UNIVERSIDADE FEDERAL DE ALAGOAS - UFAL
INSTITUTO DE COMPUTAÇÃO
CIÊNCIA DA COMPUTAÇÃO

AVALIAÇÃO DE DESEMPENHO DO *FRAMEWORK* HIBERNATE EM UM SISTEMA CENTRADO EM DADOS

Trabalho de Conclusão de Curso

Maceió, 2015

Universidade Federal de Alagoas
Instituto de Computação

AVALIAÇÃO DE DESEMPENHO DO *FRAMEWORK* HIBERNATE EM UM SISTEMA CENTRADO EM DADOS

Artur Maia Pereira

Monografia Apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação do Instituto de Computação da Universidade Federal de Alagoas.

Orientador:
PATRICK HENRIQUE DA SILVA BRITO

Maceió, 2015

Agradecimentos

Primeiramente, queria agradecer aos meus pais, por estarem sempre ao meu lado, ajudando na minha educação e garantindo tudo que eu sempre precisei para seguir com a minha vida.

Ao professor Patrick Brito que me deu a oportunidade de trabalhar com ele no projeto Falibras ao longo de dois anos e aceitou orientar o meu TCC, sempre com muita paciência para corrigir todos os problemas que apareceram ao longo deste caminho.

Agradeço aos meus familiares, principalmente tios e tias, que sempre me apoiaram durante a minha graduação. A Jesus, que sempre me aceitou em sua casa, e aos amigos, tanto os que me acompanham desde o colégio, quanto os que conheci durante a graduação, o grupo dos Papitos.

Por fim, queria agradecer a todos os professores e funcionários da secretaria do instituto de computação que contribuíram com a minha graduação, em especial ao professor Fábio Cunha e ao funcionário Marcelo de Gusmão.

Resumo 

O trabalho apresentado nesta monografia visa evoluir a forma de comunicação entre a aplicação Falibras e seu Banco de Dados. Desenvolvido na Universidade Federal de Alagoas, o Falibras tem como objetivo principal realizar a tradução automática de uma ou mais frases do português escrito para gestos de LIBRAS. Todo o processo de tradução é centrado em dados, transformando a via de comunicação entre os módulos do Falibras e seu banco de dados em um ponto crítico do sistema, justificando a escolha desse sistema como alvo de avaliação do trabalho proposto. Antes da execução deste trabalho, o sistema Falibras se comunicava com seu banco de dados MySQL utilizando um driver JDBC correspondente. Foi realizada uma refatoração na camada de persistência do sistema Falibras de modo que o *framework* Hibernate fosse incorporado ao sistema. Depois, foram realizados testes, baseados no método *Goal-Question-Metric* (GQM), para verificar o impacto do Hibernate no desempenho do Falibras em cenários reais. Por fim, foi constatado que a adoção do Hibernate proporcionou melhorias significativas no desempenho do sistema.

Palavras-chave: Arquitetura centrada em dados, Falibras, LIBRAS, MySQL, Hibernate, GQM.

Abstract

The following work aims to evolve the form of communication between the Falibras system and your database. Developed at the Universidade Federal de Alagoas, the Falibras has as main objective execute the automatic translation of one or more sentences from Portuguese to LIBRAS gestures. The entire process of translation is data-centric, making the communication path between the Falibras modules and your database in a critical point of the system, justifying the choice of this system as an evaluation target of the proposed work. Before the execution of this work, the Falibras system communicated with its MySQL database using the corresponding JDBC driver. Was performed a refactoring in the persistence layer of the Falibras system so that the framework Hibernate was built into the system. Afterwards, tests were conducted, based on the Goal-Question-Metric (GQM) technique, to check the impact of Hibernate on Falibras performance in real scenarios. Finally, it was noted that the adoption of Hibernate provided significant improvements for the system performance

Keywords: Data-centric Architecture, Falibras, LIBRAS, MySQL, Hibernate, GQM.

Lista de Figuras

Sumário

1. Introdução

Atualmente, com o avanço das tecnologias, os softwares estão ocupando cada vez mais espaço na sociedade, resolvendo problemas de alta complexidade pertencentes a diferentes contextos, sendo praticamente impossível viver sem interagir com qualquer tipo de sistema de informação.

Devido a essa crescente complexidade, surgiu a necessidade de estruturar os softwares de acordo com padrões predefinidos, dando origem ao conceito de Arquitetura de Software. Segundo Sommerville (2007), Arquitetura de Software é um projeto em que se tenta estabelecer uma organização dos componentes do sistema de modo que satisfaça seus requisitos funcionais e não funcionais, e que irá servir como guia para o desenvolvimento do sistema.

O software estudado neste trabalho, o sistema Falibras, se baseia numa arquitetura centralizada em dados, onde um repositório de dados fica no centro da arquitetura e é compartilhado por outros componentes que atualizam, adicionam e removem os dados contidos no repositório (PRESSMAN, 2011). Softwares que seguem esse estilo arquitetural são caracterizados por apresentarem um grande número de consultas à base de dados, podendo levar à criação de um gargalo no canal de comunicação entre a aplicação e o banco de dados, limitando o desempenho e a escalabilidade do sistema como um todo.

1.1. Motivação e Objetivos

O sistema Falibras, que se trata de um software com uma arquitetura centrada em dados, foi desenvolvido com o propósito de realizar traduções automáticas do português para a Língua Brasileira de Sinais (LIBRAS), onde seus módulos interagem através de modificações e consultas ao banco de dados. Com isso, o modo como é feita a comunicação entre a aplicação e o banco de dados tem uma importância crucial no desempenho do sistema como um todo. Além disso, por se tratar de um recurso centralizado, o banco de dados também pode prejudicar a escalabilidade do software, isto é, pode se tornar um recurso que limita o número de requisições simultâneas tratadas pelo sistema.

Na sua configuração atual, o sistema Falibras utiliza a biblioteca JDBC (*Java Database Connectivity*) para realizar a comunicação entre a aplicação Java e o Sistema de Gerenciamento de Banco de Dados (SGBD) - MySQL.

Este trabalho objetiva alterar a forma de comunicação entre a aplicação e o banco de dados do Falibras através da utilização do *framework* Hibernate e analisar o seu impacto no desempenho do sistema. Para isso, foi necessário realizar uma refatoração na camada de persistência do Falibras e, após concluir a refatoração, foram realizados testes quantitativos, baseados em cenários reais, para se quantificar o impacto do uso do Hibernate no desempenho do Falibras.

Com base no resultado dos testes será possível decidir qual dessas formas é mais adequada para a comunicação entre o sistema Falibras e o seu SGBD. Além disso, esta pesquisa poderá servir como suporte para outros programadores decidirem sobre como será implementada a comunicação entre aplicação e banco de dados, especialmente em softwares que adotam uma arquitetura de software centrada em dados.

1.2. Estrutura

O trabalho apresentado a seguir está estruturado da seguinte maneira:

- O Capítulo 2 traz uma fundamentação teórica a respeito dos temas que embasam esta pesquisa. Sendo assim, serão apresentados conceitos sobre o SGBD MySQL, a API JDBC, o *framework* Hibernate e a linguagem LIBRAS.
- O Capítulo 3 traz detalhes da refatoração do sistema Falibras, para adequá-lo ao *framework* Hibernate. Apresenta também a abordagem *Goal-Question-Metric* (GQM) usada para guiar os testes, além dos cenários e das características do ambiente onde os testes serão realizados.
- No Capítulo 4 serão apresentados os resultados obtidos através dos testes e uma breve discussão sobre eles.
- O Capítulo 5 trata de apresentar trabalhos relacionados e suas diferenças com as discussões apresentado nesta monografia.
- Para finalizar, o Capítulo 6 apresenta a conclusão e as possibilidades de trabalhos futuros.

2. Fundamentação Teórica

2.1. MySQL

O software MySQL é o segundo Sistema de Gerenciamento de Banco de Dados (SGBD) mais utilizado do mundo[1], ficando atrás apenas do Oracle. Utiliza a linguagem de propósito específico SQL (*Structured Query Language*), e foi desenvolvida com o objetivo de construir, atualizar e realizar consultas em banco de dados relacionais.

O MySQL foi desenvolvido pela empresa sueca *MySQL AB* em 1995 que, posteriormente, foi adquirida pela corporação *Sun Microsystems*. Atualmente, seus direitos autorais e marcas registradas pertencem a *Oracle Corporation*. Em relação a sua licença, o MySQL pode ser usado como um produto de código aberto, licenciado sob a GPL[2] (GNU *General Public License*). Caso o usuário deseje utilizar o MySQL em alguma aplicação comercial, ele precisará comprar a versão comercial da ferramenta licenciada pela Oracle[3] (MySQL, 2005).

Uma ferramenta classificada como SGBD tem como objetivo gerenciar o acesso e realizar a manutenção de um banco de dados, facilitando a manipulação das informações contidas nele. Além disso, ele deve garantir a persistência dos dados, de maneira segura, por um longo período de tempo (GARCIA-MOLINA; ULLMAN; WIDOM; 2008).

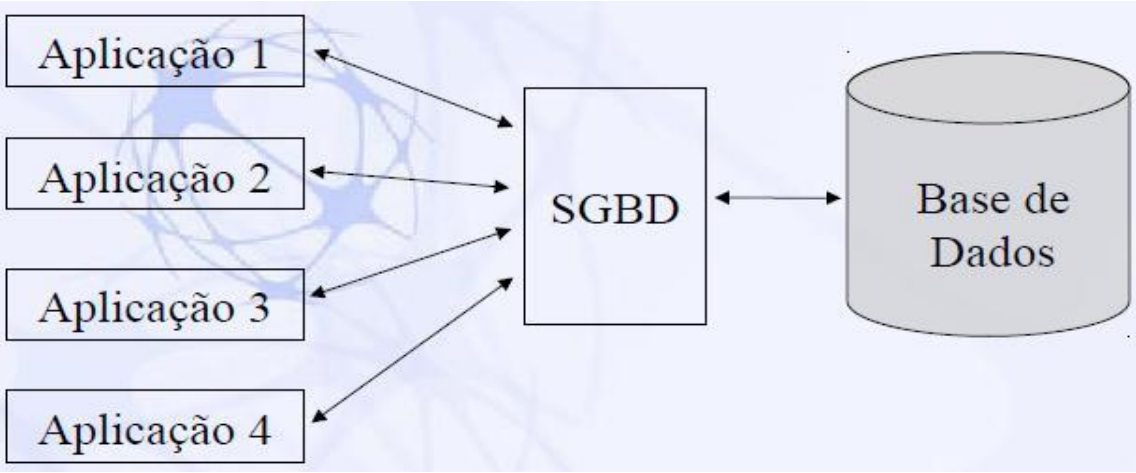

Figura 1: SGBD como interface entre as aplicações e o banco de dados

Fonte: Disponível em: <https://addst.wordpress.com/2011/10/28/sgbd/ >. Acesso em: 07 maio, 2015

[1]DB-Engines Ranking, Disponível em: < http://db-engines.com/en/ranking >. Acesso em: 06 maio, 2015.
[2] Licença GPL < http://www.fsf.org/licenses/ >, Acesso em: 06 maio, 2015.
[3] Licença Comercial MySQL < http://www.mysql.com/about/legal/ >, Acesso em: 06 maio, 2015.

De acordo com a Figura 1 podemos ver que o SGBD é um componente localizado entre a aplicação cliente e seu banco de dados que implementa as funções de acesso e manipulação dos dados, e oferece uma interface compatível com diferentes tipos de aplicações. Com isso, o SGBD facilita o desenvolvimento da aplicação, pois ela não será mais responsável pela implementação das funções de gerenciamento do banco de dados.

