**UNIVERSIDADE REGIONAL DE BLUMENAU**

**CENTRO DE CIÊNCIAS EXATAS E NATURAIS**

**CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO**

# BETA - *FRAMEWORK* PARA APLICAÇÃO DA TÉCNICA DE *BEHAVIOR TARGETING*

**RUDIMAR IMHOF**

**BLUMENAU**
**2010**

**2010/1-22**

**RUDIMAR IMHOF**

# BETA - *FRAMEWORK* PARA APLICAÇÃO DA TÉCNICA DE *BEHAVIOR TARGETING*

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Adilson Vahldick – Orientador

**BLUMENAU**
**2010**

**2010/I-22**

# BETA - *FRAMEWORK* PARA APLICAÇÃO DA TÉCNICA DE *BEHAVIOR TARGETING*

Por

**RUDIMAR IMHOF**

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

| | |
|---|---|
| Presidente: | Prof. Adilson Vahldick, MSc. – Orientador, FURB |
| Membro: | Prof. Mauro Marcelo Mattos, Dr.Eng. – FURB |
| Membro: | Prof. Marcel Hugo, MEng. – FURB |

Blumenau, 06 de julho de 2010.

Dedico este trabalho a todas as pessoas com interesse no estudo do comportamento por meio da tecnologia. A elas este trabalho é destinado.

## AGRADECIMENTOS

A Deus pois, sem ele, nada seria possível

Aos meus pais e irmão, Rudi Imhof, Gelvânia Imhof e Rudinei Imhof pelo incentivo e apoio.

A minha namorada, Taira Franciele Skerke, pela compreensão e auxílio.

Ao professor Adilson Vahldick, pela sua orientação e esforço na realização deste trabalho.

Muito obrigado.

[...] no general method for the solution of questions in the theory of probabilities can be established which does not explicitly recognise [...] those universal laws of thought which are the basis of all reasoning […].

George Boole

## RESUMO

Muitas pesquisas demonstram a dificuldade para a obtenção de padrões no comportamento humano, em especial em relação às suas preferências e opções pessoais. Recentes estudos utilizam-se de métodos de classificação baseados em critérios como região, idade, além de muitos outros, escolhidos conforme o intuito da pesquisa. Os critérios baseados em classificação normalmente utilizam-se de atributo chave, o qual indica a forma como os dados serão divididos. A segmentação comportamental, por sua vez, permite usar o próprio conjunto de dados como classificador. Neste trabalho, objetiva-se demonstrar o desenvolvimento de um *framework* empregando conceitos de *Behavior Targeting* e *Data Mining*. Faz-se uso, no presente estudo, além destes conceitos, da biblioteca *Weka*, de tecnologias como *Reflection* e *Annotations* e de alguns padrões de projeto como DAO, *Factory* e *Singleton*. Procurou-se alcançar, neste estudo, um modelo de desenvolvimento que permitisse à aplicação fornecer uma lista de preferências de vários usuários e consultar, posteriormente, no *framework*, quais preferências representam, por similaridade, o comportamento de determinado usuário, permitindo à aplicação fornecer sugestões aos usuários baseadas na consulta ao *framework*.



Palavras-chave: *Web analytics*. *Behavior targeting*. Inteligência artificial. *Framework*. *WEB* 2.0.

**ABSTRACT**

Many studies show the difficulty to acquire patterns of human behavior, particularly in relation to their preferences and options. Recent studies have used methods for classification based on criteria such as region, age, and many others, chosen according the aim of the research. The criteria based on classification often use some key attribute, which indicates how the data will be divided as behavioral targeting allows you to use their own set of data as the classifier. Showing the development of a framework, using concepts of Behavior Targeting and Data Mining, was the purpose in this work. In addition, there were also the concepts like Weka library, technologies as Reflection and Annotations and some design patterns, like DAO, Factory and Singleton. Thus, it had been achieve a development model that allows the application to provide a list of preferences for multiple users that can also be seen later in the framework, which represent preferences for similarity about the behavior from a particular user, allowing the application to provide suggestions to users based on the research done in the framework.

Key-words: Web analytics. Behavior targeting. Artificial intelligence. Framework. WEB 2.0.

## LISTA DE ILUSTRAÇÕES

## LISTA DE QUADROS

**SUMÁRIO**

# 1 INTRODUÇÃO

Há muito tempo o homem tenta entender o comportamento humano por meio da tecnologia. As tentativas atuais tentam atingir este objetivo por meio da análise do usuário em web sites. Este tema insere-se na versão mais atual da Web nomeada como Web 3.0.

Segundo Nunes (2008),Web 3.0 é a nova geração da internet, na qual o que importa é a forma como as pessoas se relacionam, o estudo do comportamento e seu meio de vida. Ela retrata a coletividade e estuda essa coletividade sob a ótica de *Web Analytics*.

*Web Analytics* são métricas para quantificar o uso de ferramentas online e, desta forma, fornecer dados para avaliação de mercado. Segundo Ribeiro (2009, p. 19). "[...] analisar o mercado é entender o macro-ambiente em que sua empresa se encontra [...]". Uma dessas métricas leva o nome de *Behavior Targeting*, que pode ser definida como "o estudo do comportamento do usuário no mercado on-line" (RIBEIRO, 2009, p. 20). Para se construir esse estudo do comportamento, é necessária alguma técnica que retrate o comportamento de maneira adaptativa. O conjunto desses comportamentos pode ser posteriormente segmentado, utilizando-se, por exemplo, uma técnica da área de inteligência artificial chamada *Data Mining*.

"*Data Mining* é uma linha de pesquisa pertencente ao campo da ciência da computação que tem por objetivo oferecer estratégias automatizadas para a análise de grandes bases de dados de empresas, procurando extrair destas fontes informações que estejam implícitas, que sejam previamente desconhecidas e potencialmente úteis" (PICHILIANI, 2008, p. 23). Com as técnicas fornecidas pela *Data Mining*, é possível segmentar os comportamentos absorvidos e criar mapas auto-organizáveis, os quais são estruturas que se organizam em função da estrutura de dados. Várias maneiras podem ser usadas para construir uma ferramenta com essas características, uma delas é construí-la como um *framework*.

"*Framework* é a estrutura de uma aplicação, ou seja, um conjunto de classes e interfaces projetado para encapsular o código comum a uma família de aplicações" (ROCHA, 2009, p. 66). Assim, esse mesmo código pode ser reutilizado inúmeras vezes.

O presente trabalho implementa um *framework* que utiliza a técnica de *Behavior Targeting* como meio para análise do comportamento do usuário em uma aplicação web. O *framework* foi especificado de tal forma que separe o que foi implementado nesse trabalho sobre *Behavior Targeting* do que possa vir a ser implementado futuramente nesse próprio *framework* para outras áreas da web 3.0, como web semântica, reputação digital, etc. A

arquitetura do *framework* foi projetada baseando-se no estilo de desenvolvimento em camadas. O *framework* tem como entradas os links escolhidos e como responsabilidade reter o comportamento e segmentá-lo com técnicas de mineração de dados, construindo, assim, mapas auto-organizáveis. O *framework* poderá fornecer sugestões de páginas para o usuário de acordo com seu grupo, além de informações dos grupos segmentados para o gestor.

## 1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho foi desenvolver um *framework* para análise das interações do usuário em uma aplicação web e fornecer dados analíticos dessas interações utilizando a técnica da segmentação comportamental.

Os objetivos específicos do trabalho foram:

a) segmentar os dados de acordo com o uso do sistema;
b) aplicar métricas para análise de perfis;
c) fornecer os grupos segmentados ao qual o usuário da aplicação pertence.

## 1.2 ESTRUTURA DO TRABALHO

Este trabalho organiza-se em quatro capítulos. O capítulo dois, fundamentação teórica, explicita os autores e discute os temas aos quais este estudo relaciona-se. O capítulo três apresenta o desenvolvimento do *framework*, desde o projeto, construção, à aplicação. Por fim, no capítulo quatro, fazem-se as considerações finais em relação aos resultados obtidos e possíveis desmembramentos futuros.

# 2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão descritos aspectos e conteúdos relevantes ao desenvolvimento do trabalho. Na seção 2.1 são classificadas as metodologias *Web Analytics* além de técnicas *Behavior Targeting*. Na seção 2.2 são apresentados aspectos relativos à *Data Mining*. Expõem-se, na seção 2.3, os fundamentos relacionados ao desenvolvimento de *framework*s. Por fim, na seção 2.4, apresentam-se trabalhos correlatos ao tema proposto.

## 2.1 WEB ANALYTICS E BEHAVIOR TARGETING

Com o surgimento da Web 3.0, cada vez mais técnicas são pesquisadas no sentido de adquirir conhecimentos sobre o usuário web e transformar de alguma maneira o volume de dados adquirido pela web 2.0 em algo útil. Nesse contexto, a *Web Analytics* fornece grupos de técnicas relacionadas à aquisição de conhecimento do usuário, ou seja, saber o que o usuário web deseja.

Para definir métricas é necessário encontrar dados qualitativos os quais, segundo Kaushik (2009, p. 12), são os seguintes:

a) monitoramento do que se fala sobre a marca (*buzz*);
b) satisfação do cliente;
c) índices de recomendação na rede (*net promoter*);
d) análise de questões subjetivas sobre a opinião do cliente;
e) tempo de uso do visitante;
f) fidelidade;
g) blog – *pulse* (tendências dos blogs).

Antes de efetuar-se a análise, é necessário conhecer os caminhos pelos quais o usuário percorreu. Para tanto, é necessário medir o fluxo de cliques efetuados por ele. Após, escolhe-se uma ferramenta *Web Analytics* para análise do site. Segundo Ribeiro (2009, p. 21), existem dois tipos de metodologia utilizada para o desenvolvimento de uma ferramenta de *Web Analytics*: *User Centric* e *Site Centric*. Ambas metodologias fundamentam-se no uso da sessão do usuário para fornecer informações relacionadas às métricas.

### 2.1.1 Fluxo de cliques

Tal como afirma Kaushik (2009, p. 23), “existem quatros modos principais de capturar os dados do fluxo de cliques: (i) *web logs*, (ii) *web beacons*, (iii) *tags* JavaScript, (iiii) *sniffers* de pacotes”.

*Web logs* são dados sobre as requisições de páginas salvas no servidor como *Web Server Logs Files* (WSLF). O procedimento para captura dos dados é detalhado no fluxograma da Figura 1.

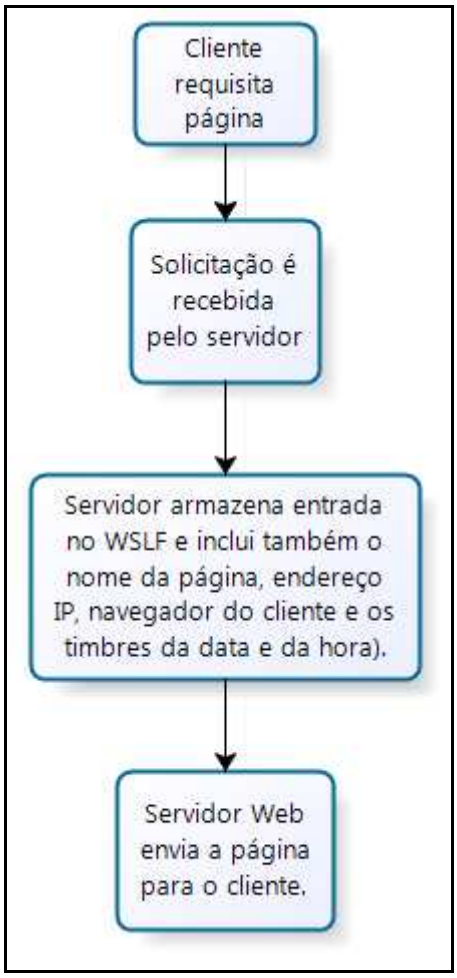



Figura 1 – Fluxograma procedimento de captura de dados em *web logs*

*Web Beacon* é utilizado para medir padrões de tráfego dos usuários de uma página a outra com objetivo de maximizar o fluxo de tráfego através da web (MELO, 2010). Os *Web Beacons* são *tags* HTML de imagens que ao serem carregadas levam consigo parâmetros que representam dados relevantes ao *site* como, por exemplo, endereço IP e duração do tempo da página.

*Web Beacon* são também chamados de *Web Bugs* e podem ser utilizados em páginas

web e e-mails, mas com diferentes finalidades. A figura 2 apresenta um exemplo de *Web Bug*. O código representa uma tag HTML de imagem em um e-mail. Ao ser carregado envia ao endereço dois parâmetros contendo o assunto do spam e endereço de correio do destinatário.

```
<img src="http://www.epublicidad.cl/cont.php?t=aventuraxlosandes&m=gonzalo
@pcw.idg.es" width="300" height="5">
```

Fonte: Melo (2010).

Figura 2 – *Tag* HTML de imagem contendo *web bug*

*Tags JavaScript* para uso de coleta de dados são relativamente parecidas com *Web Beacons*. A sequência de ações é descrita na figura 3.

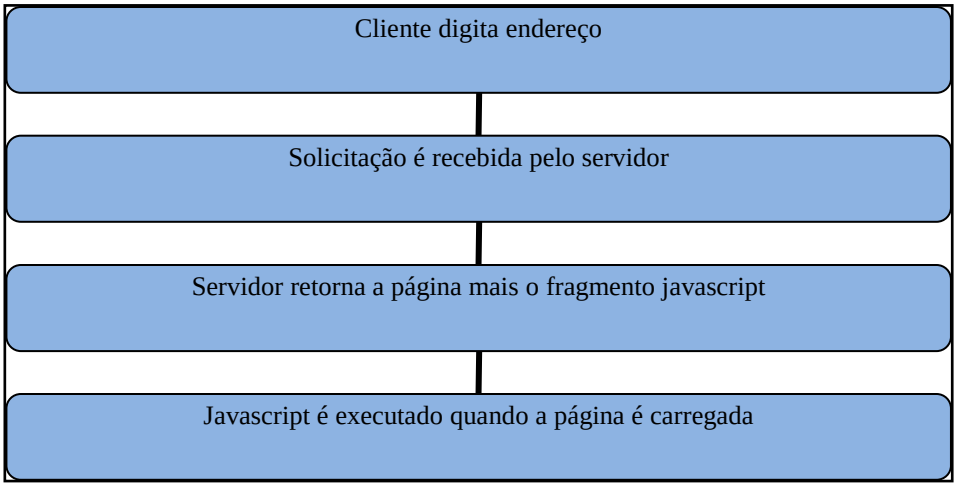



Figura 3 – Fluxo de eventos para coleta de dados com javascript

A figura 3 mostra o fluxo de eventos necessários para permitir a terceirização do serviço de *Web Analytics*, uma vez que ao carregar o código javascript o resultado pode ser devolvido para outro site que não o atual. Essa prática vem crescendo e sendo bastante utilizada hoje em dia.

*Sniffers* de pacote não tem uma utilização tão alta. Um *sniffer* é uma ferramenta capaz de interceptar o tráfego. Seu conceito na *web analytics* é simples e consiste em colocar o *sniffer* entre o usuário e o web *site*, tendo a responsabilidade de capturar o tráfego para uso da w*eb analytics*.

### 2.1.2 Site Centric

A metodologia *Site Centric*, segundo Ribeiro (2009, p. 22), tem como principal característica “[...] a medição da máquina do usuário através de um senso (medição de 100%

dos usuários que navegaram pelo seu web *site*)". Uma vantagem que esta metodologia permite é a de que ela não utiliza amostras e tem a capacidade de medir todos os usuários de seu web *site* e não somente uma parcela deles. Essa abordagem permite quantificar a navegação não por meio de pesquisas, mas pelo uso, tornando-se mais dinâmico o processo.

Em geral, ainda conforme o referido autor, esta metodologia armazena, analisando a sessão do usuário, alguns dados necessários à sua avaliação. Normalmente, um registro *site-centric* contém: (i) o usuário, (ii) informações demográficas sobre ele, (iii) o conjunto das páginas visitadas na sessão em um determinado site, (iiii) o site *ID*.

#### 2.1.2.1 Métricas Fundamentais

"Medir é o processo por meio do qual números ou símbolos são atribuídos a características das entidades do mundo real, de forma a tornar possível caracterizar cada entidade através de regras claramente definidas" (GOMES, OLIVEIRA, ROCHA, p.1, 2010). Esta é a definição para métricas de software, mas é facilmente aplicável também ao mundo web.

