostrem mais explícitos. Um projeto baseado em padrões garante facilidade de comunicação, implicando assim, na rápida compreensão do sistema, credibilidade, reusabilidade e manutenibilidade (SAMPAIO, 2007).

Os padrões de projeto variam na sua granularidade e no seu nível de abstração. Como existem muitos padrões de projeto, a Figura 2 retirada do livro de Gamma et al (2008), classifica os padrões relacionados para um melhor entendimento.

Figura 2 - Classificação de Padrões de Projeto

Escopo	Classe	Propósito		
		De criação	Estrutural	Comportamental
		Factory Method	Adapter (class)	Interpreter Template Method
	Objeto	Abstract Method Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Fonte: Gamma et al, 2008, p. 26.

Na Figura 2 os padrões de projeto estão classificados por dois critérios. O primeiro critério chamado finalidade, reflete o que um padrão faz. Os padrões podem ter finalidade de criação, estrutural ou comportamental. Os padrões de criação se preocupam com o processo de criação de objetos. Os padrões estruturais lidam com a composição de classes ou de objetos. Os padrões comportamentais caracterizam as maneiras pelas quais classes ou objetos interagem e distribuem responsabilidades (GAMMA et al, 2008).

O segundo critério, chamado escopo, especifica se o padrão se aplica primeiramente a classes ou a objetos. Os padrões para classes lidam com os relacionamentos entre classes e suas subclasses. Esses relacionamentos são estabelecidos através do mecanismo de herança, assim, eles são estáticos ou fixados em tempo de compilação. Os padrões para objetos lidam com relacionamentos entre objetos que podem ser mudados em tempo de execução e são mais dinâmicos (GAMMA et al, 2008).

Ainda de acordo com Gamma et al (2008), há outras maneiras de organizar os padrões. Alguns padrões são frequentemente usados em conjunto, e embora tenham intenções diferentes podem resultar em projetos semelhantes, conforme pode ser visto na Figura 3 a seguir.

2.2.1.1 Padrões de Criação

Os padrões de criação abstraem o processo de instanciação. Eles ajudam a tornar um sistema independente de como seus objetos são criados, compostos e representados. Um padrão de criação de classe usa a herança para variar a classe que é instanciada, enquanto que um padrão de criação de objeto delegará a instanciação para outro objeto (GAMMA et al, 2008). Se concentram na criação, composição e representação de objetos e dispõem de mecanismos que facilitam a instanciação de objetos em um sistema e que impõem restrições sobre o tipo e número de objetos que podem ser criados em um sistema (PRESSMAN, 2016).

Os padrões de criação se tornam importantes à medida que os sistemas evoluem no sentido de depender mais da composição de objetos do que da herança de classes. Quando isso acontece, a ênfase se desloca da codificação rígida de um conjunto fixo de comportamentos para a definição de um conjunto menor de comportamentos fundamentais, os quais, podem ser compostos em qualquer número para definir comportamentos mais complexos. Assim, criar objetos com comportamentos particulares exige mais do que simplesmente instanciar uma classe (GAMMA et al, 2008).

2.2.1.1.1 *Abstract Factory*

O padrão de criação *Abstract Factory* tem como objetivo fornecer uma interface para criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas (GAMMA et al, 2008).

De acordo com Gamma et al (2008), a aplicabilidade deste padrão de criação é denotado quando:

- Um sistema deve ser independente de como seus produtos são criados, compostos ou representados;
- Um sistema deve ser configurado como um produto de uma família de múltiplos produtos;
- Uma família de objetos-produto for projetada para ser usada em conjunto, e você necessita garantir essa restrição;
- Você quer fornecer uma biblioteca de classes de produtos e quer revelar somente suas interfaces, não suas implementações.

2.2.1.1.2 Builder

O padrão de criação *Builder* tem como objetivo a construção de um objeto complexo da sua representação de modo que o mesmo processo de construção possa criar diferentes representações (GAMMA et al, 2008).

De acordo com Gamma et al (2008), a aplicabilidade deste padrão de criação é denotado quando:

- O algoritmo para criação de um objeto complexo deve ser independente das partes que compõem o objeto e de como elas são montadas;
- O processo de construção deve permitir diferentes representações para o objeto que é construído.

2.2.1.2 Padrões Estruturais

Os padrões estruturais se preocupam com a forma como classes e objetos são compostos para formar estruturas maiores. Os padrões estruturais de classes utilizam a herança para compor interfaces ou implementações, enquanto, padrões estruturais de objetos descrevem maneiras de compor objetos para obter novas funcionalidades. A flexibilidade obtida pela composição de objetos provém da capacidade de mudar a composição em tempo de execução, o que é impossível com a composição estática de classes (GAMMA et al, 2008). Focalizam problemas e soluções associadas a como classes e objetos são organizados e integrados para construir uma estrutura maior (PRESSMAN, 2016).

2.2.1.2.1 Adapter

O padrão estrutural *Adapter* tem como objetivo converter a interface de uma classe em outra interface esperada pelos clientes. O *Adapter* permite que classes com interfaces incompatíveis trabalhem em conjunto o que, de outra forma, seria impossível (GAMMA et al, 2008).

De acordo com Gamma et al (2008), a aplicabilidade deste padrão estrutural é denotado quando:

- Você quiser usar uma classe existente, mas sua interface não corresponder à interface de que necessita;
- Você quiser criar uma classe reutilizável que coopere com classes não-relacionadas ou não-previstas, ou seja, classes que não necessariamente tenham interfaces compatíveis;
- Você precisar usar várias subclasses existentes, porém, for impraticável adaptar essas interfaces criando subclasses para cada uma. Um adaptador de objeto pode adaptar a interface da sua classe-mãe.

2.2.1.2.2 *Bridge*

O padrão estrutural *Bridge* tem como objetivo desacoplar uma abstração da sua implementação, de modo que as duas possam variar independentemente. Quando uma abstração pode ter uma entre várias implementações possíveis, a maneira usual de acomodá-las é usando a herança. Uma classe abstrata define a interface para a abstração, e subclasses concretas a implementam de formas diferentes. Mas essa abordagem nem sempre é suficientemente flexível. A herança liga uma implementação à abstração permanentemente, o que torna difícil modificar, aumentar e reutilizar abstrações e implementações independentemente (GAMMA et al, 2008).

De acordo com Gamma et al (2008), a aplicabilidade deste padrão estrutural é denotado quando:

- Desejar evitar um vínculo permanente entre uma abstração e sua implementação. Isso pode ocorrer, por exemplo, quando a implementação deve ser selecionada ou alterada em tempo de execução;
- Tanto as abstrações como suas implementações tiverem de ser extensíveis por meio de subclasses. Neste caso, o padrão Bridge permite combinar as diferentes abstrações e implementações e estendê-las independentemente;
- Mudanças na implementação de uma abstração não puderem ter impacto sobre os clientes, ou seja, quando o código dos mesmos não puder ser recompilado;
- Tiver uma proliferação de classes;
- Desejar compartilhar uma implementação entre múltiplos objetos e este fato deve estar oculto do cliente.

2.2.1.3 Padrões Comportamentais

Os padrões comportamentais se preocupam com algoritmos e a atribuição de responsabilidades entre objetos. Os padrões comportamentais não descrevem apenas padrões de objetos ou classes, mas também, os padrões de comunicação entre eles. Esses padrões caracterizam fluxos de controle difíceis de seguir em tempo de execução. Eles afastam o foco do fluxo de controle para permitir que você se concentre somente na maneira como os objetos são interconectados (GAMMA et al, 2008). Tratam de problemas associados à atribuição de responsabilidade entre objetos e a maneira como a comunicação é realizada entre objetos (PRESSMAN, 2016).

Os padrões comportamentais de classe utilizam a herança para distribuir o comportamento entre classes, enquanto, padrões comportamentais de objetos utilizam a composição de objetos em vez da herança (GAMMA et al, 2008).

2.2.1.3.1 *Strategy*

O padrão comportamental *Strategy* tem como objetivo definir uma família de algoritmos, encapsular cada uma delas e torná-las intercambiáveis. *Strategy* permite que o algoritmo varie independentemente dos clientes que o utilizam (GAMMA et al, 2008).

De acordo com Gamma et al (2008), a aplicabilidade deste padrão comportamental é denotado quando:

- Muitas classes relacionadas diferem somente no seu comportamento. As estratégias fornecem uma maneira de configurar uma classe com um dentre muitos comportamentos;
- Você necessita de variantes de um algoritmo. Por exemplo, pode definir algoritmos que refletem diferentes soluções de compromisso entre espaço/tempo. As estratégias podem ser usadas quando essas variantes são implementadas como uma hierarquia de classes de algoritmos;
- Um algoritmo usa dados dos quais os clientes não deveriam ter conhecimento. Use o padrão *Strategy* para evitar a exposição das estruturas de dados complexas, específicas do algoritmo;
- Uma classe define muitos comportamentos, e estes aparecem em suas operações como múltiplos comandos condicionais da linguagem. Em vez de usar muitos

comandos condicionais, mova os ramos condicionais relacionados a sua própria classe *Strategy*.

2.2.1.3.2 *Template Method*

O padrão comportamental *Template Method* tem como objetivo definir o esqueleto de um algoritmo em uma operação, postergando alguns passos para as subclasses. *Template Method* permite que subclasses redefinam certos passos de um algoritmo sem mudar a estrutura do mesmo (GAMMA et al, 2008).

De acordo com Gamma et al (2008), a aplicabilidade deste padrão comportamental é denotado quando:

- Para implementar as partes invariantes de um algoritmo uma só vez e deixar para as subclasses a implementação do comportamento que pode variar;
- Quando o comportamento comum entre subclasses deve ser fatorado e concentrado numa classe comum para evitar a duplicação de código. Este é um bom exemplo de “refatorar para generalizar”. Primeiramente, você identifica as diferenças no código existente e então separa as diferenças em novas operações. Por fim, você substitui o código que apresentava as diferenças por um método-template que chama uma dessas novas operações;
- Para controlar extensões de subclasses. Você pode definir um método-template que chama operações “gancho” em pontos específicos, desta forma permitindo extensões somente nesses pontos.

3 TRABALHOS CORRELATOS

Nesta seção, são abordados trabalhos correlacionados ao objeto de estudo desta monografia. Desta forma, são apresentados diferentes estudos e desenvolvimentos de *framework's* voltados a técnica de Inteligência Artificial, Raciocínio Baseado em Casos.

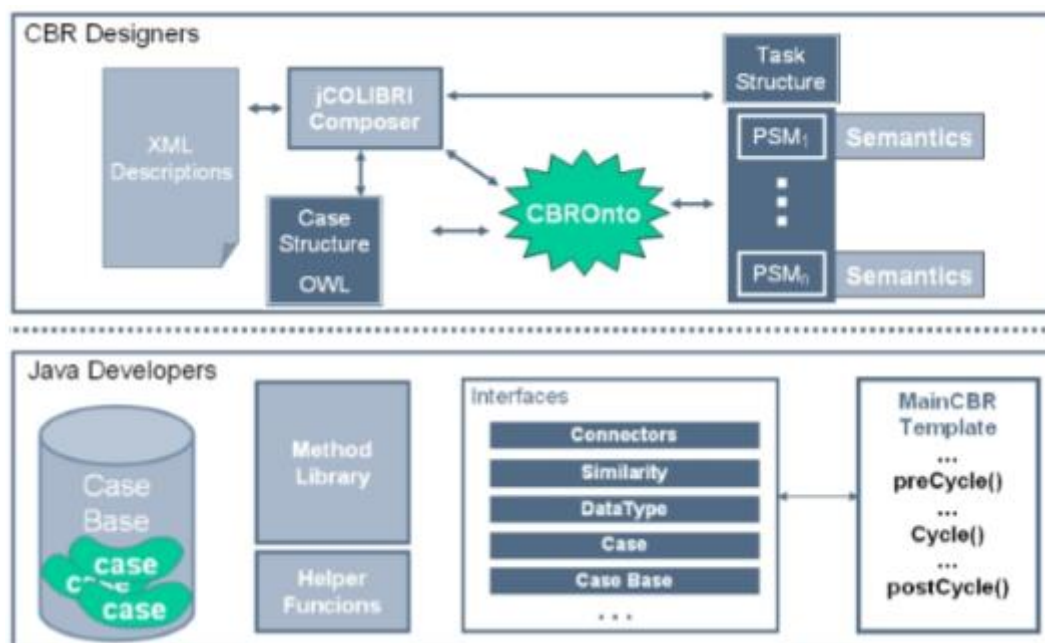
3.1 JCOLIBRI: A MULTI-LEVEL PLATFORM FOR BUILDING AND GENERATING CBR SYSTEMS

De acordo com García (2008), até no ano de 2008, o RBC já consistia de um campo de pesquisa maduro e estabelecido, porém, ainda não contava com um *framework* para apoiar o desenvolvimento de aplicações RBC de forma prática. Dessa forma, como tese de doutorado, o autor desenvolveu e apresentou um *framework* que fornece assistência de projeto e implementação a comunidade RBC.

O *framework* proposto chama-se jColibri, que tem como principal característica é sua orientação, na qual, serve tanto para desenvolvedores, quanto para designers. Esta divisão é devida a diferentes necessidades destes dois perfis de utilizador e está refletida na arquitetura multi-nível do *framework*. O nível inferior é orientado para desenvolvedores e fornece uma arquitetura para criar diversas aplicações RBC, pois, usuários desenvolvedores são aqueles que possuem habilidades técnicas e preferem implementar aplicações RBC a um nível de código fonte, enquanto, o nível superior é destinado a usuários designers e contém várias ferramentas de composição de alto nível para gerar os aplicativos RBC.

Conforme pode ser observado na Figura 4, para suportar ambos os tipos de usuários, jColibri define uma arquitetura multi-nível com duas camadas, cada uma orientada para um desses perfis. No bloco inferior, a Figura 4 demonstra os blocos básicos para construção de sistemas RBC que podem ser facilmente estendidos e reutilizados pelos programadores. No bloco superior orientado para designers, podemos perceber a inclusão de uma caixa de ferramentas que automatiza parcialmente a geração de sistemas RBC através da composição dos elementos na estrutura. Esta composição é guiada por modelos que abstraem o comportamento dos sistemas RBC e podem ser instanciados com os diferentes componentes da camada inferior. Essa instanciação é conduzida por uma representação semântica desses componentes que é baseada em uma ontologia chamada CBROnto. Portanto, a camada superior delimita outra arquitetura que organiza o conhecimento e os elementos necessários para implementar tais técnicas de composição.

Figura 4 - Arquitetura jColibri 2



Fonte: García, 2008, p. 2.

3.2 FERRAMENTA PARA DESENVOLVIMENTO DE SISTEMAS DE RACIOCÍNIO BASEADO EM CASOS

Dissertação apresentada como requisito parcial a obtenção do grau de Mestre em Computação Aplicada na Universidade do Vale do Itajaí por Kraus (2009). O autor conduz um estudo acerca dos problemas existentes em *framework's* atuais a época, e destaca como problematização, que existem diversas ferramentas para o desenvolvimento de sistemas RBC, porém, que obrigam os usuários a terem pleno domínio desta técnica. Destaca ainda, a falta de uma interface amigável para definição e representação do conhecimento e das métricas de similaridade nas ferramentas existentes. Dessa forma, traz como proposta, o desenvolvimento de uma ferramenta para o desenvolvimento de Sistemas RBC com foco no ensino da técnica, que possibilite o uso em turmas de graduação da disciplina de Inteligência Artificial.

Para ensino, a ferramenta segue como estratégia a Aprendizagem Baseada em Problemas, que é uma abordagem que utiliza problemas do mundo real, estudos de caso hipotéticos com resultados concretos e convergentes para que os aprendizes assimilem o conteúdo planejado e desenvolvam a habilidade de pensar criticamente. A ferramenta utiliza como base o framework jColibri 2, que é Open Source e modela a estrutura dos sistemas RBC,

disponibilizando um ambiente onde os usuários possam entender o funcionamento da técnica de RBC, auxiliados por um assistente e por ajudas contextualizadas, como também elaborar novos sistemas interagindo com a ferramenta.

A ferramenta é um *Shell* para desenvolvimento de sistemas RBC, que possibilita aos usuários utilizarem, ou estenderem, as definições elaboradas para outras aplicações externas. O desenvolvimento leva em consideração a ergonomia e usabilidade de sistemas para criar um ambiente de ensino e desenvolvimento de sistemas de Raciocínio Baseado em Casos, possibilitando o uso na academia para servir de auxílio aos educadores e gerar facilidade aos alunos.

3.3 UM FRAMEWORK DE RACIOCÍNIO BASEADO EM CASOS APLICADO PARA ESTRUTURAS A BASE DE CONHECIMENTO EM SISTEMAS TUTORES INTELIGENTES

Dissertação submetida ao Programa de Pós-Graduação em Ciência e Tecnologia da Computação como parte dos requisitos para obtenção do Título de Mestre em Ciência e Tecnologia da Computação. Nesta dissertação, o autor Mendes (2012) apresenta que atualmente existe a necessidade de se oferecer ensino personalizado segundo o perfil dos usuários. Nesse contexto, surgiram os sistemas tutores inteligentes, que usam técnicas de inteligência artificial para prover ensino dinâmico, considerando as habilidades e deficiências dos alunos.

Partindo desta premissa, o trabalho apresenta o desenvolvimento de um *framework* que visa auxiliar desenvolvedores de sistemas RBC, levando em consideração as características de sistemas RBC, e as dificuldades que um desenvolvedor encontra em modelar os dados na forma de casos. A ferramenta desenvolvida visa facilitar a modelagem de novos casos, bem como a introdução de novos episódios no sistema. A finalidade da utilização do *framework* é conseguir organizar toda a base de conhecimento, tanto nos tópicos teóricos a serem trabalhados com o aluno, quanto as simulações que serão posteriormente empregadas para fixar o conhecimento. Além disso, o *framework* disponibiliza medidas de similaridade para a análise da similaridade entre os casos já existentes na aplicação, apresentando valores numéricos que medem o quão similar um caso é de outro.

O *framework* foi desenvolvido em Java, e possui funcionalidades para o desenvolvedor modelar novos sistemas RBC, alimentá-los com casos, e verificar similaridades destes casos a partir de parâmetros informados pelo usuário, além de possuir todos os ciclos de

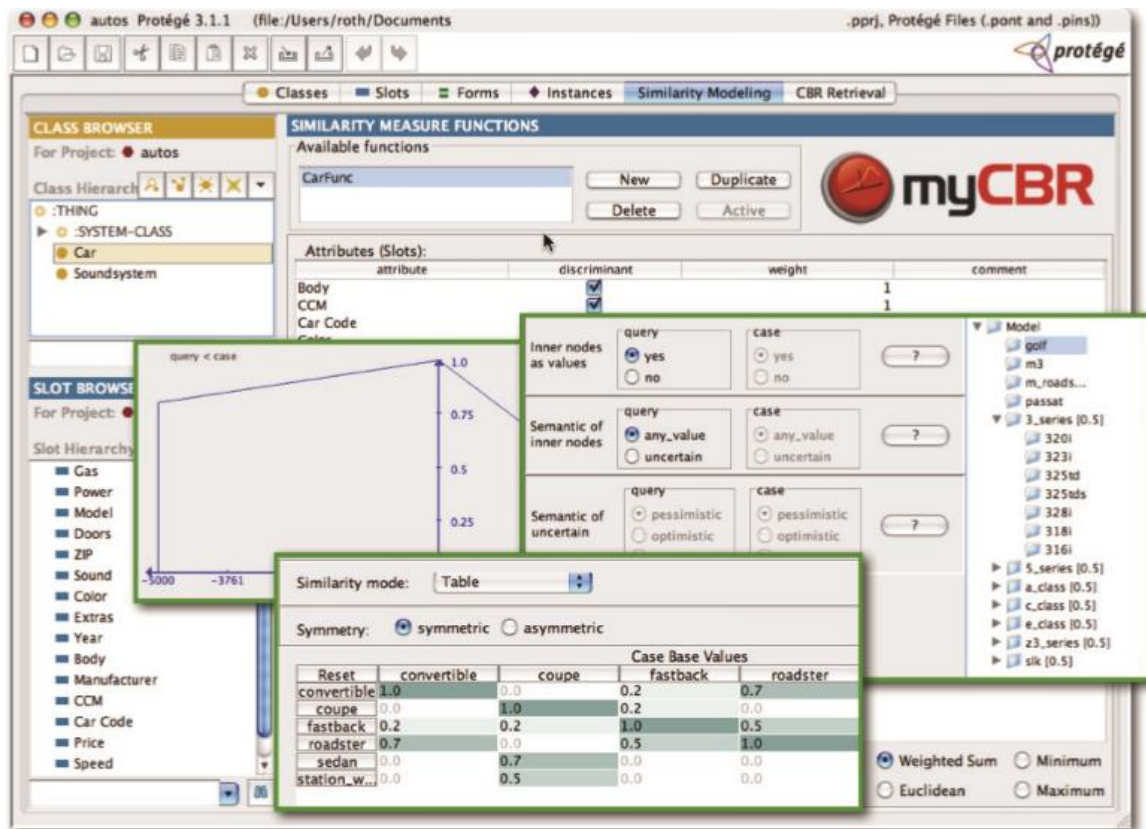
um RBC. Também permite que possam ser inseridos casos através de arquivos “.txt”, porém, respeitando a estrutura da aplicação. Como um sistema RBC pode ser desenvolvido em diferentes linguagens de programação, o *framework* possibilita ainda, a exportação dos dados modelados através de um arquivo do tipo “.txt”. Esse arquivo pode ser utilizado como entrada na aplicação em desenvolvimento. Por fim, possibilita ainda, que o desenvolvedor observe os casos que foram modelados no sistema e também os casos já inseridos nele.