O MySQL trabalha com banco de dados relacionais. Estes bancos organizam os dados em tabelas que se relacionam através de associações, de cardinalidade variável, e são descritas por um conjunto de atributos. Dentro de uma tabela, cada coluna representa um atributo e cada linha armazena os dados de uma instância de sua tabela correspondente, por exemplo, no banco de dados do sistema Falibras temos a tabela Significado que apresenta atributos (colunas) como id e nome e um relacionamento, de cardinalidade um-para-muitos, com a tabela Animação, ou seja, para cada significado existe apenas uma animação associada, mas uma animação pode estar associada a vários significados.

Outra importante característica do MySQL é o suporte a transações. Segundo Elmasri e Navathe (2010), transação é uma unidade atômica de trabalho que realiza uma sequência de operações de leitura ou escrita no banco de dados e apresenta as seguintes propriedades:

- **Atomicidade**: As operações de uma transação devem ser executadas por completo. Caso ocorra uma falha durante a transação, os efeitos parciais dela devem ser desfeitos.
- **Consistência**: A conclusão de uma transação deve levar o banco de dados de um estado consistente para outro, respeitando as regras de integridade dos dados.
- **Isolamento**: A execução de uma transação não deve sofrer interferência de outras transações que estão sendo executadas em paralelo.
- **Durabilidade**: Os efeitos de uma transação bem sucedida devem persistir no banco definitivamente.

Além do que já foi citado, vale a pena deixar registrado as seguintes características do MySQL:

- É uma ferramenta portável entre diferentes plataformas, escrito em C e C++. Além disso, apresenta interfaces compatíveis com várias linguagens de programação como Java, Python e PHP.

- Oferece suporte a multithreads, utilizadas diretamente no kernel da plataforma, aprimorando o desempenho da ferramenta.
- Disponibiliza diferentes tipos de tabelas para armazenamento dos dados. Cada tipo apresenta características específicas como tempo de resposta, volume de dados suportado, entre outras características.
- Apresenta alta escalabilidade, dando suporte a grande bases de dados que podem conter até 50 milhões de registros (instâncias)[4].

2.2. *Java Database Connectivity (JDBC)*

O *Java Database Connectivity* (JDBC) é um conjunto de classes e interfaces, uma API (sigla do inglês, *Application Programming Interface*), escrita na linguagem Java que estabelece uma conexão entre aplicação e banco de dados, permitindo que a aplicação execute instruções SQL na sua respectiva base de dados (PALMEIRA, 2013). De maneira geral, o JDBC funciona de acordo com a Figura 2, apresentada abaixo.

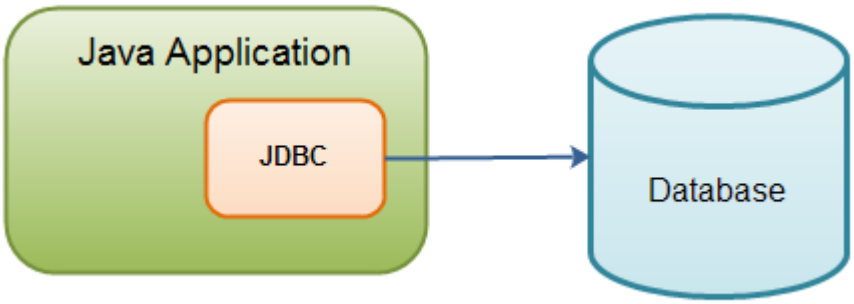

Figura 2: Abstração JDBC

Fonte: Disponível em: <http://tutorials.jenkov.com/jdbc/index.html>. Acesso em: 13 maio, 2015

Cada SGBD possui sua própria implementação da API JDBC, que são conhecidos como Driver JDBC. Geralmente, a empresa responsável pelo desenvolvimento do SGBD também desenvolve o seu driver de conexão. De acordo com a Figura 3, podemos ver o que acontece com o sistema Falibras que utiliza o driver do MySQL para realizar a comunicação entre aplicação e banco de dados.

[4] Disponível em: <http://dev.mysql.com/doc/refman/5.6/en/features.html>. Acesso em: 11 Maio, 2015.

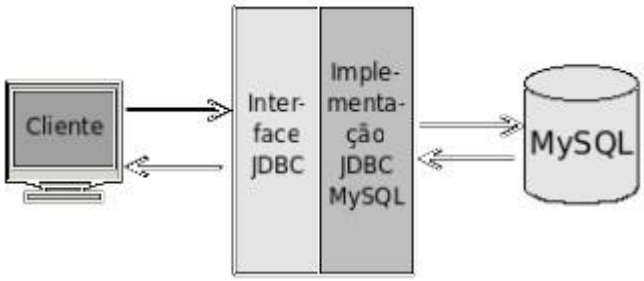

Figura 3: Abstração da conexão Aplicação-MySQL

Fonte: Disponível em: <http://www.caelum.com.br/apostila-java-web/bancos-de-dados-e-jdbc/#2-3-a-conexao-em-java> . Acesso em: 13 maio, 2015.

Dentro da API existem algumas classes fundamentais que merecem ser citadas. As classes **DriverManager** e **Connection** trabalham em conjunto para gerenciar a conexão com o banco de dados. A primeira classe é responsável pela inicialização e controle do driver, enquanto que a segunda se encarrega de estabelecer a conexão e encerrá-la quando não houver mais troca de informações com o banco de dados. Para executar instruções SQL no banco de dados usa-se objetos da classe **Statement** ou **PreparedStatement**, onde a segunda opção apresenta um melhor desempenho. Por fim, temos a classe **ResultSet** que é responsável por armazenar o conjunto de dados retornados de uma consulta ao banco de dados.

2.3. Hibernate

Hibernate é um *framework* de persistência, escrito em Java, que realiza o mapeamento entre tabelas de um banco de dados relacionais e classes (objetos) da linguagem de programação Java, conhecido como mapeamento objeto-relacional (ORM). O objetivo deste *framework* é tornar transparente para o desenvolvedor o acesso ao banco de dados, facilitando e otimizando o armazenamento e a recuperação dos dados por parte da aplicação Java, abstraindo as instruções SQL.

O Hibernate é um software livre distribuído com a licença LGPL (GNU *Lesser General Public License*). Foi desenvolvido em 2001 por uma equipe de programadores liderada por Gavin King e, atualmente, tem seus direitos pertencentes à empresa americana Red Hat.

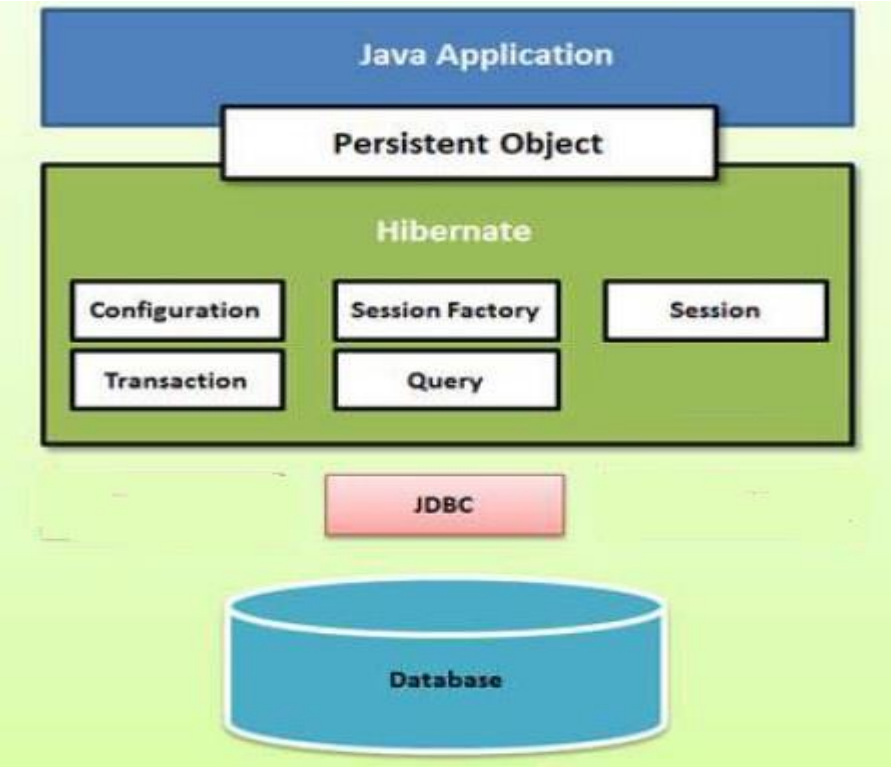

Figura 4: Arquitetura de uma aplicação com Hibernate

Fonte: Disponível em: <http://www.tutorialspoint.com/hibernate/hibernate_architecture.htm>. Acesso em: 16 maio, 2015

A Figura 4 apresenta a arquitetura de uma aplicação que utiliza o Hibernate. Como podemos ver o Hibernate não abre mão do JDBC para se conectar ao banco de dados, mas ele abstrai isso do desenvolvedor. Ele tem o intuito de libertar o programador de cerca de 95% das tarefas de programação relacionadas a persistência de dados, provendo instruções mais simples e um poderoso mecanismo de consulta, reduzindo significativamente o tempo de desenvolvimento que seria gasto com a manipulação manual dos dados através de JDBC e SQL (REDHAT, 2011).

Além da arquitetura, a Figura 4 também apresenta as principais classes do Hibernate. O objeto **Configuration** deve ser criado na inicialização da aplicação e deve receber como parâmetro todas as informações necessárias para que o Hibernate se conecte ao banco de dados e saiba quais classes do projeto deverão ser mapeadas. As informações de configuração são, normalmente, passadas através de um arquivo XML, nomeado como hibernate.cfg.xml.

O objeto **SessionFactory** é criado de acordo com as especificações do objeto Configuration e é utilizado para instanciar objetos da classe Session. Ele é um objeto pesado e deve apresentar uma única instância para cada banco de dados que interage com a aplicação, em outras palavras, este objeto irá gerenciar as sessões de comunicação entre aplicação e banco de dados. Um objeto da classe **Session** é similar ao objeto Connection do JDBC, ele que irá persistir e recuperar objetos no banco de dados através de uma sessão de comunicação.

Objetos da classe **Transaction** são utilizados para realizar transações no banco de dados e seu uso é opcional, cabendo ao desenvolvedor da aplicação decidir se seu uso é necessário. Por fim, temos a classe **Query** que é utilizada para executar as consultas no banco de dados.

Para fazer o mapeamento entre tabelas relacionais e objetos o Hibernate oferece duas possibilidades: Através da criação de arquivos XML ou através de anotações no código fonte dos objetos, que foi a opção escolhida para desenvolver este trabalho. Anotações podem ser definidas como metadados que aparecem no código fonte dos objetos mapeados e são ignorados pelo interpretador (FERNANDES; LIMA; 2007). As principais anotações do Hibernate são:

- **@Entity**: Esta anotação é usada para classificar uma classe como uma entidade do banco de dados.
- **@Table**: Permite especificar os detalhes de uma tabela que irá referenciar uma determinada entidade anotada, conhecida como classe persistente. Se essa anotação não for usada, o Hibernate utilizará o nome da classe para nomear a tabela.
- **@Id**: Usada para identificar a chave primária da classe persistente. Pode ser utilizada em conjunto com a anotação **@GeneratedValue** para que o valor da chave primária seja gerada automaticamente de acordo com uma regra predefinida.
- **@Column**: Anotação utilizada para mapear as colunas de uma tabela.

O uso de anotações diretamente na classe, ao invés do uso de arquivos XML, apresenta algumas vantagens como, por exemplo, um código mais intuitivo, visto que as anotações serão implementadas diretamente no código fonte das classes que elas estão associadas. Outra vantagem é que as anotações apresentam menos detalhes que arquivos XML, facilitando a compreensão do código, como podemos ver na Figura 5.