Algumas das ferramentas disponíveis para a metodologia *Site Centric* podem ser definidas também como métricas web, tal como elencado por Ribeiro (2009, p. 22):

a) dados de audiência: visitantes únicos, tempo de navegação no *site*;
b) tendências: audiência por dia, audiência por hora;
c) fonte de tráfego: como o visitante chegou ao site;
d) informações técnicas: computador do visitante, sistema operacional;
e) análise de conteúdo: análise de cada página do seu site;
f) dados geográficos: sobre o local de acesso do visitante;
g) funil: permite analisar o comportamento do usuário em caminhos pré-determinados no *web site*.

A ferramenta funil possibilita definir locais no *site* pelos quais o usuário passará e usar esse caminho para descobrir informações. São duas as abordagens dessa ferramenta: a abordagem que define etapas pré-estabelecidas, por onde o usuário deve passar até alcançar a página objetivo; e a mais recente, nomeada como *Hub-and-Spoke*, que coloca a página objetivo como ponto central e a partir dela múltiplas escolhas podem ser feitas.


Fonte: Mathworks (2010).

Figura 4 – Métricas para análise em sites e mídias *online*

A figura 4 demonstra as principais métricas de um site. Seus anéis seguem uma sequência lógica que pode ser resumida da seguinte maneira. A primeira análise se refere ao impacto de determinada informação. A segunda se refere às pessoas que foram atingidas pelo impacto através de seus cliques, como demonstrado no anel interações. Essas interações são contabilizadas como visitas e cada visita pode ou não se converter em uma venda.[1] A contabilização das conversões, ou seja, o que foi divulgado para o que foi vendido em função dessa divulgação é usado como critério para calcular o retorno do investimento em propaganda.

### 2.1.3 User Centric

A metodologia *User Centric*, segundo Ribeiro (2009, p. 21) "[...] possui como principal característica a medição do usuário de internet através de um painel de colaboradores (pesquisa)". Os colaboradores podem ser definidos como a amostra que a pesquisa utilizará.

Um registro *User Centric* contém: (i) o usuário, (ii) informações demográficas sobre ele, (iii) o conjunto das páginas visitadas na sessão, (iiii) o nome do *site*.

[1] O termo conversão é frequentemente utilizado para indicar a transição de um interesse para uma venda, mas pode ser utilizado também para indicar a decisão em seguir em frente no fluxo de execução que o site apresenta.

### 2.1.4 Behavior Targeting

Segmentar as métricas mais profundamente significa automaticamente que seus dados ficarão cada vez mais relevantes quando alguns fatores que os "sujam" tornarem-se uma parte menor do todo, conforme Kaushik (2009, p. 12). Existem no mercado várias segmentações sendo utilizadas e à medida que novas informações são inseridas, novas análises devem ser feitas e dessa maneira novos segmentos são criados. Algumas segmentações são aplicadas sob o número de visitas do site, outras segundo as buscas e há, ainda, a combinação de ambas. A segmentação pode fornecer tendências importantes sobre como o mercado está e como possivelmente ficará. Um tipo em especial de segmentação mede o comportamento do consumidor. A esse termo deu-se o nome de *Behavior Targeting.*

*Behavior Targeting* é uma técnica que permite segmentar o mercado de acordo com o comportamento dos usuários em relação à sua navegação. Pode ser traduzido como segmentação comportamental. Para essa técnica faz-se necessário agrupar os dados em conjuntos de consumidores, ou seja, encontrar uma maneira de agrupar as preferências manifestadas de acordo com o comportamento do consumidor. Essas preferências são expressas pelo consumidor quando da sua navegação e são necessárias métricas para transformar esse comportamento em valores numéricos que podem ser tratados pela implementação. Um meio de medir essas preferências é utilizando-se *Click-Through Rate* (CTR), a medida de cliques em anúncios pelo consumidor. "[...] Não importa como as preferências sejam expressas, você precisa de uma forma de mapeá-las para valores numéricos" (SEGARAN, 2008, p. 8). Essa forma de mapear os dados que representam o comportamento é uma maneira de fornecer informações a serem tratadas pela segmentação.

## 2.2 DATA MINING

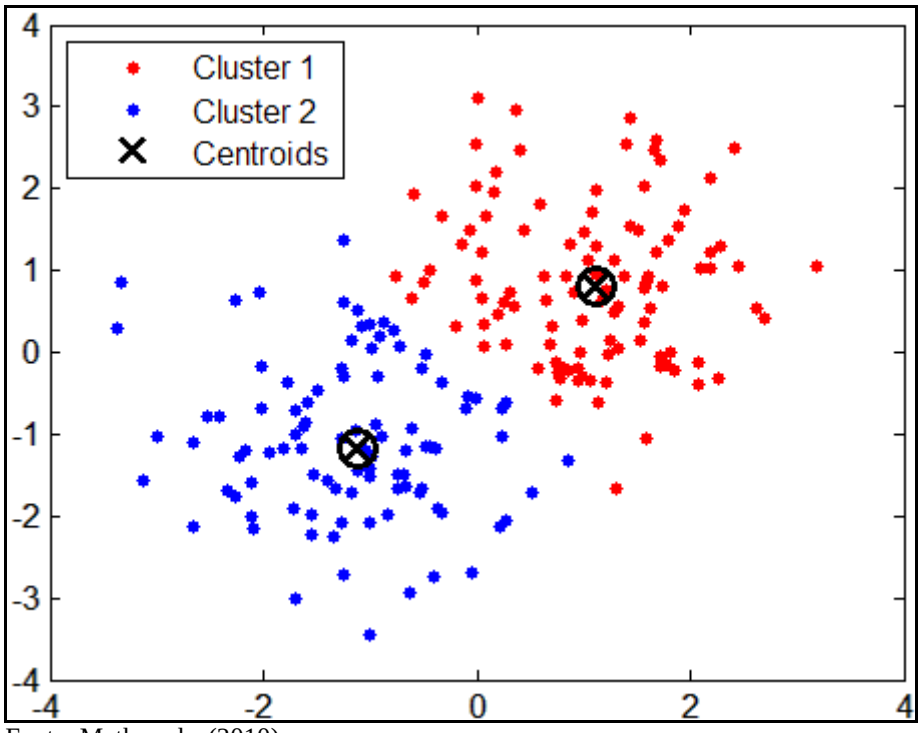
"*Data Mining* é o processo de descobrir padrões em dados." (WITTEN, FRANK, 1999, p. 7, tradução nossa). Assim, os conceitos de *Data Mining* podem ser utilizados para aplicar técnicas que segmentem os dados. Uma das técnicas de segmentação é através da utilização do algoritmo *k-means*.

"A ideia do algoritmo *k-means* (também chamado de k-médias) é fornecer uma

classificação de informações de acordo com os próprios dados" (PICHILIANI, 2008). A utilização desse algoritmo representa uma forma de reconhecer padrões em estruturas, pois permite dividir um determinado conjunto de dados em grupos na ordem de K. O princípio do algoritmo é selecionar K centros de *clusters* iniciais e em seguida iterativamente o algoritmo refina-se da seguinte maneira:

a) Cada instância *di* é atribuída a seu centro de *clusters* mais próximos, os *centroids*;
b) Cada centro de *cluster Cj* é atualizado para ser o meio de suas instâncias constituintes.

A figura 5 ilustra esse processo. Nela pode ser observado o conjunto dos dois *clusters* que compõem a execução do algoritmo e seus respectivos *centroids*.




Fonte: Mathworks (2010)

Figura 5 – Agrupamento de cluster pelos centroids no algoritmo k-means

Segundo Santos e Noronha (2010), o algoritmo emprega a distância Euclidiana para definir o centro dos grupos. Desta forma a distância euclidiana calcula a distância geométrica entre objetos. Para Global (2010), ela é a medida da distância no espaço Euclidiano, calculada como a raiz quadrada da soma das dos quadrados da diferença aritmética das coordenadas correspondentes de dois pontos apresentada na equação (1).

$$d(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \quad (1)$$

Na equação (1) pode ser visualizada a distância de deslocamento no espaço euclidiano onde $\mathbf{x} = < x_1, x_2,..., x_n >$, e $\mathbf{y} = < y_1, y_2,..., y_n >$. Assim, no espaço Euclidiano bidimensional, a

distância | AB | entre A = ($a_1$, $a_2$) e B = ($b_1$, $b_2$) é dada pela equação (2).

$$\sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2} \quad (2)$$

O resultado do calculo é usado principalmente para a o posicionamento dos elementos em relação aos *centroids* no momento da clusterização.

A clusterização, conforme Jain e Dubes (1988 apud PIMENTEL; FRANÇA; OMAR, 2003, p.497), é um método que utiliza o aprendizado não supervisionado ou auto-organizável, ou seja, não há um "professor" ou "crítico" que lhe indique o que cada padrão representa. Aprendizagem não supervisionada ou *clustering* (agrupamento) busca extrair informação relevante de dados não rotulados.

A principal vantagem na utilização do algoritmo *k-means* é que o resultado da segmentação é formado baseando-se na própria estrutura. Desta forma, a segmentação se dará à medida que novas informações são inseridas. Para a implementação das funcionalidades do algoritmo *k-means* pode ser utilizada a biblioteca *Weka*.

### 2.2.1 WEKA

*Weka* é uma coleção de algoritmos de aprendizagem para tarefas de mineração de dados (WITTEN, FRANK, 2009, p. 7, tradução nossa). *Weka* é uma biblioteca que permite a utilização de vários algoritmos na área de inteligência artificial.

Os dados podem ser manipulados no Weka através do seu formato Attribute-Relation File Format (ARFF) e pela integração com banco de dados.

Já um arquivo ARFF é um arquivo de texto puro contendo três partes:

a) relação, a primeira linha do arquivo que contém a marcação `@relation` e um nome significativo para o objeto de estudo;
b) atributos, conjunto de linhas do arquivo contendo cada um a marcação `@atributte` seguido do seu nome e tipo. O tipo de dado pode ser um dos quatro suportados pelo Weka como tipo numérico(numeric), tipo nominal(`<nominal-specification>`), tipo `string`(string) e tipo data(`<date-format>`);
c) dados, conjunto de linhas contendo a marcação `@data` somente uma vez no início do conjunto, seguida pelos dados que devem ser separados por vírgula e correspondentes aos atributos. Cada linha de dados corresponde a uma instância.

A marcação % indica início de comentários. As tags @relation, @atributte e @data

são *case insensitive*. No exemplo do quadro 1 as partes componentes de um arquivo ARRF podem ser melhor visualizadas.

```
% 1. Title: Iris Plants Database
%
% 2. Sources:
%      (a) Creator: R.A. Fisher
%      (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
%      (c) Date: July, 1988
%
@RELATION iris

@ATTRIBUTE sepallength  NUMERIC
@ATTRIBUTE sepalwidth   NUMERIC
@ATTRIBUTE petallength  NUMERIC
@ATTRIBUTE petalwidth   NUMERIC
@ATTRIBUTE class        {Iris-setosa,Iris-versicolor,Iris-virginica}

@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
```

Fonte: Rorohiko (2008).

Quadro 1 – Exemplo de arquivo ARFF

Existem as seguintes formas para se utilizar o *Weka*: (i) a partir de sua interface gráfica (*Weka GUI Chooser*), (ii) por meio de scripts, (iii) fazendo sua integração com Java.

*Weka GUI Chooser* é chamada a interface gráfica do *Weka*. As funções mais importantes podem ser encontradas na opção *Explorer*. A execução do *Weka GUI Chooser* consiste em fornecer a entrada de dados através de um arquivo ARRF ou uma conexão com o banco de dados, escolher um dos modelos de mineração de dados tais como *classifie*, *cluster*, *association* e então escolher um dos algoritmos do modelo e otimizá-lo pelas opções disponíveis.

*Weka* fornece uma biblioteca em um arquivo chamado *weka.jar*. Esse arquivo pode ser integrado ao projeto que passa a acessar os métodos da biblioteca. A figura 6 exemplifica o uso de *weka* no Java. As linhas 10 a 13 são importações para biblioteca do *weka*. As linhas 20 e 21 criam dois atributos. As linhas 24 a 28 povoam o vetor criado na linha 23. A linha 29 inclui o vetor em um atributo chamado classes. As linhas 31 e 32 criam um vetor onde fica armazenado o atributo inicial.

```
10  import weka.core.Attribute;
11  import weka.core.FastVector;
12  import weka.core.Instance;
13  import weka.core.Instances;
14
15  public class CriaARFF
16  {
17      public static void main(String[] args)
18      {
19          // First let's create the attributes.
20          Attribute x = new Attribute("x");
21          Attribute y = new Attribute("y");
22          // Third attribute is nominal.
23          FastVector classesLabels = new FastVector(5);
24          classesLabels.addElement("A");
25          classesLabels.addElement("B");
26          classesLabels.addElement("C");
27          classesLabels.addElement("D");
28          classesLabels.addElement("E");
29          Attribute classes = new Attribute("classes",classesLabels);
30          // Create a vector of attributes information.
31          FastVector attributes = new FastVector(3);
32          attributes.addElement(x);
```

Fonte: Santos (2005).

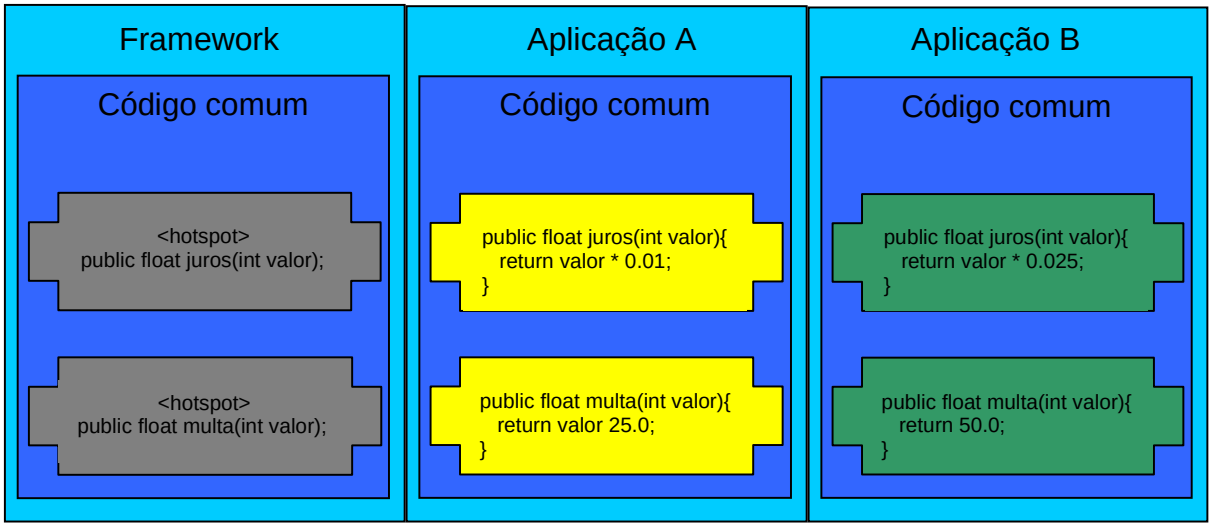
Figura 6 – Integração do weka com java

## 2.3 DESENVOLVIMENTO DE *FRAMEWORKS*

Segundo Pressman (2006, p. 204), *framework* – ou arcabouço – é um esqueleto com uma coleção de pontos conectáveis. Assim *frameworks* podem fornecer funcionalidades genéricas a uma gama de aplicações específicas permitindo maior abstração no tratar dos dados.

No *framework*, o fluxo de controle não é responsabilidade do desenvolvedor, e sim do *framework* (PRESSMAN, 2006, p. 203). Desta forma, as funcionalidades providas pelo *framework* são muito dinâmicas e podem ser modificadas pela extensão de suas características.

“Grande parte do trabalho existente no projeto de um *framework* consiste em identificar suas partes fixas (*frozen spots*) e variáveis (*hot spots*)”, (ROCHA, 2009, p. 77). Essa classificação facilita o reuso e a implementação do que deve ser genérico e o que pode ser especializado. A Figura 7 demonstra o uso de *hot spots* nas aplicações A e B.



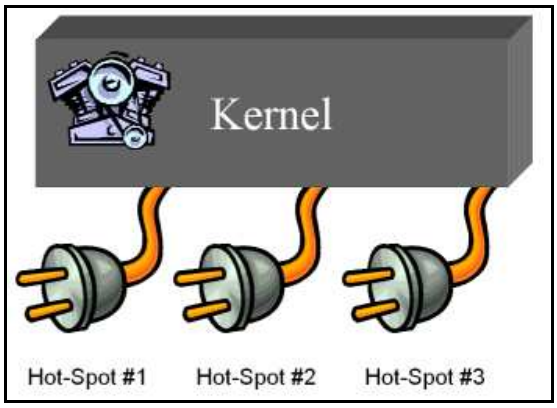

Fonte: Rocha (2009, p.77).