A ferramenta foi testada através de um estudo de caso, utilizando um sistema tutor para novos pilotos de helicóptero. Foram modelados casos de dois tipos distintos: casos teóricos e práticos. Para ambos os casos, foi possível inserir novos episódios e mensurar o quão semelhante os outros casos existentes são do caso em estudo (MENDES, 2012).

3.4 THE OPEN-SOURCE TOOL MYCBR

MyCBR é uma ferramenta Open-Source desenvolvida pelo Centro de Pesquisa Alemão de Inteligência Artificial (DFKI). A principal motivação para a implementação myCBR foi a necessidade de uma ferramenta de fácil manuseio para construção de protótipos de aplicações RBC, tanto para pesquisa como para pequenos projetos industriais, com o mínimo de esforço. A ideia principal da criação da ferramenta veio da antiga ferramenta CBR-Works, que foi descontinuada. Na Figura 5 é apresentada a interface da ferramenta.

Figura 5 - Interface MyCBR



Fonte: Bahls; Berghofer, 2007, p. 1.

A versão atual do myCBR tem foco na similaridade baseada na etapa de recuperação do ciclo RBC, visto que é a principal funcionalidade das aplicações.

O principal objetivo do myCBR é minimizar o esforço para construir aplicações RBC que requerem alta intensidade cognitiva nas medidas de similaridade. Por isso, ele fornece uma boa interface gráfica para modelar vários tipos de atributos específicos e medidas de similaridade, possibilitando avaliar a qualidade final da recuperação das informações. Afim de reduzir também o esforço da etapa anterior de definição de uma representação de casos adequada, ele utiliza ferramentas para gerar automaticamente o processo de representação de dados brutos (BAHLS; BERGHOFER, 2007).

3.5 FRAMEWORK PARA SISTEMA DE RACIOCÍNIO BASEADO EM CASOS

Trabalho de Conclusão de Curso apresentado ao Centro Universitário Vila Velha como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação. Neste

trabalho, os autores afirmam que com o crescente número de transações comerciais de varejo efetuadas na rede mundial de computadores, torna-se eminente a busca por conquistar estes consumidores. Pensando nisso, os desenvolvedores dos sites de comércio eletrônico têm sido obrigados a inovar para manter-se na briga pelo mercado. Sabe-se que para um vendedor oferecer o produto correto a um cliente (com mais chances de ser vendido), este deve identificar algumas características do interessado de acordo com o escopo. Num ambiente online, isto se torna difícil, devido às limitações que são impostas pela tradicional forma de navegar, escolher um produto e comprar na internet.

O Raciocínio Baseado em Casos, ramo da Inteligência Artificial, pode influenciar fortemente de maneira positiva neste aspecto, pois tem como premissa resolver um dado caso a partir de soluções de casos análogos que foram bem sucedidas no passado. Com o uso desta tecnologia, é possível sugerir produtos, negociar descontos e entregas, além de conhecer seus clientes, assim como um vendedor de carne e osso. Esta é a solução abordada por este trabalho onde foi desenvolvido um framework para esta tecnologia e também um protótipo de um sistema de vendas de carros online.

O *framework* foi projetado de acordo com as quatro fases básicas do ciclo de vida, e é utilizado no momento da pré-venda, na qual um cliente está procurando um produto na qual deseja comprar, o *framework* é acionado e o RBC pode ser aplicado para oferecer catálogos de produtos inteligentes, capazes de oferecer um produto ou serviço similar, caso o produto requisitado pelo cliente não esteja disponível, ou, então, capazes simplesmente de oferecer um produto com base em um conjunto de características fornecidas pelo usuário (ERVATTI; ANDRADE; SILVA, 2010).

4 DESENVOLVIMENTO

Nesta seção, são abordadas as especificações do *framework* através da engenharia de requisitos e modelagem UML. Posteriormente, apoiados na engenharia de *software* orientada a objetos e em padrões de projeto, são detalhados o desenvolvimento e a documentação do *framework* proposto. Por fim, são documentados os testes e análise dos resultados obtidos.

4.1 ESPECIFICAÇÕES FORMAIS

Nesta seção, são apresentados os requisitos do *framework*, juntamente com o diagrama de casos de uso e diagrama de classes, modelado através da linguagem UML.

4.1.1 Requisitos

De acordo com Pressman (2016), a Engenharia de Requisitos nos permite que examinamos o contexto do trabalho de software a ser realizado, as necessidades específicas a que o projeto e a construção devem atender, as prioridades que orientam a ordem na qual o trabalho deve ser concluído, e as informações, funções e comportamentos que terão um impacto profundo no projeto resultante, dessa forma, define o comportamento ou características esperadas da aplicação.

Os requisitos podem ser classificados como funcionais ou não-funcionais, e nas próximas seções, serão abordados os requisitos do *framework* proposto.

4.1.1.1 Requisitos Funcionais

De acordo com Pressman (2016), os requisitos funcionais tratam de funções que o sistema deve fornecer e como deve se comportar em determinadas situações. Em outras palavras, os requisitos funcionais descrevem as funções que o produto deverá realizar em benefício dos usuários (FILHO, 2000).

Quadro 1 - Requisitos Funcionais

Requisitos Funcionais
RF01: Permitir modelar um sistema de Raciocínio Baseado em Casos.
RF02: Permitir selecionar o banco de dados desejado entre as três opções disponíveis.
RF03: Permitir estruturar as tabelas e campos do banco de dados necessários para a busca de informações pelo <i>framework</i> .
RF04: Permitir gerar o <i>framework</i> com extensão (.jar), ou seja, uma biblioteca Java.
RF05: Permitir efetuar a recuperação de casos na base de casos, através de métricas de similaridade.
RF06: Permitir efetuar a reutilização de casos.
RF07: Permitir efetuar a retenção de casos.
RF08: Permitir efetuar a revisão de casos.
RF09: O <i>framework</i> deverá retornar ao usuário os casos similares em forma decrescente.
RF10: O <i>framework</i> deverá mostrar a solução do caso similar selecionado.
RF11: O <i>framework</i> deverá retornar informações do resultado do processamento das funções chamadas pelo usuário.

Fonte: Elaboração do autor, 2017.

4.1.1.2 Requisitos Não-Funcionais

Um requisito não funcional pode ser descrito como um atributo de qualidade, de desempenho, de segurança ou como uma restrição geral em um sistema. É estabelecido um conjunto de diretrizes de engenharia de software para o sistema a ser construído. Isso inclui diretrizes de melhor prática, mas também trata do estilo arquitetural e do uso de padrões de projeto. Então, é feita uma lista de requisitos não-funcionais, como por exemplos, requisitos que tratam da usabilidade, teste, segurança ou manutenção. Uma tabela simples lista requisitos não-funcionais como rótulos de coluna e as diretrizes de engenharia de software como rótulos de linha (PRESSMAN, 2016).

Quadro 2 - Requisitos Não-Funcionais

Requisitos Não-Funcionais
RNF01: O <i>framework</i> deverá ser implementado na Linguagem Java.
RNF02: O <i>framework</i> deverá permitir conexão com os bancos de dados Firebird, PostgreSQL e MySQL.
RNF03: O <i>framework</i> deverá ser uma biblioteca Java, extensão (.jar), para permitir integração em outras linguagens através do console.
RNF04: O <i>framework</i> deverá ser implementado apoiado nos padrões de projeto de criação <i>Abstract Factory</i> e <i>Builder</i> , nos padrões de projeto estruturais <i>Adapter</i> e <i>Bridge</i> e nos padrões comportamentais <i>Strategy</i> e <i>Template Method</i> .

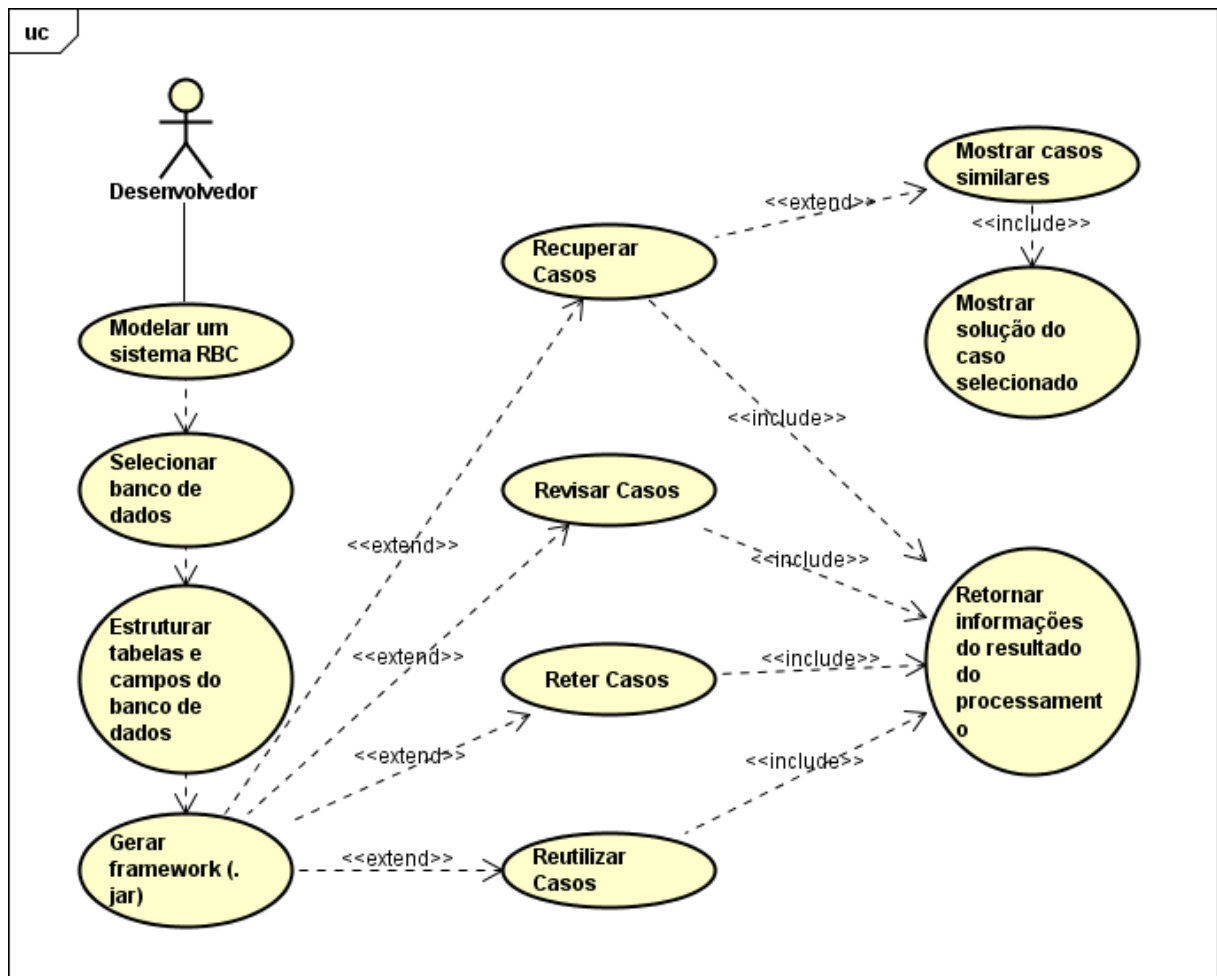
Fonte: Elaboração do autor, 2017.

4.1.2 Diagramas de Casos de Uso

De acordo com Pressman (2016), um caso de uso conta uma jornada estilizada sobre como um usuário desempenhando um de uma série de papéis possíveis, interage com o sistema sob um conjunto de circunstâncias específicas. Essa abordagem auxilia na modelagem de requisitos funcionais e conduzem todo o processo de desenvolvimento do sistema (RAMOS, 2006).

Nesta seção é apresentado pela Figura 6, o diagrama de caso do uso do *framework* proposto.

Figura 6 - Diagrama de Casos de Uso

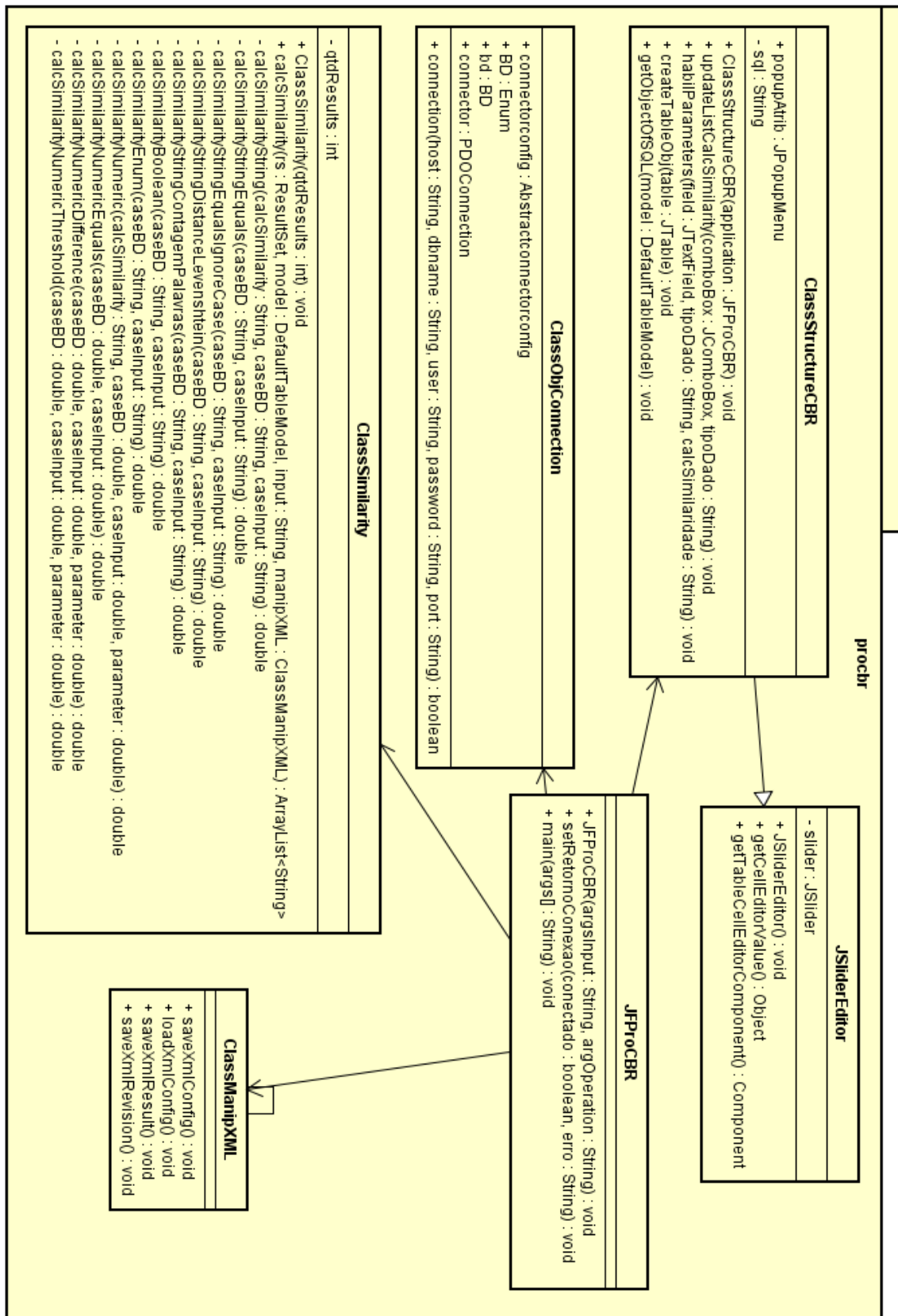


Fonte: Elaboração do autor, 2017.

4.1.3 Diagrama de Classes

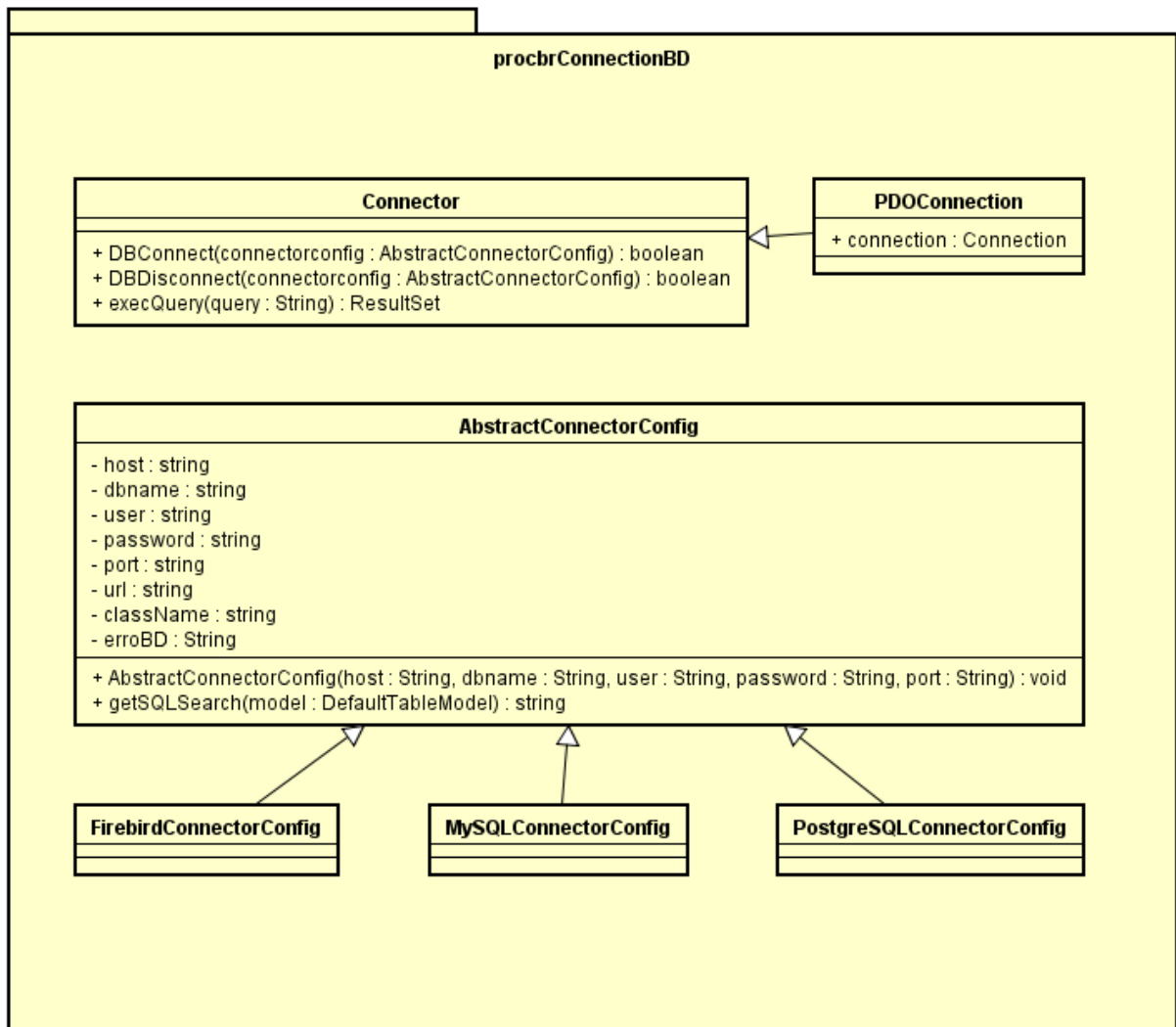
De acordo com Larman (2007), os diagramas de classes são usados para modelagem estática de objetos, ilustrando as classes, interfaces e suas associações de um projeto. É utilizado pelos desenvolvedores para visualizar um modelo de domínio.

Nesta seção é apresentado pela Figura 7 e Figura 8 separados pelos dois pacotes existentes no projeto para uma melhor visualização, o diagrama de classes do *framework* proposto.

Figura 7 - Diagrama de Classes pacote *prochr*

Fonte: Elaboração do autor, 2017.

Figura 8 - Diagrama de Classes pacote *procbrConnectionBD*



Fonte: Elaboração do autor, 2017.

4.2 IMPLEMENTAÇÃO

Nesta seção são descritos os métodos, técnicas e ferramentas utilizadas no desenvolvimento do *framework* proposto, e detalhada a sua implementação.