Exemplo 1: Mapeamento via Anotação na Classe

```
import javax.persistence.* ;
@Entity
public class Sample {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    public Integer id;
    public String name;
}
```

Exemplo 2: Mapeamento via Arquivo XML

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE
   hibernate-mapping
   PUBLIC
   "-//Hibernate/Hibernate Mapping DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd
">
<hibernate-mapping default-access="field">
   <class name="Sample">
      <id type="int" column="id">
         <generator class="native"/>
      </id>
      <property name="name" type="string"/>
   </class>
</hibernate-mapping>
```

Figura 5: Comparação das formas de mapeamento objeto-relacional.

Fonte: Disponível em: <http://howtodoinjava.com/2014/09/26/pros-and-cons-of-hibernate-annotations-vs-mappings/> Acesso em: 19 maio, 2015

Para realização das consultas, o Hibernate disponibiliza uma poderosa linguagem de consulta, conhecida como *Hibernate Query Language* (HQL). Semelhante ao SQL, mas ao invés de executar as operações nas tabelas e colunas, o HQL trabalha com objetos persistentes e seus atributos (REDHAT, 2011). Com isso, o desenvolvedor pode focar o seu trabalho apenas em objetos, deixando para o Hibernate a responsabilidade de traduzir as consultas para o SQL da maneira mais otimizada possível.

2.4. LIBRAS

A LIBRAS, ou Língua Brasileira de Sinais, é a língua utilizada pela maioria dos deficientes auditivos no Brasil e é derivada do alfabeto manual francês. Ao contrário das línguas faladas, sua percepção e produção são baseadas no modelo visto-gestual, ou seja, utilizam-se movimentos gestuais e expressões faciais percebidos pela visão. Mas a LIBRAS não se trata apenas de mímicas, ela é composta por diferentes níveis linguísticos como sintaxe, morfologia e semântica. Apresenta uma estrutura gramatical diferente do português, com suas próprias regras sintáticas e variações regionais, sendo muito comum encontrar deficientes auditivos que se comunicam bem em LIBRAS, mas não conseguem ler textos em português.

Com isso, em 2002, o governo brasileiro sancionou a Lei Nº 10.436/2002 que reconhece a Língua Brasileira de Sinais como meio legal de comunicação e expressão, que constituem um sistema linguístico completo de transmissão de ideias e fatos (BRASIL, 2002). Tais ideias são representadas por meio de sinais compostos por cinco parâmetros: A configuração das mãos, o ponto de articulação (local do corpo onde o sinal deve ser realizado), movimentação, orientação (direção) e, por fim, expressão facial e/ou corporal (KOJIMA; SEGALA; 2008).

3. Desenvolvimento e Avaliação do Trabalho

Este trabalho pode ser dividido basicamente em duas fases. Na primeira fase foi realizado um refatoramento no código do Falibras para que o *framework* Hibernate pudesse ser incorporado ao sistema. Já a segunda fase consistiu de testes de desempenho, baseados na abordagem *Goal-Question-Metric* (GQM) que será explicada mais adiante, para constatar os efeitos do Hibernate no Falibras.

3.1. Configuração do ambiente para o Hibernate

A incorporação do Hibernate ao Falibras pode ser descrita em três passos. Primeiro, foi necessário configurar e manter a comunicação entre aplicação e SGBD via Hibernate. Como já foi citado no capítulo anterior, os objetos do Hibernate responsáveis por está conexão precisam receber como parâmetro um arquivo com propriedades referentes ao banco de dados e as classes do sistema que serão mapeadas para as tabelas relacionais, que são informações cruciais para sua inicialização. No nosso caso, usaremos um arquivo XML, nomeado de **hibernate.cfg.xml**, que é apresentado na Figura 6.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
3  "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
4  <hibernate-configuration>
5    <session-factory>
6      <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
7
8      <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
9      <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/falibras</property>
10     <property name="hibernate.connection.username">root</property>
11     <property name="hibernate.connection.password">root</property>
12     <property name="hibernate.hbm2ddl.auto">update</property>
13
14     <mapping class="br.ufal.ic.falibras.common.metamodel.Animacao"/>
15     <mapping class="br.ufal.ic.falibras.common.metamodel.CategoriaGramatical"/>
16     <mapping class="br.ufal.ic.falibras.common.metamodel.EsperaFrase"/>
17     <mapping class="br.ufal.ic.falibras.common.metamodel.Frase"/>
18     <mapping class="br.ufal.ic.falibras.common.metamodel.Genero"/>
19     <mapping class="br.ufal.ic.falibras.common.metamodel.Numero"/>
20     <mapping class="br.ufal.ic.falibras.common.metamodel.Palavra"/>
21     <mapping class="br.ufal.ic.falibras.common.metamodel.Pessoa"/>
22     <mapping class="br.ufal.ic.falibras.common.metamodel.Significado"/>
23     <mapping class="br.ufal.ic.falibras.common.metamodel.SignificadosCategoria"/>
24     <mapping class="br.ufal.ic.falibras.common.metamodel.Tempo"/>
25     <mapping class="br.ufal.ic.falibras.common.metamodel.reader.Silaba"/>
26     <mapping class="br.ufal.ic.falibras.common.metamodel.sintatic.SintaticRule"/>
27     <mapping class="br.ufal.ic.falibras.common.metamodel.sintatic.ItemRegraMS"/>
28     <mapping class="br.ufal.ic.falibras.common.metamodel.translator.ItemRegraMT"/>
29     <mapping class="br.ufal.ic.falibras.common.metamodel.translator.TranslationRule"/>
30
31     </session-factory>
32 </hibernate-configuration>
```

Figura 6: Arquivo hibernate.cfg.xml utilizado para inicialização do Hibernate

Fonte: Autoria Própria.

Como podemos ver o **hibernate.cfg.xml** contém informações sobre o driver JDBC (com.mysql.jdbc.Driver) que será utilizado, os parâmetros necessários para se conectar ao banco de dados (IP, porta, nome do banco, login e senha), além do caminho de todas as classes do sistema que irão fazer parte do mapeamento objeto-relacional. A propriedade opcional hibernate.hbm2ddl.auto permite que o Hibernate crie as tabelas do banco caso elas não existam e faça a correção das tabelas de acordo com o parâmetros definidos nas anotações. O código apresentado na Figura 7 mostra a classe **HibernateUtility** que vai carregar o arquivo XML de configuração e implementa as instruções necessárias para a inicialização do Hibernate.

```
1  package br.ufal.ic.falibras.common.persistence.hibernate;
2
3  import org.hibernate.Session;
6
7  public class HibernateUtility {
8
9      private static SessionFactory factory;
10     static {
11         //Bloco estatico que inicializa o Hibernate
12         try {
13             factory = new AnnotationConfiguration().configure("hibernate.cfg.xml").buildSessionFactory();
14             System.out.println( ">> HIBERNATE INICIADO COM SUCESSO." );
15
16         }catch (Exception e){
17             System.out.println( ">> FALHA NA INICIAÇÃO DO HIBERNATE." +e.toString());
18             e.printStackTrace();
19             e.getMessage();
20             factory = null;
21         }
22     }
23
24     public static Session getSession() {
25         //Retorna uma sessão aberta
26         Session session = factory.openSession();
27         System.out.println("session is connected " + session.isConnected() ) ;
28         return session;
29     }
30 }
```

Figura 7: Classe HibernateUtility

Fonte: Autoria Própria

Após concluir a inicialização do hibernate e estabelecer corretamente a conexão entre o sistema Falibras e seu banco de dados, nosso próximo passo foi a realização do mapeamento objeto-relacional através de anotações nas classes que representam as tabelas. A Figura 8 traz dois exemplos de definição da tabela **Significado**, via SQL e via Hibernate, pertencente ao banco de dados do Falibras. Como podemos ver pelo Exemplo 1, utilizando anotações do Hibernate, não há necessidade de implementar nenhuma instrução SQL, todas as propriedades da tabela são desenvolvidas em Java na própria classes que está sendo mapeada.

Exemplo 1: Definição da Tabela via Anotações

```java
@Entity
@Table(name = "Significado")
public class Significado {

    @Id
    @Column(name="id", nullable=false)
    private int id;

    @Column(name="nome", nullable=false)
    private String nome;

    @Column(name="certeza")
    private int certeza = -1;

    @Column(name="ignorado_trand_padrao", nullable=false)
    private boolean eIgnoradoTraducaoPadrao;

    @Column(name="protegido_exclusao", columnDefinition = "TINYINT")
    @Type(type = "org.hibernate.type.NumericBooleanType")
    private boolean protegidoExclusao = false;

    @Column(name="protegido_alteracao",columnDefinition = "TINYINT")
    @Type(type = "org.hibernate.type.NumericBooleanType")
    private boolean protegidoAlteracao= false;

    @OneToOne
    @JoinColumn(name="id_animacao", referencedColumnName="id")
    private Animacao animacao;
}
```

Exemplo 2: Definção da Tabela via Script SQL

```sql
CREATE TABLE `significado` (
  `id` int(5) NOT NULL AUTO_INCREMENT,
  `nome` varchar(40) NOT NULL,
  `ignorado_trand_padrao` tinyint(1) NOT NULL,
  `protegido_exclusao` tinyint(1) DEFAULT '0',
  `protegido_alteracao` tinyint(1) DEFAULT '0',
  PRIMARY KEY (`id`),
  UNIQUE KEY `nome` (`nome`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=latin1;
```

Figura 8: Definição de uma tabela via Anotação (1) e via SQL (2)

Fonte: Autoria Própria

Por fim, para completar o último passo da configuração do Hibernate, foi preciso realizar uma refatoração em todas as consultas do sistema, trocando as instruções em SQL por instruções similares na linguagem HQL, mudando o foco das consultas de tabelas e colunas para objetos e seus atributos. As Figuras 9 e 10 apresentam as duas formas de realizar consultas. O exemplo apresentado na Figura 9 mostra a utilização do HQL, enquanto o exemplo apresentado na Figura 10 utiliza SQL.

```java
@Override
public Animacao obterAnimacao(String nomeAnimacao) throws ErroBDException {
    String hql = "FROM Animacao anim WHERE anim.nome = :nomeAnimacao";

    try {
        Query query = session.createQuery(hql);
        query.setParameter("nomeAnimacao", nomeAnimacao);
        List<Animacao> rs = query.list();

        if (rs.iterator().hasNext()) {
            return rs.get(0);
        } else {
            return null;
        }
    } catch (HibernateException e) {
        throw new ErroBDException(properties.getProperty("ErroConsultarAnimacao"), e);
    }
}
```

Figura 9: Exemplo de Consulta com o HQL

Fonte: Autoria Própria

```java
@Override
public Animacao obterAnimacao(String nomeAnimacao) throws ErroBDException{
    String sql = "SELECT id,nome, descricao, duracao_milis, arquivo FROM sign WHERE nome = '" + nomeAnimacao + "'";
    try {
        ResultSet rs = executarConsulta(sql);
        if (rs.next()) {
            return new Animacao(rs.getInt(1), rs.getString(2), rs.getString(3), rs.getInt(4), rs.getBytes(5));
        } else {
            return null;
        }
    } catch (SQLException e) {
        throw new ErroBDException(properties.getProperty("ErroConsultarAnimacao"), e);
    } catch (ErroConexaoException e) {
        throw new ErroBDException(properties.getProperty("ErroConsultarAnimacao"), e);
    } catch (ErroExecucaoException e) {
        throw new ErroBDException(properties.getProperty("ErroConsultarAnimacao"), e);
    }
}
private ResultSet executarConsulta(String consultaSql) throws ErroConexaoException, ErroExecucaoException {
    ResultSet rs = null;

    try {
        Statement stmt = getConexao().prepareStatement(consultaSql);
        rs = stmt.executeQuery(consultaSql);
    } catch (ErroConexaoException e) {
        throw new ErroExecucaoException(properties.getProperty("ErroConexaoBD") + " Usuario: " + usuario + " Senha: " + senha, e);
    } catch (SQLException e) {
        throw new ErroExecucaoException(properties.getProperty("ErroExecucaoComando") + consultaSql, e);
    }
    return rs;
}
```

Figura 10: Exemplo de Consulta via SQL

Fonte: Autoria Própria

Voltando a Figura 6, podemos observar que 16 classes da aplicação, de diferentes pacotes, pertencem ao mapeamento. Todas elas foram anotadas e, em alguns casos, refatoradas de modo que as tabelas geradas pelas anotações tivessem os mesmo atributos e relacionamentos apresentados nas tabelas criadas sem o uso do hibernate, através de scripts executados diretamente no SGBD MySQL. No total, 23 tabelas foram criadas pelo Hibernate devido aos relacionamentos binários e ternários existentes no esquema do banco de dados do Falibras. Em relação à manipulação do banco de dados, que é de responsabilidade da classe *BDController*, com o JDBC esta classe apresenta cerca de 106 funções que implementam instruções SQL de diferentes tipos (*select, insert, delete, update*). Com a refatoração da classe *BDController*, de acordo com a linguagem de consulta oferecida pelo Hibernate, houve uma redução da quantidade de funções, sendo necessário apenas 94 funções para que a manipulação dos dados com o Hibernate fosse equivalente ao uso do JDBC puro, onde todas essas funções implementam instruções HQL.