Figura 7 - *Hot Spots*

Os *frozen spots* são ponto imutáveis do *framework* em relação à aplicação. Os *frozen spots* constituem-se então como o núcleo do *framework* e desta forma é o ponto onde boa parte das tarefas a que o *framework* se propõe acontecem.

Os *hot spots* são os pontos flexíveis do *framework*. *Frameworks* não são executáveis, portanto para usá-los deve-se instanciá-los explicitamente através da implementação do aplicativo.

A figura 8 ilustra composição do *kernel* do *framework*, ou *frozen spots* com seus respectivos pontos flexíveis ou *hotspots*.




Fonte: Markiewicz e Lucena(2009).

Figura 8 – Núcleo do framework com seus *hotspots*

## 2.4 TRABALHOS CORRELATOS

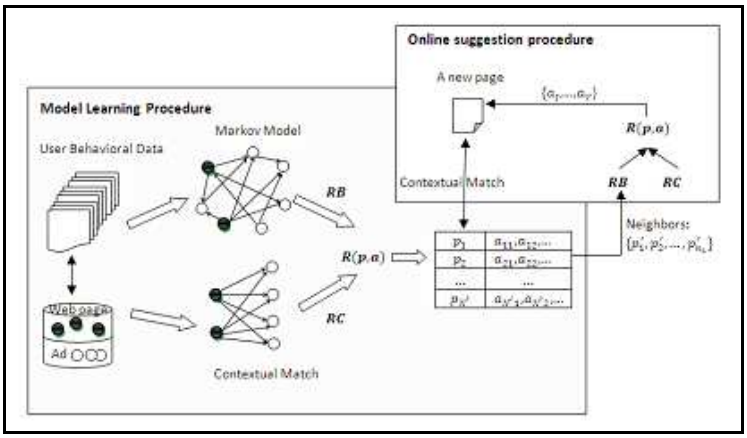
Apresentam-se, a seguir, trabalhos semelhantes ao tema proposto. Na seleção foram encontrados os seguintes projetos: A *Markov Chain Model for Integration Behavior Targeting into Contextual Advertising* de Li (2009), *Probabilistic Latent Semantic User Segmentation for Behavior Targeted Advertising* de Wu et al. (2009), *Large-Scale Behavioral Targeting* de Chen, Pavlov e Canny (2009).

### 2.4.1 A Markov Chain Model for Integration Behavior Targeting into Contextual Advertising

Li et al. (2009) propõe a utilização do Modelo de Markov para aplicação de *Behavior Targeting* em estratégias de publicidade. A maior parte dos métodos clássicos de segmentação baseia-se apenas em interesses históricos dos usuários. Li et al. (2009) estuda uma técnica que forneça publicidade baseada na interação do usuário.

Nesse trabalho, a *Behavior Targeting* é usada como forma de fornecer anúncios direcionando os usuários por meio da análise do comportamento de navegação. A maneira clássica de colocar os anúncios nas páginas é baseando-se no seu próprio contexto e o trabalho demonstra um modelo baseado não somente no contexto, nem em comportamento, mas na soma das duas maneiras.

Esse modelo leva em consideração a relevância contextual e comportamental. A Figura 9 demonstra um conjunto de treinamento usando o Modelo de Markov baseado nos dados que representam o comportamento do usuário. Esses dados possuem três tipos de entradas: (i) anúncios candidatos, (ii) o conjunto de páginas visitadas pelos usuários que clicaram nos anúncios candidatos e (iii) os usuários que navegaram pelas páginas que contém anúncios candidatos.




Fonte: Li et al.(2009).

Figura 9 - Modelo arquitetural de Li

### 2.4.2 Probabilistic Latent Semantic User Segmentation for Behavior Targeted Advertising

Wu et al (2009) sugerem como solução para *Behavior Targeting* a utilização do algoritmo *Probabilistic Latent Semantic* (PLS). A idéia principal do algoritmo consiste em segmentar os usuários e seus comportamentos de maneira semântica, visto que, segundo os autores, o algoritmo PLS, comparado a algoritmos clássicos como *k-means* e *Cluto*, pode melhorar a CTR em até 100%.

O algoritmo definido por Wu et al (2009) é baseado nas pesquisas que o usuário efetua e seu funcionamento trata esses dados. Os termos digitados nas consultas dos usuários são utilizados para sua segmentação.

### 2.4.3 Large-Scale Behavioral Targeting

A idéia central do trabalho de Chen, Pavlov e Canny (2009) é segmentar comportamentos do usuário web em larga escala usando para isso *grids* computacionais. Para atingir esse objetivo, o trabalho utilizou o *framework* MapReduce da Google para suportar grandes volumes de dados em *grids*.

O trabalho usa para mapear comportamentos um modelo matemático, que permite a contagem de eventos independentes que ocorrem a uma unidade de observação, e aplica nesse modelo um algoritmo com métricas para reter o comportamento usando-o em larga escala no *framework MapReduce*.

# 3 DESENVOLVIMENTO

Este capítulo tem por objetivo demonstrar o processo de desenvolvimento do *framework*. Para tal, compõe-se dos requisitos do *framework*, sua especificação e informações relacionadas à sua implementação. É por meio destes conteúdos que se propõe, aqui, elencar as tecnologias utilizadas e a operacionalidade do *framework*. Ao final, são apresentados os resultados obtidos, os quais também são discutidos e avaliados.

## 3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Abaixo são apresentados os Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF) da ferramenta:

a) o *framework* deverá permitir capturar por usuário as páginas visitadas e a seqüência de visitas (RF);
b) o *framework* deverá permitir fornecer como saídas que usuários pertencem aos grupos segmentados (RF);
c) o *framework* deverá permitir coletar dados estatísticos utilizando um funil de navegação com *Hub-and-Spoke* (RNF);
d) necessita ser desenvolvido baseado em filtros JEE (RNF);
e) necessita ser compatível com MySQL para persistência dos dados (RNF);
f) deve utilizar a biblioteca Weka para mineração de dados (RNF);
g) deve ser compatível com Apache Tomcat (RNF).

## 3.2 ESPECIFICAÇÃO

A seguir podem ser visualizadas a especificação e técnicas utilizadas para construção do *framework*.

### 3.2.1 Diagrama de casos de uso

O diagrama de casos de uso foi construído de maneira a tornar o uso do framework o mais acessível possível. Desta forma, a utilização e o acesso a ele acontecem de maneira facilitada, no intuito de permitir que o processamento da segmentação fique o máximo possível sob responsabilidade do *framework*, restando à APP somente fornecer dados e solicitar respostas, ou seja, solicitar o resultado da segmentação processada pelo *framework* conforme demonstra a figura 10.

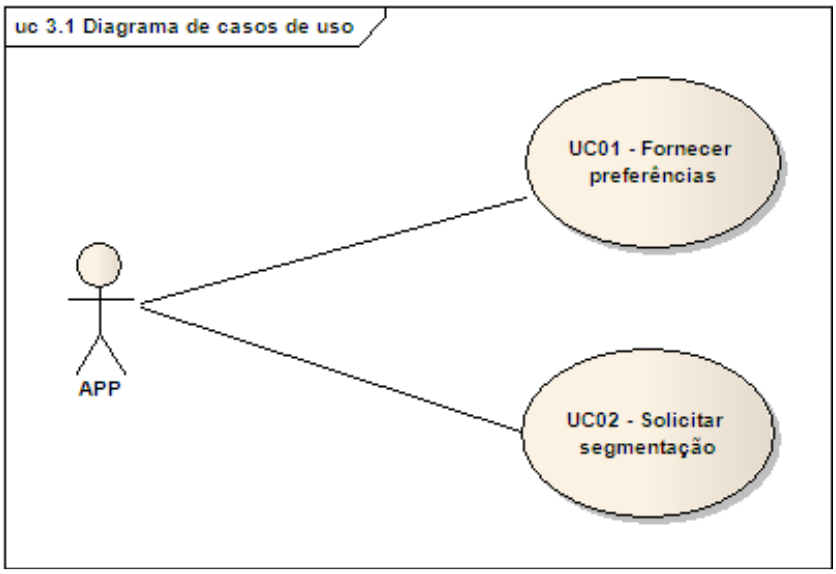



Figura 10 – Diagrama de casos de uso do *framework*

Os cenários para o caso de uso da Figura 10 são mostrados no quadro 2 e 3.

**UC01 - Fornecer preferências**

***public UseCase:*** No caso de uso fornece preferências a aplicação instancia uma classe modelo da própria aplicação. Essa classe deve conter atributos que representam as preferências a ser segmentadas.

***Cenários***

Executar listagem {Principal}.

1. aplicação faz a instância de uma classe modelo que contenha as preferências para segmentação;
2. aplicação adiciona cada objeto em uma lista de preferências da aplicação;
3. framework recebe a lista para segmentação

Quadro 2 – Cenário para o caso de uso `Fornecer Preferências`

**UC02 - Solicitar segmentação**

***public UseCase:*** No cenário solicita segmentação a aplicação deve chamar o método receberPreferencias ao implementar a uma interface ReceberPreferencias. Esse método fará a comunicação com o framework e devolverá as preferências do cluster no qual o usuário esta inserido.

***Cenários***

Retornar preferências {Principal}.

1. Aplicação executa chamada ao método getSegmentacao.
2. Framework busca id do usuário passado pela APP
3. Framework busca preferências do cluster do usuário
4. Framework devolve preferências ao usuário.

Quadro 3 – Cenário para o caso de uso `Solicitar Segmentação`

### 3.2.2 Diagrama de atividades

As figuras 11, 12 e 13 ilustram, respectivamente, os diagramas de atividades dos processos do *framework*.

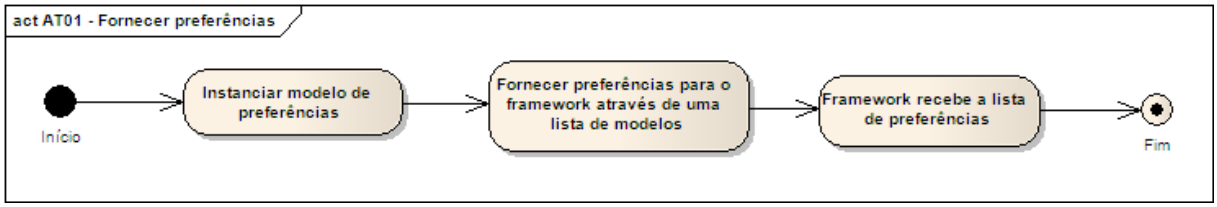



Figura 11 – Diagrama de atividades do processo `Fornecer preferências`

A figura 11 ilustra a maneira pela qual ocorre o fluxo de dados entre a APP e o *framework*. Num primeiro momento, a APP instancia um modelo de sua escolha o qual deverá conter atributos possíveis de tipo `Integer, Double` ou discretos, os quais devem ser obrigatoriamente anotados através da importação da anotação nomeada como AtributoDiscreto contida no próprio *framework*. Em seguida, a APP armazena esses modelos em uma lista e fornece a lista para o *framework*, o qual a recebe e assume o controle das atividades.

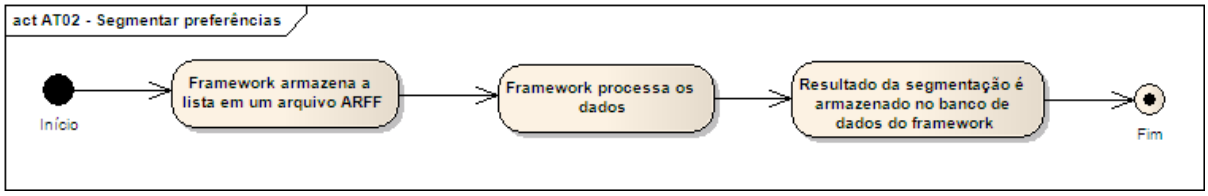


Figura 12 – Diagrama de atividades do processo `Segmentar preferências`

No diagrama da figura 12 a lista é recebida pelo *framework* e armazenada em um arquivo ARFF. Os dados do arquivo são processados e posteriormente armazenados no banco de dados do framework.

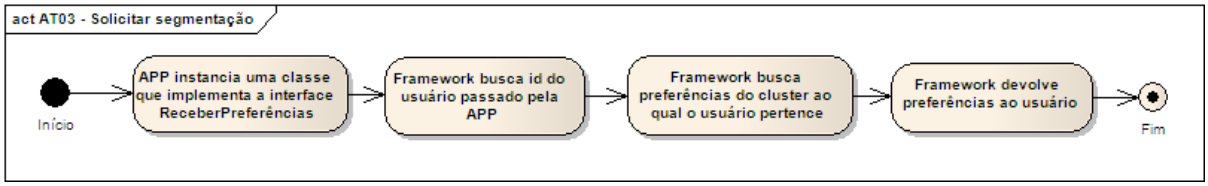



Figura 13 – Diagrama de atividades do processo `Solicitar segmentação`

A figura 13 mostra o processo que a APP usa para descobrir as preferências do usuário através do *framework*. Primeiro, a APP implementa uma interface fornecida com o *framework*. A APP utilizará, então, um método no qual passará o código do usuário desejado. Nesse momento, o *framework* busca o código do usuário, além das preferências do seu cluster e devolve o resultado para o usuário através do próprio método chamado.

### 3.2.3 Diagrama de classes

A ferramenta está dividida nas camadas de controle responsável por receber e processar os dados, os modelos de classes do *framework* e uma camada DAO para acesso ao *DataWarehouse.*

A figura 14 ilustra o diagrama para camada de controle.

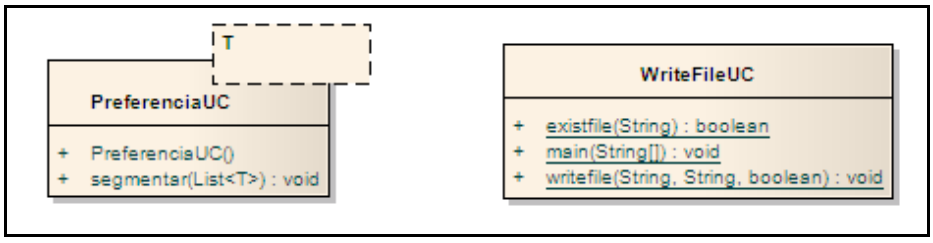



Figura 14 – Diagrama de classes do *package* `controle`

A camada de controle da figura 14 é responsável pela recepção da lista, processamento das informações e manipulação dos dados através das camadas `modelo` e `DAO`. A classe `PreferenciaUC` é a que detém o método `segmentar()`, o qual é chamado pela aplicação quando da passagem da lista. A classe `WriteFileUC` é responsável pela gravação do arquivo ARRF. Os métodos de responsabilidade dessas classes podem ser visualizados nos quadros 4 e 5.

| Método | Descrição |
|---|---|

| | |
|---|---|
| `segmentar(List<T>) : void` | Recebe a lista de preferências passada pela aplicação, constrói o arquivo ARFF executa a segmentação e faz a chamada ao DAO para gravação no banco de dados |

Quadro 4 – Descrição dos métodos da classe `PreferenciaUC`

| Método | Descrição |
|---|---|
| `existfile(String): boolean` | Responde com booleano a existência de um arquivo ARFF passando como parâmetro o caminho |
| `writefile(String, String, boolean): void` | Faz a escrita de um arquivo ARFF passando como parâmetro o caminho, a String a ser gravada, e um atributo append indicando a substituição do arquivo |