4.2.1 Técnicas e Ferramentas

No desenvolvimento do *framework*, foi utilizada a linguagem de programação Java 8 utilizando o ambiente de desenvolvimento Netbeans 8.1. Para a aplicação de testes, foi utilizada a linguagem Object Pascal e o ambiente de desenvolvimento Delphi XE6. O

framework permite conexão com três bancos de dados distintos. Firebird, PostgreSQL e MySQL, para testes foram escolhidas as versões Firebird 2.5, PostgreSQL 9.6 e MySQL 5.7.14.

O *framework* proposto, foi desenvolvido apoiado na Engenharia de Software Orientada a Objetos (OOSE), para garantir a organização, manutenibilidade e extensibilidade do projeto, sendo assim, utilizou-se os padrões de projeto de criação, estruturais e comportamentais para garantir todos esses requisitos.

No desenvolvimento do projeto, foram utilizadas bibliotecas desenvolvidas por terceiros, são elas:

- JpanelSlider, utilizada para criar um efeito estiloso de “deslize” ao ir para a próxima tela;
- Jdom, biblioteca para manipulação de arquivos .xml, na qual foram utilizados para guardar as configurações da estrutura do *framework*, e também como forma adicional de retornar os resultados ao usuário;
- PostgreSQL versão 42.1.1, utilizada para efetuar conexão com o banco de dados PostgreSQL;
- Jaybird 3.0.1, utilizada para efetuar conexão com o banco de dados Firebird;
- Mysql Connector Java versão 5.1.42, utilizada para efetuar conexão com o banco de dados MySQL.

4.2.2 Codificação

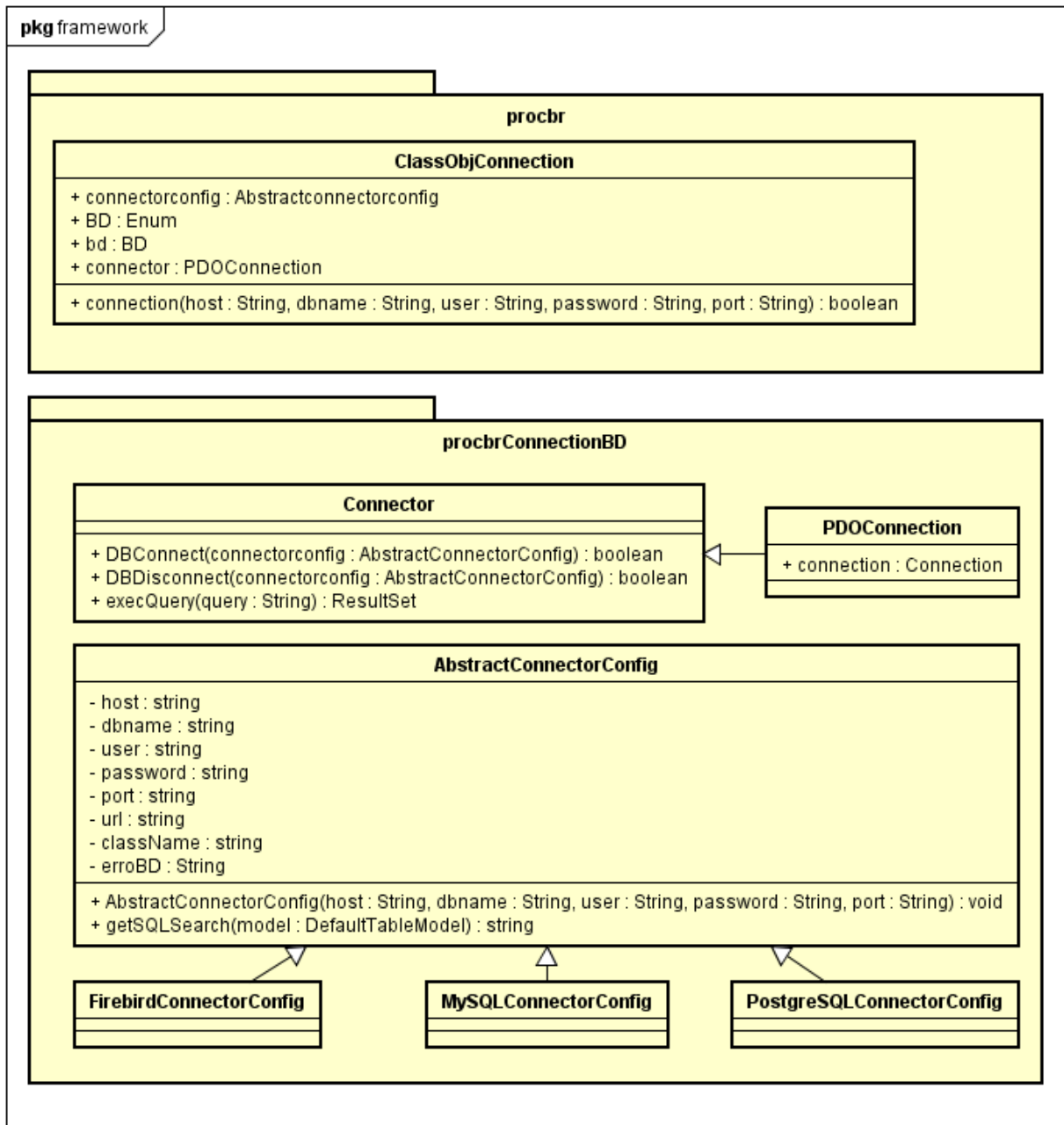
Nesta seção serão apresentados os códigos de algumas das principais funções do *framework*.

4.2.2.1 Conexão com Bancos de Dados

Conforme proposto, o *framework* possui conexão com três bancos de dados distintos, são eles, Firebird, PostgreSQL e MySQL.

Para as classes de conexão, foi utilizado um pacote separado, na qual é apresentado a seguir na Figura 9, no pacote principal *procbr* existe apenas uma classe referente a conexão, que é a classe principal que instancia as classes de conexão.

Figura 9 - Diagrama de Classes de Conexão



Fonte: Elaboração do autor, 2017.

Conforme pode ser observado na Figura 9, a classe *ClassObjConnection* do pacote *procbr* é a classe principal da conexão com os bancos de dados, nela será selecionado o banco de dados utilizado, possui a variável que mantém a conexão ativa enquanto a aplicação estiver rodando, e por fim, possui também a função que instanciará uma nova classe abstrata do banco de dados.

Quadro 3 – Código classe *ClassObjConnection*

```

12 public class ClassObjConnection {
13
14     public AbstractConnectorConfig connectorconfig;
15     public BD bd;
16     public PDOConnection connector;
17
18     public ClassObjConnection() {
19         //
20     }
21
22     public enum BD {
23         FIREBIRD, MYSQL, POSTGRESQL;
24     }
25
26     public boolean connection(String host, String dbname, String user, String
27         password, String port) {
28         if (bd.equals(BD.FIREBIRD)) {
29             connectorconfig = new FirebirdConnectorConfig(host, dbname, user,
30                 password, port);
31         } else if (bd.equals(BD.MYSQL)) {
32             connectorconfig = new MySQLConnectorConfig(host, dbname, user,
33                 password);
34         } else if (bd.equals(BD.POSTGRESQL)) {
35             connectorconfig = new PostgreSQLConnectorConfig(host, dbname,
36                 user,
37                 password);
38         }
39
40         connector = new PDOConnection();
41         if (connector.DBConnect(connectorconfig))
42             return true;
43         else
44             return false;
45     }
46 }

```

Fonte: Elaboração do autor, 2017.

A interface *Connector* possui a declaração dos métodos utilizados para conectar, desconectar e executar um *sql*. Por ser apenas uma interface, as funções estão desenvolvidas dentro da classe *PDOConnection*, na qual, implementa a interface *Connector*.

Quadro 4 - Código classe *PDOConnection*

```

18 public class PDOConnection implements Connector {
19
20     public Connection connection;
21
22     @Override
23     public boolean DBConnect(AbstractConnectorConfig connectorconfig) {
24         boolean isConnected = false;
25
26         try {
27             if (connectorconfig.getClassName() != "") {
28                 Class.forName(connectorconfig.getClassName()).newInstance();
29             }

```

```

30         this.connection =
DriverManager.getConnection(connectorconfig.getUrl(),
31                             connectorconfig.getUser(),
connectorconfig.getPassword());
32         isConnected = true;
33     } catch (SQLException e) {
34         e.printStackTrace();
35         connectorconfig.setErroBD(e.getMessage());
36         isConnected = false;
37     } catch (ClassNotFoundException e) {
38         e.printStackTrace();
39         connectorconfig.setErroBD(e.getMessage());
40         isConnected = false;
41     } catch (InstantiationException e) {
42         e.printStackTrace();
43         connectorconfig.setErroBD(e.getMessage());
44         isConnected = false;
45     } catch (IllegalAccessException e) {
46         e.printStackTrace();
47         connectorconfig.setErroBD(e.getMessage());
48         isConnected = false;
49     }
50
51     return isConnected;
52 }
53
54 @Override
55 public boolean DBDisconnect(AbstractConnectorConfig connectorconfig) {
56     boolean isConnected = false;
57
58     try {
59         if (connectorconfig.getClassName() != "") {
60             Class.forName(connectorconfig.getClassName()).newInstance();
61         }
62         this.connection =
DriverManager.getConnection(connectorconfig.getUrl(),
63                             connectorconfig.getUser(),
connectorconfig.getPassword());
64         this.connection.close();
65         isConnected = true;
66     } catch (SQLException e) {
67         connectorconfig.setErroBD(e.getMessage());
68         isConnected = false;
69     } catch (ClassNotFoundException e) {
70         connectorconfig.setErroBD(e.getMessage());
71         isConnected = false;
72     } catch (InstantiationException e) {
73         connectorconfig.setErroBD(e.getMessage());
74         isConnected = false;
75     } catch (IllegalAccessException e) {
76         connectorconfig.setErroBD(e.getMessage());
77         isConnected = false;
78     }
79
80     return isConnected;
81 }
82
83 @Override
84 public ResultSet execQuery(String query) {
85     Statement st;
86     ResultSet rs;
87
88     try {
89         st = this.connection.createStatement();
90         rs = st.executeQuery(query);
91         return rs;
92     } catch (SQLException e) {
93         e.printStackTrace();

```

```

94         }
95         return null;
96     }
97
98 }

```

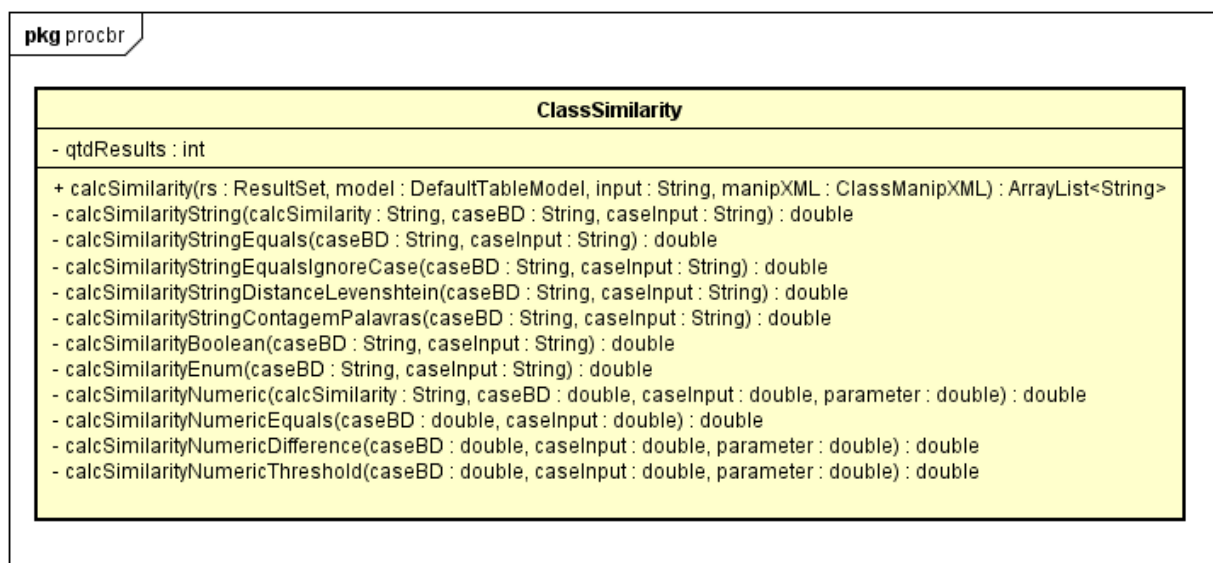
Fonte: Elaboração do autor, 2017.

Por fim, dentro do pacote de classes de conexão, existe a classe *AbstractConnectorConfig*, esta classe serve apenas para declarar as variáveis e instanciar seus respectivos *getters* e *setters* necessários para conexão, como por exemplo, caminho do banco, usuário e senha. Outras três classes são extensões dessa classe, que são, *FirebirdConnectorConfig*, *MySQLConnectorConfig* e *PostgreSQLConnectorConfig*. As três possuem poucas diferenças, sendo elas, um método chamado de *getSQLSearch(DefaultTableModel model)*, que irá retornar os atributos da estrutura do RBC, com base no SQL informado pelo usuário, além disso, elas setam as variáveis *Url* e *ClassName*, conforme cada uma das diferentes tecnologias de banco empregadas exigem.

4.2.2.2 Cálculo de Similaridade

Para realizar o cálculo de similaridade, foi implementada uma classe chamada *ClassSimilarity* apresentada na Figura 10.

Figura 10 - Diagrama de Classe *ClassSimilarity*



Fonte: Elaboração do autor, 2017.

Tanto as similaridades locais quanto a global, são calculadas dentro da função *calcSimilarity*. Como parâmetros de entrada são passados o resultado da execução do sql de busca dos dados no banco de dados e a estrutura do RBC definidos anteriormente pelo usuário na configuração, também é informado uma variável do tipo String na qual está concatenado os dados que o usuário deseja calcular a similaridade. Os dados de entrada devem ser informados como argumento na hora de chamar o *framework* pelo prompt de comando ou aplicação externa, respeitando a estrutura do RBC já definida, informando os campos na ordem correta e separados por ponto e vírgula “;”.

A classe também é encarregada, de efetuar a ordenação dos resultados de forma decrescente, ou seja, do caso mais similar ao menos similar, respeitando também, a quantidade de casos que irão retornar configurados anteriormente.

Por fim, o resultado da função será um *ArrayList* do tipo *String*, contendo os campos soluções configurados e o valor da similaridade de cada caso. Como forma alternativa, o sistema poderá salvar o retorno em arquivo do tipo *xml* contendo os valores de entrada passados, os atributos e soluções de cada caso, e o valor da similaridade desse caso, dessa forma, aplicações externas que chamaram a função de cálculo de similaridade do *framework*, poderão fazer o tratamento dos resultados conforme necessário e sem restrições.

Quadro 5 - Código função *calcSimilarity*

```

18 public class ClassSimilarity {
19
20     private int qtdResults;
21
22     public ClassSimilarity(int qtdResults) {
23         this.qtdResults = qtdResults;
24     }
25
26     public static String rTrim(String s) {
27         int i = s.length();
28         while (i > 0 && Character.isWhitespace(s.charAt(i - 1))) {
29             i--;
30         }
31         return s.substring(0, i);
32     }
33
34     public ArrayList<String> calcSimilarity(ResultSet rs, DefaultTableModel
model,
35         String input, ClassManipXML manipXML) throws SQLException {
36         ArrayList<String> arrayAtrib = new ArrayList<String>();
37         ArrayList<String> arraySolut = new ArrayList<String>();
38
39         ArrayList<String> arrayAtribResult = new ArrayList<String>();
40         ArrayList<String> arraySolutResult = new ArrayList<String>();
41
42         ArrayList<Double> arraySimil = new ArrayList<Double>();
43

```

```

44     String arrayInput[] = input.split(";");
45
46     for (int i = 0; i <= model.getRowCount() - 1; i++) {
47         if ((Boolean) model.getValueAt(i, 5)) { //Is Atribute
48             arrayAtrib.add(model.getValueAt(i, 0).toString());
49         }
50
51         if ((Boolean) model.getValueAt(i, 6)) { //Is Solution
52             arraySolut.add(model.getValueAt(i, 0).toString());
53         }
54     }
55
56     while (rs.next()) {
57         double similarity = 0;
58         double weight = 0;
59         double totalWeight = 0;
60         double globalSimilarity = 0;
61         String retConcat = "";
62         String atribConcat = "";
63         int cont = 0;
64         for (int i = 0; i <= model.getRowCount() - 1; i++) {
65             if ((Boolean) model.getValueAt(i, 5)) { //Is Atribute
66                 weight = Double.parseDouble(model.getValueAt(i,
3).toString());
67                 totalWeight += weight;
68
69                 if (model.getValueAt(i, 1).equals("String")) {
70                     similarity +=
(calcSimilarityString(model.getValueAt(i,
71                         2).toString(), rTrim(rs.getString(i + 1)),
72                         arrayInput[cont])) * weight;
73                 } else if (model.getValueAt(i, 1).equals("Boolean")) {
74                     similarity +=
(calcSimilarityBoolean(rTrim(rs.getString(i + 1)),
75                         arrayInput[cont])) * weight;
76                 } else if (model.getValueAt(i, 1).equals("Numeric")) {
77                     similarity +=
(calcSimilarityNumeric(model.getValueAt(i,
78                         2).toString(), rs.getDouble(i + 1),
79                         Double.parseDouble(arrayInput[cont]),
80                         Double.parseDouble(model.getValueAt(i,
4).toString()))
81                         * weight;
82                 } else if (model.getValueAt(i, 1).equals("Enum")) {
83                     similarity +=
(calcSimilarityEnum(rTrim(rs.getString(i + 1)),
84                         arrayInput[cont])) * weight;
85                 }
86
87                 atribConcat += rs.getString(i + 1) + ";";
88                 cont++;
89             }
90
91             if ((Boolean) model.getValueAt(i, 6)) { //Is Solution
92                 retConcat += rTrim(rs.getString(i + 1)) + ";";
93             }
94         }
95
96         //Calcular Similaridade Global
97         globalSimilarity = (similarity / totalWeight) * 100;
98
99         if (arraySolutResult.size() == 0) {
100             arraySolutResult.add(retConcat);
101             arrayAtribResult.add(atribConcat);
102             arraySimil.add(globalSimilarity);
103         } else {
104             int index = arraySolutResult.indexOf(retConcat);
105             if (index > - 1) {

```

```

106         if (arraySimil.get(index) < globalSimilarity) {
107             arraySimil.set(index, globalSimilarity);
108         }
109     } else {
110         int size = arraySimil.size();
111         for (int i = 0; i <= size - 1; i++) {
112             if (globalSimilarity > arraySimil.get(i)) {
113                 for (int j = size - 1; j > i - 1; j--) {
114                     if ((j == size - 1) && (j < getQtdResults() -
115 1)) {
116                         arraySimil.add(arraySimil.get(j));
117                     }
118                 } else if (j < getQtdResults() - 1) {
119                     arraySimil.set(j + 1, arraySimil.get(j));
120                     arraySolutResult.set(j + 1,
121 arrayAtribResult.set(j + 1,
122 arrayAtribResult.get(j));
123                     }
124                     arraySimil.set(i, globalSimilarity);
125                     arraySolutResult.set(i, retConcat);
126                     arrayAtribResult.set(i, atribConcat);
127                     break;
128                 } else if ((i == size - 1) && (i < getQtdResults() -
129 1)) {
130                     arraySimil.add(globalSimilarity);
131                     arraySolutResult.add(retConcat);
132                     arrayAtribResult.add(atribConcat);
133                 }
134             }
135         }
136     }
137 }
138
139 manipXML.saveXmlResult(arrayAtrib, arraySolut, arrayAtribResult,
140 arraySolutResult, arraySimil, arrayInput);
141
142 for (int i = 0; i <= arraySolutResult.size() - 1; i++) {
143     arraySolutResult.set(i, arraySolutResult.get(i) +
144 String.format("%.2f",
145 arraySimil.get(i)));
146 }
147 return arraySolutResult;
148 }

```

Fonte: Elaboração do autor, 2017.

As funções que realizam o cálculo das similaridades locais, efetuarão o cálculo e retornarão sempre um valor entre 0 e 1, sendo 1 totalmente similar e 0 nada similar, para que, desta forma, os valores retornados são padronizados para a função principal calcular a similaridade global levando em consideração o peso do atributo apresentado.

Na função *calcSimilarityStringEquals* o sistema apenas irá comparar a igualdade entre as palavras ou textos passados como parâmetros, dos casos retornados do banco de dados,

incluindo diferenças de caixa alta nos textos. Os valores retornados serão apenas 0 ou 1, sendo respectivamente, nada similar, e totalmente similar.

Quadro 6 - Código função *calcSimilarityStringEquals*

```
165     private double calcSimilarityStringEquals (String caseBD, String
caseInput) {
166         if (caseBD.equals(caseInput)) {
167             return 1;
168         } else {
169             return 0;
170         }
171     }
```

Fonte: Elaboração do autor, 2017.

A função *calcSimilarityStringEqualsIgnoreCase* também irá comparar a igualdade entre as palavras ou textos passados como parâmetro, dos casos retornados do banco de dados, porém, não leva em consideração diferenças de letras com caixa alta nos textos. Os valores retornados serão apenas 0 ou 1, sendo respectivamente, nada similar, e totalmente similar.