3.2. Método *Goal-Question-Metric* (GQM)

O método GQM, segundo Basili et al.(1994), é uma abordagem que especifica um sistema de medição baseado em metas que foca em um conjunto de questões usadas para definir as métricas que irão ajudar na interpretação dos dados. O modelo GQM pode ser divido em três níveis, são eles:

- **Nível Conceitual (*Goal*):** Envolve a definição do objetivo levando em consideração o artefato que será testado, o que pretende ser alcançado ou medido e em qual contexto o objeto de teste está inserido.
- **Nível Operacional (*Question*):** Formado por um conjunto de questões que tentam caracterizar o artefato avaliado a partir de uma determinada perspectiva e, quando respondidas, fornecem informações que ajudarão a atingir o objetivo inicial.
- **Nível Quantitativo (*Metric*):** Identifica um conjunto de métricas quantitativas usadas para responder as questões geradas anteriormente.

A estrutura hierárquica do GQM, apresentada na Figura 11, começa com um determinado objetivo (propósito, objeto, contexto, ponto de vista) que é refinado em uma ou mais questões que, por fim, irão gerar as métricas. Vale ressaltar que uma determinada métrica pode ser usada para responder várias questões de diferentes objetivos.

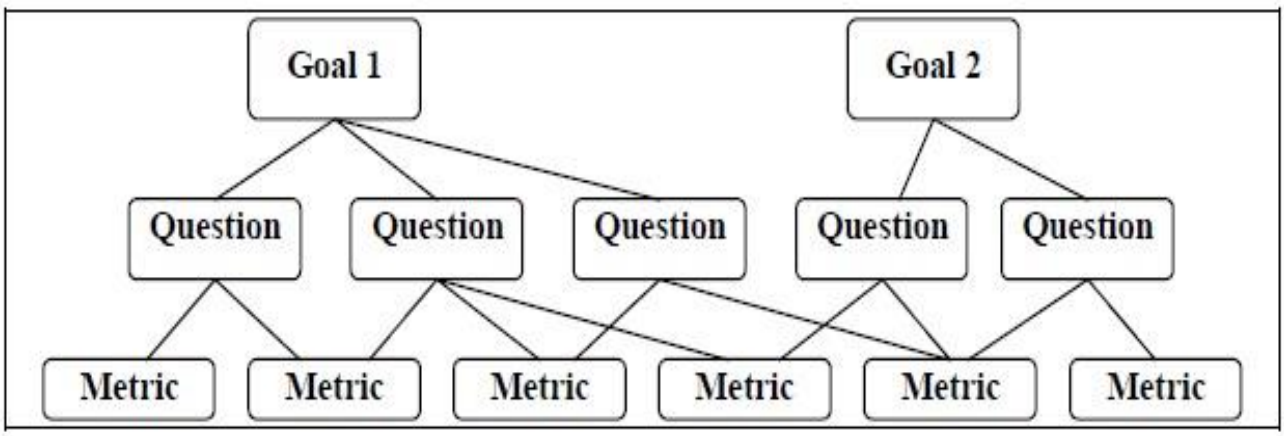

Figura 11: Estrutura Hierárquica da Técnica GQM

Fonte: BASILI et al. (1994)

Adaptando a técnica GQM para o presente estudo temos as seguintes definições:

- **Goal:**
  - ➢ **Propósito:** Comparação;
  - ➢ **Artefato:** *Framework* Hibernate
  - ➢ **Problema:** Desempenho
  - ➢ **Ponto de vista:** Equipe do projeto Falibras
  - ➢ **Contexto:** Analisar o comportamento do Hibernate em uma determinada plataforma.
- **Question:**
  - ➢ Qual a influência do *framework* Hibernate no desempenho do sistema Falibras em cenários reais?
  - ➢ O tempo de inicialização do sistema Falibras (*framework*, aplicação e SGBD) apresenta relevância no processo de tradução de possíveis cenários?
- **Metrics:**
  - ➢ Tempo de inicialização do sistema com JDBC.
  - ➢ Tempo de inicialização do sistema com o Hibernate
  - ➢ Tempo para executar todas as traduções, em diferentes cenários, utilizando JDBC.
  - ➢ Tempo para executar todas as traduções, em diferentes cenários, utilizando Hibernate.

3.3. Alvo de Testes: Sistema FALIBRAS

Atualmente, o uso da Web e das tecnologias de comunicação tem crescido de forma exponencial, acelerando cada vez mais a disseminação de informações. Porém, ainda há uma parte significativa da sociedade que não consegue ter acesso a essas informações devido a barreiras linguísticas. Dentre estes estão os deficientes auditivos que, no Brasil, possuem como língua nativa a Língua Brasileira de Sinais (LIBRAS), sustentada no canal visto-gestual, sendo, o português, transmitido na forma oral-auditiva, a sua segunda língua. Essa diferença no canal de comunicação, juntamente com as divergências morfossintáticas de ambas as linguagens, traz como consequência para os deficientes auditivos dificuldades na leitura e escrita de textos em português, excluindo os surdos do acesso a informações no meio digital.

Foi a partir deste cenário que surgiu o projeto Falibras. Idealizado em 2001, no Instituto de Computação da UFAL, o projeto tem como objetivo promover a inclusão digital e social dos surdos, além de tentar auxiliar na comunicação entre surdos e ouvintes, tudo isto através de um sistema que realiza a tradução automática de textos em português para gestos em LIBRAS.

3.3.1. Arquitetura

A versão atual do sistema Falibras, proposta por Rodrigues et al. (2012), está disposta em uma arquitetura heterogênea, formada a partir de características dos seguintes estilos e padrões arquiteturais:

- ***Model-View-Controller* (MVC):** Possibilita uma separação bem clara entre dados (*Model*), funcionalidades (*Controller*) e interfaces (*View*). Esta divisão em camadas tem como objetivo facilitar a integração de novos componentes ao sistema Falibras.
- **Centrada em Dados:** Trata da integração entre os módulos de funcionalidades, como analisadores e geradores de gramática, e o banco de dados. Por se tratar de um sistema baseado em regras, o Falibras apresenta uma intensa troca de informações entre a camada de controle e a camada de dados, e é nessa área que o presente trabalho atuará, tentando aprimorar o desempenho dessa comunicação.
- **Cliente-Servidor:** Tem como objetivo aumentar a escalabilidade da arquitetura, descentralizando a execução do software, fazendo com que funcionalidades altamente acopladas com o módulo cliente sejam alocadas em uma mesma máquina. Este estilo foi levado em consideração devido à extensão do projeto para uma versão Web, conhecida com Falibras-Web, que está em desenvolvimento.

A Figura 12 apresenta uma abstração da configuração atual da arquitetura do Falibras, tomando como base o estilo MVC.

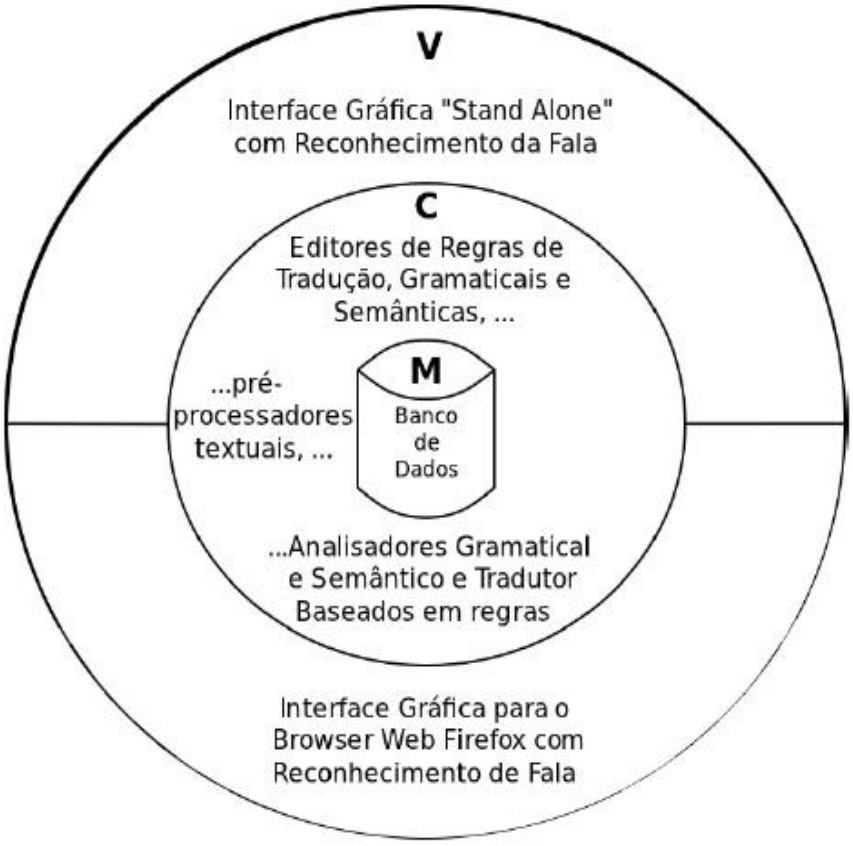

Figura 12: Diagrama da Arquitetura do Falibras, baseado no estilo MVC

Fonte: RODRIGUES et al. (2012, p. 5)

3.3.2. Processo de Tradução Atual

Na configuração atual do sistema Falibras, de acordo com Franco et al. (2012), o processo de tradução é executado por um tradutor híbrido que combina dois componentes, são eles:

- **Tradução por transferência sintática:** Neste módulo a tradução é baseada na sintaxe, projeta-se uma árvore sintática da gramática da língua fonte para uma correspondente na gramática da língua alvo. Possui um analisador léxico, um analisador sintático baseado em grafos, um analisador de contexto e um gerador de tradução em LIBRAS. Todas as regras de tradução são fixas e especificadas diretamente na aplicação Java.
- **Tradução por memória de tradução:** Nesta abordagem, o módulo de tradução realiza consultas a uma base dados composta por exemplos de traduções realizadas por um tradutor humano em busca de fragmentos similares ao texto de entrada. Fica a cargo do módulo de tradução escolher o exemplo mais adequado ao contexto em questão.