Quadro 5 – Descrição dos métodos da classe `WriteFileUC`

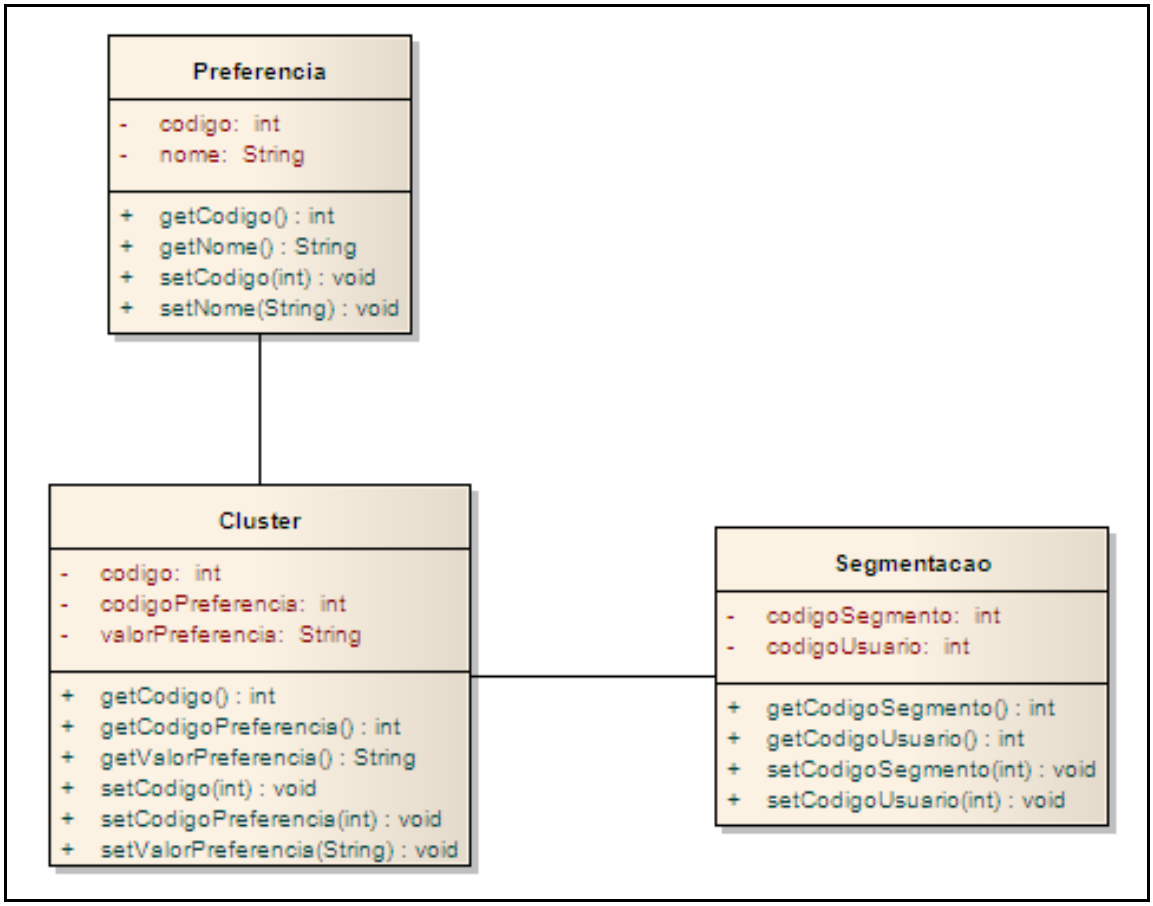



Figura 15 – Diagrama de classes do *package* `modelo`

A figura 15 apresenta o diagrama para o *package* `modelo`. Este *package* contém as classes responsáveis pela mineração dos dados. A classe `Preferencia` é um modelo para a

lista de preferências contendo o código e nome da preferência que a APP passará. A classe `Cluster` contém o código do *cluster*, as preferências deste *cluster* e os valores das preferências deste *cluster*. A classe `Segmentacao` contém o código do segmento, ou seja, o código do *cluster* no qual o usuário está inserido e a identificação desse usuário.

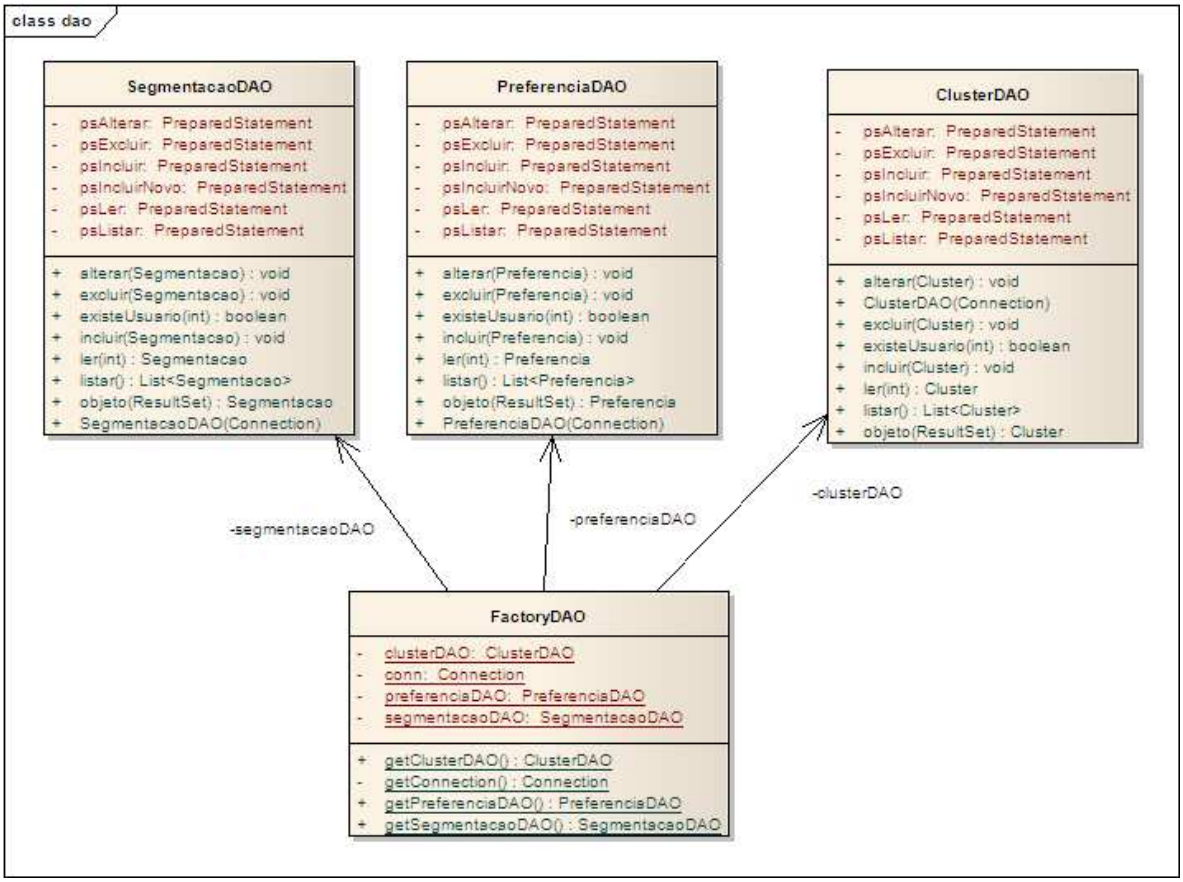



Figura 16 – Diagrama de classes para o *package* `DAO`

A camada `DAO` apresentada na figura 16 tem as classes de acesso ao *DataWarehouse*. Para este acesso a classe `Factory` fornece uma fábrica de objetos únicos (padrão *Singleton*) de conexão com o banco de dados. Para as classes `SegmentacaoDAO`, `PreferenciaDAO` e `ClusterDAO` a construção dos métodos segue lógica semelhante. No quadro 6 são elencados os principais métodos de cada uma das classes.

| Método | Descrição |
|---|---|
| `getConnection(): Connection;` | Retorna uma conexão com o banco |
| `getPreferenciaDAO(): PreferenciaDAO`<br>`getClusterDAO(): ClusterDAO`<br>`getSegmentacaoDAO(): SegmentacaoDAO` | Retorna um objeto único de conexão com o banco |
| `existe(): boolean` | Checa a existência de um objeto de tipo da classe |
| `incluir(Objeto): void` | Inclui um objeto da classe |

| | |
|---|---|
| `alterar(Objeto): void` | Altera um objeto da classe |
| `excluir(Objeto): void` | Exclui um objeto da classe |
| `ler(): Segmentacao` | Efetua a leitura no banco dos valores de um objeto da classe |
| `listar(): List<Segmentacao>` | Retorna uma lista com os objetos presentes na tabela correspondente do `Data WareHouse` |
| `objeto(): Segmentacao` | Constrói os atributos do objeto para uso pelos outros métodos |

Quadro 6 – Descrição dos métodos das classes do pacote `DAO`

### 3.2.4 Diagrama de sequência

O diagrama de sequência da figura 17 demonstra o fluxo das execuções do framework na chamada do método segmentar da classe `PreferenciasUC`. Primeiro a instanciação e inclusão das preferências no banco de dados do framework. Após, a escrita do arquivo ARFF. Em seguida, a instanciação e inclusão dos clusters neste banco de dados. Finalizando, a instanciação e inclusão do resultado da segmentação.

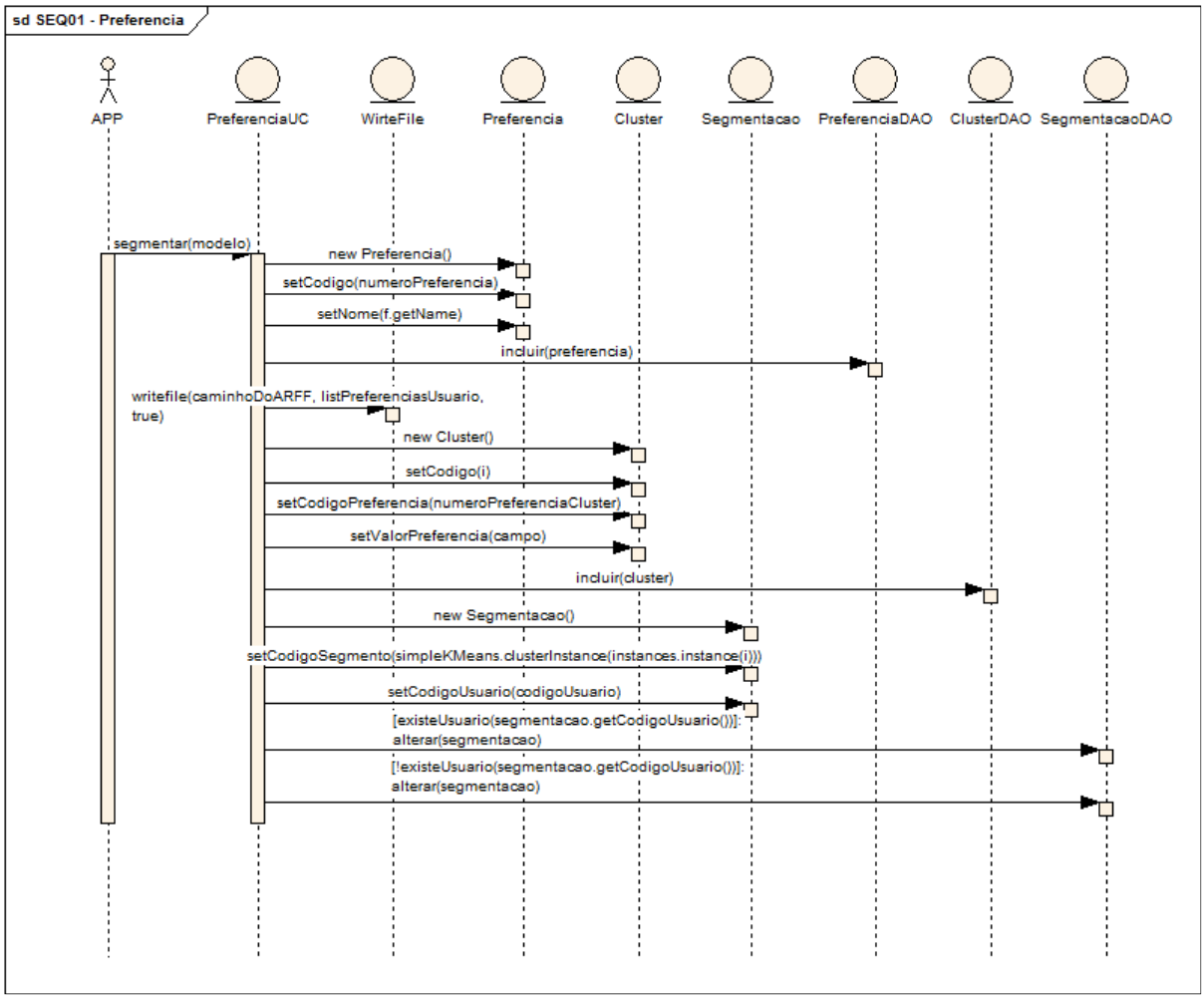



Figura 17 – Diagrama de sequência do *framework* Beta

### 3.2.5 Arquitetura do software

A arquitetura do *framework* foi construída em torno das necessidades que a aplicação deve apresentar no momento do uso. Para isso, inicialmente desenvolveu-se uma aplicação de teste. Em torno disso, procurou-se atingir o objetivo da pesquisa e, simultaneamente, manter compatibilidade com a aplicação.

A figura 18 ilustra o modelo de arquitetura adotado para *app* teste. O modelo utiliza MVC e o padrão DAO para acesso a base de dados.

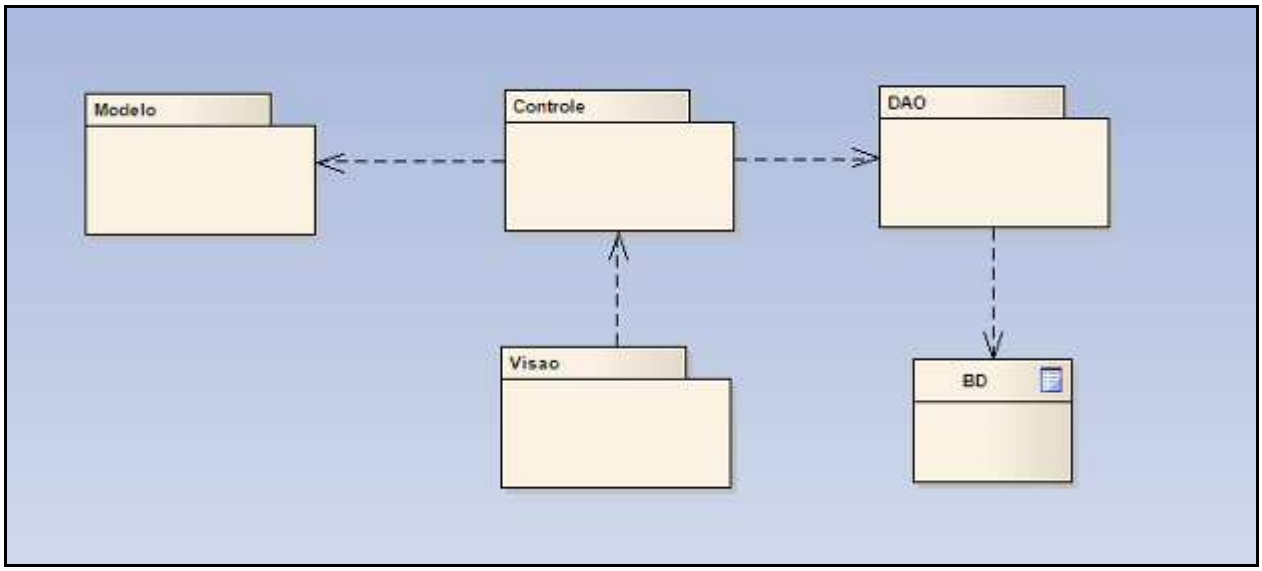



Figura 18 – Arquitetura da APP que fará uso do *framework*

Através da referida *app*, procurou-se alcançar o modelo de arquitetura que se adequasse ao desenvolvimento do *framework*. No início da implementação, a escolha de filtros foi elencada utilizando-se o estilo pipes e filtros. Por este estilo lidar com várias etapas de processamento não se observou necessidade para o seu uso. Optou-se, então, pelo estilo em camadas, por ser este o que melhor adaptou-se aos problemas encontrados além de não apresentar um modelo de arquitetura que exija implementações adicionais desnecessárias ao desenvolvimento do framework. A figura 19 ilustra o modelo de arquitetura do *framework*.

