

MINISTÉRIO DA EDUCAÇÃO  
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA  
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE MINAS GERAIS  
CAMPUS SÃO JOÃO EVANGELISTA

Curso Superior de Sistemas de Informação

Douglas Souto de Souza  
João Paulo Medina Passos

**UMA SOLUÇÃO WEB PARA O PROBLEMA DE ALOCAÇÃO DE HORÁRIOS  
ACADÊMICOS**

São João Evangelista  
2013

Douglas Souto de Souza  
João Paulo Medina Passos

## **UMA SOLUÇÃO WEB PARA O PROBLEMA DE ALOCAÇÃO DE HORÁRIOS ACADÊMICOS**

Monografia apresentada ao curso de Sistemas de Informação do Instituto Federal de Minas Gerais – Campus São João Evangelista, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Esp. Bruno de Souza Toledo  
Coorientador: Me. Rosinei Soares de Figueiredo

São João Evangelista  
2013

### **FICHA CATALOGRÁFICA**

Elaborada pelo Serviço Técnico da Biblioteca do  
Instituto Federal Minas Gerais – Campus São João Evangelista

S729u      SOUZA, Douglas Souto de, 1986 -

Uma solução Web para o problema de alocação de horários acadêmicos./ Douglas Souto de Souza; João Paulo Medina Passos. São João Evangelista, MG: IFMG – Campus São João Evangelista, 2013.

63 p.: il.

Trabalho de Conclusão de Curso - TCC (graduação) apresentado ao Instituto Federal Minas Gerais – Campus São João Evangelista – IFMG, Curso de Bacharelado em Sistemas de Informação, 2013.

Orientador: Prof. Esp. Bruno de Souza Toledo

Coorientadores: Prof. Me. Rosinei Soares de Figueiredo

1. Web. 2. Gestão acadêmica. 3. Documentos Web. I. Instituto Federal Minas Gerais – Campus São João Evangelista. Curso de Bacharelado em Sistemas de Informação. II. Título.

CDD 005.276

Douglas Souto de Souza  
João Paulo Medina Passos

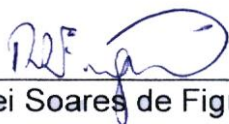
## **UMA SOLUÇÃO WEB PARA O PROBLEMA DE ALOCAÇÃO DE HORÁRIOS ACADÊMICOS**

Monografia apresentada ao curso de Sistemas de Informação do Instituto Federal de Minas Gerais – Campus São João Evangelista, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.



---

Bruno de Souza Toledo (Orientador) – IFMG-SJE



---

Rosinei Soares de Figueiredo (Coorientador) – IFMG-SJE



---

Ricardo Bittencourt Pimentel – IFMG-SJE

São João Evangelista, 13 de Novembro de 2013.

*Este trabalho é dedicado a  
todos aqueles que nos ajudaram  
a concluir o presente curso. Em especial a  
Maria Medina Cardoso Netto e a Gildete Souto de Souza*

## **AGRADECIMENTOS**

Agradeço primeiramente a Deus. Aos professores e colegas do curso, nossa admiração e agradecimentos. Aos familiares e amigos, o nosso respeito, sempre.

Basta-me que, onde quer que nasçam homens, se possa fazer deles o que proponho; e que, tendo feito deles o que proponho, se tenha feito o que há de melhor, tanto para eles como para os outros.  
(ROUSSEAU, 1968)

## RESUMO

O presente trabalho teve como foco o desenvolvimento de um software para plataforma Web, como solução para o problema de alocação de horários acadêmicos, delimitado por um estudo de caso relacionado ao curso de Bacharelado em Sistemas de Informação do Instituto Federal de Minas Gerais – campus de São João Evangelista. A alocação de horários acadêmicos é um problema clássico de combinação, classificado como NP-difícil. O problema de alocação de horários se resume em distribuir aulas das disciplinas das turmas de cada curso em um horário disponível durante os dias letivos da semana, sem confrontar os horários de professores, turmas e salas de aulas. Esse problema combinatório de horários é conhecido na literatura acadêmica como *timetabling*. Devido às particularidades do problema de *timetabling* de cada instituição, uma vez que cada uma considera objetivos e restrições distintas ao criar a grade de horários acadêmicos, delimitou-se o curso de Sistemas de Informação do Instituto Federal de Minas Gerais como instância do problema de *timetabling* estudado. Ao final deste trabalho, obteve-se um *software* utilizando a linguagem de programação *Java* e seus principais frameworks de desenvolvimento, o mesmo foi capaz de gerar uma grade de horários básica, respeitando as restrições mais importantes exigidas no curso de Sistemas de Informação. O software utiliza um algoritmo baseado em heurística computacional própria para fazer as alocações dos horários em uma grade. Considera-se este trabalho como um marco inicial para resolução do problema de Alocação de Horários do IFMG-SJE. Espera-se que, com ele, seja possível inspirar novos pesquisadores a explorar este vasto campo de pesquisa.

Palavras-chave: Grade de Horários; Timetabling; Java; Framework.



## **ABSTRACT**

The presented assignment focuses on the development of a software on the web platform as a solution to the problem of allocation of academic time schedules, delimited by a case study in the course of Bachelor's degree of Information Systems in the Federal Institute of Minas Gerais - campus of Sao Joao Evangelista. The allocation of academic time schedules is a classic problem of combination, classified as NP - hard. The allocation problem boils down to distribute schedule classes of subjects of the classes of each course in a time available during the school days of the week without confronting the time schedules of teachers, classes and classrooms. This combinatorial problem of schedule is known in the academic literature as timetabling. Due to the particular timetabling problem of each institution, which each have different goals and constraints when creating the grid of academic time schedules. Was delimited that the course of Information Systems at the Federal Institute of Minas Gerais as an instance of the problem studied timetabling. At the end of this work, a software has been developed using the Java programming language and its key development frameworks, it was able to generate a basic schedule grid, respecting the most important constraints required in the course of Information Systems. The software uses an algorithm based on its own computational heuristic, to make allocations of schedules on a grid. This assignment is considered as a starting point for solving the problem of allocation of time schedules of IFMG - SJE. Where it is expected to inspire new researchers to explore this vast study field.