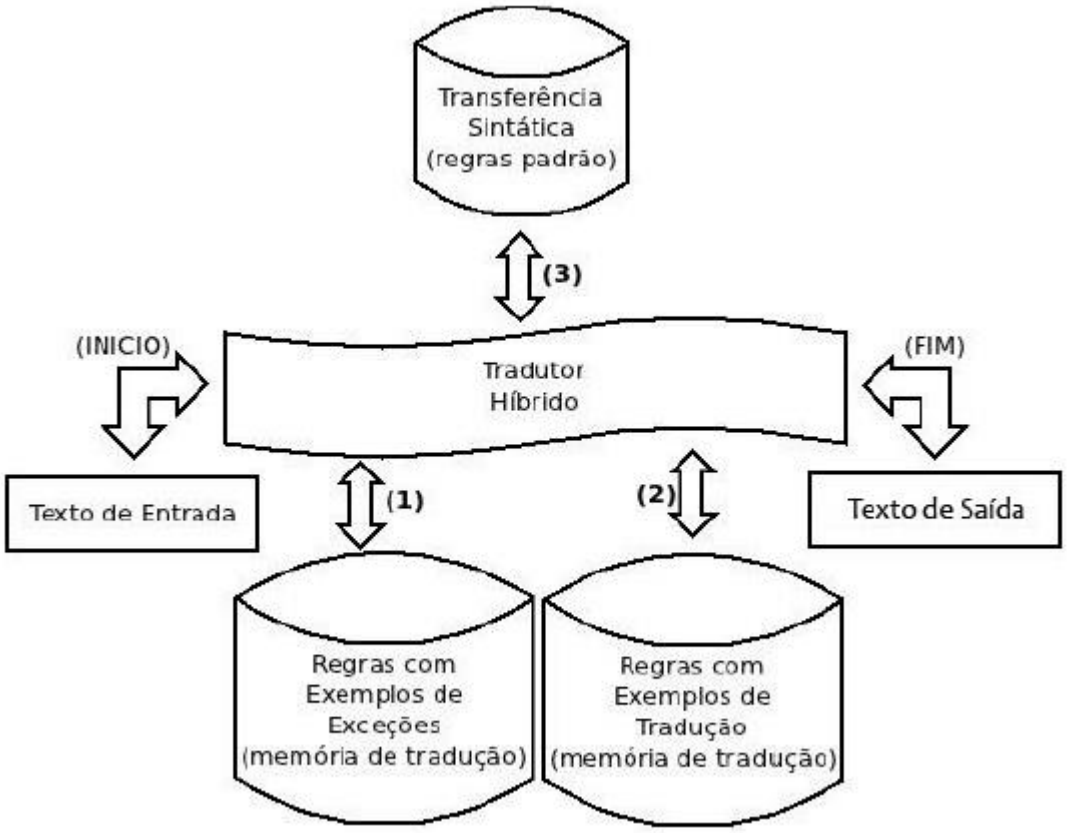

Figura 13: Processo de Tradução do Sistema Falibras

Fonte: FRANCO et al. (2012, p. 6)

A Figura 13 apresenta o esquema de tradução do Falibras, que funciona da seguinte maneira (FRANCO et al., 2012): Dado um texto de entrada, a primeira ação do sistema é verificar se o texto é similar a algum exemplo de exceção a regra cadastrado na base de dados. Caso não se encaixe em uma exceção, procuram-se exemplos, na memória de tradução, que possam servir de analogia para a tradução do texto de entrada. No último caso, se não for possível encaixar o texto nas regras cadastradas na memória de tradução, a tradução será executada através de transferência sintática.

Como podemos perceber o processo de tradução do sistema Falibras realiza uma série de consultas ao banco de dados, quanto maior for o texto a ser traduzido, maior será a troca de dados entre aplicação e SGBD. Isso poderá criar um ponto de estrangulamento que irá prejudicar o desempenho do sistema como um todo. É nessa área que o presente trabalho atuará, promovendo alterações no Falibras, visando aperfeiçoar a comunicação entre aplicação e banco de dados para melhorar o desempenho e o rendimento geral do Falibras.

3.4. Cenários de Teste

O foco dos testes realizados foi no tempo de execução do sistema Falibras. Este tempo de execução corresponde ao tempo gasto pelo Falibras para inicializar seus componentes e realizar a tradução de uma ou mais frases, dependendo do cenário em questão. Como já foi explicado em uma sessão anterior, o processo de tradução de uma frase compreende várias consultas ao banco de dados onde, de acordo com o estudo apresentado por Oliveira (2015), os métodos obterAnimacao(*String*) e identificarRegrasMT(Frase) executam a maior parte da tradução. Vale ressaltar que, para os testes realizados, apenas o módulo de tradução por transferência sintática foi utilizado, pois o módulo baseado em memória de tradução não continha nenhum exemplo base cadastrado. Para quantificar esses testes de desempenho, foram criadas variáveis, inicializadas em zero, e utilizadas para armazenar o tempo de processamento da tradução que vai ser calculado com o auxílio de funções de uma API padrão do Java, a java.lang.System.

Para que os testes pudessem apresentar resultados relevantes, foram idealizados quatro cenários reais do Falibras variando de acordo com o tamanho da frase a ser traduzida. Vale ressaltar que o tempo registrado não foi de apenas uma tradução, o tempo computado no final da execução é referente a 100 (cem) traduções do mesmo tipo de frase, ou seja, definido uma determinada frase, sua avaliação consiste em 100 (cem) traduções dela. Os cenários idealizados foram os seguintes:

- **Frase Curta:** “Ele é brasileiro?”.
- **Frase Média: “**Você já ouviu falar sobre os índios? Quem são eles? Viram o que as pessoas falam e pensam sobre os índios no Brasil e sobre a nossa história? O que eles comem?”.
- **Parágrafo:** “Você já ouviu falar sobre os índios? Quem são eles? Viram o que as pessoas falam e pensam sobre os índios no Brasil e sobre a nossa história? O que eles comem? Eles vivem no norte do Brasil, na floresta, longe da sociedade e vivem de pesca do rio e caça. Também trabalham com artesanato. Eles fazem parte da história do Brasil. As escolas ensinam sobre a história dos índios?”.
- **Redação (20 linhas):**

“Você já ouviu falar sobre os índios? Quem são eles? Viram o que as pessoas falam e pensam sobre os índios no Brasil e sobre a nossa história? O que eles comem? Eles vivem no norte do Brasil, na floresta, longe da sociedade e vivem de pesca do rio e caça. Também

trabalham com artesanato. Eles fazem parte da história do Brasil. As escolas ensinam sobre a história dos índios?”

“Você já ouviu falar sobre os índios? Quem são eles? Viram o que as pessoas falam e pensam sobre os índios no Brasil e sobre a nossa história? O que eles comem? Eles vivem no norte do Brasil, na floresta, longe da sociedade e vivem de pesca do rio e caça. Também trabalham com artesanato. Eles fazem parte da história do Brasil. As escolas ensinam sobre a história dos índios?”

“Você já ouviu falar sobre os índios? Quem são eles? Viram o que as pessoas falam e pensam sobre os índios no Brasil e sobre a nossa história? O que eles comem? Eles vivem no norte do Brasil, na floresta, longe da sociedade e vivem de pesca do rio e caça. Também trabalham com artesanato. Eles fazem parte da história do Brasil. As escolas ensinam sobre a história dos índios?”

“Você já ouviu falar sobre os índios? Quem são eles? Viram o que as pessoas falam e pensam sobre os índios no Brasil e sobre a nossa história? O que eles comem? Eles vivem no norte do Brasil, na floresta, longe da sociedade e vivem de pesca do rio e caça. Também trabalham com artesanato. Eles fazem parte da história do Brasil. As escolas ensinam sobre a história dos índios?”

Para cada cenário iremos fazer dois registros de tempo. Primeiro, que iremos chamar de **tradução simples,** vamos computar apenas o tempo do processo de tradução (classificação sintática do texto de entrada e definição da árvore sintática da linguagem fonte e da linguagem alvo) e busca de animações, depois vamos registrar o tempo de **tradução completa** que compreende o tempo de execução completo do Falibras, começando do início da aplicação até a última iteração de tradução, ou seja, engloba o tempo de tradução simples mais o tempo necessário para inicializar os componentes do sistema (Hibernate, MySQL, módulos de tradução, entre outros).

Os testes foram realizados em um computador pessoal que apresenta um ambiente configurado da seguinte maneira:

- Sistema operacional Windows 7 Home Basic, Service Pack 1, 64 bits.
- Processador Intel(R) Core(TM) i5-2410M, 2.30GHz
- Memória RAM de 8,00GB.
- Fabricante ASUSTeK Computer Inc.

- Eclipse Standard/SDK, versão: Luna Service Release 2 (4.4.2)
- Hibernate versão 4.3.6.Final 17-07-2014
- ApectJ, versão: 1.8.5
- MySQL Community Server, versão: 5.5.27 (GPL)
- Driver JDBC: mysql-connector-java-5.1.7

4. Análise dos Resultados

Baseado nos dados obtidos através dos testes realizados em todos os cenários será apresentado a seguir uma série de gráficos que explicam as informações coletadas neste presente trabalho. Vale lembrar que para cada cenário os testes foram executados 10 (dez) vezes para que no final fosse possível obter uma média aceitável.

4.1. Desempenho Tradução Simples

Como já foi definida anteriormente, a tradução simples corresponde apenas ao processo de tradução, ou seja, classificação sintática do texto de entrada de acordo com a memória de tradução e regras sintáticas predefinidas, mais a busca pelas animações de cada palavra identificada.

4.1.1. Frase Curta

Neste cenário a utilização do Hibernate teve um desempenho levemente melhor, como podemos ver no gráfico apresentado na Figura 14. Essa diferença varia entre dois e três segundos onde, no pior caso, o Hibernate executa em 6,45 segundos enquanto que o MySQL (sem o auxílio do Hibernate) processa a tradução em 9,25 segundos. Como este é um cenário de uso real, a diferença de desempenho alcançada aqui não tem um impacto significativo para o sistema Falibras.