Quadro 7 - Código função *calcSimilarityStringEqualsIgnoreCase*

```
173     private double calcSimilarityStringEqualsIgnoreCase (String caseBD, String
caseInput) {
174         if (caseBD.equalsIgnoreCase(caseInput)) {
175             return 1;
176         } else {
177             return 0;
178         }
179     }
```

Fonte: Elaboração do autor, 2017.

A função *calcSimilarityStringDistanceLevenshtein* retorna um valor entre 0 e 1 com base na quantidade de operações necessárias para transformar uma palavra na outra, sendo 1, totalmente similar, um valor entre 0 e 1 calculado de acordo com o tamanho da palavra e o número de operações necessárias para deixar as palavras iguais, e 0, caso as duas palavras ou texto sejam totalmente diferentes, ou seja, é necessário trocar todas as letras da palavra, para transformá-la na outra comparada.

Quadro 8 - Código função *calcSimilarityStringDistanceLevenshtein*

```

181     private double calcSimilarityStringDistanceLevenshtein(String caseBD,
String caseInput) {
182         double result = 0;
183         int sizeCaseBD = caseBD.length();
184         int sizeCaseInput = caseInput.length();
185         int[][] matriz = new int[sizeCaseBD + 1][sizeCaseInput + 1];
186
187         //Se uma das duas palavras for vazia a similaridade é 0
188         if (caseBD.equals(caseInput)) {
189             return 1;
190         } else if (sizeCaseBD == 0) {
191             return 0;
192         } else if (sizeCaseInput == 0) {
193             return 0;
194         }
195         for (int i = 1; i <= sizeCaseBD; i++) {
196             for (int j = 1; j <= sizeCaseInput; j++) {
197                 int custo = (caseInput.charAt(j - 1) == caseBD.charAt(i - 1))
? 0 : 1;
198
199                 matriz[i][j] = Integer.min(Integer.min(matriz[i - 1][j] + 1,
200                     matriz[i][j - 1] + 1), matriz[i - 1][j - 1] + custo);
201             }
202         }
203
204         if ((matriz[sizeCaseBD][sizeCaseInput]) > 0) {
205             if (matriz[sizeCaseBD][sizeCaseInput] >= sizeCaseBD) {
206                 return 0;
207             } else {
208                 result = (double) matriz[sizeCaseBD][sizeCaseInput] /
sizeCaseBD;
209                 return result;
210             }
211         }
212
213         return 1;
214     }

```

Fonte: Elaboração do autor, 2017.

A função *calcSimilarityStringContagemPalavras* tem como objetivo a comparação de frases ou textos. Compara a quantidade de palavras iguais entre o texto informado como parâmetro, e o dado retornado da consulta no banco de dados. Essa função irá retornar um valor entre 0 e 1, sendo 1 textos totalmente similares, um valor entre 0 e 1 calculado com base na quantidade de palavras e o número de palavras iguais, quanto maior a frase, menor o peso de cada palavra diferente, e 0, caso as frases sejam totalmente diferentes e não possuam nenhuma palavra igual.

Quadro 9 - Código função *calcSimilarityStringContagemPalavras*

```

216     private double calcSimilarityStringContagemPalavras(String caseBD, String
caseInput) {

```

```

217     double result = 0;
218     int cont = 0;
219
220     if (caseBD.equalsIgnoreCase(caseInput)) {
221         return 1;
222     } else if (caseInput.trim() == "") {
223         return 0;
224     }
225
226     String arrayCaseBD[] = caseBD.split(" ");
227     String arrayCaseInput[] = caseInput.split(" ");
228
229     for (int i = 0; i <= arrayCaseBD.length - 1; i++) {
230         for (int j = 0; j <= arrayCaseInput.length - 1; j++) {
231             if (arrayCaseBD[i].equalsIgnoreCase(arrayCaseInput[j])) {
232                 cont++;
233             }
234         }
235     }
236
237     if (cont == 0) {
238         return 0;
239     } else if (cont >= arrayCaseBD.length) {
240         return 1;
241     } else {
242         return (double) cont / arrayCaseBD.length;
243     }
244 }

```

Fonte: Elaboração do autor, 2017.

A função *calcSimilarityBoolean* compara valores booleanos, ou seja, *true* ou *false*, irá retornar 1 em casos onde os valores comparados sejam iguais, e 0 onde os mesmos são diferentes.

Quadro 10 - Código função *calcSimilarityBoolean*

```

246     private double calcSimilarityBoolean(String caseBD, String caseInput) {
247         if (caseBD.equalsIgnoreCase(caseInput)) {
248             return 1;
249         } else {
250             return 0;
251         }
252     }

```

Fonte: Elaboração do autor, 2017.

A função *calcSimilarityEnum* tem como objetivo comparar parâmetros passados como entrada com valores que são pré-definidos no banco de dados, ou seja, não são campos livres para serem digitados pelo usuário do banco de dados, dessa forma, utilizar essa função evita que erros de digitação possam alterar o resultado do cálculo de similaridade. O retorno será 0 ou 1, sendo 0 nada similar, e 1 totalmente similar.

Quadro 11 - Código função *calcSimilarityEnum*

```

254     private double calcSimilarityEnum(String caseBD, String caseInput) {
255         if (caseBD.equalsIgnoreCase(caseInput)) {
256             return 1;
257         } else {
258             return 0;
259         }
260     }

```

Fonte: Elaboração do autor, 2017.

A função *calcSimilarityNumericEquals* faz a comparação entre números inteiros ou fracionários passados como parâmetro de entrada e dos casos retornados do banco de dados. Nessa função, o retorno é apenas 0 e 1 sendo respectivamente nada similar e totalmente similar.

Quadro 12 - Código função *calcSimilarityNumericEquals*

```

277     private double calcSimilarityNumericEquals(double caseBD, double
caseInput) {
278         if (caseBD == caseInput) {
279             return 1;
280         } else {
281             return 0;
282         }
283     }

```

Fonte: Elaboração do autor, 2017.

A função *calcSimilarityNumericDifference* faz a comparação entre números inteiros ou fracionários passados como parâmetro de entrada e dos casos retornados do banco de dados. Os valores retornados será entre 0 e 1. Para o cálculo de similaridade entre 0 e 1, o *framework* levará em consideração o valor de parâmetro configurado anteriormente pelo usuário, quanto maior a distância entre os valores informados, menor será o resultado da similaridade, sendo o limite de diferença o parâmetro, neste caso, ao atingir o limite de diferença o resultado será 0, nada similar.

Quadro 13 - Código função *calcSimilarityNumericDifference*

```

285     private double calcSimilarityNumericDifference(double caseBD, double
caseInput, double parameter) {
286         double weight = 1;
287
288         if (parameter > 0) {
289             weight = (double) 1 / parameter;
290         }
291     }

```

```

292         if (caseBD < caseInput) {
293             if ((caseBD + parameter) >= caseInput) {
294                 double difference = caseInput - caseBD;
295                 return 1 - (difference * weight);
296             } else {
297                 return 0;
298             }
299         } else if (caseBD > caseInput) {
300             if ((caseBD - parameter) <= caseInput) {
301                 double difference = caseBD - caseInput;
302                 return 1 - (difference * weight);
303             } else {
304                 return 0;
305             }
306         } else {
307             return 1;
308         }
309     }

```

Fonte: Elaboração do autor, 2017.

A função *calcSimilarityNumericThreshold* faz a comparação entre números inteiros ou fracionários passados como parâmetros de entrada e dos casos retornados do banco de dados. O valor de retorno será 0 ou 1. Na configuração da estrutura do RBC, o usuário irá definir através de um parâmetro o valor da faixa limite na qual o *framework* irá considerar essa comparação como igual, ou seja, enquanto o valor passado como parâmetro de entrada estiver dentro da faixa do valor do caso somado ou diminuído pelo parâmetro configurado anteriormente, o sistema irá retornar 1, totalmente similar, caso contrário, 0 nada similar.

Quadro 14 - Código função *calcSimilarityNumericThreshold*

```

311     private double calcSimilarityNumericThreshold(double caseBD, double
caseInput,
312         double parameter) {
313         if (((caseBD - parameter) <= caseInput) && ((caseBD + parameter) >=
caseInput)) {
314             return 1;
315         } else {
316             return 0;
317         }
318     }

```

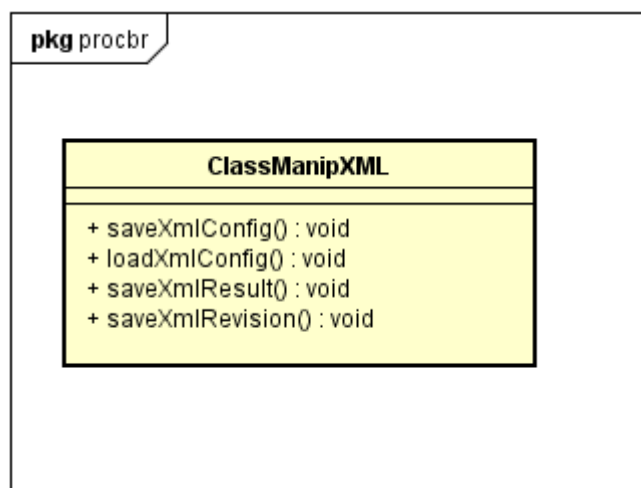
Fonte: Elaboração do autor, 2017.

4.2.2.3 Manipulação de XML

Para uma melhor extensibilidade do *framework*, foi implementada uma classe chamada ClassManipXML que possui as funções de manipulação de *xml*. Para a manipulação foi utilizada uma biblioteca de terceiros chamada de Jdom que já possui diversas funções implementadas.

O diagrama de classes da classe pode ser visto a seguir na Figura 11.

Figura 11 - Diagrama de Classes *ClassManipXML*



Fonte: Elaboração do autor, 2017.

A função *saveXmlConfig* irá salvar o XML contendo todas as informações de configuração do RBC, conexão com banco de dados, SQL utilizado na busca dos dados do banco de dados e toda a estrutura do RBC, atributos, soluções, cálculo de similaridade, peso e parâmetros.

Quadro 15 - Código função *saveXmlConfig*

```

38  public void saveXmlConfig(ClassObjConnection objConnection,
39  ClassStructureCBR structureCBR, DefaultTableModel model, int qtdResults) {
40      Document doc = new Document();
41      Element root = new Element("config");
42      //Configurações de Conexão
43      Element connection = new Element("connection");
44      Attribute database = new Attribute("database",
45  objConnection.bd.toString());
46      connection.setAttribute(database);
47      Element host = new Element("host");
48      host.setText(objConnection.connectorconfig.getHost());
49      connection.addContent(host);
50      Element dbname = new Element("dbname");
51      dbname.setText(objConnection.connectorconfig.getDbname());
52      connection.addContent(dbname);
53      Element user = new Element("user");
54      user.setText(objConnection.connectorconfig.getUser());
55      connection.addContent(user);
56      Element password = new Element("password");
  
```

```

58 password.setText(Cripto(objConnection.connectorconfig.getPassword()));
59 connection.addContent(password);
60 Element port = new Element("port");
61 port.setText(objConnection.connectorconfig.getPort());
62 connection.addContent(port);
63
64 root.addContent(connection);
65
66 //Configurações da Estrutura do RBC
67 Element structure = new Element("structurecbr");
68
69 Element sql = new Element("sql");
70 Element sqlChild = new Element("sql");
71 sqlChild.setText(structureCBR.getSql());
72 sql.addContent(sqlChild);
73 structure.addContent(sql);
74
75 Element nrResults = new Element("quantityResults");
76 Element nrResultsChild = new Element("quantityResults");
77 nrResultsChild.setText(String.valueOf(qtdResults));
78 nrResults.addContent(nrResultsChild);
79 structure.addContent(nrResults);
80
81 for (int i = 0; i <= model.getRowCount() - 1; i++) {
82     Element attribute = new Element("attribute" + i);
83
84     Element nome = new Element("nome");
85     nome.setText(model.getValueAt(i, 0).toString());
86     attribute.addContent(nome);
87
88     Element tipoDado = new Element("tipodado");
89     tipoDado.setText(model.getValueAt(i, 1).toString());
90     attribute.addContent(tipoDado);
91
92     Element calculosimilaridade = new Element("calculosimilaridade");
93     calculosimilaridade.setText(model.getValueAt(i, 2).toString());
94     attribute.addContent(calculosimilaridade);
95
96     Element peso = new Element("peso");
97     peso.setText(model.getValueAt(i, 3).toString());
98     attribute.addContent(peso);
99
100    Element parametro = new Element("parametro");
101    parametro.setText(model.getValueAt(i, 4).toString());
102    attribute.addContent(parametro);
103
104    Element atributo = new Element("atributo");
105    atributo.setText(model.getValueAt(i, 5).toString());
106    attribute.addContent(atributo);
107
108    Element solucao = new Element("solucao");
109    solucao.setText(model.getValueAt(i, 6).toString());
110    attribute.addContent(solucao);
111
112    structure.addContent(attribute);
113 }
114
115 root.addContent(structure);
116 doc.setRootElement(root);
117
118 XMLOutputter xout = new XMLOutputter();
119 OutputStream out = null;
120 try {
121     out = new FileOutputStream(new
122 File("config.xml").getAbsolutePath());
123 } catch (FileNotFoundException ex) {

```

```

123         Logger.getLogger(ClassManipXML.class.getName()).log(Level.SEVERE,
null, ex);
124     }
125     try {
126         xout.output(doc, out);
127     } catch (IOException ex) {
128         Logger.getLogger(ClassManipXML.class.getName()).log(Level.SEVERE,
null, ex);
129     }
130
131 }

```

Fonte: Elaboração do autor, 2017.

A função *loadXmlConfig* fará a leitura do arquivo de configuração, para que dessa forma, o *framework* consiga conectar e estruturar o RBC ao ser chamado por uma aplicação externa.

Quadro 16 - Código função *loadXmlConfig*

```

133     public void loadXmlConfig(ClassObjConnection objConnection,
ClassStructureCBR structureCBR, DefaultTableModel model, ClassSimilarity
similarity) {
134         int i = 0;
135         //File file = new File("config.xml").getAbsolutePath();
136         String className = "/" + this.getClass().getName().replace('.', '/')
+ ".class";
137         String path = this.getClass().getResource(className).toString();
138         try {
139             path = java.net.URLDecoder.decode(path, "UTF-8");
140         } catch (UnsupportedEncodingException ex) {
141             Logger.getLogger(ClassManipXML.class.getName()).log(Level.SEVERE,
null, ex);
142         }
143         path = String.valueOf(path.toCharArray(), path.indexOf("file:") +
6, path.length() - (path.indexOf("file:") + 6));
144         if (path.indexOf("ProCBR.jar") > -1) {
145             path = String.valueOf(path.toCharArray(), 0, path.length() -
(path.length() - path.indexOf("ProCBR.jar")));
146         } else {
147             path = String.valueOf(path.toCharArray(), 0, path.length() -
(path.length() - (path.indexOf("ProCBR") + 6)));
148         }
149         path += "/config.xml";
150         File file = new File(path);
151         SAXBuilder sb = new SAXBuilder();
152
153         try {
154             Document doc = sb.build(file);
155             Element root = doc.getRootElement();
156             List elements = root.getChildren();
157             Iterator iterator = elements.iterator();
158             while (iterator.hasNext()) {
159                 if (i == 0) {
160                     Element element = (Element) iterator.next();
161                     objConnection.bd = ClassObjConnection.BD.valueOf(
element.getAttributeValue("database"));
162                     objConnection.connection(element.getChildText("host"),
element.getChildText("dbname"),
element.getChildText("user"),
163                     Decripto(element.getChildText("password")),

```

```

167         element.getChildText("port"));
168     } else {
169         Element element = (Element) iterator.next();
170         List elementsStructure = element.getChildren();
171         Iterator iteratorStructure =
elementsStructure.iterator();
172         int contAtrib = 0;
173         while (iteratorStructure.hasNext()) {
174             Element elementAtributes = (Element)
iteratorStructure.next();
175             if (contAtrib == 0) {
176                 structureCBR.setSql(elementAtributes.getChildText("sql"));
177                 contAtrib++;
178             } else if (contAtrib == 1) {
179                 similarity.setQtdResults(Integer.parseInt(
180 elementAtributes.getChildText("quantityResults")));
181                 contAtrib++;
182             } else {
183                 model.addRow(new Object[]{
184                     elementAtributes.getChildText("nome"),
185                     elementAtributes.getChildText("tipodado"),
186                     elementAtributes.getChildText("calculosimilaridade"),
187                     elementAtributes.getChildText("peso"),
188                     elementAtributes.getChildText("parametro"),
189                     new
Boolean(elementAtributes.getChildText("atributo")),
190                     new
Boolean(elementAtributes.getChildText("solucao"))});
191             }
192         }
193     }
194     i++;
195 }
196 } catch (JDOMException ex) {
197     Logger.getLogger(ClassManipXML.class.getName()).log(Level.SEVERE,
null, ex);
198 } catch (IOException ex) {
199     Logger.getLogger(ClassManipXML.class.getName()).log(Level.SEVERE,
null, ex);
200 }
201 }
202 }

```

Fonte: Elaboração do autor, 2017.

A função *saveXmlResult* irá salvar o arquivo contendo os resultados da operação de cálculo de similaridade. Este XML irá conter todas as informações necessárias para que sejam tratadas posteriormente na aplicação externa que estiver chamando o *framework*, são elas, os atributos passados como parâmetro de entrada e todos os casos ordenados de forma decrescente, cada um contendo os seus atributos, soluções, e o valor da similaridade calculado. A quantidade de registros retornados no XML são previamente configurados.

Quadro 17 - Código função saveXmlResult

```

203     public void saveXmlResult(ArrayList<String> arrayAtrib, ArrayList<String>
arraySolut,
204         ArrayList<String> arrayAtribResult, ArrayList<String>
arraySolutResult,
205         ArrayList<Double> arraySimil, String[] arrayInput) {
206         Document doc = new Document();
207
208         Element root = new Element("results");
209
210         //Dados de Entrada
211         Element inputData = new Element("inputData");
212         for (int i = 0; i <= arrayInput.length - 1; i++) {
213             Element temp = new Element(arrayAtrib.get(i));
214             temp.setText(arrayInput[i]);
215             inputData.addContent(temp);
216         }
217         root.addContent(inputData);
218
219         //Casos
220         Element cases = new Element("cases");
221         for (int i = 0; i <= arraySolutResult.size() - 1; i++) {
222             Element tempCase = new Element("case" + (i + 1));
223
224             Element attributes = new Element("attributes");
225             String strArrTemp[] = arrayAtribResult.get(i).split(";");
226             for (int j = 0; j <= arrayAtrib.size() - 1; j++) {
227                 Element tempAttributes = new Element(arrayAtrib.get(j));
228                 tempAttributes.setText(strArrTemp[j]);
229                 attributes.addContent(tempAttributes);
230             }
231             tempCase.addContent(attributes);
232
233             Element solutions = new Element("solutions");
234             String strArrTemp2[] = arraySolutResult.get(i).split(";");
235             for (int j = 0; j <= arraySolut.size() - 1; j++) {
236                 Element tempSolutions = new Element(arraySolut.get(j));
237                 tempSolutions.setText(strArrTemp2[j]);
238                 solutions.addContent(tempSolutions);
239             }
240             tempCase.addContent(solutions);
241
242             Element similarity = new Element("similarity");
243             similarity.setText(String.format("%.2f", arraySimil.get(i)));
244             tempCase.addContent(similarity);
245
246             cases.addContent(tempCase);
247         }
248         root.addContent(cases);
249         doc.setRootElement(root);
250
251         XMLOutputter xout = new XMLOutputter();
252
253         OutputStream out = null;
254         try {
255             String className = "/" + this.getClass().getName().replace('.',
"/") + ".class";
256             String path = this.getClass().getResource(className).toString();
257             try {
258                 path = java.net.URLDecoder.decode(path, "UTF-8");
259             } catch (UnsupportedEncodingException ex) {
260
261             }
262             Logger.getLogger(ClassManipXML.class.getName()).log(Level.SEVERE, null, ex);
263         }

```

```

262         path = String.copyValueOf(path.toCharArray(),
path.indexOf("file:") + 6, path.length() - (path.indexOf("file:") + 6));
263
264         if (path.indexOf("ProCBR.jar") > -1) {
265             path = String.copyValueOf(path.toCharArray(), 0,
path.length() - (path.length() - path.indexOf("ProCBR.jar")));
266         } else {
267             path = String.copyValueOf(path.toCharArray(), 0,
path.length() - (path.length() - (path.indexOf("ProCBR") + 6)));
268         }
269         path += "/result.xml";
270         File file = new File(path);
271         out = new FileOutputStream(file);
272     } catch (FileNotFoundException ex) {
273         Logger.getLogger(ClassManipXML.class.getName()).log(Level.SEVERE,
null, ex);
274     }
275     try {
276         xout.output(doc, out);
277     } catch (IOException ex) {
278         Logger.getLogger(ClassManipXML.class.getName()).log(Level.SEVERE,
null, ex);
279     }
280 }

```

Fonte: Elaboração do autor, 2017.