Keywords: Time schedules; Timetabling; Java; Framework.

## SUMÁRIO

1 INTRODUÇÃO .....	9
2 FUNDAMENTAÇÃO TEÓRICA .....	12
2.1 O conceito de <i>timetabling</i> .....	12
2.1.1 Tipos de <i>timetabling's</i> escolares .....	12
2.1.2 Restrições de <i>timetabling</i> .....	13
2.2 Inteligência Artificial.....	14
2.3 Java.....	15
2.3.1 Uma breve história sobre a Linguagem Java.....	16
2.4 Hibernate .....	16
2.5 XML .....	18
2.6 Framework .....	19
3 REVISÃO BIBLIOGRÁFICA .....	20
4 METODOLOGIA E PROCEDIMENTOS.....	23
4.1 Delimitação do esboço.....	23
4.2 Java Web .....	24
4.3 Padrão MVC.....	25
4.4 Ferramentas utilizadas.....	28
4.4.1 Instalação do Java.....	28
4.4.2 Instalação do Apache Tomcat .....	29
4.4.3 Instalação do MySQL e MySQL Workbench.....	30
4.4.4 Instalação do NetBeans .....	30
4.4.5 Instalação e configuração do Hibernate no NetBeans .....	31
4.4.6 Instalação do JSF .....	32
4.5 Função do software AHA .....	34
4.6 Interface do CRUD .....	37
4.7 Lógica do gerador de horários .....	42
4.7.1 O métodos gerarGrade() da classe motor .....	42
4.7.2 O método gerarHorario() da classe Motor .....	47
4.7.3 O método buscarDisciplina() da classe Motor .....	48
4.7.4 O método procurarVaga() da classe Motor .....	49
4.7.5 O método validar() da classe restricaoohard .....	51
4.8 Tratamento dos dados .....	52
5 ANÁLISE DOS RESULTADOS.....	54
6 CONSIDERAÇÕES FINAIS .....	58
REFERÊNCIAS BIBLIOGRÁFICAS .....	59
ANEXO.....	61

## 1 INTRODUÇÃO

Muitas instituições de ensino dedicam horas (ou dias) de trabalho na criação da Grade de Horários Escolar (GHE). Segundo Bardadym (1996), a construção manual dessa GHE normalmente é uma tarefa penosa e complexa que requer vários dias de trabalho, além de consumir uma razoável quantidade de recursos humanos das instituições de ensino. Ao final da elaboração da GHE, é possível perceber que nem sempre os resultados obtidos agradam os alunos e professores, por exemplo, quando as aulas de uma determinada disciplina são alocadas em horários fragmentados, causando assim uma segmentação na aprendizagem. Mesmo existindo vários softwares no mercado que geram a grade de horários de forma automática, seu uso é pouco frequente já que esses softwares propõem soluções genéricas que pouco atendem às necessidades específicas exigidas na construção das GHE das instituições.

Constantino (2009) e Braz Júnior (2000) evidenciam que o problema de alocação de horários (*timetabling problem*) é um problema clássico que tem sido tema de uma série de conferências científicas internacionais, tais como o *Practice and Theory of Automated Timetabling* (PATAT).

O problema de *timetabling* possui um escopo de aplicações bastante abrangente. Dentro desse escopo é que se encontra o problema abordado neste trabalho, cujo foco está voltado para a geração de horários acadêmicos. Realizou-se o desenvolvimento de um software Web, utilizando heurística computacional própria, para a geração de uma Grade de Horários Acadêmicos (GHA) básica, com restrições baseado nas necessidades específicas do curso de Sistemas de Informação (SI) do Instituto Federal de Minas Gerais – Campos São João Evangelista (IFMG-SJE).

Cita-se, como relevância, o fato de que o sistema foi baseado no estudo de caso do curso SI do IFMG-SJE, o que servirá como marco inicial para a criação de um modelo que englobe as necessidades, não apenas do curso SI, mas do campus como um todo. Podendo, assim, resolver este problema específico do campus com eficácia.

O problema da *timetabling* consiste na combinação de diversas variáveis (alunos, professores, salas e horários disponíveis, por exemplo), gerando uma

sequência de combinações que devem satisfazer diversos tipos de restrições em um tempo previamente fixado (PAIM; GREIS, 2008).

Segundo Shaerf (1999), no âmbito escolar, o problema de *timetabling* é dividido em três categorias: *school timetabling* (aulas de ensino médio), *course timetabling* (aulas de ensino superior), e *examination timetabling* (marcação de avaliações). Boa parte das soluções desenvolvidas para estes problemas são baseadas em métodos que utilizam técnicas de Inteligência Artificial (IA).

O trabalho teve como objetivo geral desenvolver uma solução *Web* para o problema de Alocação de Horários Acadêmicos baseado no Estudo de Caso do IFMG-SJE delimitado pelo curso de Sistemas de Informação e pelo Prédio Escolar II Centro de Tecnologia da Informação. Como objetivos específicos, destacaram-se: a) desenvolver um sistema capaz de gerar uma grade de horários básica de maneira automática e rápida; b) criar um *software Web* conciso, implementado no padrão MVC, que possua uma boa adaptabilidade, em que adequações futuras possam ser acrescentadas com facilidade, sendo assim expansível para os demais prédios e curso do IFMG-SJE.

A implementação do *software* realizou-se na plataforma Java *Web*, foi utilizado IDE<sup>1</sup> NetBeans 7.3 e os *frameworks* *JavaServer Faces* (JSF) para interface e o *Hibernate* para manipulação da base de dados, para a qual se utilizou do SGBD MySQL. É interessante ressaltar que todas as ferramentas e *frameworks* são de licença gratuita e de uma eficácia aprovada pela comunidade *Web*.