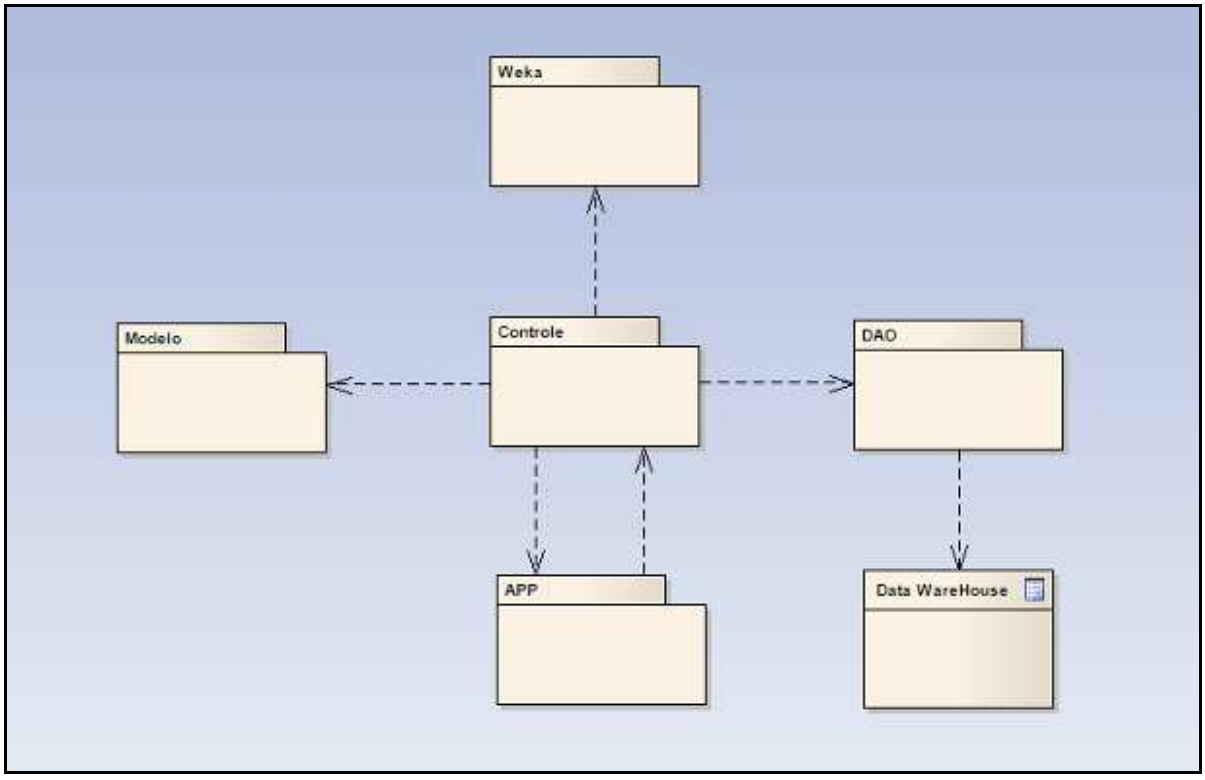



Figura 19 – Arquitetura do *framework*

Como pode ser visto na figura 19, a aplicação acessa e requisita dados do *framework* através da sua camada de controle. A camada de controle do *framework* é responsável por acessar as classes do modelo, a biblioteca do *Weka* e povoar através do *partner DAO* o banco de dados do *framework.*

### 3.2.6 Modelo Relacional

A figura 20 apresenta a modelagem do *DataWarehouse* projetado para execução com o *framework*. No modelo podem ser visualizadas as tabelas *Preferencia*, *Cluster* e *Segmentacao*. A tabela *Preferencia* armazena as possíveis preferências do *cluster*. A tabela *Cluster* armazena o código de identificação do *cluster*, os códigos de preferências e os valores para essas preferências. A tabela segmentação armazena o código do cluster de nome *cd_segmento* e o código do usuário.

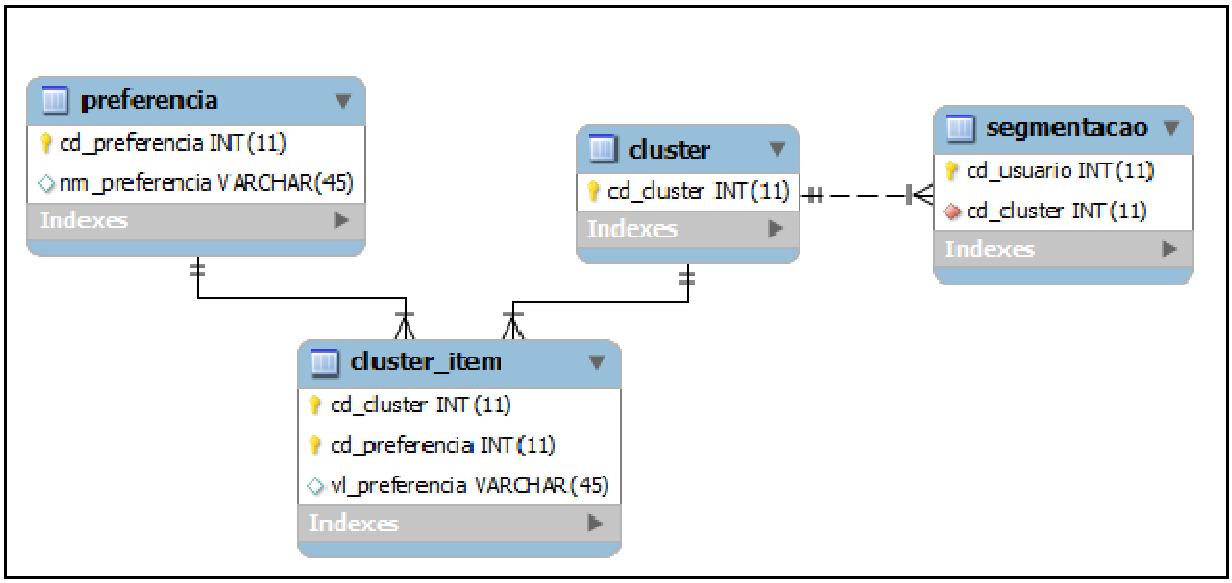



Figura 20 – Dicionário de dados

## 3.3 IMPLEMENTAÇÃO

A implementação do *framework* procurou seguir o processo de desenvolvimento de software, a divisão de responsabilidades do *framework* foi construída em camadas como já demonstrado na escolha do estilo arquitetural. A partir disso, foram necessárias tecnologias no

decorrer do desenvolvimento e desta forma chegou-se à implementação.

A seguir são discutidas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

### 3.3.1 Tecnologias e padrões utilizados

As seguintes tecnologias e padrões foram utilizadas para a construção do *framework*:

a) linguagem Java: linguagem de altíssimo nível, alta portabilidade e que permite integração com *Weka*;
b) IDE eclipse: IDE bastante usada e com vários recursos além de uma gama de linguagens suportadas;
c) biblioteca *WEKA*: biblioteca com vários recursos para área de inteligência artificial;
d) estilo arquitetural em camadas: estilo bastante conhecido e difundido devido a flexibilidade na construção e baixo acoplamento fornecido;
e) *reflection*: recurso que permite a um objeto conhecer a si mesmo;
f) *anotations*: uso bastante variado, sendo aqui usado para marcar alguns atributos e diferenciá-los;
g) padrão de projeto DAO: *Data Acess Object*, permite fornecer um objeto que execute todas as etapas necessárias para ligação e operação com banco de dados,
h) padrão de projeto *Factory*: uma fábrica de objetos que aqui é utilizada para criação de objetos de conexão com o banco de dados;
i) padrão de projeto *Singleton*: permite a instância única de objetos;
j) banco de dados MySQL: banco de dados muito utilizado e que permitiu executar todas as operações necessárias ao desenvolvimento.

### 3.3.2 Código fonte

O fluxo de execução do framework é controlado na quase totalidade por dois métodos da classe `Preferencia` nomeados como `segmentar` e `getSegmentacao.` A partir destes métodos outros objetos do framework são criados e outros métodos chamados. O código dos métodos pode ser melhor visualizado no diagrama de atividades da figura 21 e no quadro 7 os quais descrevem o método `segmentar` e no quadro 8 o qual descreva o método

`getSegmentacao`.

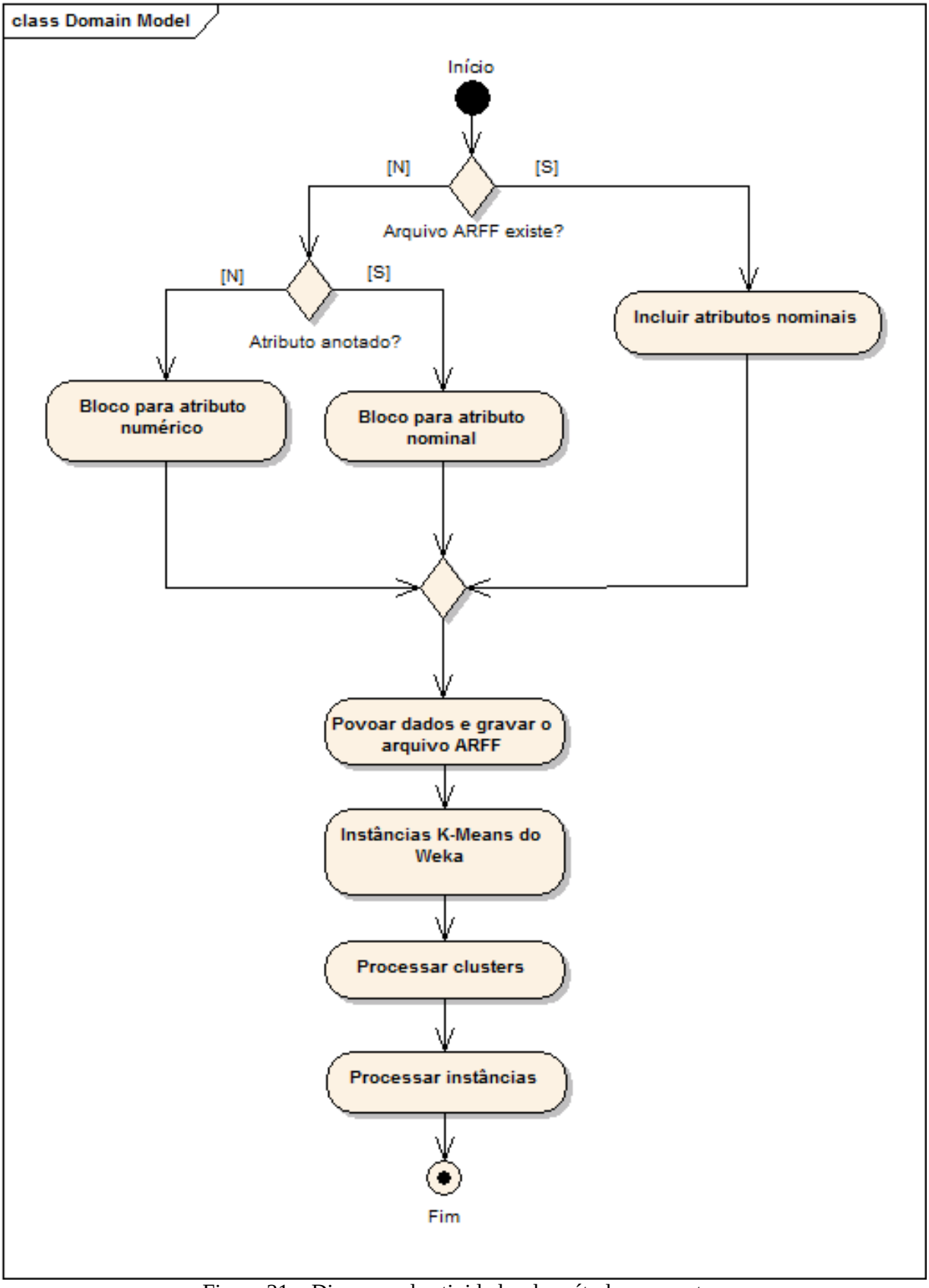



Figura 21 – Diagrama de atividades do método segmentar

```
 1 public void segmentar(List<T> preferenciasList, int numClusters, String
   caminhoDoARFF) throws Exception {
 2    if (preferenciasList.size() == 0)
 3           return;
 4    T preferenciaObjeto = preferenciasList.get(0);
 5    Class classe = preferenciaObjeto.getClass();
 6    Field[] fs = classe.getDeclaredFields();
 7    boolean append = true;
 8    String listPreferenciasUsuario = "";
 9    if (!WriteFileUC.exitsfile(caminhoDoARFF)) {
10       listPreferenciasUsuario = "@RELATION BeTa\n\n";
11       int numeroPreferencia = 1;
12       for (Field f : fs) {
13          if (f.isAnnotationPresent(AtributoDiscreto.class)) {
14             f.setAccessible(true);
15             Set<String> conjuntoDiscreto = new HashSet<String>();
16             for (T preferenciax : preferenciasList) {
17                conjuntoDiscreto.add(f.get(preferenciax).toString());
18             }
19             String valoresDiscretos = "";
20             boolean primeiraColuna = true;
21             for (Iterator<String> iterator = conjuntoDiscreto.iterator();
   iterator.hasNext();) {
22                if (primeiraColuna) {
23                   valoresDiscretos = valoresDiscretos + (String)
   iterator.next();
24                   primeiraColuna = false;
25                } else {
26                   valoresDiscretos = valoresDiscretos + ", " + (String)
   iterator.next();
27                }
28             }
29             Preferencia preferencia = new Preferencia();
30             preferencia.setCodigo(numeroPreferencia);
31             preferencia.setNome(f.getName());
32             PreferenciaDAO dao = FactoryDAO.getPreferenciaDAO();
33             dao.incluir(preferencia);
34             listPreferenciasUsuario = listPreferenciasUsuario + "@ATTRIBUTE " +
   f.getName() + "{" + valoresDiscretos + "}\n";
35             numeroPreferencia++;
36          } else {
37             f.setAccessible(true);
38             Preferencia preferencia = new Preferencia();
39             preferencia.setCodigo(numeroPreferencia);
40             preferencia.setNome(f.getName());
41             PreferenciaDAO dao = FactoryDAO.getPreferenciaDAO();
42             dao.incluir(preferencia);
43             listPreferenciasUsuario = listPreferenciasUsuario + "@ATTRIBUTE " +
   f.getName() + " NUMERIC\n";
44             numeroPreferencia++;
45          }
46       }
47       listPreferenciasUsuario = listPreferenciasUsuario + "\n" + "@DATA" + "\n";
48       append = false;
49    } else {
50       StringBuffer instancesStringBuffer = null;
51       Instances instances = null;
52       File file = new File(caminhoDoARFF);
53       ArffLoader arffLoader = new ArffLoader();
54       arffLoader.setFile(file);
55       instances = arffLoader.getDataSet();
56       instancesStringBuffer = new StringBuffer(instances.toString());
57       for (Field f : fs) {
58          if (f.isAnnotationPresent(AtributoDiscreto.class)) {
59             f.setAccessible(true);
60             Set<String> conjuntoDiscreto = new HashSet<String>();
```

```
 61             for (T preferenciax : preferenciasList) {
 62                conjuntoDiscreto.add(f.get(preferenciax).toString());
 63             }
 64             for (Iterator<String> iterator = conjuntoDiscreto.iterator();
    iterator.hasNext();) {
 65                int numValues = instances.attribute(f.getName()).numValues();
 66                boolean thereValueAttribute = false;
 67                String nextIterator = iterator.next();
 68                for (int i = 0; i < numValues; i++) {
 69                   String valueAttribute =
    instances.attribute(f.getName()).value(i);
 70                   if (nextIterator.equals(valueAttribute)) {
 71                      thereValueAttribute = true;
 72                   }
 73                }
 74                if (!thereValueAttribute) {
 75                   Integer offset = instancesStringBuffer.indexOf(f.getName());
 76                   char getName[] = f.getName().toCharArray();
 77                   offset = offset + getName.length + 2;
 78                   nextIterator = nextIterator + ", ";
 79                   instancesStringBuffer.insert(offset, nextIterator);
 80                   listPreferenciasUsuario = instancesStringBuffer.toString();
 81                   append = false;
 82                }
 83                f.setAccessible(true);
 84             }
 85          }
 86       }
 87    }
 88    boolean primeiraColuna = true;
 89    for (T preferenciax : preferenciasList) {
 90       listPreferenciasUsuario = listPreferenciasUsuario + "\n";
 91       primeiraColuna = true;
 92       for (Field f : fs) {
 93          f.setAccessible(true);
 94          if (!Modifier.isTransient(f.getModifiers()))
 95             if (!primeiraColuna) {
 96                listPreferenciasUsuario = listPreferenciasUsuario + "," +
    f.get(preferenciax);
 97             } else {
 98                listPreferenciasUsuario = listPreferenciasUsuario +
    f.get(preferenciax);
 99                primeiraColuna = false;
100             }
101          }
102       }
103       WriteFileUC.writefile(caminhoDoARFF, listPreferenciasUsuario, append);
104
105       Instances instances = null;
106       SimpleKMeans simpleKMeans = null;
107       try {
108          File file = new File(caminhoDoARFF);
109          ArffLoader arffLoader = new ArffLoader();
110          arffLoader.setFile(file);
111          instances = arffLoader.getDataSet();
112          simpleKMeans = new SimpleKMeans();
113          simpleKMeans.setNumClusters(numClusters);
114          simpleKMeans.buildClusterer(instances);
115          Instances instancesCentroids = simpleKMeans.getClusterCentroids();
116          String copyInstance = "";
117          for (int i = 0; i < instancesCentroids.numInstances(); i++) {
118             copyInstance = instancesCentroids.instance(i).copy() + "\n";
119             char[] charArrayCopyInstance = copyInstance.toCharArray();
120             String valorPreferencia = "";
121             int numeroPreferenciaCluster = 1;
122             for (int j = 0; j < charArrayCopyInstance.length; j++) {
123                if (charArrayCopyInstance[j] == ',' || j ==
    charArrayCopyInstance.length - 1) {
```

```
124                     Cluster cluster = new Cluster();
125                     cluster.setCodigo(i + 1);
126                     ClusterItem clusterItem = new ClusterItem();
127                     clusterItem.setCodigo(i + 1);
128                     clusterItem.setCodigoPreferencia(numeroPreferenciaCluster);
129                     clusterItem.setValorPreferencia(valorPreferencia);
130                     ClusterDAO daoCluster = FactoryDAO.getClusterDAO();
131                     if (daoCluster.existeCluster(cluster.getCodigo())) {
132                        daoCluster.alterar(cluster);
133                     } else {
134                        daoCluster.incluir(cluster);
135                     }
136                     ClusterItemDAO daoClusterItem =
    FactoryDAO.getClusterItemDAO();
137                     if (daoClusterItem.existeCluster(clusterItem.getCodigo(),
    clusterItem.getCodigoPreferencia())) {
138                        daoClusterItem.alterar(clusterItem);
139                     } else {
140                        daoClusterItem.incluir(clusterItem);
141                     }
142                     valorPreferencia = "";
143                     numeroPreferenciaCluster++;
144                  } else {
145                     valorPreferencia = valorPreferencia +
    charArrayCopyInstance[j];
146                  }
147               }
148            }
149            for (int i = 0; i < instances.numInstances(); i++) {
150               Segmentacao segmentacao = new Segmentacao();
151 segmentacao.setCodigoCluster(simpleKMeans.clusterInstance(instances.instance(i))
    + 1);
152               copyInstance = instances.instance(i).copy() + "\n";
153               char[] charArrayCopyInstance = copyInstance.toCharArray();
154               String codigoUsuario = "";
155               boolean incluiuCodigoUsuario = false;
156               for (int j = 0; j < charArrayCopyInstance.length; j++) {
157                  if (!incluiuCodigoUsuario) {
158                     if (charArrayCopyInstance[j] == ',' || j ==
    charArrayCopyInstance.length - 1) {
159                        int codigoUsuarioInt = Integer.parseInt(codigoUsuario);
160                        segmentacao.setCodigoUsuario(codigoUsuarioInt);
161                        incluiuCodigoUsuario = true;
162                     } else {
163                        codigoUsuario = codigoUsuario + charArrayCopyInstance[j];
164                     }
165                  }
166               }
167               SegmentacaoDAO dao = FactoryDAO.getSegmentacaoDAO();
168               try {
169                  if (dao.existeUsuario(segmentacao.getCodigoUsuario())) {
170                     dao.alterar(segmentacao);
171                  } else {
172                     dao.incluir(segmentacao);
173                  }
174               } catch (Exception e) {
175                  e.printStackTrace();
176               }
177            }
178         } catch (Exception e) {
179            e.printStackTrace();
180         }
181 }
```