A função *saveXmlRevision* irá salvar em arquivo os dados informados com a revisão, para que dessa forma, o usuário consiga reter em seu banco de dados os dados revisados.

Quadro 18 - Código função *saveXmlRevision*

```

282     public void saveXmlRevision(String input, DefaultTableModel model) {
283         Document doc = new Document();
284
285         String arrayInput[] = input.split(";");
286
287         Element root = new Element("revision");
288
289         for (int i = 0; i <= model.getRowCount() - 1; i++) {
290             Element tempElement = new Element(model.getValueAt(i,
0).toString());
291             tempElement.setText(arrayInput[i]);
292             root.addContent(tempElement);
293         }
294
295         doc.setRootElement(root);
296
297         XMLOutputter xout = new XMLOutputter();
298
299         OutputStream out = null;
300         try {
301             String className = "/" + this.getClass().getName().replace('.',
'/') + ".class";
302             String path = this.getClass().getResource(className).toString();
303             try {
304                 path = java.net.URLDecoder.decode(path, "UTF-8");
305             } catch (UnsupportedEncodingException ex) {
306
307             }
308             Logger.getLogger(ClassManipXML.class.getName()).log(Level.SEVERE, null, ex);

```

```

307         }
308         path = String.copyValueOf(path.toCharArray(),
path.indexOf("file:") + 6, path.length() - (path.indexOf("file:") + 6));
309         if (path.indexOf("ProCBR.jar") > -1) {
310             path = String.copyValueOf(path.toCharArray(), 0,
path.length() - (path.length() - path.indexOf("ProCBR.jar")));
311         } else {
312             path = String.copyValueOf(path.toCharArray(), 0,
path.length() - (path.length() - (path.indexOf("ProCBR") + 6)));
313         }
314         path += "/revision.xml";
315         File file = new File(path);
316         out = new FileOutputStream(file);
317     } catch (FileNotFoundException ex) {
318         Logger.getLogger(ClassManipXML.class.getName()).log(Level.SEVERE,
null, ex);
319     }
320     try {
321         xout.output(doc, out);
322     } catch (IOException ex) {
323         Logger.getLogger(ClassManipXML.class.getName()).log(Level.SEVERE,
null, ex);
324     }
325 }

```

Fonte: Elaboração do autor, 2017.

4.2.3 Interfaces do Sistema

Nesta seção serão apresentadas as interfaces da aplicação na qual será configurada a conexão com o banco de dados e estruturado o RBC, necessários para o funcionamento do *framework*.

4.2.3.1 Configuração de Banco de Dados

Esta tela possui todas as configurações necessárias para a conexão com o banco de dados selecionado, além disso, efetua a conexão com o banco, e caso negativo, retorna um *log* ao usuário contendo o motivo do problema ao tentar conectar.

Figura 12 - Interface para Conexão com Banco de Dados

The screenshot shows a window titled "Framework ProCBR" with a standard Windows title bar. The main content area has a title "Framework ProCBR" and a section "Selecione o Banco de Dados" with three radio buttons: "Firebird" (selected), "MySQL", and "PostgreSQL". Below this is a section "Informe os dados para efetuar a conexão" with several input fields: "Host" (localhost), "Nome BD" (D:\OneDrive\Aula BCC\TCC\TCC\Projetos Teste\Bancos de), "Usuário" (SYSDBA), "Senha" (masked with asterisks), "Porta" (3050), and "Log Conexão" (Contectado!). There is a "Testar Conexão" button below the "Log Conexão" field and a "Próximo" button in the bottom right corner.

Framework ProCBR

Selecione o Banco de Dados

☒ Firebird ☐ MySQL ☐ PostgreSQL

Informe os dados para efetuar a conexão

Host: localhost

Nome BD: D:\OneDrive\Aula BCC\TCC\TCC\Projetos Teste\Bancos de

Usuário: SYSDBA

Senha: *****

Porta: 3050

Log Conexão: Contectado!

Testar Conexão

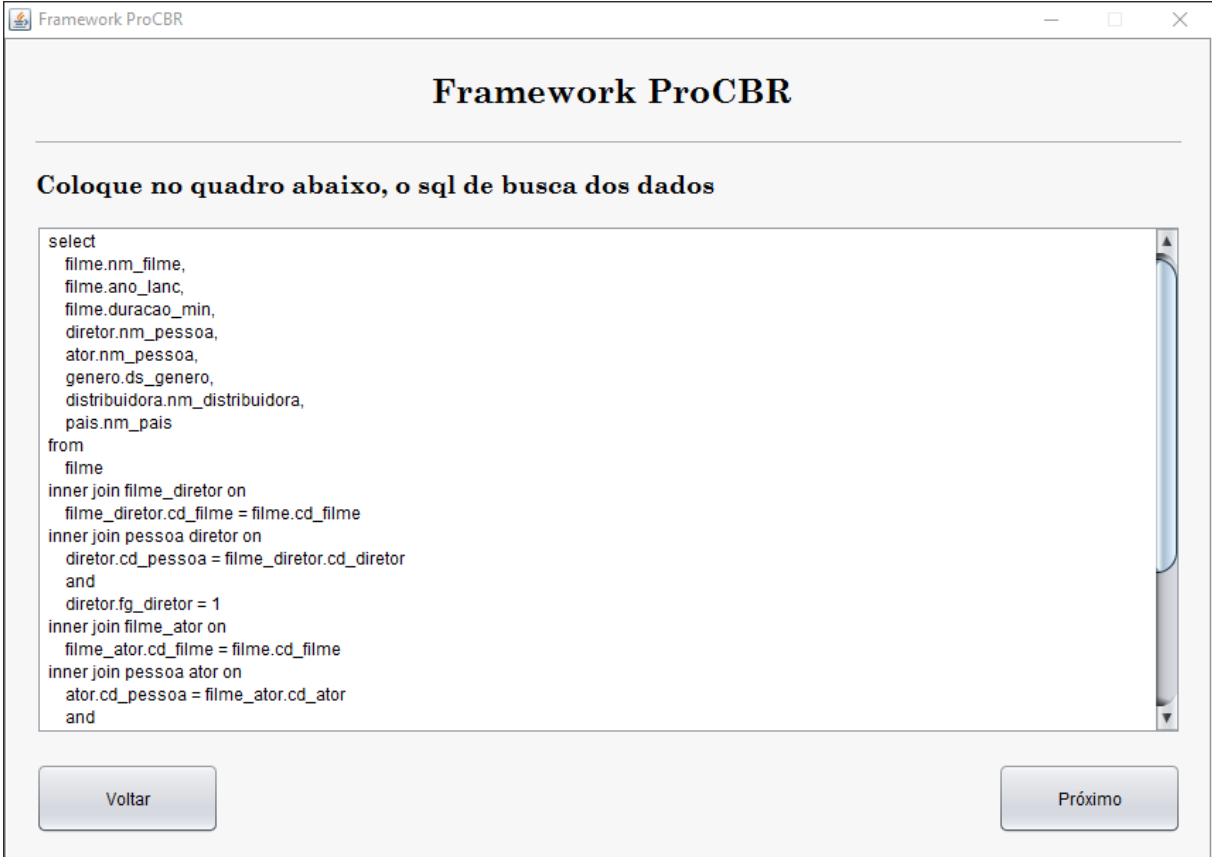
Próximo

Fonte: Elaboração do autor, 2017.

4.2.3.2 SQL para Consulta do Banco de Dados

Nesta tela será informado o SQL definido pelo usuário que efetuará a consulta ao banco de dados cada vez que for chamada a função de cálculo de similaridade, além disso, o sistema leva em consideração os campos informados na consulta como atributos a serem configurados na próxima tela, ou seja, todos os campos que serão atributos ou soluções, deverão ser previamente informados neste SQL.

Figura 13 - Interface para Configuração do SQL



The screenshot shows a window titled "Framework ProCBR" with a subtitle "Coloque no quadro abaixo, o sql de busca dos dados". Below the subtitle is a text area containing a SQL query. At the bottom of the window are two buttons: "Voltar" (Back) on the left and "Próximo" (Next) on the right.

Framework ProCBR

Coloque no quadro abaixo, o sql de busca dos dados

```
select
  filme.nm_filme,
  filme.ano_lanc,
  filme.duracao_min,
  diretor.nm_pessoa,
  ator.nm_pessoa,
  genero.ds_genero,
  distribuidora.nm_distribuidora,
  pais.nm_pais
from
  filme
inner join filme_diretor on
  filme_diretor.cd_filme = filme.cd_filme
inner join pessoa diretor on
  diretor.cd_pessoa = filme_diretor.cd_diretor
and
  diretor.fg_diretor = 1
inner join filme_ator on
  filme_ator.cd_filme = filme.cd_filme
inner join pessoa ator on
  ator.cd_pessoa = filme_ator.cd_ator
and
```

Voltar **Próximo**

Fonte: Elaboração do autor, 2017.

4.2.3.3 Estrutura do RBC

Nesta tela serão configurados todos os atributos da estrutura do RBC, seu peso, tipo, cálculo de similaridade que deverá ser utilizado, e caso possuir, os parâmetros. Também deverá ser configurado se esse campo é um atributo ou solução, sendo os atributos considerados nos cálculos de similaridades, e a solução apenas campos que retornarão nos resultados.

Será configurado ainda, a quantidade de resultados que retornarão ao usuário.

Figura 14 - Interface para Estruturação do RBC

The screenshot shows the 'Framework ProCBR' application window. It features a table with the following columns: Nome, Tipo de Dado, Cálculo de Similaridade, Peso, Parâmetro, Atributo, and Solução. The table lists several attributes with their respective data types, similarity calculations, weights, parameters, and checkboxes for attribute and solution selection. Below the table is a slider for 'Nr. de Casos no Retorno' ranging from 0 to 1000, and a 'Finalizar' button.

Nome	Tipo de Dado	Cálculo de Similaridade	Peso	Parâmetro	Atributo	Solução
nm_filme	String	Igual	5		<input type="checkbox"/>	<input checked="" type="checkbox"/>
ano_lanc	Numeric	Threshold	6	10	<input checked="" type="checkbox"/>	<input type="checkbox"/>
duracao_min	Numeric	Diferença	1	35	<input checked="" type="checkbox"/>	<input type="checkbox"/>
nm_pessoa	String	Igual (Ignore Case)	2		<input checked="" type="checkbox"/>	<input type="checkbox"/>
nm_pessoa	String	Distância de Levenshtein	2		<input checked="" type="checkbox"/>	<input type="checkbox"/>
ds_genero	String	Distância de Levenshtein	8		<input checked="" type="checkbox"/>	<input type="checkbox"/>
nm_distribuidora	String	Distância de Levenshtein	1		<input checked="" type="checkbox"/>	<input type="checkbox"/>
nm_pais	String	Contagem de Palavras	1		<input checked="" type="checkbox"/>	<input type="checkbox"/>

Nr. de Casos no Retorno: 0 100 200 300 400 500 600 700 800 900 1000

Finalizar

Fonte: Elaboração do autor, 2017.

4.2.4 Arquivos XML

Nesta seção serão apresentados os arquivos XML que guardam a configuração da estrutura do RBC, retornam os resultados aos usuários, e guardam a revisão do caso efetuada pelo usuário para uma possível retenção.

4.2.4.1 Arquivo de Configuração

Neste arquivo XML são salvas todas as configurações efetuadas pelo usuário, configurações de conexão com banco de dados, SQL de busca e Estrutura do RBC. Essa configuração deve ser efetuada antes da chamada dos métodos de cálculo de similaridade, pois, o *framework* baseia-se nele para efetuar todas as operações necessárias.

Quadro 19 - Arquivo XML de Configuração

```
<?xml version="1.0" encoding="UTF-8"?>
<config><connection
database="FIREBIRD"><host>localhost</host><dbname>D:\OneDrive\Aula
BCC\TCC\TCC\Projetos Teste\Bancos de
Dados\BANCO.FDB</dbname><user>SYSDBA</user><password>iãöøçôîçû</password>
<port>3050</port></connection><structurechr><sql><sql>select
filme.nm_filme, filme.ano_lanc, filme.duracao_min, diretor.nm_pessoa,
ator.nm_pessoa, genero.ds_genero, distribuidora.nm_distribuidora,
pais.nm_pais from filme inner join filme_diretor on
filme_diretor.cd_filme = filme.cd_filme inner join pessoa_diretor on
diretor.cd_pessoa = filme_diretor.cd_diretor and diretor.fg_diretor = 1
inner join filme_ator on filme_ator.cd_filme = filme.cd_filme inner join
pessoa_ator on ator.cd_pessoa = filme_ator.cd_ator and ator.fg_ator = 1
inner join filme_genero on filme_genero.cd_filme = filme.cd_filme inner
join genero on genero.cd_genero = filme_genero.cd_genero inner join
distribuidora on distribuidora.cd_distribuidora = filme.cd_distribuidora
inner join pais on pais.cd_pais =
filme.cd_nacionalidade</sql></sql><quantityResults><quantityResults>4</qu
antityResults></quantityResults><attribute0><nome>nm_filme</nome><tipodado
>String</tipodado><calculosimilaridade>Igual</calculosimilaridade><peso>5
</peso><parametro/><atributo>false</atributo><solucao>true</solucao></atr
ibute0><atribute1><nome>ano_lanc</nome><tipodado>Numeric</tipodado><calcu
losimilaridade>Diferença</calculosimilaridade><peso>2</peso><parametro>5<
/parametro><atributo>true</atributo><solucao>false</solucao></atribute1><
atribute2><nome>duracao_min</nome><tipodado>Numeric</tipodado><calculosim
ilaridade>Threshold</calculosimilaridade><peso>1</peso><parametro>30</par
ametro><atributo>true</atributo><solucao>false</solucao></atribute2><atri
bute3><nome>nm_pessoa</nome><tipodado>String</tipodado><calculosimilarida
de>Igual (Ignore
Case)</calculosimilaridade><peso>4</peso><parametro/><atributo>true</atri
buto><solucao>false</solucao></atribute3><atribute4><nome>nm_pessoa</nome
><tipodado>String</tipodado><calculosimilaridade>Distância de
Levenshtein</calculosimilaridade><peso>2</peso><parametro/><atributo>true
</atributo><solucao>false</solucao></atribute4><atribute5><nome>ds_genero
</nome><tipodado>Enum</tipodado><calculosimilaridade>Igual</calculosimila
ridade><peso>5</peso><parametro/><atributo>true</atributo><solucao>false<
/solucao></atribute5><atribute6><nome>nm_distribuidora</nome><tipodado>St
ring</tipodado><calculosimilaridade>Contagem de
Palavras</calculosimilaridade><peso>1</peso><parametro/><atributo>true</a
tributo><solucao>false</solucao></atribute6><atribute7><nome>nm_pais</nom
e><tipodado>String</tipodado><calculosimilaridade>Igual (Ignore
Case)</calculosimilaridade><peso>1</peso><parametro/><atributo>true</atri
buto><solucao>false</solucao></atribute7></structurechr></config>
```

Fonte: Elaboração do autor, 2017.

4.2.4.2 Arquivo de Resultados

Neste arquivo são salvos os resultados da função de cálculo de similaridade. A quantidade de casos nesse arquivo será configurada anteriormente pelo usuário.

No arquivo serão salvas todas as informações para um eventual tratamento das informações por uma aplicação externa, informações como, atributos de entrada informados, todos os casos com seus atributos, soluções e similaridade calculada.

Quadro 20 - Arquivo XML de Resultados

```
<?xml version="1.0" encoding="UTF-8"?>
<results><inputData><ano_lanc>2016</ano_lanc><duracao_min>123</duracao_min>
<nm_pessoa>David Ayer</nm_pessoa><nm_pessoa>Will
Smith</nm_pessoa><ds_genero>Ação</ds_genero><nm_distribuidora>Warner
Bros</nm_distribuidora><nm_pais>Estados
Unidos</nm_pais></inputData><cases><case1><atributes><ano_lanc>2016</ano_lanc>
<duracao_min>123</duracao_min><nm_pessoa>David
Ayer</nm_pessoa><nm_pessoa>Will
Smith</nm_pessoa><ds_genero>Ação</ds_genero><nm_distribuidora>Warner
Bros</nm_distribuidora><nm_pais>Estados
Unidos</nm_pais></atributes><solutions><nm_filme>Esquadrão
Suicida</nm_filme></solutions><similarity>100,00</similarity></case1><case2>
<atributes><ano_lanc>2012</ano_lanc><duracao_min>143</duracao_min><nm_pessoa>Joss
Whedon</nm_pessoa><nm_pessoa>Robert Downey
Jr</nm_pessoa><ds_genero>Ação</ds_genero><nm_distribuidora>Disney</nm_distribuidora>
<nm_pais>Estados
Unidos</nm_pais></atributes><solutions><nm_filme>Os
Vingadores</nm_filme></solutions><similarity>53,28</similarity></case2><case3>
<atributes><ano_lanc>2016</ano_lanc><duracao_min>110</duracao_min><nm_pessoa>David
Yates</nm_pessoa><nm_pessoa>Alexander
Skarsgard</nm_pessoa><ds_genero>Aventura</ds_genero><nm_distribuidora>Warner
Bros</nm_distribuidora><nm_pais>Estados
Unidos</nm_pais></atributes><solutions><nm_filme>A Lenda de
Tarzan</nm_filme></solutions><similarity>37,83</similarity></case3><case4>
<atributes><ano_lanc>2014</ano_lanc><duracao_min>144</duracao_min><nm_pessoa>Peter
Jackson</nm_pessoa><nm_pessoa>Ian
McKellen</nm_pessoa><ds_genero>Aventura</ds_genero><nm_distribuidora>Warner
Bros</nm_distribuidora><nm_pais>Estados
Unidos</nm_pais></atributes><solutions><nm_filme>O Hobbit A Batalha dos
Cinco
Exércitos</nm_filme></solutions><similarity>34,58</similarity></case4></cases>
</results>
```

Fonte: Elaboração do autor, 2017.

4.2.4.3 Arquivo de Revisão

Neste arquivo será salvo um caso revisado pelo usuário para uma posterior retenção, vale ressaltar, que a retenção deve ser feita pelo usuário, pois, o *framework* não possui capacidade para guardar informações no banco de dados configurado, visto que, cada banco é estruturado de forma diferente e particular do desenvolvedor.

Quadro 21 - Arquivo XML de Revisão

```
<?xml version="1.0" encoding="UTF-8"?>
<revision><nm_filme>Esquadrão
Suicida</nm_filme><ano_lanc>2016</ano_lanc><duracao_min>123</duracao_min>
<nm_pessoa>David Ayer</nm_pessoa><nm_pessoa>Will
Smith</nm_pessoa><ds_genero>Ação</ds_genero><nm_distribuidora>Warner
Bros</nm_distribuidora><nm_pais>Estados Unidos</nm_pais></revision>
```

Fonte: Elaboração do autor, 2017.

5 RESULTADOS

Com o intuito de validar as funcionalidades desenvolvidas no *framework*, foram elaborados diversos testes separados em quatro seções, testes de conexão com diferentes bancos de dados, testes de resultado de cálculos de similaridade, testes de comunicação com diferentes tecnologias de programação e teste de revisão de resultados. Todos os testes foram feitos com a estrutura do RBC sendo configurada para simular um recomendador de filmes com base nos atributos informados, dessa forma, tanto os bancos de dados, quanto a aplicação externa de teste, foram modelados com base nessa estrutura.

Para testes de conexão com bancos, foram criados três bancos de dados contendo a mesma estrutura e cada um deles desenvolvido com uma tecnologia diferente conforme proposto neste trabalho.

O banco de dados Firebird foi escolhido para testes de cálculo de similaridade, dessa forma, ele contém mais dados preenchidos. Ainda para os testes de cálculo de similaridade, serão testados diferentes atributos de entrada.

Para validar ainda a proposta de que diferentes tecnologias de linguagens de programação consigam comunicar-se com o *framework*, os testes foram realizados utilizando a chamada do *framework* via *prompt de comando* e desenvolvido também, uma aplicação na linguagem *Object Pascal* utilizando a *IDE Delphi XE6*. Após a chamada do *framework*, essa aplicação irá ler o retorno salvo em arquivo XML e mostrar através de um *grid* aos usuários.

Por fim, a estrutura da chamada do *framework* pelo *Prompt de Comando* é “java – jar “Caminho do *framework*” *arg1 arg2*”, sendo *arg1* os atributos de entrada e *arg2* que não é obrigatório informar, deve ser informado a palavra *revision* caso deseje gravar um arquivo xml com os dados para posterior retenção. Conforme apresentado anteriormente, os argumentos de atributos de entrada devem respeitar a estrutura de atributos definida, assim como sua ordem, e os campos separados por “;”, no quadro 22 a seguir, podemos acompanhar um exemplo de chamada do *framework* com atributos via *prompt de comando*.

Quadro 22 - Exemplo chamada do *framework* via *prompt de comando*

```
java -jar "D:\OneDrive\Aula BCC\TCC\TCC\ProCBR\dist\ProCBR.jar" "2016;123;David
Ayer;Will Smith;Ação;Warner Bros;Estados Unidos"
```

Fonte: Elaboração do autor, 2017.