Como resultado, chegou-se ao *software* AHA (Alocação de Horários Acadêmicos), com o qual se obteve êxito em todos os objetivos conforme citado nas considerações finais deste trabalho.

Este trabalho organizou-se em capítulos. O primeiro capítulo aqui descrito apresentou o tema, relevância, abrangência, problematização, justificativa, objetivos, considerações e organização do trabalho. No segundo capítulo, abordou-se a fundamentação teórica, descrevendo conceitos necessários para a sustentação teórica do trabalho, destacando os autores Schaerf (1999) e Souza (2000). O terceiro capítulo tratou da revisão bibliográfica, onde mostrou-se os trabalhos correlatos ao problema do tema, destacando-se os autores Freitas *et. al.* (2007),

---

<sup>1</sup> IDE, do inglês *Integrated Development Environment* ou Ambiente Integrado de Desenvolvimento, é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo.

Carvalho (2011) e Fernandes, Pereira e Freitas (2002). No quarto capítulo, descreveu-se a metodologia utilizada para a realização do trabalho. No quinto capítulo, fez-se a apresentação do resultado e no sexto capítulo as considerações finais.

## 2 FUNDAMENTAÇÃO TEÓRICA

Fundamentação teórica é a parte do trabalho onde se apresentam os autores selecionados para sustentação e abordagem do tema. Segundo Lakatos e Marconi (2010), a fundamentação teórica é construída através de um levantamento de toda a bibliografia já publicada, em forma de livros, revistas, publicações avulsas e imprensa escrita.

### 2.1 O conceito de *timetabling*

Pain e Greis (*apud* SILVA e BURKE, 2003) definem *timetabling* como um problema típico de elaboração de grades de horários de forma automatizada. Os problemas de *timetabling* podem ser classificados de acordo com o tipo de evento ao qual se quer aplicar a grade de horários: área esportiva, hospitalar, educacional, entre outras. Na área escolar, encontrou-se um modelo do problema de alocação de horários que consiste em agendar uma sequência de encontros (aulas, avaliações) entre professores e estudantes em um determinado horário e local (sala, laboratório).

#### 2.1.1 Tipos de *timetabling* escolares

De acordo com Souza (2000), os problemas de programação de horários escolares mais frequentemente abordados podem ser classificados em três grupos: *school timetabling* (aulas de ensino médio), *course timetabling* (aulas de ensino superior), e *examination timetabling* (marcação de avaliações).

Segundo Souza (2000), o *school timetabling* consiste em agrupar os estudantes em turmas com o mesmo plano de aula. Essa instância do problema possui uma quantidade de recursos menor, pois uma turma já possui uma sala especificada e seu grupo de alunos, em sua maioria, é predeterminado, o que faz com que a complexidade seja reduzida. Em *course timetabling* os estudantes são considerados individualmente, onde há um conjunto de disciplinas e para cada disciplina uma carga horária. O problema consiste em alocar as aulas das disciplinas aos períodos disponibilizados respeitando a disponibilidade e capacidade das salas existentes, de modo que nenhum estudante tenha duas ou mais aulas ao mesmo

tempo. Já em *examination timetabling*, consiste em agendar avaliações para diversas disciplinas. Para cada disciplina existe uma avaliação, as mesmas não devem ocorrer em um mesmo dia e horário e a marcação de provas em dias consecutivos devem ser evitadas.

### 2.1.2 Restrições de *timetabling*

As restrições são um assunto essencial na geração das grades de horários, pois é em torno delas que se define de qual maneira a grade é construída. Pode-se entender como restrições um conjunto de regras que devem ser impostas para a combinação dos recursos disponíveis de uma instituição. Como exemplo, pode-se considerar os horários de disponibilidade dos professores e o número de salas e laboratórios.

Estas restrições podem ser classificadas em dois grandes grupos:

- Restrições **essenciais (hard)** são aquelas que devem ser obrigatoriamente satisfeitas;
- Restrições **não-essenciais (soft)** são aquelas que podem ser violadas, mas que devem ser satisfeitas da melhor forma possível. (SCHAERF, 1999).

Ainda segundo Schaerf (1999), as restrições do tipo *hard* são as que não podem ser violadas em hipótese alguma, pois causariam a inviabilidade da grade de horário. Um exemplo desse tipo de restrição seria a sobreposição de aula de um professor, pois o mesmo não pode lecionar mais de uma aula em um determinado horário. Já as restrições do tipo *soft* são as que se desejam alcançar, mas a sua violação não inviabiliza a grade de horários. A preferência de um professor em lecionar sua disciplina no horário diurno em função de algum compromisso pessoal é um exemplo de restrição *soft*. No Quadro 1 são apresentados os tipos de restrições em horários acadêmicos.

**Quadro 1 – Tipos de restrições em horários acadêmicos**

Restrição	Descrição
Sobreposição de professores	Um professor não pode lecionar mais do que uma aula em um dado horário.
Sobreposição de turmas	Uma turma não pode ter mais do que uma aula em um dado horário.
Sobreposição de salas especializadas	Em um dado horário, o número de salas especializadas requeridas não pode ser maior que o número de salas

	especializadas disponíveis.
Restrições de pré-alocação	Certas aulas precisam ser pré-allocadas a um conjunto específico de horários.
Indisponibilidade de professores	Uma aula não pode ser alocada a um horário durante o qual um professor não esteja disponível.
Indisponibilidade de turmas	Uma aula não pode ser alocada a um horário durante o qual uma turma não esteja disponível.
Restrições geográficas	É preciso que tenha tempo suficiente para que uma turma possa se deslocar de um prédio a outro quando as aulas são ministradas em locais distantes. Assim, duas aulas ministradas em dois locais distantes e envolvendo um mesmo professor ou uma mesma turma não podem ser alocadas em horários consecutivos.
Restrições de compacidade	Cada professor deseja uma agenda de aulas com o menor número possível de buracos e aulas isoladas. O quadro de horário das turmas é, a princípio, compacto, uma vez que o número de horários disponibilizados para cada turma é, em geral, exatamente igual ao número de aulas constantes em seu currículo (gerando a inexistência de “buracos” na tabela de horário de aulas das turmas).
Restrições de precedência	O quadro de horário de uma turma não pode conter sequências de matérias consideradas difíceis, pois assim tornaria inviável para os alunos.
Restrições de preparação de salas especializadas	Uma sala especializada não está disponível quando ela está sendo preparada para uma aula específica.
Intervalo variável de refeição	Em determinadas instituições o horário de intervalo deve acontecer após dois horários consecutivos da mesma disciplina.