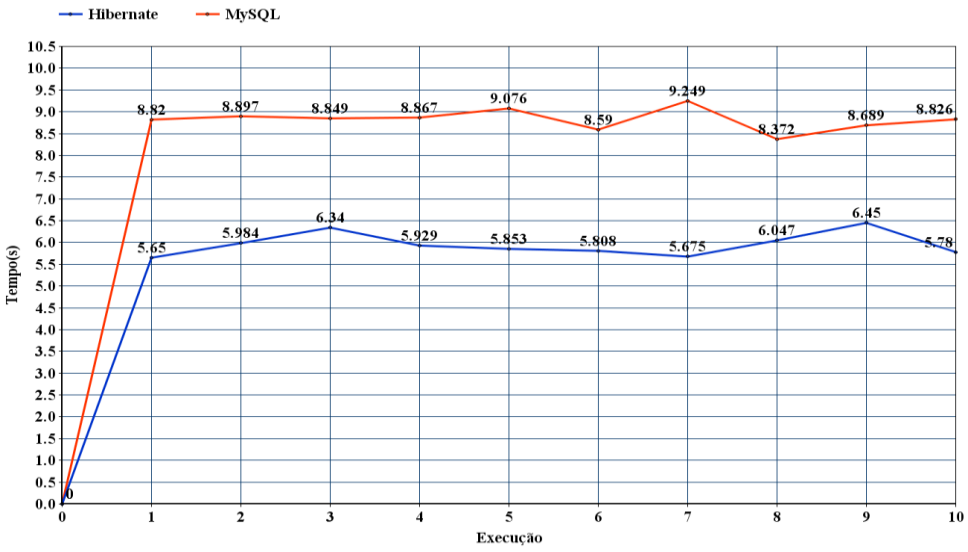

Figura 14: Tempo de Tradução Simples do Cenário Frase Curta

Fonte: Autoria Própria

4.1.2. Frase Média

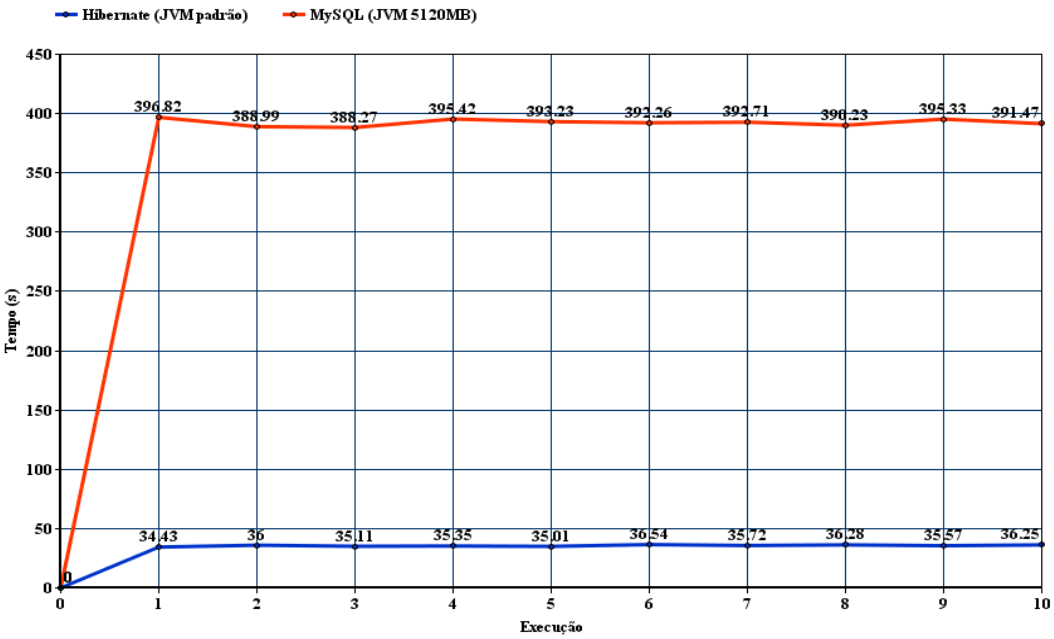

Figura 15: Tempo de Tradução Simples do Cenário Frase Média

Fonte: Autoria Própria

Para concluir os testes deste cenário com o MySQL puro, foi preciso aumentar a memória RAM padrão da Máquina Virtual Java (JVM) para 5.120 Megabytes. Com a memória RAM padrão da JVM, o Hibernate conseguiu processar todas as traduções, enquanto que o MySQL parou na execução 42 de 100 devido a um estouro de memória (*Exception in thread "main" java.lang.OutOfMemoryError: Java heap space*).

Após fazer as alterações necessárias na JVM para conclusão dos testes, os resultados apresentaram uma diferença significativa no desempenho do sistema com e sem o Hibernate. No pior caso, o Hibernate levou cerca 36,5 segundos, enquanto que o MySQL sozinho, no melhor caso, levou cerca de 388,3 segundos. Sendo assim, temos que o uso do Hibernate reduz o tempo de tradução em aproximadamente 6 minutos, além de apresentar um consumo menor de memória da JVM.

4.1.3. Parágrafo

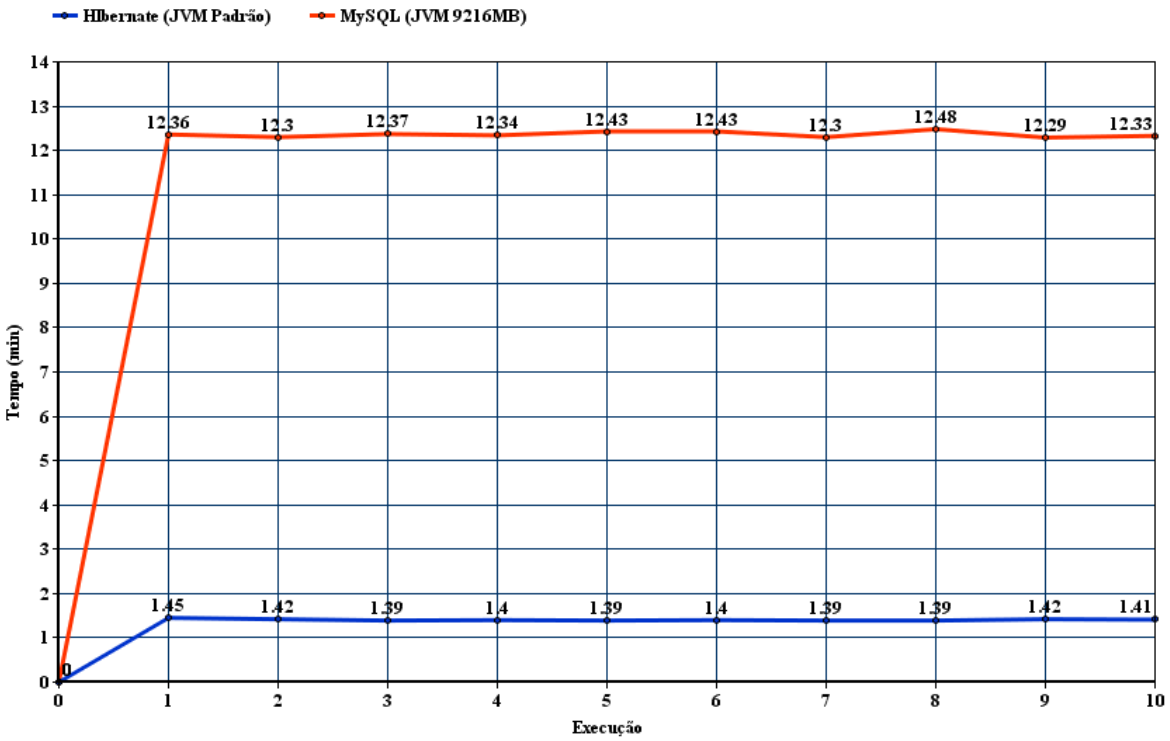

Figura 16: Tempo de Tradução Simples do Cenário Parágrafo

Fonte: Autoria Própria

Novamente, a ausência do Hibernate precisou de um aumento de memória da JVM para concluir sua execução. Utilizando a memória padrão da JVM o MySQL acusou o estouro da memória na iteração 21 de 100. Para processar todas as traduções foi necessário disponibilizar 9.216 MB de memória para a JVM, enquanto que com o Hibernate todas as traduções foram completadas normalmente, sem necessitar de alterações na configuração padrão da JVM.

Concluindo todas as iterações de tradução, mais uma vez o Hibernate apresentou um desempenho melhor que o MySQL puro. No melhor caso, o MySQL levou 12,29 minutos (737.395 segundos), já o Hibernate, no pior caso, concluiu as traduções em 1,45 minutos (86.726 segundos). A diferença de tempo (aproximadamente 11 minutos) e de consumo de memória apresentadas nesse cenário são perceptíveis e têm um forte impacto no trabalho do usuário final.

4.1.4. Redação

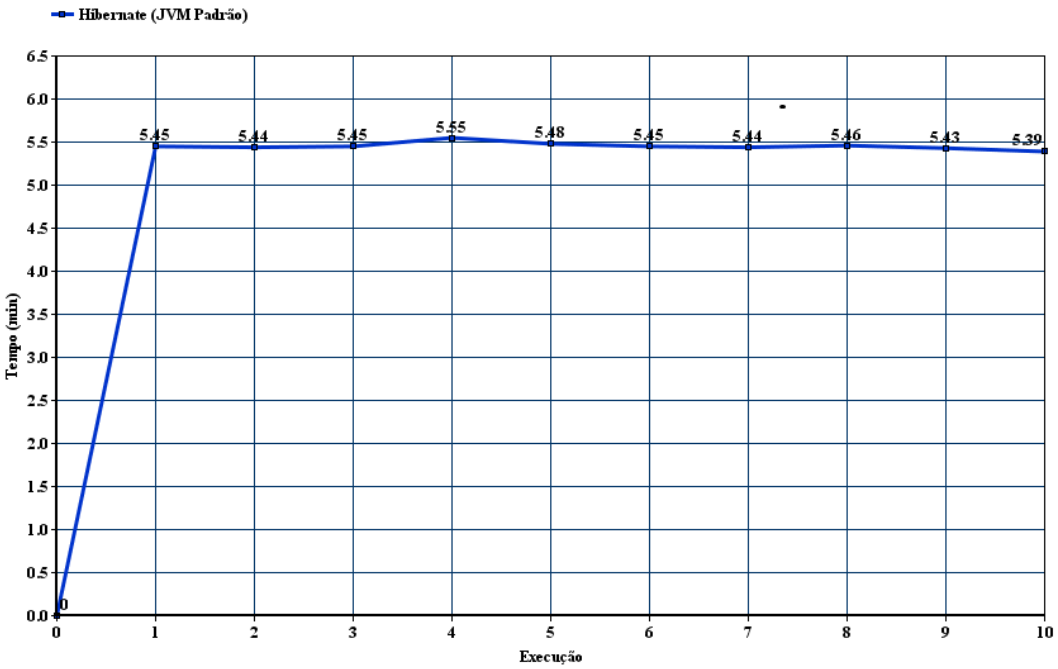

Figura 17: Tempo de Tradução Simples Cenário Redação

Fonte: Autoria Própria

Para este cenário, apenas os teste com o Hibernate foram concluídos. Para o MySQL, utilizando a memória padrão da JVM, o teste apresentou falha na iteração 6 de 100. Aumentando a memória disponível da JVM para 10.240MB o teste continuou apresentando estouro da memória, dessa vez na iteração 30 de 100. Com isso, os testes com o MySQL foram interrompidos devido ao alto consumo de memória. A Figura 17 apresenta o gráfico com os resultados obtidos nos testes com o Hibernate que leva, em média, 5,4 minutos para processar todas as traduções.

4.2. Desempenho Tradução Completa

Essa próxima fase de testes tem como objetivo registrar o tempo de tradução completa, a fim de analisar o impacto do tempo de inicialização de todos os componentes do sistema Falibras no processo de tradução.

4.2.1. Frase Curta

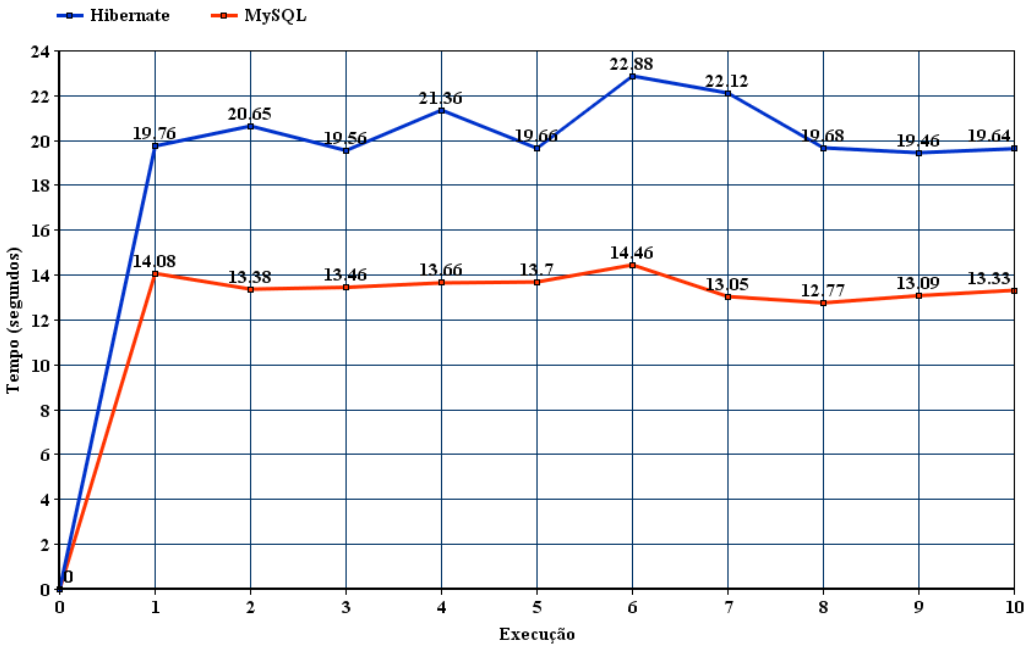

Figura 18: Tempo de Tradução Completa para Frase Curta

Fonte: Autoria Própria

Para este cenário, os testes sem o Hibernate apresentaram melhores resultados, como podemos ver na Figura 18. Levando em consideração os resultados obtidos anteriormente na tradução simples para este mesmo cenário, percebe-se que a inicialização do Hibernate prejudica o desempenho do sistema caso ele esteja sendo usado para traduzir pequenas frases. No pior caso com o Hibernate, apenas o processo de tradução levou 6,45 segundos, enquanto que a tradução completa levou 22.88 segundos. Sendo assim, apesar do Hibernate otimizar as consultas, sua inicialização apresenta um pequeno, mas perceptível, acréscimo de tempo.