Quadro 7 – Método `segmentar`

A assinatura do método `segmentar` que é chamado pela aplicação para efetuar a segmentação inclui três parâmetros sendo: (i) a lista de preferências passada pela aplicação representado por `List<T> preferenciasList`; (ii) o número de clusters para segmentação representado por `int numClusters`; (iii) o caminho para leitura e gravação do ARFF representado por `String caminhoDoARFF`. A assinatura do método pode ser melhor visualizada na linha 1.

Ao receber os parâmetros da aplicação o *framework* inicia no mesmo instante o processo de recepção da lista de preferências. A recepção da lista de preferências é feita usando a api de reflexão do Java. O início do processo de recepção da lista pode ser melhor visualizado na linha 4.

Na linha citada é inicialmente instanciado um objeto `preferenciaObjeto` de tipo `T`. Este objeto recebe as preferências do índice 0 da lista. O índice 0 contem os atributos de classe do objeto `preferenciaObjeto`. Prosseguindo na linha 5 é feita a intanciação de um objeto do tipo `Class` que recebe em tempo de execução a classe do objeto `preferenciaObjeto`. Na linha 6 é feita instanciação de uma lista de fields recebendo as fields declaradas da classe.

Antes do início da contrução do ARFF ainda são criadas alguns atributos necessários ao processamento do ARFF, demonstrados na linha 7 e 8.

Na linha 9 é demonstrado o atributo `append` que tem por responsabilidade indicar a substituição ou incremento do ARFF e na linha 10 o atributo `listaPreferenciasUsuario` que tem a função de armazenar a estrutura a ser incluída no ARFF.

Ao início da construção do ARFF é testada a existência do arquivo. Para o caso do arquivo ARFF já existir este deve estar obrigatoriamente salvo no local informado pela aplicação. Se o arquivo ARFF não existir são iniciados os passos para sua construção. A linha 9 mostra o início desse processo.

O atributo `listPreferenciasUsuario` recebe o cabeçalho do ARFF na linha10. É criado então um atributo para contabilizar o numero de preferências recebidas pela aplicação na linha 11. Neste momento as `fields` são iteradas em busca dos atributos que podem ser numéricos ou discretos na linha 12.

Na linha 13 a descoberta do atributo discreto é feita por meio da anotação AtributoDiscreto contida no *framework*. A anotação pode ser visualizada no quadro 8.

```
package framework.modelo;

import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;

@Retention(RetentionPolicy.RUNTIME)
public @interface AtributoDiscreto {

}
```

Quadro 8 – Anotação `AtributoDiscreto`

A anotação do quadro 8 permite ao *framework* diferenciar os atributos numéricos dos discretos. Se o teste reconhecer o atributo discreto no quadro 7 o primeiro passo é delegar acesso aos atributos da `field` como mostrado na linha 14. Após todos os valores dessa `field` são lidos e armazenados em um conjunto nas linha 15 a 18. A escolha de um conjunto permite guardar valores não duplicados evitando processamento desnecessário ao percorrer posteriormente esses dados. Também a criação de um atributo nomeado como `valoresDiscretos` para gravar por meio de uma iteração do conjunto a lista de valores possíveis para o atributo nas linhas 21 a 28. Então o nome do atributo é salvo como uma preferência no `Data WareHouse` através de um DAO. O DAO utilizado para salvar a preferência é exposto no quadro 9.

```
 1 package framework.dao;
 2
 3 public class PreferenciaDAO {
 4    private PreparedStatement psIncluir;
 5    private PreparedStatement psAlterar;
 6    private PreparedStatement psExcluir;
 7    private PreparedStatement psLer;
 8    private PreparedStatement psListar;
 9    private PreparedStatement psIncluirNovo;
10
11    public PreferenciaDAO(Connection con) throws SQLException {
         psIncluir = con.prepareStatement("INSERT INTO preferencia
12 (cd_preferencia, nm_preferencia) VALUES (?, ?)");
         psAlterar = con .prepareStatement("UPDATE preferencia SET
13 cd_preferencia=?, nm_preferencia=? WHERE cd_preferencia=?");
         psExcluir = con.prepareStatement("DELETE FROM preferencia WHERE
14 cd_preferencia=?");
         psLer = con.prepareStatement("SELECT cd_preferencia, nm_preferencia
15 FROM preferencia WHERE cd_preferencia=?");
         psIncluirNovo = con.prepareStatement("SELECT cd_preferencia,
16 nm_preferencia FROM preferencia WHERE cd_preferencia=?");
         psListar = con.prepareStatement("SELECT cd_preferencia,
17 nm_preferencia FROM preferencia");
18    }
19
20    public boolean existeUsuario(int codigo) throws Exception {
21       psIncluirNovo.setInt(1, codigo);
22       ResultSet var = psIncluirNovo.executeQuery();
23       boolean var2 = false;
24       if (var.next()) {
25          var2 = true;
26       }
27       var.close();
28       return var2;
29    }
30
```

```
31     public void incluir(Preferencia preferencia) throws SQLException {
32        psIncluir.setInt(1, preferencia.getCodigo());
33        psIncluir.setString(2, preferencia.getNome());
34        psIncluir.executeUpdate();
35     }
36
37     public void alterar(Preferencia preferencia) throws SQLException {
38        psAlterar.setInt(1, preferencia.getCodigo());
39        psAlterar.setString(2, preferencia.getNome());
40        psAlterar.setInt(3, preferencia.getCodigo());
41        psAlterar.executeUpdate();
42     }
43
44     public void excluir(Preferencia preferencia) throws SQLException {
45        psExcluir.setInt(1, preferencia.getCodigo());
46        psExcluir.setInt(2, preferencia.getCodigo());
47        psExcluir.executeUpdate();
48     }
49
50     public Preferencia ler(int cd) throws SQLException {
51        Preferencia preferencia = null;
52        psLer.setInt(1, cd);
53        ResultSet rs = psLer.executeQuery();
54        if (rs.next()) {
55           preferencia = objeto(rs);
56        }
57        return preferencia;
58     }
59
60     public List<Preferencia> listar() throws SQLException {
61        List<Preferencia> lista = new ArrayList<Preferencia>();
62        ResultSet rs = psListar.executeQuery();
63        while (rs.next()) {
64           lista.add(objeto(rs));
65        }
66        rs.close();
67        return lista;
68     }
69
70     public Preferencia objeto(ResultSet rs) throws SQLException {
71        Preferencia preferencia = new Preferencia();
72        preferencia.setCodigo(rs.getInt("cd_preferencia"));
73        preferencia.setNome(rs.getString("nm_preferencia"));
74        return preferencia;
75     }
76  }
```

Quadro 9 – Método responsável pela inclusão para o DAO de preferência

O quadro 9 mostra inicialmente os atributos para conexão com o banco das linhas 4 a 9. Mostra também o construtor da classe `PreferenciaDAO` na linha 11 com suas operações SQL e os demais métodos ao qual vale ressaltar o método `incluir` chamado na linha 33 do quadro 7. Este método recebe um objeto de Preferência a linha 31 do quadro 9, seta seus valores e efetua a operação nas linhas 32 a 34. No quadro 7 finalizando o processamento do atributo discreto a string que armazena os dados do ARFF chamada `listPreferenciasUsuario` recebe os dados do atributo discreto juntamente com seus valores possíveis na linha 34. Então o atributo `numeroPreferencia` é incrementado na linha 35 preparando a iteração para próxima preferência.

Para o caso do atributo não estar anotado no teste do quadro 7 na linha 13 é efetuado o

processamento descrito no bloco que consta nas linhas 36 a 45 do quadro 7.

Nas linhas citadas ocorre o processamento do atributo numérico que tem sequência de implementação semelhante ao atributo discreto, mas sem o teste para anotação e também sem valores possíveis para o atributo por ser esse um atributo numérico.

Para terminar a estrutura do ARFF no caso de não existência do arquivo é inserido também o código mostrada nas linhas 47 e 48. Nessas linhas a variável que armazena a estrutura do ARFF recebe o último item a ser incluído no ARFF anteriormente aos dados. Além disso, é atribuído ao `append` o valor `true` indicando a substituição do arquivo se esse existir.

Se o arquivo já existir como pode ser visualizado no teste da linha 9 é executada então o início da rotina de checagem do arquivo. Na linha 50 é criado um buffer para armazenar os dados, em seguida é instanciado um `Instances` presente na biblioteca do Weka que recebera as instâncias do ARFF. É criado então um `File(caminhoDoARFF)` na linha 52 com o arquivo referenciado e posteriormente carregado através do `ArffLoader`. No carregador é setado o arquivo e então é atribuída a instância os dados presentes no arquivo. Todos os dados da instância são atribuídos ao `buffer` para manipulação.

Com os dados no buffer é iniciada a verificação dos dados do ARFF conforme os novos dados fornecidos na lista de preferências. Essa verificação é necessária em função dos atributos discretos necessitarem de valores possíveis para os atributos antes da entrada dos dados do ARFF. O bloco citado pode ser visualizado nas linhas 57 a 87. Nesse bloco a rotina de verificação do ARFF para atributo discreto com existência do arquivo tem início semelhante a rotina de não existência do arquivo até o ponto onde o conjunto é iterado na linha 64. A partir deste ponto o número de valores possíveis do atributo que esta no ARFF é atribuído ao inteiro `numValues` e então criado um atributo para sinalizar a existência do valor do atributo na linha 66. O string `nextIterator` guarda o próximo valor do conjunto. É efetuada a iteração dos valores do atributo ARFF na linha 68. Nessa iteração que ocorre da linha 68 a 73, o valor do conjunto que contem dados da lista passada pela aplicação é comparado com o valor do atributo ARFF. Se os valores forem iguais é atribuído verdade ao atributo `thereValueAttribute` indicando que o valor foi encontrado no ARFF. Em seguida ocorre na linha 74 o teste para verificar se o valor não existe no ARFF. Nesse caso é criado o atributo offset para guardar o endereço do nome do atributo no ARFF. Também um array de char que contem o numero de caracteres do nome a linha 76. Então o `offset` recebe seus valores mais o tamanho do nome mais dois caracteres correspondentes ao espaço

em branco e a vírgula na linha 77. Com isso o `offset` guarda o local exato para inserção no ARFF. O atributo `nextIterator` recebe o valor do atributo mais a virgula. O valor é inserido através do método `instancesStringBuffer.insert(offset, nextIterator)`. A lista de preferências recebe o `buffer` e ao `append` é atribuído `false` indicando a substituição do arquivo.

Após a rotina de verificação são efetuados o povoamento dos dados e a gravação do arquivo nas linhas 88 a 103. Após a criação de um `booleano` na linha 88 para informar a coluna lida é efetuada a iteração da lista de preferências na linha 89. A string que recebe as preferências para gravação no ARFF recebe uma quebra de linha para posicionar os dados após a marcação `@DATA` do ARFF na linha 90. A `primeiraColuna` recebe `true` para informar o posicionamento na primeira coluna do arquivo na linha seguinte. As fields passam a ser iteradas das linhas 92 a 102 e a partir daí, se o posicionamento não estiver na `primeiraColuna` são adicionados na string `listPreferenciasUsuario` os valores das preferências mais a vírgula. Se o posicionamento se encontra na `primeiraColuna` são adicionados na string somente o valor da preferência. Findado o processo a gravação é procedida à gravação do arquivo na linha 103 utilizando a atributo `append` para indicar a substituição ou incremento do arquivo.

Com a finalização da entrada de dados, inclusão das preferências e gravação do arquivo é iniciado o processamento dos clusters. Bloco este que vai das linhas 105 a 148. O bloco é iniciado com a instanciação dos objetos `intances` e `simpleKMeans` nas linhas 105 e 106. É então iniciado um bloco de exceção onde o arquivo é carregado e recebido na instancia nas linha 108 a linha 111. Dentro do bloco é criado um novo objeto `SimpleKMeans` para construção do conjunto de `clusters`. É setado nesse objeto o numero de clusters na linha 113 e efetuada a chamada de um método passando as instancias do ARFF como parâmetro na linha 114. Na linha 115 é criada uma instancia para recebimento dos centroids também realizado na mesma linha. Então é criado um atributo string para receber as copias de cada instancia dos centroids.

Findado esse processo é iniciado o código que fará o armazenamento e atualização dos clusters. A linha 117 mostra a iteração das instancias dos centroids. A cópia da instancia atual de índice i é feita na linha 118. Essa cópia é então convertida para um `Array` de `char`, criando uma `String` para armazenar o valor das preferências do cluster e um `int` para o número da preferência nas linhas 119, 120 e 121. Um `Array` de `char` é então iterado em busca do caractere vírgula ou final de linha da cópia. Nesse laço são criados objetos de

`Cluster` e `ClusterItem` e setados seus atributos das linhas 124 a 129. É realizada então a criação de um `DAO` e procedido o teste para verificar a existência e inclusão ou alteração dependendo do resultado. Esse processo ocorre em ambos os objetos e pode ser visualizado das linhas 130 a 143. Nas linhas 142 e 143 o valor da preferência é limpo para a próxima iteração assim como incrementado o atributo `numeroPreferenciaCluster.` Na linha 144 é executada a cláusula `else` para o caso de ainda não ter encontrado vírgula ou fim de linha que indica a leitura por completo da preferência. Nesse caso ao `valorPreferencia` é adicionado o próximo caractere.