Fonte: Souza, 2000

## 2.2 Inteligência Artificial

Ao longo dos anos, pôde-se notar o grande avanço da ciência as diversas áreas de conhecimento, o que proporciona uma vida mais saudável e mais confortável às pessoas, tanto nos lares como nos ambientes de trabalho. Junto com esse avanço, cresce a expectativa do que ainda está por vir trazendo mais melhorias para a população mundial. Um exemplo no ramo da medicina seria a descoberta da cura da AIDS, ou até mesmo a cura para o câncer.

Na computação, assim como nas outras áreas, o crescente avanço dos computadores e de suas linguagens também faz crescer as expectativas sobre os próximos passos no avanço dessa ciência que possui tanto potencial. É possível citar, como uma das mais promissoras e desafiadoras das vertentes da ciência da computação, o ramo da Inteligência Artificial. Segundo Ganascia citado por Fernandes (2005), filósofos e cientistas, já há muito tempo, se dedicaram a análise de vários aspectos cognitivos da inteligência humana. Hoje, como todas as outras ciências que antigamente pertenciam ao campo da filosofia e acabaram se tornando



independentes, a inteligência passou a ser objeto de estudo da ciência chamada de Inteligência Artificial (IA). Bittencourt (2006) cita que a IA se tornou um ramo da Ciência da Computação, oficialmente, em 1956.

Inteligência vem do latim *inter* que significa entre, e *legere* que significa escolher. Inteligência é aquilo que permite ao homem escolher entre duas opções ou ainda a habilidade de realizar de forma eficiente uma tarefa. Já artificial provém do latim *artificiale*, que significa algo não natural (FERNANDES 2005). Dessa forma, Inteligência Artificial é uma forma de fazer uma máquina escolher a melhor opção entre várias apresentadas a ela.

Boose citado por Fernandes (2005) caracterizou IA como multidisciplinar e interdisciplinar, onde ela não se atém somente a sua área específica, mas é totalmente dependente e abrangente de outras áreas do conhecimento.

Bittencourt (2006) descreveu como objetivo da IA a criação de modelos para a inteligência e a construção de sistemas computacionais baseados nesses modelos.

[...] a Inteligência Artificial busca entender a mente humana e imitar seu comportamento, levantando questões tais como: Como ocorre o pensar? Como o homem extrai o conhecimento do mundo? Como a memória, os sentidos e a linguagem ajudam no desenvolvimento da inteligência? Como surgem as ideias? Como a mente processa as informações e tira conclusões decidindo uma coisa ao invés da outra? Essas são algumas perguntas que a IA precisa responder para simular o raciocínio humano e implementar aspectos da inteligências (FERNANDES apud BOOSE, 2005).

## 2.3 Java

O software para a geração de horários foi desenvolvido utilizando-se a Linguagem Java. Portanto, foi necessário aprender a utilizar a referida linguagem.

Java é uma linguagem de programação e uma plataforma de computação lançada pela Sun Microsystems em 1995[...] Java foi projetado para permitir o desenvolvimento de aplicações portáteis de alto desempenho para a mais ampla variedade possível de plataformas de computação (OBTENHA INFORMAÇÕES SOBRE A TECNOLOGIA JAVA,2 013)

Java é executada pela JVM - Java Virtual Machine, sendo ela a responsável pelo grande poder de integração que tem a linguagem Java. Pois tendo a JVM instalada em qualquer computador é possível rodar um software feito em Java,

indiferente se este computador possui Linux, Windows, MAC OS, Mobile ou outros sistemas operacionais (DEITEL, 2008).

### 2.3.1 Uma breve história sobre a Linguagem Java

Java foi desenvolvida na década de noventa, mais precisamente em 1991 pela empresa Sun Microsystems. Tinha como objetivo inicial possibilitar a convergência entre computadores, equipamentos eletrônicos e eletrodomésticos.

Na época a linguagem era chamada de Oak (em português, “carvalho”), pois era essa árvore que James Gosling avistava da janela da sede do projeto [...] O resultado do projeto foi um controle remoto chamado \*7 (*StarSeven*), que tinha uma interface gráfica sensível ao toque e era capaz de interagir como diversos equipamentos. A linguagem *Oak*, na época, surgiu justamente para controlar internamente esse equipamento. [...] O controle remoto em si não vingou: ele estava muito a frente de seu tempo. Na época, as empresas de TV a cabo e vídeo por demanda não tinham condições de viabilizar o negócio [...] Depois disso, James Gosling foi encarregado de adaptar a linguagem para a internet, surgindo em 1995, assim, a plataforma Java [...] Uma das principais diferenças entre a plataforma Java e as demais linguagens existentes na época é que o Java é executado sobre JVM, ou *Java Virtual Machine*. Qualquer plataforma de hardware ou equipamento eletrônico que possa executar a máquina virtual conseguirá executar Java. Isso justifica o slogan “*write once, run anywhere*”, ou, em português, “escreva uma vez, rode em qualquer lugar” (LUCKOW; MELO, 2010).

Java possui muitas plataformas, mas possui três principais que são: Java SE, Java EE e Java ME.