5.1 TESTES DE CONEXÃO COM DIFERENTES TECNOLOGIAS DE BANCOS DE DADOS

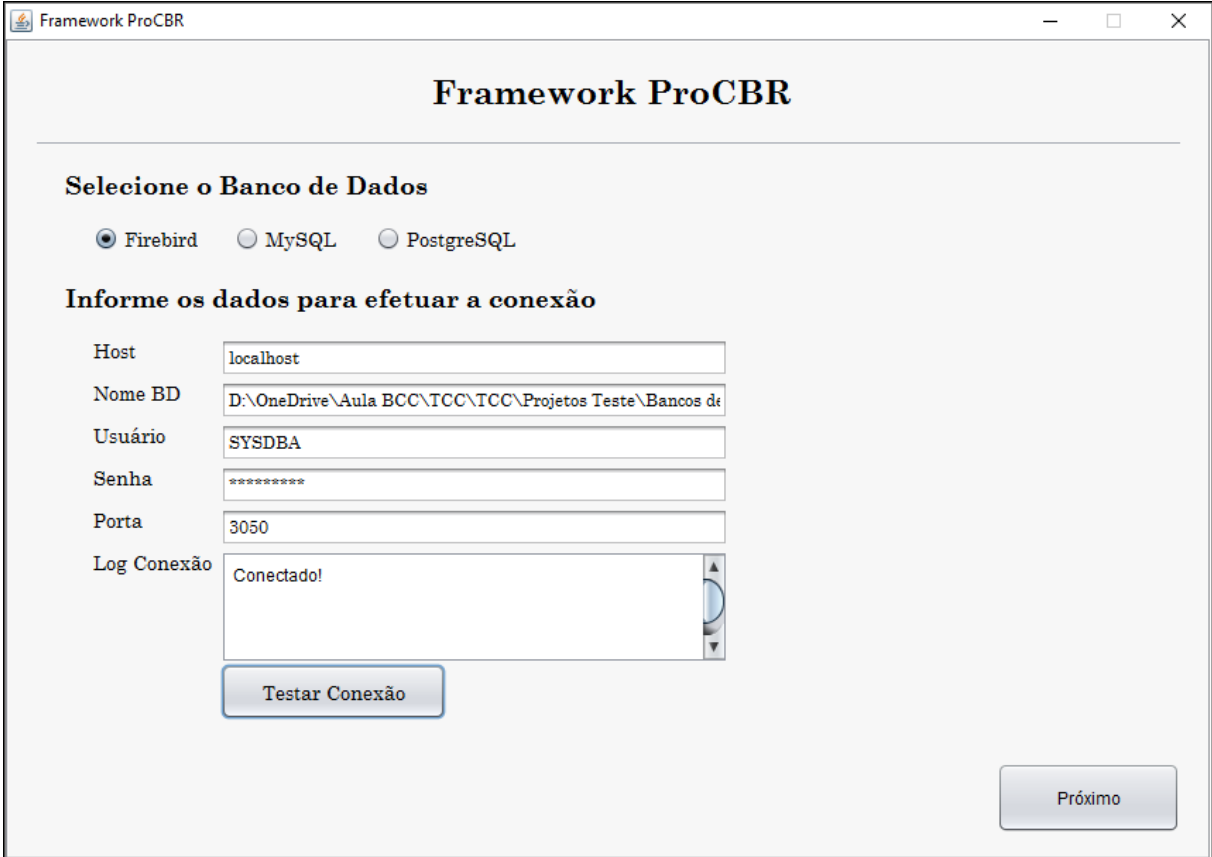
A conexão com diferentes tecnologias de banco de dados possui duas particularidades. A conexão em Java é feita com bibliotecas de terceiros desenvolvidas pelas próprias fabricantes do banco de dados utilizando um padrão, dessa forma, as bibliotecas de conexão sendo padronizadas, a biblioteca *DriverManager* disponível com o próprio Java consegue efetuar a conexão de forma idêntica em todas as tecnologias, necessitando apenas que o desenvolvedor informe a biblioteca de conexão com o banco que deve ser utilizado.

A segunda particularidade é referente a estrutura do RBC, conforme visto em seções anteriores, o *framework* pelo SQL informado pelo usuário, irá tratar os campos que serão definidos como atributos ou soluções posteriormente pelo usuário, dessa forma, como cada tecnologia de banco possui particularidades em seu SQL, o tratamento feito pelo *framework* é diferente em cada uma dessas tecnologias.

5.1.1 Teste 1 – Conexão com banco de dados Firebird

A Figura 15 a seguir, mostra as informações passadas ao *framework* para efetuar uma conexão com sucesso com o banco de dados Firebird.

Figura 15 - Conexão com banco de dados Firebird



The screenshot shows a window titled "Framework ProCBR" with a standard Windows title bar. The main content area has a title "Framework ProCBR" and a section "Selecione o Banco de Dados" with three radio buttons: "Firebird" (selected), "MySQL", and "PostgreSQL". Below this is a section "Informe os dados para efetuar a conexão" with several input fields: "Host" (localhost), "Nome BD" (D:\OneDrive\Aula BCC\TCC\TCC\Projetos Teste\Bancos de), "Usuário" (SYSDBA), "Senha" (masked with asterisks), "Porta" (3050), and "Log Conexão" (Contectado!). A "Testar Conexão" button is located below the "Log Conexão" field. A "Próximo" button is located at the bottom right of the window.

Framework ProCBR

Selecione o Banco de Dados

☒ Firebird ☐ MySQL ☐ PostgreSQL

Informe os dados para efetuar a conexão

Host: localhost

Nome BD: D:\OneDrive\Aula BCC\TCC\TCC\Projetos Teste\Bancos de

Usuário: SYSDBA

Senha: *****

Porta: 3050

Log Conexão: Contectado!

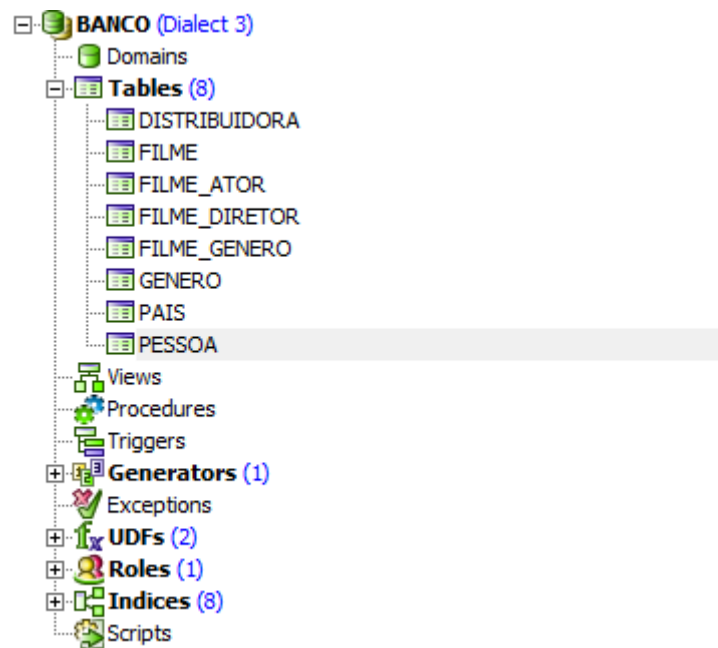
Testar Conexão

Próximo

Fonte: Elaboração do autor, 2017.

A Figura 16 a seguir, mostra a estrutura do banco de dados. O gerenciador de banco de dados utilizado foi o IBExpert.

Figura 16 - Estrutura do banco de dados Firebird



Fonte: Elaboração do autor, 2017.

O Quadro 23 mostra o SQL utilizado na configuração da estrutura do RBC.

Quadro 23 - SQL Firebird

```
select
    filme.nm_filme,
    filme.ano_lanc,
    filme.duracao_min,
    diretor.nm_pessoa,
    ator.nm_pessoa,
    genero.ds_genero,
    distribuidora.nm_distribuidora,
    pais.nm_pais
from
    filme
inner join filme_diretor on
    filme_diretor.cd_filme = filme.cd_filme
inner join pessoa diretor on
    diretor.cd_pessoa = filme_diretor.cd_diretor
    and
    diretor.fg_diretor = 1
inner join filme_ator on
    filme_ator.cd_filme = filme.cd_filme
inner join pessoa ator on
    ator.cd_pessoa = filme_ator.cd_ator
    and
    ator.fg_ator = 1
inner join filme_genero on
    filme_genero.cd_filme = filme.cd_filme
inner join genero on
    genero.cd_genero = filme_genero.cd_genero
```

```

inner join distribuidora on
    distribuidora.cd_distribuidora = filme.cd_distribuidora
inner join pais on
    pais.cd_pais = filme.cd_nacionalidade

```

Fonte: Elaboração do autor, 2017.

5.1.2 Teste 2 – Conexão com banco de dados MySQL

A Figura 17 a seguir, mostra as informações passadas ao *framework* para efetuar uma conexão com sucesso com o banco de dados MySQL.

Figura 17 - Conexão com banco de dados MySQL

Framework ProCBR

Framework ProCBR

Selecione o Banco de Dados

☐ Firebird ☒ MySQL ☐ PostgreSQL

Informe os dados para efetuar a conexão

Host: localhost

Nome BD: banco

Usuário: root

Senha: *****

Porta: 8080

Log Conexão: Conectado!

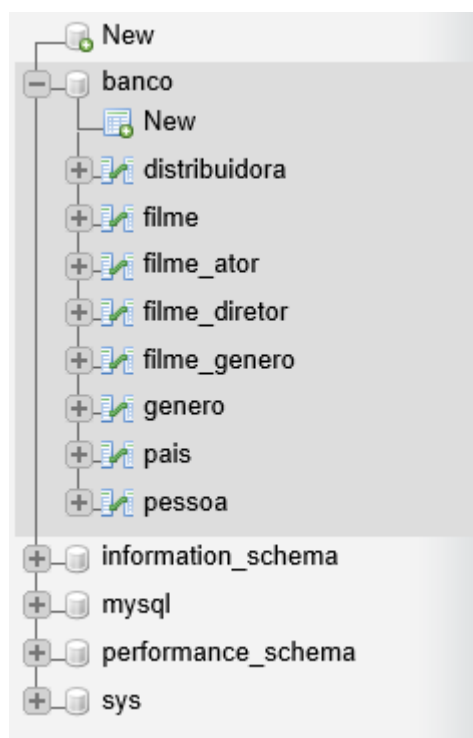
Testar Conexão

Próximo

Fonte: Elaboração do autor, 2017.

A Figura 18 a seguir, mostra a estrutura do banco de dados. O gerenciador de banco de dados utilizado foi o phpmyadmin do servidor wamp.

Figura 18 - Estrutura do banco de dados MySQL



Fonte: Elaboração do autor, 2017.

O Quadro 24 mostra o SQL utilizado na configuração da estrutura do RBC.

Quadro 24 - SQL MySQL

```
select
    filme.nm_filme,
    filme.ano_lanc,
    filme.duracao_min,
    diretor.nm_pessoa,
    ator.nm_pessoa,
    genero.ds_genero,
    distribuidora.nm_distribuidora,
    pais.nm_pais
from
    filme
inner join filme_diretor on
    filme_diretor.cd_filme = filme.cd_filme
inner join pessoa diretor on
    diretor.cd_pessoa = filme_diretor.cd_diretor
and
    diretor.fg_diretor = 1
inner join filme_ator on
    filme_ator.cd_filme = filme.cd_filme
inner join pessoa ator on
    ator.cd_pessoa = filme_ator.cd_ator
and
    ator.fg_ator = 1
inner join filme_genero on
```

```

filme_genero.cd_filme = filme.cd_filme
inner join genero on
    genero.cd_genero = filme_genero.cd_genero
inner join distribuidora on
    distribuidora.cd_distribuidora = filme.cd_distribuidora
inner join pais on
    pais.cd_pais = filme.cd_nacionalidade

```

Fonte: Elaboração do autor, 2017.

5.1.3 Teste 3 – Conexão com banco de dados PostgreSQL

A Figura 19 a seguir, mostra as informações passadas ao *framework* para efetuar uma conexão com sucesso com o banco de dados PostgreSQL.

Figura 19 - Conexão com banco de dados PostgreSQL

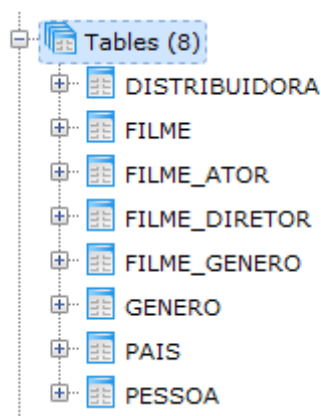
The screenshot shows a window titled "Framework ProCBR" with a standard Windows title bar. The main content area has a light gray background and contains the following elements:

- Selezione o Banco de Dados:** Three radio buttons are present: "Firebird", "MySQL", and "PostgreSQL". The "PostgreSQL" option is selected.
- Informe os dados para efetuar a conexão:** A section with several input fields:
 - Host:** "localhost"
 - Nome BD:** "postgres"
 - Usuário:** "postgres"
 - Senha:** "*****"
 - Porta:** "5432"
 - Log Conexão:** A text area containing "Conectado!" with a vertical scrollbar on the right.
- Testar Conexão:** A button located below the "Log Conexão" field.
- Próximo:** A button located in the bottom right corner of the window.

Fonte: Elaboração do autor, 2017.

A Figura 20 a seguir, mostra a estrutura do banco de dados. O gerenciador de banco de dados utilizado foi o pgAdmin.

Figura 20 - Estrutura do banco de dados PostgreSQL



Fonte: Elaboração do autor, 2017.

O Quadro 25 mostra o SQL utilizado na configuração da estrutura do RBC.

Quadro 25 - SQL PostgreSQL

```
SELECT
    public."FILME"."NM_FILME",
    public."FILME"."ANO_LANC",
    public."FILME"."DURACAO_MIN",
    DIRETOR."NM_PESSOA",
    ATOR."NM_PESSOA",
    public."GENERO"."DS_GENERO",
    public."DISTRIBUIDORA"."NM_DISTRIBUIDORA",
    public."PAIS"."NM_PAIS"
FROM
    public."FILME"
INNER JOIN public."FILME_DIRETOR" ON
    public."FILME_DIRETOR"."CD_FILME" = public."FILME"."CD_FILME"
INNER JOIN public."PESSOA" DIRETOR ON
    DIRETOR."CD_PESSOA" = public."FILME_DIRETOR"."CD_DIRETOR"
    AND
    DIRETOR."FG_DIRETOR" = 1
INNER JOIN public."FILME_ATOR" ON
    public."FILME_ATOR"."CD_FILME" = public."FILME"."CD_FILME"
INNER JOIN public."PESSOA" ATOR ON
    ATOR."CD_PESSOA" = public."FILME_ATOR"."CD_ATOR"
    AND
    ATOR."FG_ATOR" = 1
INNER JOIN public."FILME_GENERO" ON
    public."FILME_GENERO"."CD_FILME" = public."FILME"."CD_FILME"
INNER JOIN public."GENERO" ON
    public."GENERO"."CD_GENERO" = public."FILME_GENERO"."CD_GENERO"
INNER JOIN public."DISTRIBUIDORA" ON
    public."DISTRIBUIDORA"."CD_DISTRIBUIDORA" = public."FILME"."CD_DISTRIBUIDORA"
INNER JOIN public."PAIS" ON
    public."PAIS"."CD_PAIS" = public."FILME"."CD_NACIONALIDADE"
```

Fonte: Elaboração do autor, 2017.

5.1.4 Resultados testes de conexão

A seguir na Figura 21, podemos verificar que o *framework* definiu os campos da estrutura do RBC com base nos campos selecionados no SQL. Os campos retornados do SQL são iguais independente da estrutura de banco utilizada.

Figura 21 - Estrutura RBC com base no SQL

Nome	Tipo de Dado	Cálculo de Similaridade	Peso	Parâmetro	Atributo	Solução
nm_filme	String	Igual	5		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
ano_lanc	String	Igual	5		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
duracao_min	String	Igual	5		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
nm_pessoa	String	Igual	5		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
nm_pessoa	String	Igual	5		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
ds_genero	String	Igual	5		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
nm_distribuidora	String	Igual	5		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
nm_pais	String	Igual	5		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Fonte: Elaboração do autor, 2017.

Por fim, podemos acompanhar na Figura 22, o resultado da execução via *Prompt de Comando*. Para o retorno foi utilizado um arquivo de configuração, na qual, estava configurado para retornar apenas 4 resultados. Da mesma forma que a estrutura do RBC, o retorno da execução é idêntico independente da tecnologia de banco de dados utilizada.

Figura 22 - Retorno execução do *framework*

```

C:\Users\Fernando>java -jar "D:\OneDrive\Aula BCC\TCC\TCC\ProCBR\dist\ProCBR.jar"
"2016;123;David Ayer;Will Smith;Acção;Warner Bros;Estados Unidos"
Esquadrão Suicida;100,00
Os Vingadores;53,28
A Lenda de Tarzan;37,83
O Hobbit A Batalha dos Cinco Exércitos;34,58

```

Fonte: Elaboração do autor, 2017.

5.2 TESTES DE RESULTADOS DE CÁLCULOS DE SIMILARIDADE

Os testes a seguir foram realizados para verificar a integridade dos valores de similaridade calculados pelo *framework*.

Conforme mencionado anteriormente, a estrutura do RBC está configurada para simular um programa recomendador de filmes com base em alguns atributos.

Todos os testes a seguir foram realizados chamando o *framework* pelo *prompt de comando*. Na configuração foi definido um retorno dos cinco melhores resultados, e a solução será sempre o nome do filme.

A tabela 1 a seguir, mostra as configurações dos atributos utilizada nos testes.

Tabela 1 - Configurações da estrutura do RBC utilizadas nos testes

Nome	Tipo Dado	Calc. Similaridade	Peso	Parâmetro
Ano Lançamento	Numeric	Diferença	2	5
Duração	Numeric	Threshold	1	30
Diretor	String	Igual (Ignore Case)	4	
Ator	String	Distância de Levenshtein	2	
Gênero	Enum	Igual	5	
Distribuidora	String	Contagem de Palavras	1	
País	String	Igual (Ignore Case)	1	

Fonte: Elaboração do autor, 2017.

5.2.1 Teste 1 – 100% de similaridade

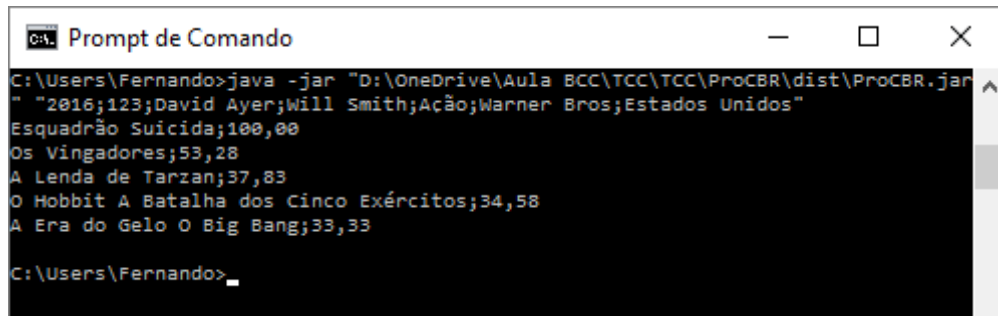
A Tabela 2 a seguir, mostra os atributos de entrada utilizados no Teste 1.

Tabela 2 - Atributos de entrada Teste 1

Nome Atributo	Valor do Atributo
Ano Lançamento	2016
Duração	123
Diretor	David Ayer
Ator	Will Smith
Gênero	Ação
Distribuidora	Warner Bros
País	Estados Unidos

Fonte: Elaboração do autor, 2017.

Figura 23 - Resultado Teste 1



```

C:\Users\Fernando>java -jar "D:\OneDrive\Aula BCC\TCC\TCC\ProCBR\dist\ProCBR.jar"
" 2016;123;David Ayer;Will Smith;Ação;Warner Bros;Estados Unidos"
Esquadrão Suicida;100,00
Os Vingadores;53,28
A Lenda de Tarzan;37,83
O Hobbit A Batalha dos Cinco Exércitos;34,58
A Era do Gelo O Big Bang;33,33
C:\Users\Fernando>

```

Fonte: Elaboração do autor, 2017.

Conforme podemos observar na figura 23, o primeiro filme do resultado está com 100% de similaridade. Os atributos de entrada que foram passados, são exatamente os mesmos cadastrados no banco para esse filme, dessa forma, o retorno de 100% de similaridade está correto.

5.2.2 Teste 2 – Distância de Levenshtein

Tabela 3 - Atributos de entrada Teste 2

Nome Atributo	Valor do Atributo
Ano Lançamento	2016
Duração	123
Diretor	David Ayer
Ator	João Smith
Gênero	Ação
Distribuidora	Warner Bros
País	Estados Unidos

Fonte: Elaboração do autor, 2017.