4.2.2. Frase Média

A tradução completa apresentou resultados semelhantes à tradução simples para esse cenário. Com o Hibernate, a execução dos testes levou, no pior caso, aproximadamente 56 segundos, cerca de 20 segundos a mais que o teste de tradução simples. Para concluir os testes sem o Hibernate, foi necessário aumentar, novamente, a memória disponível da JVM para 5.120 Megabytes. Além do alto consumo de memória o MySQL continuou apresentando um aumento

significativo no tempo de execução, no melhor caso, ele levou cerca de 6 minutos e 41 segundos para concluir as traduções.

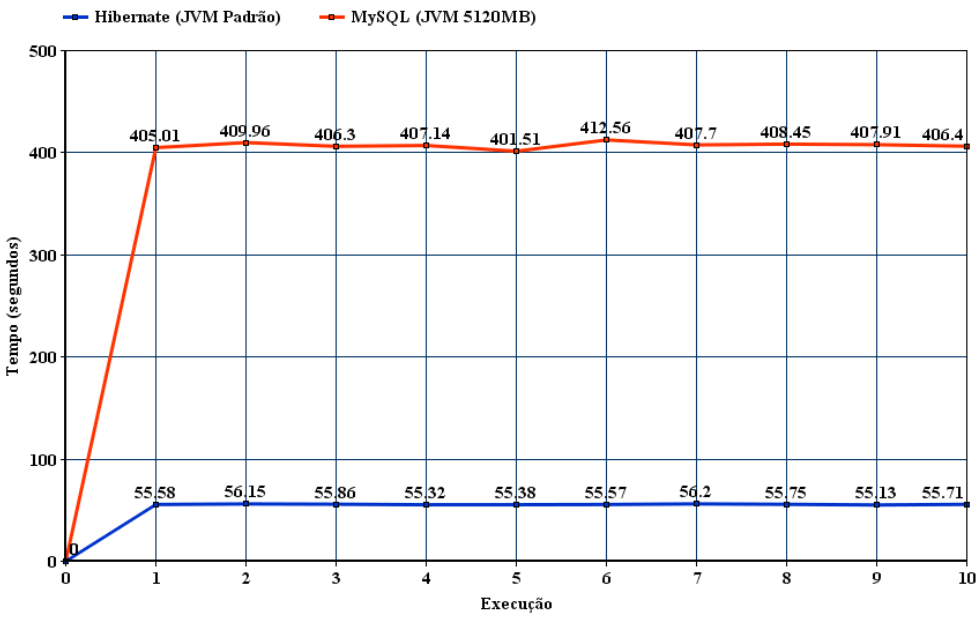

Figura 19: Tempo de Tradução Completa para Frase Média

Fonte: Autoria Própria

4.2.3. Parágrafo

Neste cenário, o uso do Hibernate apresentou, mais uma vez, melhorias, tanto no desempenho, quanto em relação ao consumo de memória. Utilizando apenas o MySQL, foi necessário alterar a memória RAM da JVM para 9126 MB e o teste completo foi concluído, no melhor caso, em 12 minutos 24 segundos, enquanto que o Hibernate, com memória padrão, executou o teste, no pior caso, em 1 minuto e 47 segundos, como podemos ver na Figura 20.

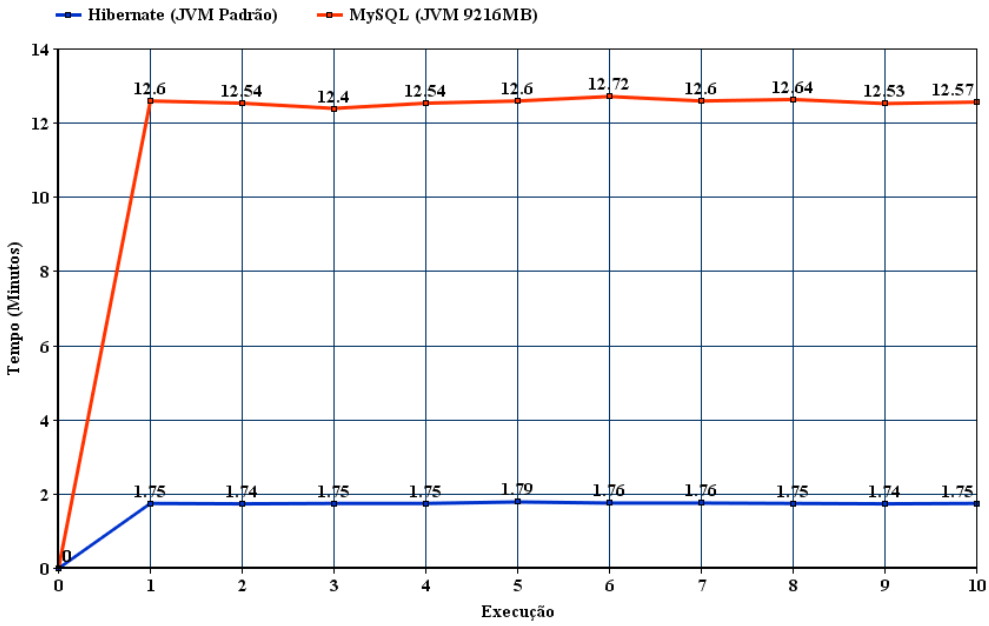

Figura 20: Tempo de Tradução Completa do Cenário Parágrafo

Fonte: Autoria Própria

4.2.4. Redação

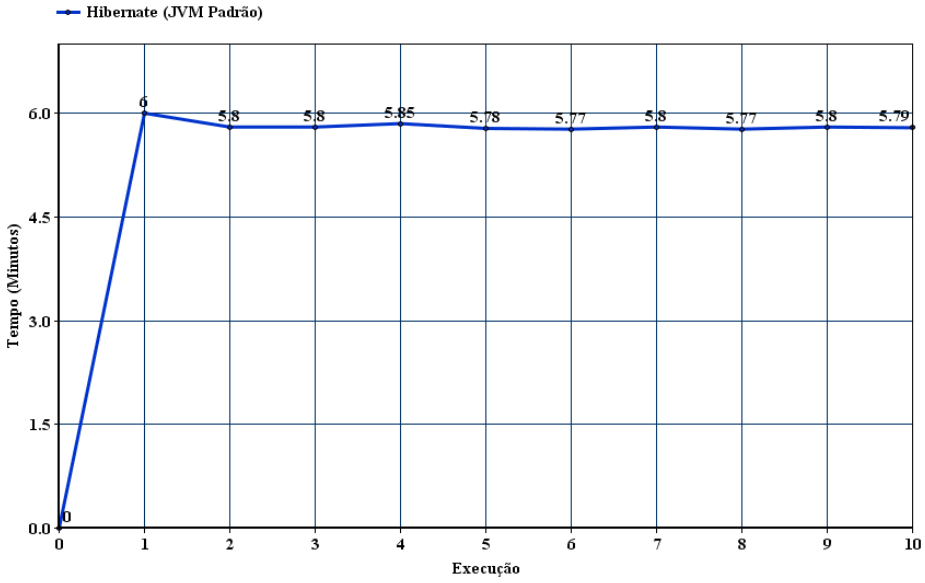

Figura 21: Tempo de Tradução Completa do Cenário Redação

Fonte: Autoria própria

Por fim, para o cenário redação, foi registrado o tempo da tradução completa apenas com o uso do Hibernate, pois a execução desse teste diretamente com o MySQL apresentou um alto

consumo de memória da JVM, sendo preciso mais de 10.240 MB para sua conclusão, tornando o experimento desnecessário, pois nem todos os usuários vão possuir máquinas com alta disponibilidade de memória. Sendo assim, a execução desse teste sem o Hibernate foi abandonada apresentando erro na iteração 6 de 100 com a memória disponível padrão da JVM. Os resultados obtidos nos testes com o Hibernate são apresentados na página anterior, na Figura 21.

4.3. Média Geral das Traduções para cada Cenário

Os gráficos apresentados nas Figuras 22 e 23 representam a média do tempo de execução do processo de tradução simples e de tradução completa para todos os cenários reais propostos, levando em consideração os testes apresentados nas sessões anteriores (4.1 e 4.2).