- a) Java SE<sup>2</sup> é a plataforma básica de desenvolvimento para Java Desktop;
- b) Java EE<sup>3</sup> é a que permite o desenvolvimento para web;
- c) Java ME<sup>4</sup>, desenvolvimento de aplicações para dispositivos móveis.

## 2.4 Hibernate

Segundo Luckow e Melo (2010), *Hibernate* é um *Framework* baseado nas técnicas de *Object Relational Mapping*<sup>5</sup> de persistência de dados que tem como

<sup>2</sup> Java Platform Standard Edition

<sup>3</sup> Java Platform Enterprise Edition

<sup>4</sup> Java Platform Micro Edition

finalidade armazenar objetos Java em bases de dados relacionais, fornecendo uma visão orientada a objetos de dados relacionais existentes. De maneira direta, pode-se dizer que o *Hibernate* é uma ferramenta automatizada e transparente de persistir objetos que pertençam a uma aplicação nas respectivas tabelas em um banco relacional, automatizando os processos repetitivos referentes à manipulação de dados.

A principal característica do *Hibernate* é a possibilidade de representar as tabelas do banco de dados relacional através de classes, obtendo assim o registro de cada tabela sendo representado por instâncias de sua respectiva classe. O *Hibernate* gera as chamadas SQL (*Structured Query Language*) e libera o desenvolvedor do trabalho manual da conversão dos dados resultante, mantendo o programa possível de ser portátil para quaisquer bancos de dados SQL. Porém, essa conversão de dados causa um pequeno aumento no tempo de execução se comparado com os modelos de banco de dados relacionais (LUCKOW; MELO, 2010).

Através do uso de arquivos de configuração XML, o framework faz o mapeamento dos dados contidos nas colunas de uma tabela em uma base de dados relacional para os atributos de uma classe Java. A Figura 1 ilustra o funcionamento do *framework* no mapeamento dos objetos da aplicação para as tabelas de uma base de dados.

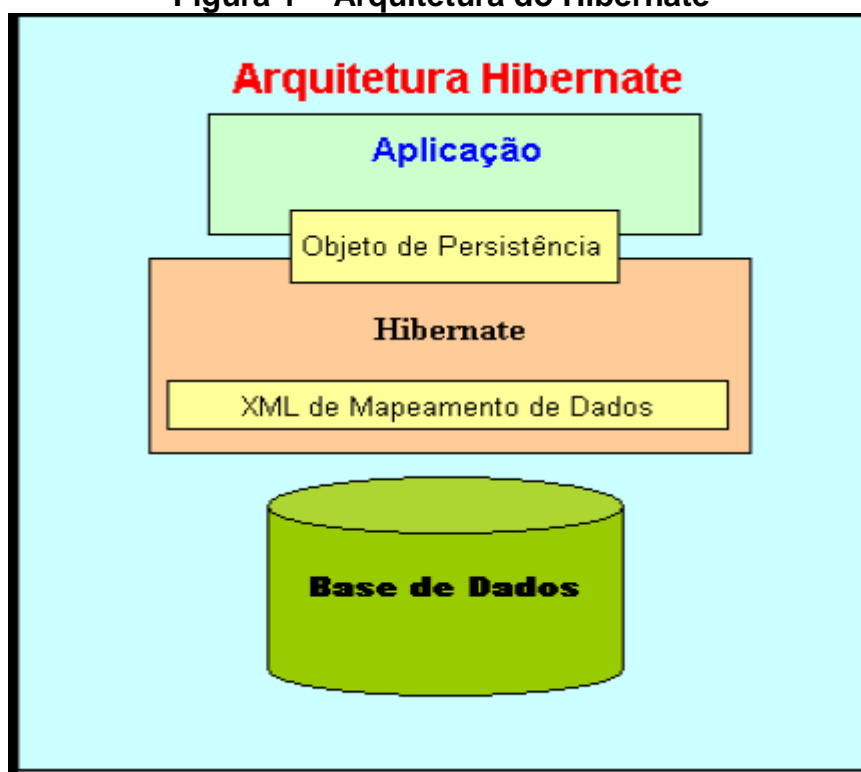
O objeto inserido no banco de dados é passado para o *Hibernate*, que por sua vez valida a estrutura desse objeto através do arquivo XML de mapeamento de dados. Se a estrutura de dados do objeto e da tabela na base de dados for compatível, o *Hibernate* inicia uma transação na base de dados e faz a persistência das informações.

Pelo fato do *Hibernate* ser um *framework* baseado em orientação a objetos, é obtida uma aplicação totalmente feita com orientação a objetos com o uso de suas ferramentas, ficando assim livre de trabalhar diretamente com a base de dados. Segundo Luckow e Melo (2010) a abstração de tabelas de banco de dados por classes torna o desenvolvimento mais simples e intuitivo.

---

<sup>5</sup> Técnica de desenvolvimento orientada a objetos para construção de bancos de dados relacionais.

Figura 1 – Arquitetura do Hibernate



Fonte: Luckow, Melo, 2010

## 2.5 XML

O XML é a abreviação de *eXtensible Markup Language*. Segundo Holzner (2001), trata-se de uma linguagem de marcação para a criação de documentos com dados organizados hierarquicamente, tais como textos, banco de dados ou desenhos vetoriais. Esta linguagem é uma especificação técnica desenvolvida pela W3C (*World Wide Web Consortium*) que é uma organização responsável pela padronização da Internet. O objetivo do XML é superar as limitações do HTML, que é o padrão das páginas da Web. A linguagem XML é definida como o formato universal para dados estruturados na Web. Esses dados estão compreendidos em tabelas, desenhos, parâmetros de configuração e exportação de dados.

A utilização do XML é parte crucial para o funcionamento do *framework Hibernate*, pois somente através de arquivos de configuração no padrão XML que o *framework* pode fazer todo o trabalho de mapeamento de tabelas, colunas, restrições, índices, relacionamentos das bases de dados para os objetos Java. É por meio desses arquivos de configuração que a persistência desses objetos é feita de maneira correta nas bases de dados relacionais. (LUCKOW; MELO, 2010).