Finalizado o bloco dos centroids é dado continuidade à execução com o bloco responsável por armazenar as instâncias do conjunto de dados das preferências. O bloco citado pode ser visto da linha 149 a 181. O bloco é iniciado com a iteração das instancias dos dados que representam as preferências na linha 149. A seguir é criado um objeto de `Segmentacao` e setado com o número do cluster correspondente a instancia recebido através do método `clusterInstance` com a passagem do parâmetro que representa a instancia mais o número 1. Este um ajuste necessário para inclusão no `Data WareHouse`. Na linha 152 é criado uma cópia para leitura da instancia que na linha a seguir é convertida para um `Array` de caracteres. Prossegue o algoritmo com a criação de uma String para armazenar o código do usuário na linha 154 e a criação de um atributo para verificação da inclusão na linha 155. O `array` é percorrido de forma idêntica ao bloco que inicia na linha 122. Ao encontrar a vírgula ou final de linha o atributo `codigoUsuario` é convertido para `int` para posterior armazenamento através da sua passagem no método `setCodigoUsuario` do objeto `segmentação.` Se a vírgula ou final de linha não forem encontrados a execução prossegue varrendo o `Array` e adicionando caracteres a `String` que guarda o código. Para terminar o bloco na linha 167 é criado um `DAO` e efetuada a gravação ou alteração dependente do teste da linha 169. O bloco que armazena os segmentos finaliza o método `segmentar` da classe `Preferencia`.

A classe `Preferencia` tem ainda outro método que é responsável por devolver o resultado do processamento para a aplicação quando solicitado. O método citado pode ser visualizado no quadro 10.

```
 1 public List<Preferencia> getSegmentacao(int usuario) throws Exception {
 2    SegmentacaoDAO daoSegmentacao = FactoryDAO.getSegmentacaoDAO();
 3    Segmentacao segmentacao = daoSegmentacao.ler(usuario);
 4    ClusterItemDAO daoClusterItem = FactoryDAO.getClusterItemDAO();
 5    List<ClusterItem> clusterItemList =
   daoClusterItem.listar(segmentação.getCodigoCluster());
 6    ArrayList<Preferencia> preferenciaList = new ArrayList<Preferencia>();
 7    for (ClusterItem clusterItem : clusterItemList) {
 8       PreferenciaDAO daoPreferencia = FactoryDAO.getPreferenciaDAO();
 9       Preferencia preferencia = null;
10       preferencia = daoPreferencia.ler(clusterItem.getCodigoPreferencia());
11       preferencia.setValor(clusterItem.getValorPreferencia());
12       preferenciaList.add(preferencia);
13    }
14    return preferenciaList;
15 }
```

Quadro 10 – Método `getSegmentacao`

O método `getSegmentacao` apresenta primeiramente na linha 1 a assinatura do método o qual recebe como parâmetro o código do usuário a efetuar a busca. Após são instanciados objetos `DAO` para busca do segmento onde o usuário esta inserido. A busca do segmento na linha 3, a busca das preferências dos `clusters` relacionado com o segmento na linha 5, a busca das preferências dos valores do `cluster` no percorrimento do `Array` da linha 7 a 13 e por fim a armazenagem de todos esses dados em uma lista de preferências que é retornada para aplicação na linha 14.

## 3.4 ESTUDO DE CASO

A figura 21 mostra a modelagem do banco de dados da aplicação que faz uso do *framework*. Nela, somente as informações necessárias à implementação do *framework* que se restringem às tabelas cliente, pedido, pedido item e produto são apresentadas.

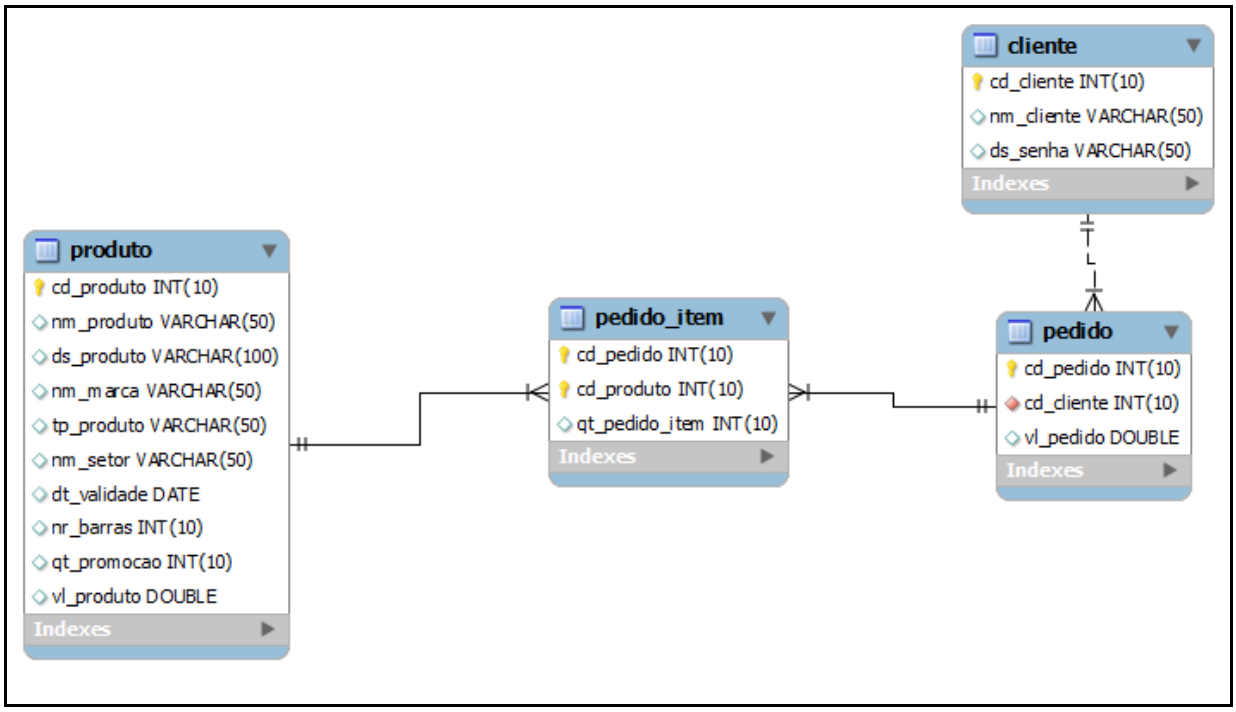



Figura 22 – *Data WareHouse* para mineração de dados pelo *framework*

Por meio do *Data WareHouse* da Figura 21, a aplicação pode fornecer os dados necessários para o *framework* efetuar a segmentação como exposto no quadro 7.

```
 1 public void incluir(int codigoPedido, int codigoProduto, int quantidade)
   throws Exception {
 2    PedidoItem pedidoItem = new PedidoItem();
 3    pedidoItem.setCodigoPedido(codigoPedido);
 4    pedidoItem.setCodigoProduto(codigoProduto);
 5    pedidoItem.setQuantidade(quantidade);
 6    PedidoItemDAO dao = FactoryDAO.getPedidoItemDAO();
 7    dao.incluir(pedidoItem);
 8    ProdutoDAO daoProduto = FactoryDAO.getProdutoDAO();
 9    Produto produto = daoProduto.ler(pedidoItem.getCodigoProduto());
10    PedidoDAO daoPedido = FactoryDAO.getPedidoDAO();
11    Pedido pedido = daoPedido.ler(codigoPedido);
12    Preferencia preferencia = new Preferencia();
13    preferencia.setCodigo(pedido.getCodigoCliente());
14    preferencia.setBarras(produto.getBarras());
15    preferencia.setDescricao(produto.getDescricao());
16    preferencia.setMarca(produto.getMarca());
17    preferencia.setNome(produto.getNome());
18    preferencia.setPromocao(produto.getPromocao());
19    preferencia.setSetor(produto.getSetor());
20    preferencia.setTipo(produto.getTipo());
21    preferencia.setValidade(produto.getValidade());
22    preferencia.setValor(produto.getValor());
23    List<Preferencia> listaPreferencia = new ArrayList<Preferencia>();
24    listaPreferencia.add(preferencia);
25    PreferenciaUC<Preferencia> preferenciax = new
   PreferenciaUC<Preferencia>();
26    preferenciax.segmentar(listaPreferencia, 5,
   "C:/Users/RUDIMAR/Documents/beta");
27 }
```

Quadro 11 – Método da aplicação que realiza a chamada ao *framework*

No quadro 11 a aplicação seta os valores das preferências do usuário da linha 12 a 22. Esse objeto `Preferencia` é propriedade da aplicação e pode ser criado com quantos

atributos forem necessários. O quadro 12 demonstra uma parte da classe preferências da aplicação.

```
1  public class Preferencia {
2     private int codigo;
3     @AtributoDiscreto
4     private String marca;
5     @AtributoDiscreto
6     private String nome;
7     @AtributoDiscreto
8     private String descricao;
9     @AtributoDiscreto
10    private String tipo;
11    @AtributoDiscreto
12    private String setor;
13    private int validade;
14    private int barras;
15    private int promocao;
16    private Double valor;
```

Quadro 12 – Classe `Preferencia` da aplicação

No quadro 12, expõem-se os atributos para uma classe `Preferencia` utilizada pela aplicação. Nesta classe, pode ser visto ainda como anotar um atributo discreto se surgir à necessidade, sendo esta anotação necessária para indicar ao *framework* que o atributo anotado é do tipo nominal e pode aceitar uma lista de valores declarados no início do ARFF. Esse é o caso da linha 3 à linha 12.

Após a criação do objeto de `Preferencia` e atribuição dos valores, é necessária a criação de uma lista de preferências e adição a ela nas linhas 23 e 24. Na linha 25, instancia um novo objeto de `PreferenciaUC` do *framework*. Nesse objeto, faz-se a chamada ao método segmentar da linha 26 passando como parâmetro a lista, a quantidade de *clusters* e o local para salvamento do ARFF demonstrando assim o uso do *framework* pela aplicação.

Na busca pelo resultado, procedeu-se à aplicação de alguns testes que objetivavam visualizar o funcionamento do *framework*, assim como também mostrar, após a segmentação, a estrutura do arquivo ARFF completo e verificar no modelo relacional do framework os resultados apresentados.

Para isso, foram criados na aplicação que faz uso do *framework* 10 usuários e 50 produtos distintos. A figura 22 ilustra a criação do usuário de código 39 e a figura 23 a adição dos produtos de 1 a 5.


Figura 23 – Criação do usuário


Figura 24 – Adição de 5 produtos distintos

O processo das figuras 22 e 23 é repetido várias vezes até completarem-se 10 usuários, cada qual com cinco produtos distintos. Cada produto tem característica próprias. Estas características foram removidas da aplicação para facilitar a demonstração. O quadro 26 mostra parte do arquivo ARFF após o processo de inclusão dos 10 usuários e seus produtos relacionados.

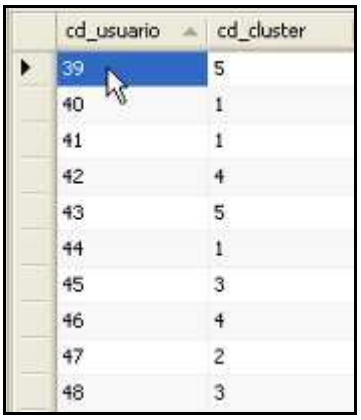
```
@relation BeTa
@attribute codigo numeric
@attribute marca {marca1,
marca2,marca3,marca4,marca5,marca6,marca7,marca8,marca9,marca10,marca11,marca
rca22,marca23,marca24,marca25,marca26,marca27,marca28,marca29,marca30,marca31
a41,marca42,marca43,marca44,marca45,marca46,marca47,marca48,marca49,marca50}
@attribute nome {produto50,
produto49,produto48,produto47,produto46,produto45,produto44,produto43,produto
uto34,produto33,produto32,produto31,produto30,produto29,produto28,produto27,p
9,produto18,produto17,produto16,produto15,produto14,produto13,produto12,produ
,produto2, produto1}
@attribute descricao {descricao50,
descricao49,descricao48,descricao47,descricao46,descricao45,descricao44,descr
7,descricao36,descricao35,descricao34,descricao33,descricao32,descricao31,des
o24,descricao23,descricao22,descricao21,descricao20,descricao19,descricao18,d
cao11,descricao10,descricao9,descricao8,descricao7,descricao6,descricao5,desc
@attribute tipo {tipo50,
tipo49,tipo48,tipo47,tipo46,tipo45,tipo44,tipo43,tipo42,tipo41,tipo40,tipo39,
tipo27,tipo26,tipo25,tipo24,tipo23,tipo22,tipo21,tipo20,tipo19,tipo18,tipo17,
5,tipo4,tipo3,tipo2,tipo1}
@attribute setor {setor1,
setor2, setor3, setor4, setor5, setor6, setor7, setor8, setor9, setor10, setor11, setor
tor22, setor23, setor24, setor25, setor26, setor27, setor28, setor29, setor30, setor31
r41, setor42, setor43, setor44, setor45, setor46, setor47, setor48, setor49, setor50}
@attribute validade numeric
@attribute barras numeric
@attribute promocao numeric
@attribute valor numeric

@data
39,marca50,produto1,descricao1,tipo1,setor50,1,1,40,10
39,marca49,produto2,descricao2,tipo2,setor49,2,2,30,100
```

Quadro 13 – Arquivo ARFF após processo de inclusão

No quadro 26 é possível conferir a estrutura dos atributos e o início dos dados após a marcação `@data`. Nesse mesmo quadro, é possível identificar o código do usuário 39 informado pela aplicação e presente no ARFF. Este foi o usuário escolhido para monitoramento e aplicação de testes. A figura 24 apresenta os dados desse usuário, a figura 25, por sua vez, o *cluster* no qual este usuário está inserido e a figura 26, as preferências do *cluster* deste usuário.

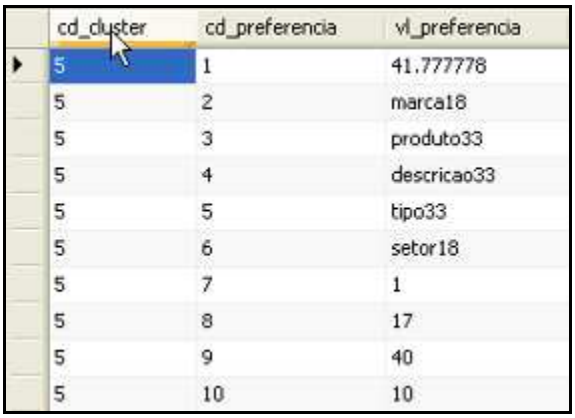
| cd_cliente | nm_cliente | ds_senha |
|---|---|---|
| 39 | rudi1 | rudi1 |
| 40 | rudi2 | rudi2 |
| 41 | rudi3 | rudi3 |
| 42 | rudi4 | rudi4 |
| 43 | rudi5 | rudi5 |
| 44 | rudi6 | rudi6 |
| 45 | rudi7 | rudi7 |
| 46 | rudi8 | rudi8 |
| 47 | rudi9 | rudi9 |
| 48 | rudi10 | rudi10 |

Figura 25 – Informações do usuário monitorado para o teste



| cd_usuario | cd_cluster |
|---|---|
| 39 | 5 |
| 40 | 1 |
| 41 | 1 |
| 42 | 4 |
| 43 | 5 |
| 44 | 1 |
| 45 | 3 |
| 46 | 4 |
| 47 | 2 |
| 48 | 3 |

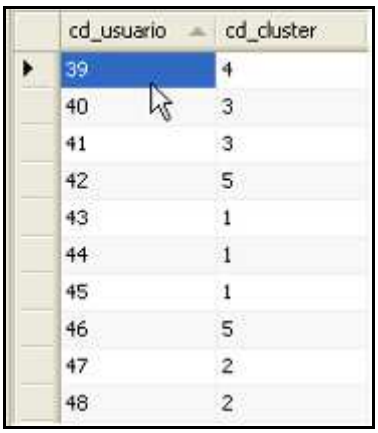
Figura 26 – A tabela segmentação mostra o cluster do usuário



| cd_cluster | cd_preferencia | vl_preferencia |
|---|---|---|
| 5 | 1 | 41.777778 |
| 5 | 2 | marca18 |
| 5 | 3 | produto33 |
| 5 | 4 | descricao33 |
| 5 | 5 | tipo33 |
| 5 | 6 | setor18 |
| 5 | 7 | 1 |
| 5 | 8 | 17 |
| 5 | 9 | 40 |
| 5 | 10 | 10 |

Figura 27 – A tabela `cluster_item` ilustrando as preferências do cluster do usuário

Após as informações das figuras 24, 25 e 26, realizou-se a adição de cinco produtos no usuário 39. Foram os produtos 46 a 50, produtos esses diferentes dos selecionados anteriormente para este usuário que eram os produtos de 1 a 5, conforme figura 23. Após nova consulta à segmentação, obteve-se o mesmo resultado. O usuário de código 39 permaneceu no *cluster* 5. Tentou-se então adicionar novamente os produtos de 46 a 50 ao usuário 39 simulando uma nova compra dos mesmos produtos tentando mostrar o interesse do usuário nestes produtos. Após processamento e consulta a tabela `Segmentacao`, obteve-se o resultado mostrado na figura 27.



| cd_usuario | cd_cluster |
|---|---|
| 39 | 4 |
| 40 | 3 |
| 41 | 3 |
| 42 | 5 |
| 43 | 1 |
| 44 | 1 |
| 45 | 1 |
| 46 | 5 |
| 47 | 2 |
| 48 | 2 |

Figura 28 – Segmentacao do usuário após inclusão repetida dos mesmos produtos

O resultado dos testes evidenciou a mudança do usuário 39 do cluster 5 para o 4 indicando que, a medida que suas escolhas mudaram, sua segmentação mudou, fazendo, assim, com que as preferências fornecidas sejam as do cluster 4, e não mais do cluster 5.