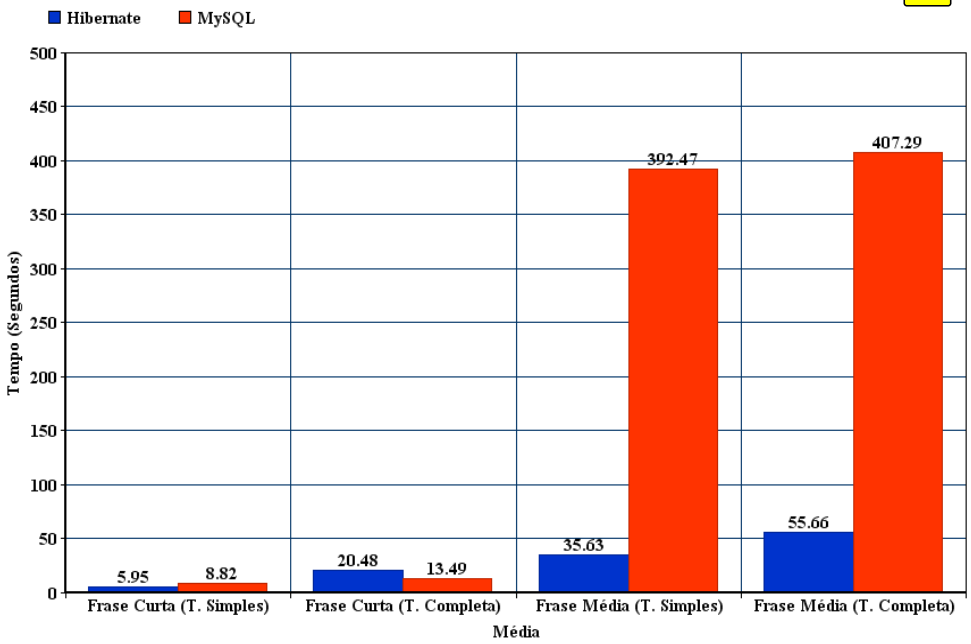

Figura 22: Tempo Médio de Tradução para Cenários Frase Curta e Frase Média

Fonte: Autoria Própria

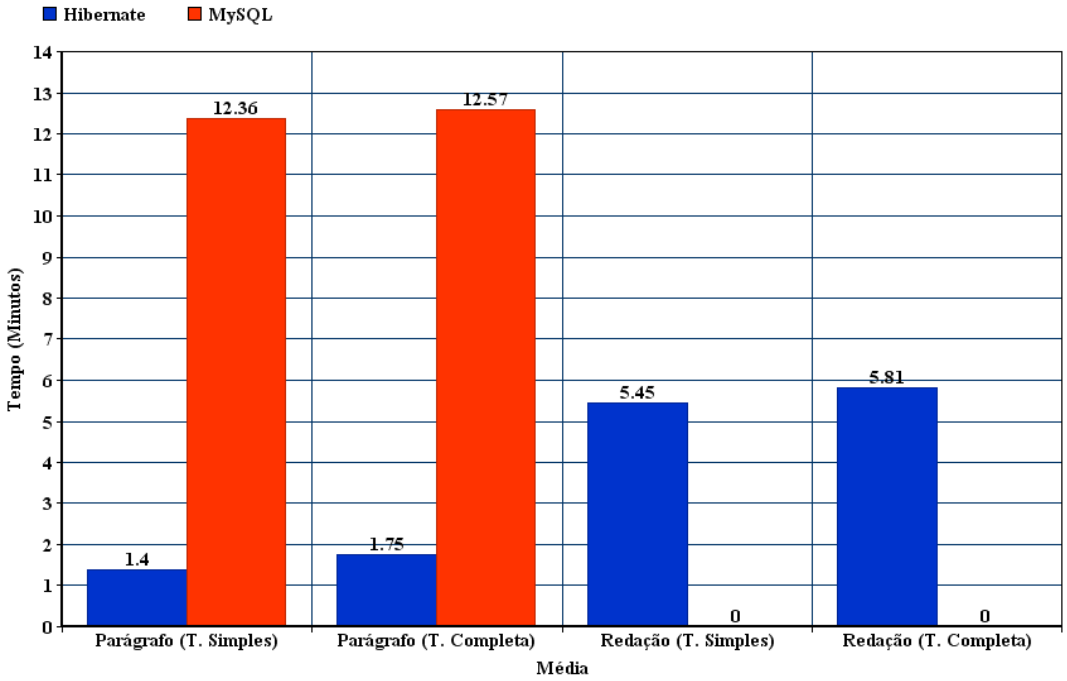

Figura 23: Tempo Médio de Tradução para Cenários Parágrafo e Redação

Fonte: Autoria Própria

Após executar os testes em todos os cenários, podemos perceber claramente, através das Figuras 22 e 23, que, com exceção da tradução completa do cenário frase Curta, a execução do sistema Falibras apresentou um melhor desempenho quando se utiliza o *framework* Hibernate. Quanto maior foi o texto a ser traduzido, maior foi a diferença de desempenho registrada, por exemplo, para o cenário Frase Curta da tradução simples, a utilização do Hibernate baixou o tempo de tradução em 2,87 segundos, já no cenário Parágrafo, o Hibernate apresentou uma redução de aproximadamente 11 minutos de acordo com as médias registradas, tanto para a tradução simples, quanto para a tradução completa.

Portanto, apesar do sistema Falibras levar alguns segundos a mais para inicializar, o uso do framework Hibernate reduz significativamente o tempo e o consumo de memória do processo de tradução do Falibras, provando que sua forma de acesso ao banco de dados e a linguagem de consulta HQL são poderosas ferramentas de otimização.

Como o Falibras visa auxiliar deficientes auditivos através da tradução automática de segmentos de texto e não apenas funcionar como um dicionário português-LIBRAS, a utilização do Hibernate pode trazer grandes benefícios para o usuário final.

5. Trabalhos Relacionados

No contexto do Falibras, o trabalho apresentado por Oliveira (2015) analisa o desempenho e o rendimento do sistema Falibras com dois SGBDs diferentes, o HSQLBD e o MySQL, através de cenários reais da aplicação e tomando como base para os testes a técnica GQM. Foi baseado neste estudo que nós decidimos usar o MySQL para se conectar com o Hibernate, pois ele apresentou resultados melhores que o HSQLDB.

Em relação à abordagem *Goal-Question-Metric*, vale a pena citar o estudo realizado por Soares (2011), que utilizou esta técnica para avaliar o desempenho e a escalabilidade de tecnologias de persistência semântica.

No caso do Hibernate, existem alguns trabalhos que avaliam seu desempenho em relação ao driver JDBC do MySQL em outros contextos como, por exemplo, Silva (2007), que avalia o desempenho do Hibernate em uma aplicação que apresenta uma arquitetura cliente-servidor. Foram realizadas consultas simples, sem clausulas de restrição (*Where*), em uma ou mais tabelas simultaneamente. Por fim, observou-se que o Hibernate apresentou uma redução de código e uma maior portabilidade a aplicação em relação ao SGBD utilizado. Porém, os testes finais apresentaram perda de desempenho em uma escala de milissegundos, levando a autora a concluir que, se a performance não for um fator crítico, onde alguns milissegundos não irão atrapalhar o trabalho do usuário final, então o Hibernate é vantajoso devido a facilidade de desenvolvimento.

Por fim, vale ressaltar que o Falibras não é o único software brasileiro que realiza a tradução automática do português para LIBRAS. Um exemplo disso é a ferramenta **SIGNSIM** (CAMPOS et al., 2000) que funciona como um dicionário entre essas línguas, realizando a tradução palavra por palavra sem interpretar o contexto delas. Existe também o **Sign WebMessage** (SOUZA; VIEIRA; 2006) que se trata de uma ferramenta de comunicação assíncrona na Web que apresenta as mensagens tanto em português quanto em LIBRAS, através de uma tradução direta, ou seja, sem o processamento sintático das frases a serem traduzidas.

6. Considerações Finais

O estudo apresentado foca no processo de tradução do sistema Falibras, visando melhorar o seu desempenho. Para isso, foi incorporado ao software o *framework* Hibernate, para realizar o mapeamento objeto-relacional, abstraindo e otimizando boa parte da comunicação entre a aplicação Java e o Sistema de Gerenciamento de Banco de Dados (SGBD), que no nosso caso é o MySQL.

Após concluir a fase de refatoração, foi realizada uma bateria de testes para verificar o impacto do Hibernate no desempenho do Falibras. Tomando como base a abordagem *Goal-Question-Metric* (GQM), foram idealizados quatro cenários do Falibras para a execução dos testes, que variam de acordo com o tamanho do texto a ser traduzido. Para verificar o desempenho de cada cenário, foi registrado o tempo de inicialização do sistema mais o tempo de 100 (cem) iterações do processo de tradução do seu texto correspondente.

Analisando os resultados obtidos com a execução dos testes, foi possível concluir que o Hibernate diminui o tempo e o consumo de memória do processo de tradução do Falibras, principalmente nos cenários que apresentaram grandes textos a serem traduzidos, como, por exemplo, o caso do cenário Parágrafo, onde as traduções realizadas com o Hibernate foram processadas em aproximadamente 2 minutos, enquanto que as traduções sem a presença do Hibernate levarem cerca de 12 minutos para serem concluídas, apresentando uma melhoria de 83% no desempenho do Falibras. Resumindo, quanto maior o texto, maior o ganho de desempenho do sistema Falibras com o Hibernate.

Em linhas gerais, no contexto do sistema Falibras, a utilização do *framework* Hibernate para a conexão entre a aplicação e o SGBD mostrou ser uma alternativa viável e mais promissora, oferecendo uma maneira simples de realizar o mapeamento objeto-relacional, além de oferecer uma linguagem de consulta expressiva, abstraindo do desenvolvedor a necessidade de implementar instruções na sintaxe SQL. Além do mais, os ganhos de desempenho reforçam a adoção do *framework.*

Para trabalhos futuros, pretende-se avaliar outras características do Hibernate em relação ao Falibras como, por exemplo, o seu rendimento (*throughput*), que corresponde à quantidade de traduções que o sistema consegue processar em um determinado intervalo de tempo, ou a sua escalabilidade. Existe também a possibilidade de avaliar o desempenho do *framework* Hibernate combinado com outros SGBDs relacionais.

7. Referências

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. **The Goal Question Metric Paradigm**, 1994. Disponível em: <https://www.cs.umd.edu/~basili/publications/technical/T89.pdf>. Acesso em: 29 maio, 2015.

BRASIL, **Lei nº 10.436 de 24 de abril de 2002**. Disponível em: <http://www.planalto.gov.br/ccivil_03/leis/2002/l10436.htm>. Acesso em: 21 maio, 2015.

CAMPOS, M. B.; GIRAFFA, L. M. M.; SANTAROSA, L. M. C. **SIGNSIM: uma ferramenta para auxílio à aprendizagem da língua brasileira de sinais**. In: V Congresso Ibero-Americano de informática na Educação – RIBIE, Chile. 2000.

ELMASRI, R.; NAVATHE, S. B. **Fundamentals of Database Systems**. 6th Ed. USA: Addison-Wesley, 2010, p. 751-755.

FERNANDES, R. G.; LIMA, G. A. F. **Hibernate com Anotações**. 2007. Disponível em: <http://www.futurepages.org/wiki/lib/exe/fetch.php?media=quickstart:hibernate_anotacoes.pdf>. Acesso em: 18 maio, 2015.

FRANCO, N. M.; BRITO, P. H. S.; CORADINE, L. C. **FALIBRAS: Uma Ferramenta Flexível para Promover Acessibilidade de Pessoas Surdas.** In: Congresso Internacional de Informática Educativa, 2012, Santiago, Chile. TISE, 2012.

GARCIA-MOLINA, H.; ULLMAN, J. D.; WIDOM, J. **Database Systems: The Complete Book**. 2th Ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2008, p.1-12.

KOJIMA, C. K.; SEGALA, S. R. **Libras – Língua Brasileira de Sinais: A imagem do pensamento**. Volumes 1, 2, 3, 4 e 5. São Paulo: Editora Escala, 2008.

MySQL. **MySQL 5.5 Reference Manual**, MySQL Standards Compliance. 2015.

OLIVEIRA, E. B. C. **Comparação Sistemática de Desempenho e Rendimento do SGBDS HSQLDB e MySQL no Contexto do Sistema Falibras.** Trabalho de Conclusão de Curso (Graduação em Ciência da Computação). Arapiraca: Universidade Federal de Alagoas - UFAL, 2015.

PALMEIRA, T. V. V. **Aprendendo Java com JDBC**. 2013. Disponível em: <http://www.devmedia.com.br/aprendendo-java-com-jdbc/29116>. Acesso em: 13 maio. 2015.

PRESSMAN, Roger S. **Engenharia de Software: Uma Abordagem Profissional**. 7.Ed. Porto Alegre: AMGH Editora, 2011.

REDHAT. **JBoss Enterprise Application Platform 4.3 Hibernate Reference Guide.** Red Hat Documentation Group, 2011.

RODRIGUES, V. P.; BRITO, P. H. S.; MAGALHÃES, T. B. S.; FELIX, N. R. S.; BARBOSA A. A. **Remodelagem da Arquitetura do Sistema Falibras**. In: ERBASE, 2012, Juazeiro-BA. WTICGBASE, 2012.

SILVA, Carolina Fernanda. **Análise e Avaliação do Framework Hibernate em uma Aplicação Cliente/Servidor**. Monografia defendida e aprovada na FAJ em 11 de Dezembro de 2007.

SOARES, E. E. **Evaluation of Performance and Scalability of Semantic Persistence Technologies**. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) Universidade Federal de Alagoas, Maceió, AL, 2011.

SOMMERVILLE, I. **Engenharia de Software**. 8. Ed. Tradução Selma Shin Melnikoff; Reginaldo Arakaki; Edilson de Andrade Barbosa. São Paulo: Pearson, 2007.

SOUZA, V. C.; VIEIRA, R. **Uma Proposta para Tradução Automática entre Libras e Português no Sign WebMessage**. 2006. Disponível em:

<https://ead2.moodle.ufsc.br/pluginfile.php/64082/mod_resource/content/1/SOUZA%2C%20VIEIRA%2C%202006.pdf>. Acesso em: 04 junho, 2015.