## 2.6 Framework

Segundo Fayad e Schmidt (1997), *frameworks* representam um conjunto de classes criadas para resolver um problema específico na área de *software*. Um *framework* não se trata de um *software* executável, mas sim um modelo de dados com funcionalidades para várias aplicações. O conjunto de classes de um *framework* tem como objetivo a reutilização de funcionalidades já desenvolvidas, trazendo um modelo de arquitetura em um domínio específico de *software*. Essas classes são implementadas em uma linguagem de programação específica e são usadas para auxiliar o desenvolvimento de *software*.

Em um projeto de *software*, o uso de um *framework* pode incluir programas de apoio, bibliotecas de código, linguagens de script e outros *softwares* para ajudar a desenvolver e juntar diferentes componentes do seu projeto (FAYAD; SCHMIDT, 1997).

Soares Filho (2006) explica em seu trabalho que:

Um framework se diferencia de uma simples biblioteca de funcionalidades para software, pois esta se concentra apenas em oferecer implementação de funcionalidades, sem definir a reutilização de uma solução de arquitetura. Já o framework pode atingir uma funcionalidade específica, por configuração, durante a programação de uma aplicação. (p.17).

Com o uso de um *framework* é possível diminuir as dificuldades encontradas na fase de desenvolvimento de um *software*, pois o mesmo oferece um mecanismo para reutilizar soluções prontas na resolução de problemas que surgem da construção de aplicações com requisitos em comum.

### 3 REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta os trabalhos de alguns autores que abordam a temática proposta. São abordados os trabalhos de Freitas *et. al.* (2007); Carvalho (2011) e Fernandes, Pereira e Freitas (2002).

Freitas e outros (2007) apresentam, em seu artigo, uma ferramenta que foi construída para satisfazer as necessidades da Faculdade Ruy Barbosa (FRB) – Salvador – BA – Brasil. Nesse artigo, os autores salientam que é mais fácil verificar o resultado apresentado e, depois, modificar o mesmo, do que conseguir solução ótima já na primeira tentativa, pois o processo de alocação de horário escolar é um problema de otimização combinatória da classe dos NP-Difíceis, e essa solução em tempo polinomial seria inviável.

No desenvolvimento da ferramenta, implementada com base em algoritmos genéticos, obteve-se, após várias gerações de uma população, um resultado quase sem conflitos de horários. Mesmo com a ferramenta não tendo alcançado um resultado ótimo, a solução alcançada permitiu a realização de ajustes manuais que possibilitaram a definição de uma solução aceitável e utilizável.

Carvalho (2011) explica que a utilização de métodos de otimização, como algoritmos determinísticos, pode garantir uma solução ótima para problemas combinatórios. Porém, quando o problema é de um porte maior, a utilização de técnicas otimizadas pode gerar uma solução inviável. Devido a essas razões, Carvalho (2011) resolveu utilizar métodos heurísticos como forma de resolução destes problemas, pois tais métodos são capazes de gerar uma solução boa independente da natureza do problema. O método utilizado foi *Iterated Local Search* (ILS), que consiste em explorar o espaço de soluções por meio de uma sequência de perturbações a uma solução, seguidas de buscas locais.

Fernandes, Pereira e Freitas (2002) apresentam um algoritmo evolutivo híbrido, ilustrado no Quadro 2, para resolver o problema de programação de horários em escolas. O algoritmo híbrido proposto é um método evolutivo na qual a população inicial é gerada por um procedimento construtivo e, em cada geração, o melhor indivíduo é submetido a um refinamento pelo método de Busca Tabu<sup>6</sup>.

---

<sup>6</sup> Algoritmo baseado em procedimentos heurísticos que permitem explorar o espaço de busca e encontrar novas soluções além daquelas encontradas em uma busca local.

### Quadro 2: Algoritmo Evolutivo Híbrido

```

procedimento Híbrido( );
1. Inicializar a população;
2.  $t \leftarrow 0$ ;
3. Gerar uma população inicial  $P(t)$ ;
4. Avaliar  $P(t)$ ;
5. enquanto “os critérios de parada não forem satisfeitos” faça
    6. Cruzar selecionados ;
    7. Mutar Resultantes;
    8. Avaliar  $P(t)$ ;
    9. Refinar o melhor indivíduo de  $P(t)$  usando Busca Tabu;
    10. Selecionar  $P(t+1)$  à partir de  $P(t)$ ;
    11.  $t \leftarrow t+1$ ;
12. fim-enquanto;
13. Retornar melhor indivíduo;
fim Híbrido;
  
```

Fonte: Fernandes, Pereira e Freitas 2002.

No trabalho de Kotsko, Machado e Santos (2005), intitulado “Otimização na alocação de professores na construção de uma grade de horários escolares”, é tratada a construção otimizada de horários escolares de turmas em escolas de ensino fundamental e médio, utilizando técnicas da Pesquisa Operacional.

Na construção do modelo são utilizadas restrições correspondentes a exigências administrativas como: máximo de duas aulas diárias por professor em uma mesma turma, aulas vagas dos professores preferencialmente as primeiras e/ou últimas, disponibilidades dos professores quanto a dias da semana; preferências de três professores por atuarem em três dias quaisquer dos cinco dias da semana e restrições para assegurar uma aula por turma e uma aula por professor em um mesmo horário (KOTSKO, MACHADO, SANTOS, 2005).

No trabalho de Kotsko, Machado e Santos (2005), para se organizar aulas aos 21 professores de maneira que em doze turmas cada uma tenha 25 aulas semanais, distribuídas em 5 aulas diárias em um único turno, para disciplinas e número de aulas previamente atribuídas aos respectivos professores, a resolução do problema foi dividida em etapas, sendo elas: 1º - Identificação de dados e variáveis do problema; 2º - Identificação das restrições comuns a modelos de designação e peculiares ao problema sendo resolvido; 3º - Modelagem matemática do Problema de Programação Linear Binária (PPLB) e implementação computacional; 4ª - Solução com o Lingo e adequação da solução à prática usual.