Figura 24 - Resultado Teste 2

```

C:\Users\Fernando>java -jar "D:\OneDrive\Aula BCC\TCC\TCC\ProCBR\dist\ProCBR.jar"
"2016;123;David Ayer;João Smith;Ação;Warner Bros;Estados Unidos"
Esquadrão Suicida;92,50
Os Vingadores;53,28
A Lenda de Tarzan;37,83
O Hobbit A Batalha dos Cinco Exércitos;36,67
A Era do Gelo O Big Bang;33,33

```

Fonte: Elaboração do autor, 2017.

No teste 2 no atributo de entrada “ator”, foi passado “João Smith”, ao invés de “Will Smith” como era o esperado, dessa forma, utilizando a função *Distância de Levenshtein*, o *framework* teve de efetuar quatro trocas de letras para chegar a palavra “Will”, sendo assim, diminuiu em 7,5% a similaridade, com um resultado de 92.5%.

5.2.3 Teste 3 – Contagem de Palavras

Tabela 4 - Atributos de entrada Teste 3

Nome Atributo	Valor do Atributo
Ano Lançamento	2016
Duração	123
Diretor	David Ayer
Ator	João Smith
Gênero	Ação
Distribuidora	Alamo
País	Estados Unidos

Fonte: Elaboração do autor, 2017.

Figura 25 - Resultado Teste 3

```

C:\Users\Fernando>java -jar "D:\OneDrive\Aula BCC\TCC\TCC\ProCBR\dist\ProCBR.jar"
" 2016;123;David Ayer;João Smith;Ação;Alamo;Estados Unidos"
Esquadrão Suicida;86,25
Os Vingadores;53,28
A Era do Gelo O Big Bang;33,33
A Lenda de Tarzan;31,58
O Hobbit A Batalha dos Cinco Exércitos;30,42
C:\Users\Fernando>

```

Fonte: Elaboração do autor, 2017.

Neste teste foram alterados dois atributos, conforme teste anterior, foi mantido o atributo de entrada “ator” como “João Smith”, e alterado também, o atributo “Distribuidora” de “Warner Bros” para “Alamo”. No atributo “Distribuidora” foi utilizado o cálculo de similaridade *contagem de palavras*, dessa forma, o *framework* verificou se a palavra informada era idêntica a palavra informada no banco de dados. Como o resultado é negativo, isso diminuiu ainda mais a similaridade com relação ao teste 02, perdendo agora, 13,75% de similaridade, resultando em 86,25%.

5.2.4 Teste 4 – Threshold

Tabela 5 - Atributos de entrada Teste 4

Nome Atributo	Valor do Atributo
Ano Lançamento	2016
Duração	100
Diretor	David Ayer
Ator	Will Smith
Gênero	Ação
Distribuidora	Warner Bros
País	Estados Unidos

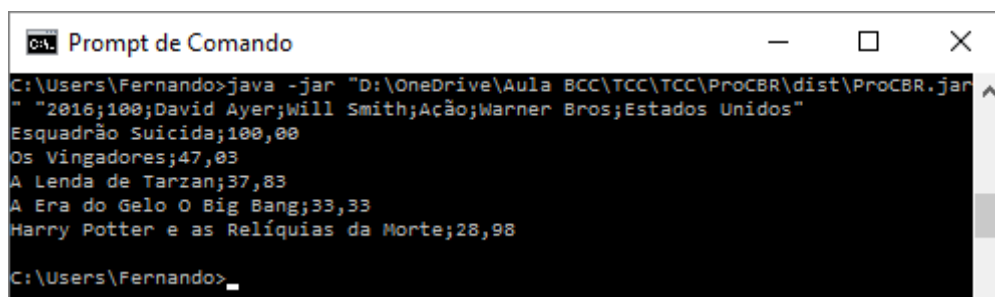
Fonte: Elaboração do autor, 2017.

Tabela 6 - Atributos de entrada Teste 4

Nome Atributo	Valor do Atributo
Ano Lançamento	2016
Duração	90
Diretor	David Ayer
Ator	Will Smith
Gênero	Ação
Distribuidora	Warner Bros
País	Estados Unidos

Fonte: Elaboração do autor, 2017.

Figura 26 - Resultado Teste 4



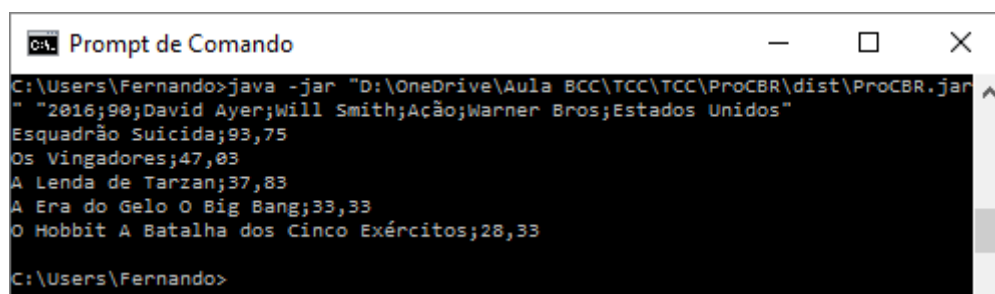
```

C:\Users\Fernando>java -jar "D:\OneDrive\Aula BCC\TCC\TCC\ProCBR\dist\ProCBR.jar"
" "2016;100;David Ayer;Will Smith;Ação;Warner Bros;Estados Unidos"
Esquadrão Suicida;100,00
Os Vingadores;47,03
A Lenda de Tarzan;37,83
A Era do Gelo O Big Bang;33,33
Harry Potter e as Relíquias da Morte;28,98
C:\Users\Fernando>

```

Fonte: Elaboração do autor, 2017.

Figura 27 - Resultado Teste 4



```

C:\Users\Fernando>java -jar "D:\OneDrive\Aula BCC\TCC\TCC\ProCBR\dist\ProCBR.jar"
" "2016;90;David Ayer;Will Smith;Ação;Warner Bros;Estados Unidos"
Esquadrão Suicida;93,75
Os Vingadores;47,03
A Lenda de Tarzan;37,83
A Era do Gelo O Big Bang;33,33
O Hobbit A Batalha dos Cinco Exércitos;28,33
C:\Users\Fernando>

```

Fonte: Elaboração do autor, 2017.

Para o teste da função *Threshold* foram feitos dois testes conforme Figura 26 e 27. No primeiro teste foi passado uma duração do filme de 100 minutos, como o filme é 123

minutos e o parâmetro de configuração desse atributo é de 30 minutos, a similaridade desse atributo ainda é de 100%, dessa forma, o filme continuou com uma similaridade de 100%. Já no segundo teste foi passado 90 minutos como atributo de duração, sendo assim, a duração ficou fora do limite mínimo, e isso diminuiu a similaridade do filme para 93,75%.

5.2.5 Teste 5 – Diferença

Tabela 7 - Atributos de entrada Teste 5

Nome Atributo	Valor do Atributo
Ano Lançamento	2013
Duração	123
Diretor	David Ayer
Ator	Will Smith
Gênero	Ação
Distribuidora	Warner Bros
País	Estados Unidos

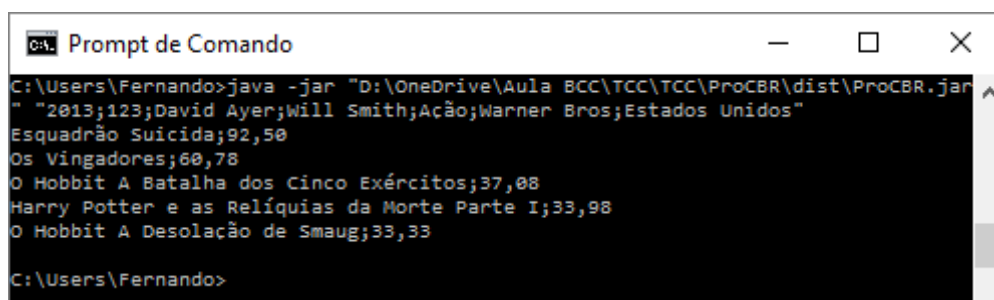
Fonte: Elaboração do autor, 2017.

Tabela 8 - Atributos de entrada Teste 5

Nome Atributo	Valor do Atributo
Ano Lançamento	2010
Duração	123
Diretor	David Ayer
Ator	Will Smith
Gênero	Ação
Distribuidora	Warner Bros
País	Estados Unidos

Fonte: Elaboração do autor, 2017.

Figura 28 - Resultado Teste 5



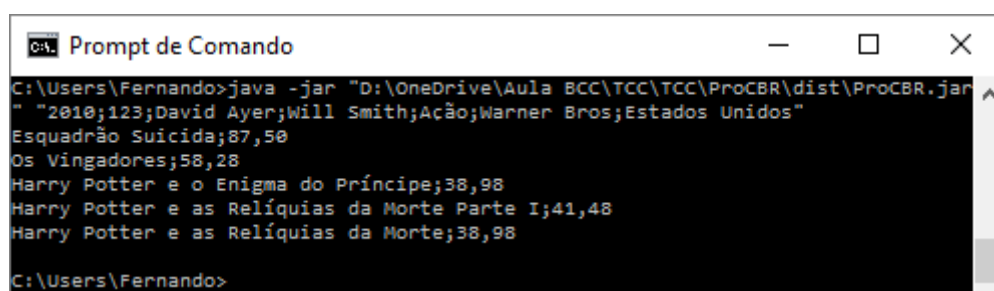
```

C:\Users\Fernando>java -jar "D:\OneDrive\Aula BCC\TCC\TCC\ProCBR\dist\ProCBR.jar"
"2013;123;David Ayer;Will Smith;Ação;Warner Bros;Estados Unidos"
Esquadrão Suicida;92,50
Os Vingadores;60,78
O Hobbit A Batalha dos Cinco Exércitos;37,08
Harry Potter e as Relíquias da Morte Parte I;33,98
O Hobbit A Desolação de Smaug;33,33
C:\Users\Fernando>

```

Fonte: Elaboração do autor, 2017.

Figura 29 - Resultado Teste 5



```

C:\Users\Fernando>java -jar "D:\OneDrive\Aula BCC\TCC\TCC\ProCBR\dist\ProCBR.jar"
"2010;123;David Ayer;Will Smith;Ação;Warner Bros;Estados Unidos"
Esquadrão Suicida;87,50
Os Vingadores;58,28
Harry Potter e o Enigma do Príncipe;38,98
Harry Potter e as Relíquias da Morte Parte I;41,48
Harry Potter e as Relíquias da Morte;38,98
C:\Users\Fernando>

```

Fonte: Elaboração do autor, 2017.

Para o teste da funcionalidade de cálculo de distância também foram efetuados dois testes conforme Figuras 28 e 29. No primeiro teste, no atributo “ano” foi passado 2013, como o cálculo de distância leva em consideração o parâmetro informado para esse atributo, e a diferença entre o atributo passado e o valor do banco de dados, nesse onde 2013 tem uma diferença de 3 anos para 2016, isso diminui a similaridade desse atributo, porém, ainda não chegou em 0%, resultando na similaridade global do filme em 92.5%. No entanto, no segundo teste foi passado o ano de 2010, e com a configuração do parâmetro em 5, esse valor saiu da faixa limite resultando no atributo ano como 0, ou seja, nada similar, dessa forma, a similaridade global do filme foi de 87.5%.

5.2.6 Teste 6 – Filmes do gênero Aventura

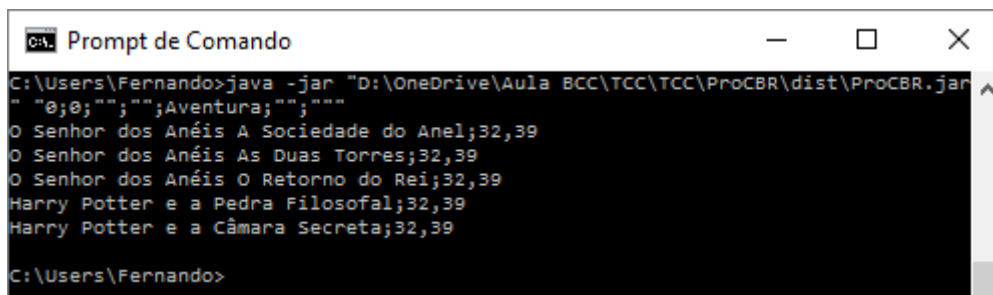
No teste 6, foram zerados todos os atributos, e passado apenas o gênero “Aventura” conforme pode ser visto na Tabela 9.

Tabela 9 - Atributos de entrada Teste 6

Nome Atributo	Valor do Atributo
Ano Lançamento	0
Duração	0
Diretor	
Ator	
Gênero	Aventura
Distribuidora	
País	

Fonte: Elaboração do autor, 2017.

Figura 30 - Resultado Teste 6



```

C:\Users\Fernando>java -jar "D:\OneDrive\Aula BCC\TCC\TCC\ProCBB\dist\ProCBB.jar"
"0;0;"";"";Aventura;"";""
O Senhor dos Anéis A Sociedade do Anel;32,39
O Senhor dos Anéis As Duas Torres;32,39
O Senhor dos Anéis O Retorno do Rei;32,39
Harry Potter e a Pedra Filosofal;32,39
Harry Potter e a Câmara Secreta;32,39
C:\Users\Fernando>

```

Fonte: Elaboração do autor, 2017.

No teste 6 foi passado apenas o atributo de filmes de “Aventura”, dessa forma, o resultado foram cinco filmes com similaridade igual de 32,39%. Vale ressaltar, que todos esses filmes possuem dois gêneros cadastrados em cada um deles, “Aventura” e “Ação”, nesses casos, o *framework* trata para trazer apenas uma vez cada filme, e sempre o resultado com a similaridade maior.

5.2.7 Teste 7 – Filmes do diretor Peter Jackson

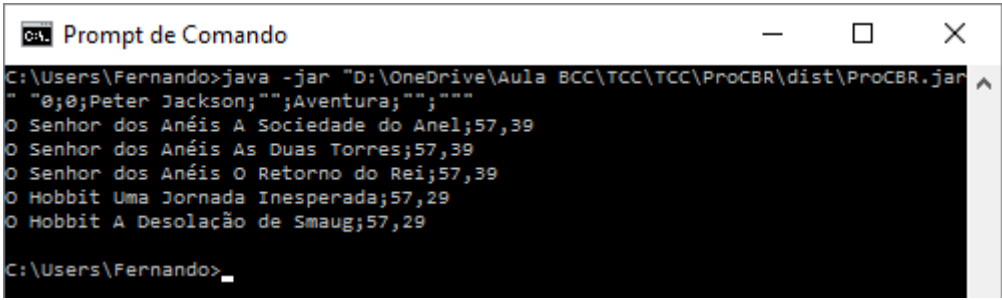
No teste 7, foram zerados todos os atributos, informando apenas o diretor “Peter Jackson” e mantido o gênero “Aventura” conforme pode ser visto na Tabela 10.

Tabela 10 - Atributos de entrada Teste 7

Nome Atributo	Valor do Atributo
Ano Lançamento	0
Duração	0
Diretor	Peter Jackson
Ator	
Gênero	Aventura
Distribuidora	
País	

Fonte: Elaboração do autor, 2017.

Figura 31 - Resultado Teste 7



```

C:\Users\Fernando>java -jar "D:\OneDrive\Aula BCC\TCC\TCC\ProCBR\dist\ProCBR.jar"
"0;0;Peter Jackson;"";Aventura;"";""
O Senhor dos Anéis A Sociedade do Anel;57,39
O Senhor dos Anéis As Duas Torres;57,39
O Senhor dos Anéis O Retorno do Rei;57,39
O Hobbit Uma Jornada Inesperada;57,29
O Hobbit A Desolação de Smaug;57,29
C:\Users\Fernando>

```

Fonte: Elaboração do autor, 2017.

Da mesma forma que no teste 6, o *framework* retornou cinco filmes com similaridades parecidas, ficando apenas os dois últimos filmes com 0.10% a menos de similaridade devido ao número de palavras e caracteres diferentes resultando em uma pequena diferença nos atributos que utilizam a função *distância de levenshtein* e *contagem de palavras*.

5.2.8 Teste 8 – Filmes do ano de 2002

No teste 8, foi informado apenas o atributo “Ano” conforme pode ser visto na Tabela 11.

Tabela 11 - Atributos de entrada Teste 8

Nome Atributo	Valor do Atributo
Ano Lançamento	2002
Duração	0
Diretor	
Ator	
Gênero	
Distribuidora	
País	

Fonte: Elaboração do autor, 2017.

Figura 32 - Resultado Teste 8

```

C:\Users\Fernando>java -jar "D:\OneDrive\Aula BCC\TCC\TCC\ProCBR\dist\ProCBR.jar"
" 2002;0;"";"";"";""
A Era do Gelo;13,54
O Senhor dos Anéis A Sociedade do Anel;13,64
O Senhor dos Anéis As Duas Torres;13,64
Harry Potter e a Câmara Secreta;13,64
O Senhor dos Anéis O Retorno do Rei;11,14
C:\Users\Fernando>

```

Fonte: Elaboração do autor, 2017.

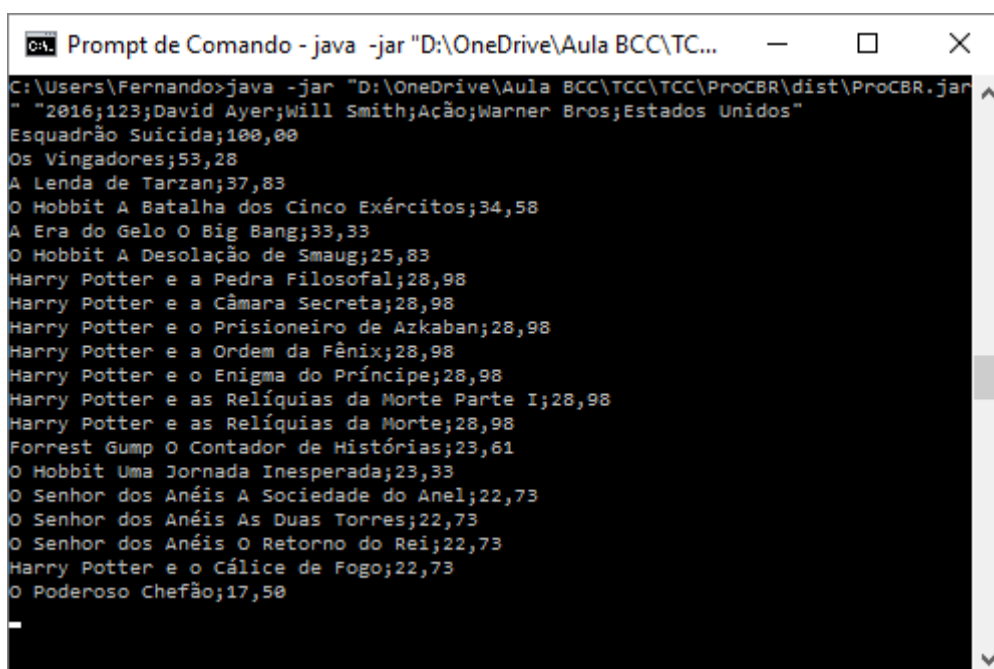
O resultado foram cinco filmes todos lançados no ano de 2002 e com similaridade parecida, com pequena diferença apenas devido ao cálculo de similaridade de *Strings* conforme descrito no Teste 7. A similaridade desses filmes ficaram baixas devido ao baixo peso do atributo “Ano”.

5.3 TESTES DE COMUNICAÇÃO COM DIFERENTES TECNOLOGIAS DE PROGRAMAÇÃO

5.3.1 Teste 1 – Comunicação com *Prompt de Comando*

Conforme descrito nas propostas deste trabalho, a comunicação com o *Prompt de Comando* do Windows, permite que diversas tecnologias diferentes de programação consigam efetuar a chamada do *framework*. Podemos acompanhar na Figura 33 a seguir, a chamada do *framework* via *Prompt de Comando*.

Figura 33 - Chamada do *framework* pelo *prompt de comando*



```

C:\Users\Fernando>java -jar "D:\OneDrive\Aula BCC\TCC\TCC\ProCBR\dist\ProCBR.jar"
"2016;123;David Ayer;Will Smith;Ação;Warner Bros;Estados Unidos"
Esquadrão Suicida;100,00
Os Vingadores;53,28
A Lenda de Tarzan;37,83
O Hobbit A Batalha dos Cinco Exércitos;34,58
A Era do Gelo O Big Bang;33,33
O Hobbit A Desolação de Smaug;25,83
Harry Potter e a Pedra Filosofal;28,98
Harry Potter e a Câmara Secreta;28,98
Harry Potter e o Prisioneiro de Azkaban;28,98
Harry Potter e a Ordem da Fênix;28,98
Harry Potter e o Enigma do Príncipe;28,98
Harry Potter e as Relíquias da Morte Parte I;28,98
Harry Potter e as Relíquias da Morte;28,98
Forrest Gump O Contador de Histórias;23,61
O Hobbit Uma Jornada Inesperada;23,33
O Senhor dos Anéis A Sociedade do Anel;22,73
O Senhor dos Anéis As Duas Torres;22,73
O Senhor dos Anéis O Retorno do Rei;22,73
Harry Potter e o Cálice de Fogo;22,73
O Poderoso Chefão;17,50
  
```

Fonte: Elaboração do autor, 2017.