Para mostrar o uso que a aplicação pode fazer do framework foi implementado na classe Logon a rotina que chama o método getSegmentacao como pode ser visto no quadro 14.

```
 1 public Logon ler(String nome, String senha) throws Exception {
 2    LogonDAO dao = FactoryDAO.getLogonDAO();
 3    Logon logon = dao.ler(nome, senha);
 4    PreferenciaUC<app.modelo.Preferencia> preferenciax = new
   PreferenciaUC<app.modelo.Preferencia>();
 5    List<framework.modelo.Preferencia> preferenciaList =
   preferenciax.getSegmentacao(logon.getCodigoUsuario());
 6    String marca = "";
 7    String tipo = "";
 8    String setor = "";
 9    int garantia = 0;
10    int promocao = 0;
11    Double valor = 0.0;
12    for (framework.modelo.Preferencia preferenciaFramework :
   preferenciaList) {
13       if (preferenciaFramework.getNome().toString() == "marca") {
14          marca = preferenciaFramework.getValor();
15       }
16       if (preferenciaFramework.getNome().toString() == "tipo") {
17          tipo = preferenciaFramework.getValor();
18       }
19       if (preferenciaFramework.getNome().toString() == "setor") {
20          setor = preferenciaFramework.getValor();
21       }
22       if (preferenciaFramework.getNome().toString() == "garantia") {
23          garantia = Integer.parseInt(preferenciaFramework.getValor());
24       }
25       if (preferenciaFramework.getNome().toString() == "promocao") {
26          promocao = Integer.parseInt(preferenciaFramework.getValor());
27       }
28       if (preferenciaFramework.getNome().toString() == "valor") {
29          valor = Double.parseDouble(preferenciaFramework.getValor());
30       }
31    }
32    ProdutoDAO produtoDAO = FactoryDAO.getProdutoDAO();
33    List produtoLista = produtoDAO.listarPreferencia(marca, tipo,
   setor, garantia, promocao, valor);
34    return logon;
35 }
```

Quadro 14 - Aplicação usando o método `getSegmentacao`

O quadro 14 mostra o método `ler` da aplicação o qual é responsável por receber os dados do logon e permitir ou negar acesso. Neste método a aplicação efetua a rotina para logon até a linha 3. A linha 4 e 5 mostram de que forma o método `getSegmentacao` é chamado. A linha 3 cria um objeto `Preferencia` do *framework* e a linha 5 chama o método `getSegmentacao` que devolve uma lista contendo as preferências do cluster ao qual o

usuário informado pela aplicação pertence. Somente estes passos são necessários para permitir a aplicação fazer uso do framework, mas para demonstrar a utilidade foram implementadas também as rotinas que iniciam na linha 6 até 35. Nestas linhas primeiramente a lista fornecida pelo método `getSegmentacao` é iterada em busca das preferências. Ao final da iteração é criado um `DAO` para acesso aos produtos. Neste `DAO` é chamado o método `listarPreferencia` que busca no banco de dados da aplicação produtos que apresentem características idênticas as preferências do cluster ao qual o usuário em questão esta relacionado. Assim, é possível fornecer anúncios que tenham alto interesse de compra pelo usuário visto que esses produtos foram sugeridos em função das preferências do cluster do usuário sendo o cluster o resultado do conjunto de preferências entre usuários com escolhas similares.

Ainda relacionando o método com o exemplo anterior, se o usuário passado na linha 5 ao método `getSegmentacao` fosse o usuário 39 demonstrado na figura 27 teríamos como resultado uma lista contendo os valores do quadro 15.

| **Preferencia** | **Valor** |
|---|---|
| marca | marca1 |
| tipo | tipo50 |
| setor | setor1 |
| garantia | 2 |
| promoção | 30 |
| valor | 302,5 |

Quadro 15 – Preferencias do cluster do `usuário 39`

Ao selecionar no banco de dados da aplicação os produtos comprados pelo usuário 39 temos os valores da figura 28.

| nm_marca | tp_produto | nm_setor | dt_validade | qt_promocao | vl_produto |
|---|---|---|---|---|---|
| marca50 | tipo1 | setor50 | 1 | 40 | 10 |
| marca49 | tipo2 | setor49 | 2 | 30 | 100 |
| marca48 | tipo3 | setor48 | 3 | 20 | 1000 |
| marca47 | tipo4 | setor47 | 4 | 10 | 10000 |
| marca46 | tipo5 | setor46 | 1 | 40 | 10 |
| marca5 | tipo46 | setor5 | 2 | 30 | 100 |
| marca4 | tipo47 | setor4 | 3 | 20 | 1000 |
| marca3 | tipo48 | setor3 | 4 | 10 | 10000 |
| marca2 | tipo49 | setor2 | 1 | 40 | 10 |
| marca1 | tipo50 | setor1 | 2 | 30 | 100 |

Figura 29 – Produtos comprados pelo `usuário 39`

Relacionando os valores do quadro 15 com os valores da figura 28 é possível concluir que a maior parte dos produtos que apresentam características idênticas as preferências

informadas pelo framework já foram comprados pelo usuário. Conclui-se também que nenhum produto já comprado apresenta todas as preferências sendo uma boa oportunidade para sugerir um produto com as características do quadro 15 visto que o este quadro representa as preferências do conjunto de pessoas com escolhas similares ao usuário. Ainda seria possível sugerir um produto com uma ou mais das preferências do quadro 15 facilitando assim a busca de tal produto.

Além do exemplo citado o framework pode ser usado em outras situações visto que seu objetivo seja efetuar a segmentação de um conjunto de dados em função das suas preferências. Alguns exemplos seriam segmentar os alunos pelas notas, freqüências e outros possíveis valores. Segmentar funcionários pelo salário, tempo serviço e demais características. Segmentar a contabilidade de empresas pelo seu fluxo de caixa além de outros exemplos a escolher.

## 3.5 RESULTADOS E DISCUSSÃO

Dentre as ferramentas correlatas analisadas na fundamentação a que apresenta maior aproximação com o framework BeTa é o modelo de Li et al. (2009) tendo esta um objetivo similar ao exemplo do estudo de caso. O modelo de Li et al. (2009) leva em consideração os anúncios como centralizadores da informação. Desta forma, se determinado usuário não clicar em um anúncio candidato esta informação não será computada embora tenha relevância ao contexto.

O framework BeTa leva em consideração as pessoas e suas escolhas visto que tais atributos representam uma parcela maior de informação gerando assim um resultado mais realista.

Por isso com os resultados demonstrados no estudo de caso, além das comparações efetuadas, é possível avaliar o desempenho final do *framework* como regular, tendo sido, os objetivos iniciais, alcançados, mas com queda de performance gradativa em relação à escalabilidade.

# 4 CONCLUSÕES

Através dos conceitos estudados e ferramentas analisadas desenvolveu-se um *framework* para analisar as preferências por meio das interações fornecidas pela aplicação web utilizando para isso a técnica *Behavior Targeting*. Realizaram-se pesquisas na área de *Web Analytics*, *Data Mining* e construção de *framework*. Os trabalhos correlatos tiveram fundamental importância e surgiram como bons exemplos de algumas outras abordagens em relação ao tema. Dessa forma as informações encontradas nessa etapa de pesquisa contribuíram altamente para a busca do objetivo inicial.

Algumas tecnologias e padrões foram utilizadas na tentativa de melhorar a forma como o *framework* foi implementado, a começar pelo uso da linguagem Java mantendo assim alta portabilidade e eficiência devido ao forte apelo da linguagem atualmente. Outra tecnologia de fundamental importância que pode ser citada é o uso da biblioteca do *Weka*, a qual contém o algoritmo *kmeans* utilizado como recurso para o desenvolvimento do *framework*. Utilizou-se também *Reflection* e *Annotation*, tecnologias essas intimamente relacionadas com a linguagem elencada. Outros padrões e boas práticas podem ser citados pela contribuição que exerceram na escrita do código fonte, tais como DAO, *Factory* e *Singleton* os quais auxiliaram a organizar o código além de aumentar o desempenho final, visto o incentivo ao uso de padrões.

Como resultado o *framework* executa as segmentações de acordo com o uso do sistema através do método `segmentar`, utiliza como métricas para análise dos perfis as preferências do usuário e fornece os grupos segmentados ao qual o usuário da aplicação pertence através do método `getSegmentacao` conforme os objetivos específicos. Como limitações a baixa escalabilidade e a escolha do algoritmo *k-means*.

## 4.1 EXTENSÕES

A escalabilidade do framework mostrou-se relativamente baixa devido ao uso do ARFF. A troca do arquivo pela integração com banco de dados é uma possível solução para esta limitação.

Em relação ao algoritmo *k-means*, este apresenta um resultado diferente a cada nova

execução. Devido a este fato, a cada nova informação segmentada pelo *framework* recorre-se a todos os dados, os quais não podem ser excluídos do ARFF, pois são utilizados na segmentação. Para resolver faz-se necessário a busca de outro algoritmo ainda não pesquisado no contexto deste trabalho.

# REFERÊNCIAS BIBLIOGRÁFICAS

CHEN, Ye; PAVLOV, Dmitry; CANNY, John F. Large-Scale Behavioral Targeting. In: INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, 15., 2009, San Jose. **Proceedings…** PARIS: ACM, 2009. p. 209-217. Disponível em: <http://portal.acm.org/citation.cfm?id=1557019.1557048&coll=portal&dl=GUIDE&CFID=8056571&CFTOKEN=65725324>. Acesso em: 11 set. 2009.

GLOBAL Professor. **Distância Euclidiana.** [S.l.], [2010]. Disponível em: <http://professorglobal.cbpf.br/mediawiki/index.php/Dist%C3%A2ncia_Euclidiana>. Acesso em: 25 maio 2010.

GOMES, Augusto; OLIVEIRA, Kathia; ROCHA, Ana R. Avaliação de processos de software baseada em medições. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 15., 2001, Rio de Janeiro. **Anais…** RIO DE JANEIRO: COPPE, 2001. p. 84-99. Disponível em: <http://www.lbd.dcc.ufmg.br:8080/colecoes/sbes/2001/006.pdf>. Acesso em: 21 fev. 2010.

GONÇALVES, Eduardo. C. Data Mining de regras de associação. **SQL Magazine,** Porto Alegre, 2008. Disponível em: <http://www.devmedia.com.br/articles/viewcomp.asp?comp=6533>. Acesso em: 05 set. 2009.

PIMENTEL Edson P.; FRANÇA Vilma F. de.; OMAR Nizam. A identificação de grupos de aprendizes no ensino presencial utilizando técnicas de clusterização. In: SIMPÓSIO BRASILERIO DE INFORMÁTICA NA EDUCAÇÃO, 3., 2003, São José dos Campos. **Proceedings…** Rio de Janeiro: SBIE, 2003 Disponível em: <http://www.br-ie.org/pub/index.php/sbie/article/view/280/0>. Acesso em: 05 abr. 2009.

KAUSHIK, Avinash. **Web analytics**: uma hora por dia. 2. ed. Rio de Janeiro: Alta Books, 2009.

LANGER, Sergio. **As métricas da web.** São Paulo, [2008]. Disponível em: <http://www.slideshare.net/gestaohipermidia/as-mtricas-da-web-1136298>. Acesso em: 29 fev. 2010.

LI, Ting et al. A markov chain model for integrating behavioral targeting into contextual advertising. In: INTERNATIONAL WORKSHOP ON DATA MINING AND AUDIENCE INTELLIGENCE FOR ADVERTISING, 3., 2009. Beijing. **Proceedings…** Paris: ACM, 2009. p. 1-9. Disponível em: <http://portal.acm.org/citation.cfm?id=1592748.1592750&coll=portal&dl=GUIDE&CFID=8056571&CFTOKEN=65725324>. Acesso em: 11 set. 2009.

MARKIEWICZ, Marcus E.; LUCENA, Carlos J. P. de. Object oriented framework development. **ACM Crossroads,** New York, v. 7, n. 4, p. 3-9, summer 2001. Disponível em: <http://portal.acm.org/citation.cfm?id=372765.372771>. Acesso em: 20 set. 2009.

THE MATHWORKS. **Kmeans.** [S.l.], [2010]. Disponível em: <http://www.mathworks.com/access/helpdesk/help/toolbox/stats/kmeans.gif>. Acesso em: 15 mar. 2010.

MELO, Camilo T. de. Web beacon. **Imasters,** São Paulo, jul. 2009. Disponível em: <http://imasters.uol.com.br/artigo/13389/desenvolvimento/web_beacon/>. Acesso em: 13 maio 2010.

NUNES, Jorge L. V. B. WEB 3.0, o que há de novo? **Linha de Código,** Porto Alegre, abr. 2008. Disponível em: <http://www.linhadecodigo.com.br/Artigo.aspx?id=1775>. Acesso em: 25 ago. 2009.

PICHILIANI, M. Data Mining na prática: algoritmo k-means. **SQL Magazine**, Porto Alegre, 2008. Disponível em: <http://www.devmedia.com.br/articles/viewcomp.asp?comp=4584&hl=*k-means*>. Acesso em: 05 set. 2009.

PRESSMAN, Roger S. **Engenharia de software**. 6. ed. São Paulo: McGraw-Hill, 2006.

RIBEIRO, G. **Web analytics**: analisando os números e gerando resultados. 2. ed. São Paulo: Creative Commons, 2009.

ROCHA, André D. Construindo frameworks em Java. **Java Magazine**, Rio de Janeiro, v. 1, n. 66, p. 76-82, Fev. 2009.

ROROHIKO, Tari. **Attribute-Relation File Format**: ARFF. The University of Waikato, New Zealand, [2008]. Disponível em: <http://www.cs.waikato.ac.nz/~ml/weka/arff.html>. Acesso em: 10 fev. 2010.

SANTOS, Rafael. **Weka na munheca**. São José dos Campos, 2005. Disponível em: <http://www.lac.inpe.br/~rafael.santos/Docs/CAP359/2005/weka.pdf>. Acesso em: 10 fev. 2010.

SANTOS, Simone M.; NORONHA, Claudio P. Padrões espaciais de mortalidade e diferenciais sócio-econômicos na cidade do Rio de Janeiro. **Cadernos de Saúde Pública**, Rio de Janeiro, v. 17, n. 5, p. 1099-1110, set. 2001. Disponível em: <http://www.scielosp.org/pdf/csp/v17n5/6319.pdf>. Acesso em: 25 maio 2010.

SEGARAN, Toby. **Programando a inteligência coletiva**: desenvolvendo aplicativos Web 2.0 inteligentes. Rio de Janeiro: Alta Books, 2008.

WITTEN, Ian H.; FRANK, Eibe. **Data Mining**. United States: Morgan Kaufmann Publishers, 1999.

WU, Xiaohui et al. Probabilistic latent semantic user segmentation for behavioral targeted advertising. In: INTERNATIONAL WORKSHOP ON DATA MINING AND AUDIENCE INTELLIGENCE FOR ADVERTISING, 3., 2009, Beijing. **Proceedings…** Paris: ACM, 2009. p. 10-17. Disponível em: <http://portal.acm.org/citation.cfm?id=1592748.1592751&coll=portal&dl=GUIDE&CFID=8056571&CFTOKEN=65725324>. Acesso em: 11 set. 2009.