No resultado final, apresentado no trabalho de Gomes, Fonseca e Santos (2005), concluiu-se que a construção de grade horária mostrou-se viável, assim podendo ser aplicada a outras escolas, desde que se disponha do *software* Lingo.



## 4 METODOLOGIA E PROCEDIMENTOS

Segundo Freitas e Prodanov (2013), metodologia é a linha de raciocínio adotada no processo de pesquisa, ou o conjunto de processos ou operações mentais que se deve empregar na investigação.

Este capítulo pretende transparecer todos os processos utilizados para o desenvolvimento deste trabalho, demonstrando a linha de raciocínio empregada para chegar aos objetivos propostos.

### 4.1 Delimitação do esboço

Delimitou-se que o trabalho fosse focado apenas ao curso de Sistemas de Informação, pelo grau de complexidade que se encontra o modelo do problema de *timetabling* no campus IFMG-SJE e, principalmente, pelo curto espaço de tempo estipulado para a conclusão deste trabalho.

O grau de complexidade encontrado no estudo de caso baseado no IFMG-SJE é justificado abaixo:

- a) Primeiramente, o campus IFMG-SJE possui vários prédios em diferentes locais com diferentes cursos, não especificamente cada curso em apenas um prédio, o que dificultaria a criação de um modelo inicial.
- b) Além disso, no IFMG-SJE são ofertados cursos na modalidade de ensino superior, ensino pós-médio e ensino técnico integrado, que é o ensino médio juntamente com o técnico. Com essas características, pode-se deduzir que a modelagem do problema de horários do IFMG-SJE é uma combinação de duas categorias de *timetabling*, a *school timetabling* e a *course timetabling*, o que torna a definição de um modelo com uma complexidade ainda maior.

Após o levantamento dos requisitos e análise das necessidades específicas, delimitou-se o escopo do problema com a finalidade de enquadrá-lo ao curto prazo de entrega do presente trabalho. Assim, a versão inicial do *software* atenderá a um modelo baseado no curso de Bacharelado de Sistemas de Informação com apenas um único turno.

No modelo seguido, tem-se uma estrutura única que não se preocupa com o deslocamento dos alunos e professores entre prédios distintos. É possível cadastrar quantas salas quiser, mas não serão considerados objetos distintos as salas de aulas e os laboratórios. As aulas serão ministradas em apenas um turno com o número de seis horários diários. As aulas serão ministradas em cinco dias da semana, ou seja, de segunda a sexta.

O *software* suporta o cadastro de vários cursos, porém, como ele foi projetado e desenvolvido com base nas características do curso de SI do IFMG-SJE, ele é mais indicado para esse contexto. Cada disciplina será de apenas um único curso, e cada turma também será de apenas um único curso. Este modelo não suportará as disciplinas optativas devido às mesmas serem ofertadas em outro turno e em conjunto com outros cursos. Devido à necessidade de toda turma conter um determinado número de dias letivos, cada turma deve conter no mínimo duas aulas por dia.

Como observado no decorrer do curso SI, durante a graduação, como hipótese por considerar 45 minutos pouco tempo para uma boa sequência do conteúdo programático, sempre é marcado mais de um horário seguido no mesmo dia para a mesma disciplina.

Sempre serão alocadas duas ou três aulas seguidas, de acordo com os créditos de cada disciplina. Exemplo: A disciplina de Algoritmos e Estruturas de Dados I que contém 6 créditos terá dois dias de aula, sendo três aulas seguidas por dia. Já disciplina de Cálculo Diferencial e Integral I, que contém 4 créditos terá dois dias de aula, sendo 2 aulas seguidas.

## 4.2 Java Web

Como linguagem utilizada no desenvolvimento, Java *Web* é considerada como a que possui a maior curva de aprendizagem na atualidade, mas mesmo assim é uma linguagem robusta com inúmeras APIs<sup>7</sup> que amenizam sua complexidade na implementação. Além de APIs, são disponibilizados inúmeros *frameworks* gratuitos e pagos que tornam as aplicações mais seguras, padronizadas

---

<sup>7</sup> API, de *Application Programming Interface* (ou Interface de Programação de Aplicativos) é um conjunto de rotinas e padrões estabelecidos por um *software* para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da implementação do *software*, mas apenas usar seus serviços.

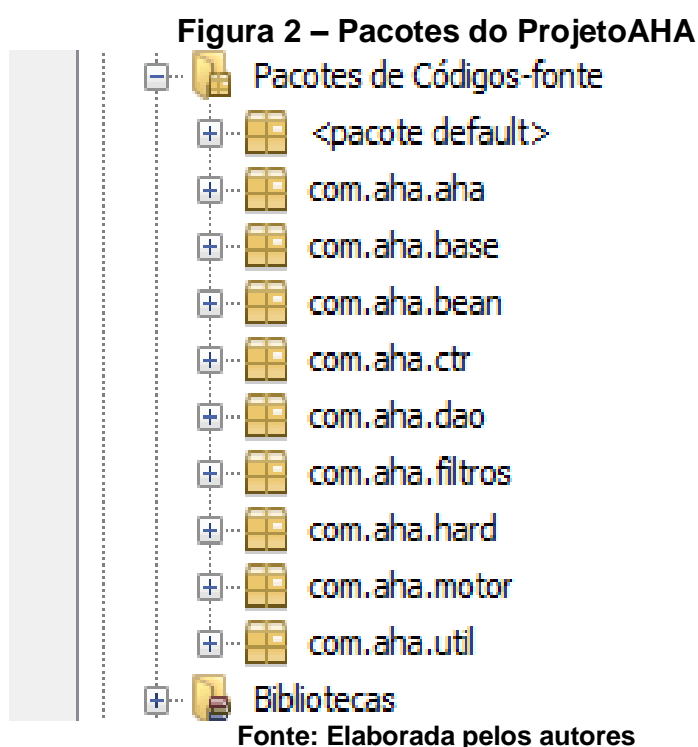
e de fácil interatividade com o usuário. Além disso, a linguagem encontra-se em franco desenvolvimento e aperfeiçoamento, conta com vários fóruns e comunidades, o que a torna uma linguagem ainda mais robusta e confiável.