O resultado conforme descrito anteriormente será de duas maneiras, na primeira, a função retorna uma variável do tipo *Array de String* onde pode ser tratada e mostrada conforme a Figura 33, porém, a tecnologia de programação empregada tem que dar suporte ao retorno do *prompt de comando*. Na segunda opção, dentro da pasta onde está o arquivo *jar* do *framework*, será salvo um arquivo XML chamado *results* que pode ser tratado e mostrado na aplicação externa, caso o usuário assim desejar.

5.3.2 Teste 2 – Comunicação com *Object Pascal Delphi XE6*

No teste 2, foi desenvolvido pelo autor uma aplicação externa utilizando a linguagem de programação *Object Pascal* e o ambiente de desenvolvimento *Delphi XE6*. Essa aplicação irá efetuar a chamada do *framework* e tratar posteriormente o arquivo XML de

resultados. Na Figura 34, podemos observar a interface da aplicação externa, junto dos atributos que serão informados ao *framework* e um *grid* contendo o resultado dessa chamada.

Figura 34 - Interface aplicação externa

Nome Filme	Taxa de Similaridade
O Hobbit A Batalha dos Cinco Exércitos	32,08
A Lenda de Tarzan	35,33
Os Vingadores	50,78
Esquadrão Suicida	97,5

IFC - Bacharelado em Ciências da Computação

Fernando Schulz - 2017

Fonte: Elaboração do autor, 2017.

Conforme pode ser visto na Figura 34, uma interface externa utilizando outra linguagem de programação consegue efetuar a chamada do *framework* e tratar o XML do retorno.

Nos quadros 26 e 27 mostrados a seguir, podemos acompanhar os códigos da chamada do *framework* e do tratamento do arquivo XML respectivamente.

Quadro 26 - Código chamada do *framework* em *Object Pascal*

```
WinExec(strCaminho, SW_HIDE);
```

Fonte: Elaboração do autor, 2017.

No *Delphi XE* é extremamente fácil rodar uma linha de código no *prompt de comando*, conforme pode ser visto no Quadro 26, apenas uma linha de código foi necessária. Na variável *strCaminho*, foi passado o mesmo comando que é informado no *prompt de comando* para efetuar a chamada no *framework* e já foi demonstrado anteriormente.

Quadro 27 - Código tratamento do XML de retorno

```
procedure TFAplicationTest.lerXmlRetorno(strCaminhoXml: String);
var
  i: integer;
  NodeCases: IXMLNode;
  NodeTempCases: IXMLNode;
  NodeSolution: IXMLNode;
begin
  vXMLDoc.LoadFromFile(strCaminhoXml);
  NodeCases := vXMLDoc.DocumentElement.childNodes.FindNode('cases');
  for i := 0 to NodeCases.childNodes.Count - 1 do
  begin
    NodeTempCases := NodeCases.childNodes.Nodes[i];
    NodeSolution := NodeTempCases.childNodes.Nodes[1];

    ClientDataSet1.Insert;
    ClientDataSet1NM_FILME.AsString :=
      NodeSolution.childNodes.FindNode('nm_filme').Text;
    ClientDataSet1TX_SIMILARIDADE.AsString :=
      NodeTempCases.childNodes.FindNode('similarity').Text;
    ClientDataSet1.Post;
  end;
end;
```

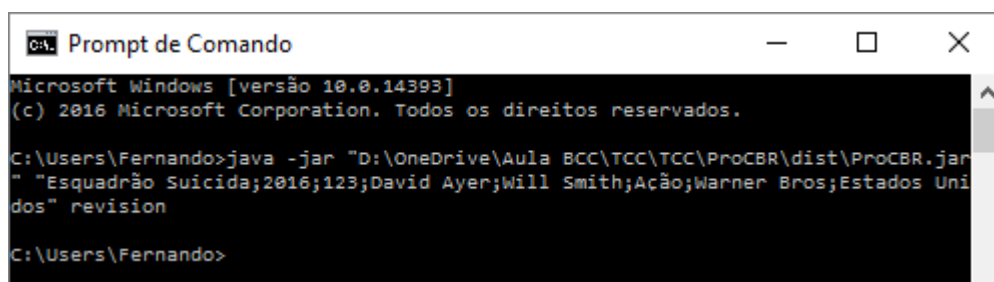
Fonte: Elaboração do autor, 2017.

O tratamento de arquivos XML's o *Delphi* também trata com componentes nativos desde versões muito antigas, dessa forma, para tratar o xml de resultado também são necessárias apenas poucas linhas de código conforme visto no Quadro 27.

5.4 TESTES ARQUIVO DE REVISÃO

A chamada do *framework* para gravar um arquivo XML com a revisão é muito parecida, as diferenças ficam apenas nos argumentos da chamada. No primeiro argumento onde são passados os atributos de entrada, deverá ser passado também os campos que são solução, lembrando que deve ser respeitada a configuração da estrutura do RBC e a ordem dos campos no SQL. No segundo argumento, deve ser informado a palavra *revision*, dessa forma, o *framework* sabe que deverá apenas gravar um arquivo XML com as informações passadas. Após gravar a revisão, o usuário poderá fazer a retenção desses dados em seu banco de dados como desejar. Podemos acompanhar na Figura 35 a seguir, uma chamada para gravar um arquivo de revisão no *framework*.

Figura 35 - Teste arquivo de revisão



```
C:\> Prompt de Comando
Microsoft Windows [versão 10.0.14393]
(c) 2016 Microsoft Corporation. Todos os direitos reservados.

C:\Users\Fernando>java -jar "D:\OneDrive\Aula BCC\TCC\TCC\ProCBR\dist\ProCBR.jar"
" "Esquadrão Suicida;2016;123;David Ayer;Will Smith;Ação;Warner Bros;Estados Uni
dos" revision

C:\Users\Fernando>
```

Fonte: Elaboração do autor, 2017.

Conforme podemos observar na Figura 35, no primeiro argumento foi informado também o nome do filme que foi configurado como solução na estrutura do RBC, e na primeira posição, pois no SQL de busca ele era o primeiro campo a ser selecionado. E no segundo argumento foi informado a palavra *revision*. O resultado dessa operação é um arquivo XML já apresentado na seção 4.2.4.3 no Quadro 32 – Arquivo de revisão.

6 CONCLUSÃO

Os sistemas de RBC possuem muitas características relevantes para sua utilização, porém, um problema importante que deve ser tratado é o armazenamento e o processamento necessário para recuperação dos casos na base de casos. Com isso, desenvolvedores deste tipo de sistema necessitam analisar as diferentes representações de conhecimento e métricas de similaridade para definir qual a mais adequada para cada tipo de domínio de aplicação.

Atualmente as soluções existentes para o desenvolvimento de sistema de RBC obrigam o usuário a terem pleno domínio sobre esta técnica, também não disponibilizam uma interface amigável e consistente para definição da representação do conhecimento e das métricas de similaridade. Além disso, outra característica que deve ser levada em consideração na elaboração de sistemas de RBC é a integração desse sistema com diferentes tecnologias, desde a forma como são armazenadas as informações, até as interfaces para apresentação das informações aos usuários.

Após uma análise realizada em trabalhos acadêmicos correlatos, foi possível constatar a falta de suporte nessas características, dessa forma, para suprir a necessidade de uma plataforma consistente e que permite integrações com diferentes tecnologias, foi desenvolvido o *framework ProCBR*, o qual consiste em uma plataforma genérica para facilitar a criação e aplicação de um sistema RBC. Permite conexão com os bancos de dados Firebird, MySQL e PostgreSQL, e por tratar-se de uma aplicação desenvolvida em Java, pode comunicar-se com diferentes tecnologias de linguagem de programação por comandos no *prompt de comando*.

Com o intuito de validar todas as funcionalidades desenvolvidas no framework, foram elaborados testes separados, sendo eles, testes para conexão com diferentes tecnologias de banco de dados, testes de resultado de cálculo de similaridade, testes de comunicação com diferentes tecnologias de programação e teste para revisão dos resultados, e em todos eles, obteve-se bons resultados, atingindo os objetivos estabelecidos. O *framework* mostrou facilidade na estruturação de um sistema RBC de qualquer natureza, e a comunicação das funções pelo *prompt de comando* mostrou boa integração com diferentes linguagens de programação.

Para os testes de cálculo de similaridade, foram elaborados oito testes distintos utilizando diferentes atributos de entrada para verificar a integridade dos cálculos de similaridade locais e global, além de resultados obtidos quando não são informados todos os atributos de entrada. Os testes foram elaborados pensando apenas na análise do cálculo das funções, e os resultados obtidos na recomendação de filmes foram os esperados, diminuindo a similaridade em casos na qual necessita de troca de letras e palavras utilizando as funções

distância de levenshtein e *contagem de palavras*, e também, em casos de atributos numéricos, onde o valor esperado possuía uma faixa de limite mínimo e máximo calculados pelas funções *distância* e *threshold*. Os demais cálculos de similaridade locais apesar de simples por serem apenas de comparação de igualdade, também obtiveram os resultados esperados.

Para trabalhos futuros, sugere-se implementar novas funcionalidades e bancos de dados disponíveis para conexão, aumentando assim, a extensibilidade e aplicabilidade do *framework* desenvolvido. Pode ser desenvolvido uma funcionalidade na qual permite que o *framework* retenha novos casos no banco de dados disponível para conexão, funcionalidade essa, que demanda bastante conhecimento de banco de dados, visto que, as estruturas dos bancos de dados configurados terão diferenças entre si, dessa forma, a funcionalidade deve possuir tratamento para diversas situações diferentes. Outra sugestão, seria melhorias na funcionalidade de cálculo de similaridade, tratando cada atributo como um *Array*, dessa forma, o *framework* pode tratar um determinado atributo com os operadores de “*and*” ou “*or*”, por exemplo, na estrutura de sugestão de filmes, o atributo “ator” recebe apenas um nome, se o *framework* possui tratamento de um campo do tipo *array*, poderíamos informar dois atores, dessa forma, o *framework* calcularia a similaridade dos filmes utilizando os dois atores, ou apenas um deles. Por fim, o *framework* poderia também, ser implementado sem a necessidade de possuir uma interface para desenvolver a estrutura do RBC, dessa forma, funcionaria independente de uma aplicação.

REFERÊNCIAS

- AAMODT, A.; PLAZA, E. **Case-Based Reasoning: Foundational Issues, Methodological Variations and System Approaches**. AICOM, 1994.
- ABDRABOU, Essam Amin M. Lotfy; SALEM, Abdel-Badeeh. **Case-based reasoning tools from shells to object-oriented frameworks**. Faculty of Computer and Information Sciences, Ain Shams University, Abbassia – Cairo, Egypt, 2008. <https://www.researchgate.net/publication/228959494_Case-based_reasoning_tools_from_shells_to_object-oriented_frameworks>. Data de acesso: 27 de Janeiro de 2017.
- ABEL, Mara. **Um Estudo sobre Raciocínio Baseado em Casos**. Universidade Federal do Rio Grande do Sul, Porto Alegre – RS, 1996. Disponível em: <<http://www.inf.ufrgs.br/bdi/wp-content/uploads/CBR-TI60.pdf>>. Data de Acesso: 15 de Abril de 2017.
- ABEL, Mara. **Raciocínio Baseado em Casos – CBR**. Instituto de Informática. Universidade Federal do Rio Grande do Sul, Porto Alegre – RS, 2002.
- BAHLS, Daniel; BERGHOFER, Thomas Roth. **Explanation Support for the Case-Based Reasoning Tool myCBR**. In Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence. July 22 - 26, 2007, Vancouver, British Columbia, Canada., pages 1844-1845. The AAAI Press, Menlo Park, California, 2007. Disponível em: <<http://mycbr-project.net/publications.html>>. Data de Acesso: 25 de Maio de 2017.
- BECKER, Elvis Bartolomeu. **Sistema de Apoio para o Diagnóstico de Enfermidades Orais Utilizando Raciocínio Baseado em Casos**. Universidade Regional de Blumenau, Blumenau – SC, 2002. Disponível em: <<http://dsc.inf.furb.br/arquivos/tccs/monografias/2002-2elvisbartolomeubeckervf.pdf>>. Data de Acesso: 26 de Abril de 2017.
- BEZERRA, E. **Princípios de Análise e Projeto de Sistemas com UML**. 3. ed. São Paulo: Elsevier, 2015.
- BURKHARD, H.-D. **Extending some Concepts of CBR Foundations of Case Retrieval Nets**. In M. Lenz et al. (eds.). Case-Based Reasoning Technology from Foundations to Applications, Springer-Verlag, 1998.
- CARVALHO, Raquel Regis Azevedo de. **Função de Crença como Ferramenta para Solucionar Diagnóstico em Raciocínio Baseado em Casos**. Instituto de Ciências Exatas, Departamento de Sistemas da Computação, UnB, Brasília – DF, 1996.
- CEAUSU, Valentina; DESPRÈS, Sylvie. **A Semantic Case-based Reasoning for Text Categorization**. University of Paris, Paris, 2007. <https://link.springer.com/chapter/10.1007/978-3-540-76298-0_53>. Data de acesso: 27 de Janeiro de 2017.
- COPLIEN, J. **Software Patterns**, 2005.
- ERVATTI, Diego Lima; ANDRADE Patrick Peyneau; SILVA, Rafael Zacché de Sá. **Framework para Sistema de Raciocínio Baseado em Casos**. Centro Universitário Vila

Velha, Vila Velha, 2010. <http://www.uvv.br/edital_doc/2010-02-monografia-final-05_835a0bfa-d20f-4fd9-8bd5-57fb22e85a72.pdf>. Data de acesso: 27 de Janeiro de 2017.

FAYAD, M.; SCHIMIDT, D.; RALPH, J. **Implementing application frameworks: object-oriented framework at work**. New York: John Wiley & Sons, 1999. p. 729.

FILHO, Wilson de Pádua Paula. **Engenharia de Software: Fundamentos, Métodos e Padrões**. Disponível em: <http://aulasprof.6te.net/Arquivos_Aulas/07-Proces_Desen_Soft/Livro_Eng_Soft_Fund_Met_Padros.pdf>. Data de Acesso: 09 de Junho de 2017.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos**. 1. ed. São Paulo: Bookman, 2008.

GARCÍA, Juan A. Recio. **JColibri: A Multi-Level Platform for Building and Generating CBR Systems**. PhD Thesis, Universidad Complutense de Madrid – Spain, 2008. Disponível em: <<http://gaia.fdi.ucm.es/files/people/juanan/thesis.pdf>>. Data de Acesso: 27 de Janeiro de 2017.

GARCÍA, Juan A. Recio; CALERO, Pedro A. González; AGUDO, Belén Díaz. **Jcolibri2: A framework for bulding Case-based reasoning systems**. Science Direct, Universidad Complutense de Madrid, Spain, 2011. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0167642312000664>>. Data de acesso: 27 de Janeiro de 2017.

HEERINGA, Wilbert. **Measuring Pronunciation Differences using Levenshtein Distance**. University of Groningen, Groninga – Holanda, 2004. Disponível em: <https://s3.amazonaws.com/academia.edu.documents/41719281/thesis.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1509039051&Signature=ubNtTo7BPYxX%2FGM5k%2B%2FPbS5gG1E%3D&response-content-disposition=inline%3B%20filename%3DMeasuring_Dialect_Pronunciation_Di_erenc.pdf>. Data de Acesso: 26 de Outubro de 2017.

HEINRICH, Luciane Tondorf. **Sistemas de Informação Aplicados a Lojas de Confeções do Alto Vale do Itajaí-SC Utilizando Raciocínio Baseado em Casos**. Universidade Regional de Blumenau, Blumenau – SC, 2000. Disponível em: <<http://campeche.inf.furb.br/tccs/2000-II/2000-2lucianetondorfheinrichvf.pdf>>. Data de Acesso: 19 de Abril de 2017.

HORSTMANN, C. **Padrões e Projeto Orientados a Objetos**. 2. ed. Porto Alegre: Bookman, 2007.

KOLODNER, Janet. **Case-Based Reasoning**. San Mateo, California: Morgan Kaufman Publishers, 1993.

KRAUS, Machado Helton. **Ferramenta para Desenvolvimento de Sistemas de Raciocínio Baseado em Casos**. Universidade do Vale do Itajaí, São José – SC, 2009. Disponível em: <<http://siaiap32.univali.br/seer/index.php/acotb/article/download/6323/3560>>. Data de acesso: 21 de Março de 2017.

LARMAN, C. **Utilizando UML e Padrões**. 3. ed. Porto Alegre: Bookman, 2007.

LEE, Rosina Weber. **Pesquisa Jurisprudencial Inteligente**. Programa de pós-graduação em Engenharia de Produção. Universidade Federal de Santa Catarina, Florianópolis – SC, 1998. Disponível em: <<http://campeche.inf.furb.br/tccs/2002-II/2002-2carloseduardodesouzasilvavf.pdf>>. Data de Acesso: 15 de Abril de 2017.

LURREGI, S. Corrêa; SOUZA, Marcel P. **Fábrica de Experiências – Raciocínio Baseado em Casos**. 1999.

MENDES, Joice Barbosa. **Um Framework de Raciocínio Baseado em Casos Aplicado para Estruturar a Base de Conhecimento em Sistemas Tutores Inteligentes**. Universidade Federal de Itajubá, Itajubá – MG, 2012. <<http://saturno.unifei.edu.br/bim/0039008.pdf>>. Data de acesso: 27 de Janeiro de 2017.

MONTEIRO, Odlaniger Lourenço Damaceno. **Aplicação de Padrões de Projeto no Desenvolvimento de Frameworks: Um Estudo de Caso**. Universidade Federal de Santa Catarina, Florianópolis – SC, 2002. Disponível em: <<https://repositorio.ufsc.br/bitstream/handle/123456789/83246/190065.pdf?sequence=1>>. Data de Acesso: 12 de Maio de 2017.

PRESSMAN, Roger S. **Engenharia de Software: Uma abordagem profissional**. 8. ed. Porto Alegre: AMGH, 2016.

RAMOS, R. A. **Treinamento prático em UML**. 1. ed. São Paulo: Digerati Books, 2006.

REINALDO, Francisco Antonio Fernandes. **Definição e Aplicação de um Framework para Desenvolvimento de Redes Neurais Modulares e Heterogêneas**. Universidade Federal de Santa Catarina, Florianópolis – SC, 2003. <<https://repositorio.ufsc.br/handle/123456789/85689>>. Data de acesso: 16 de Março de 2017.

RUSSELL, Stuart; NORVIG, Peter. **Inteligência Artificial**. 3. ed. Rio de Janeiro: Elsevier, 2004.

SILVA, Carlos Eduardo de Souza. **Sistema de Apoio para Otimização das Atividades de Suporte Técnico de uma Empresa de Desenvolvimento de Software, Utilizando Raciocínio Baseado em Casos**. Universidade Regional de Blumenau, Blumenau – SC, 2002. Disponível em: <<http://campeche.inf.furb.br/tccs/2002-II/2002-2carloseduardodesouzasilvavf.pdf>>. Data de Acesso: 30 de Abril de 2017.

SAMPAIO, C. **Guia do Java Enterprise Edition 5: Desenvolvendo Aplicações Corporativas**. 1. ed. Rio de Janeiro: Brasport, 2007.

SILVA, R. P. **Suporte ao Desenvolvimento e Uso de frameworks e Componentes**. Porto Alegre: PPGC da UFRGS, 2000.

VARELA, Geraldo Menegazzo. **Utilização de Raciocínio Baseado em Casos no Sistema para Controle e Gerenciamento de Projetos do Instituto de Pesquisa Ambiental**. Universidade Regional de Blumenau, Blumenau - SC, 1998.

VITORINO, Thiago Arreguy Silva. **Raciocínio Baseado em Casos: Conceitos e Aplicações**. Dissertação (Mestre) – Universidade Federal de Minas Gerais, Belo Horizonte – MG, 2009. Disponível em: <<http://www.ppgee.ufmg.br/documentos/Defesas/827/DM2009a.pdf>>. Data de Acesso: 29 de Abril de 2017.

WANGENHEIM, Christiane Gresse von; WANGENHEIM, Aldo von; RATEKE, Thiago. **Raciocínio Baseado em Casos**. 2. ed. São Paulo: Manole, 2003.

WATSON, Ian. **Na Introducing to Case-Based Reasoning**. Progress in Case-Based Reasoning. UK CBR 1995. Lecture Notes in Computer Science (Lecture Notes in Artificial Intelligence), vol 1020. Springer, Berlin, Heidelberg, 1995. Disponível em: <https://link.springer.com/chapter/10.1007%2F3-540-60654-8_18?LI=true>. Data de Acesso: 17 de Abril de 2017.

WATSON, Ian. **Applying Case-Based Reasoning Techniques for Enterprise Systems**. Morgan Kaufmann Publisher, California, 1997. Disponível em: <<http://dl.acm.org/citation.cfm?id=286680>>. Data de Acesso: 17 de Abril de 2017.

WEISSMANN, Henrique Lobo. **Vire o jogo com Spring Framework**. São Paulo: Casa do Código, 2012.