### 4.3 Padrão MVC

Segundo Luckow e Melo (2010), MVC (*Model – View – Controller*) é um padrão de arquitetura muito utilizada no desenvolvimento web, ele determina que um sistema seja separado em três camadas chamadas *Model*, *View* e *Controller*.

Seu principal objetivo é definir como as camadas devem interagir, e não como as camadas devem se separar. Sua principal característica que a camada *Model* não pode conhecer a camada *View* e a camada *View* não pode conhecer a camada *Controller*.

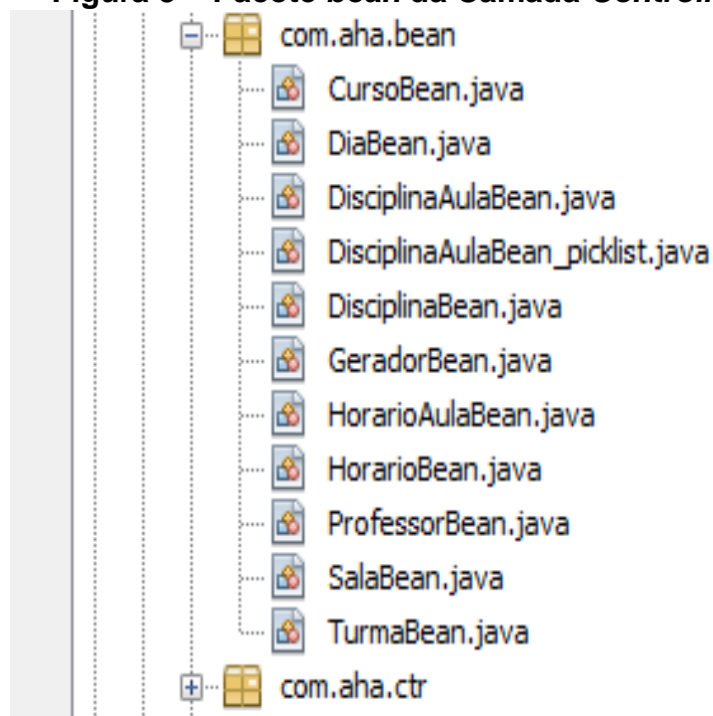
Na Figura 2 é apresentada todos os pacotes de códigos de desenvolvimento do ProjetoAHA.



Na Figura 3, é apresentado o pacote *bean* e suas respectivas classes. Todas as classes contidas nesse pacote são a representação da camada de *Controller* do

padrão MVC. Sua função é buscar informações da camada *Model* para gerar a *View* e por receber as informações da *View* e enviar para a *Model*.

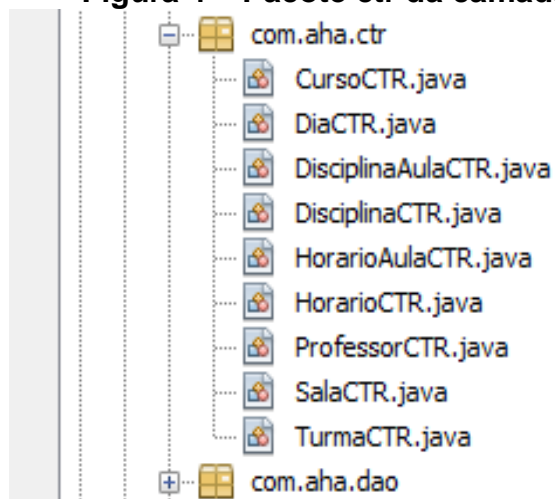
**Figura 3 – Pacote *bean* da Camada *Controller***



Fonte: Elaborada pelos autores

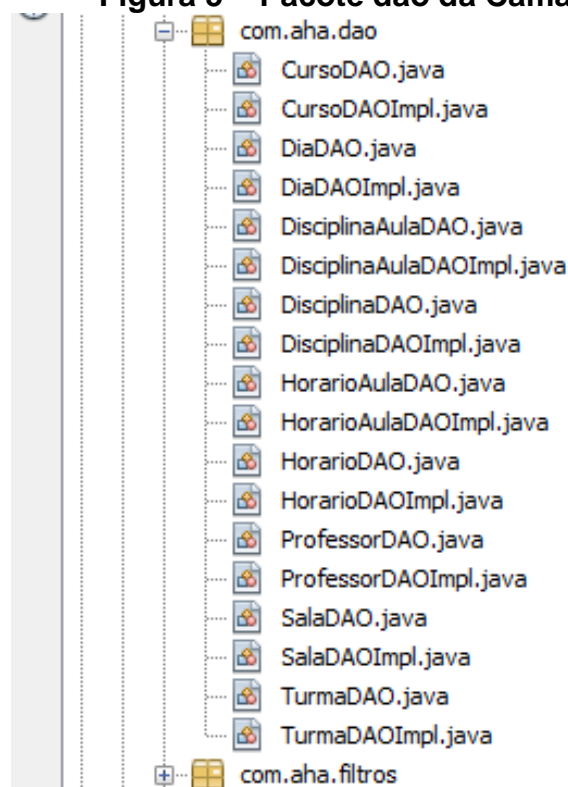
Na Figura 4 e 5 são apresentados os pacotes que representam a camada *Model* do modelo MVC. Essa camada *Model* é responsável pelo acesso a dados e regra de negócio. As classes do pacote ctr cuidam das regras de negócio e as classes do pacote dao cuidam do acesso a dados do sistema.

**Figura 4 – Pacote *ctr* da camada *Model***



Fonte: Elaborada pelos autores

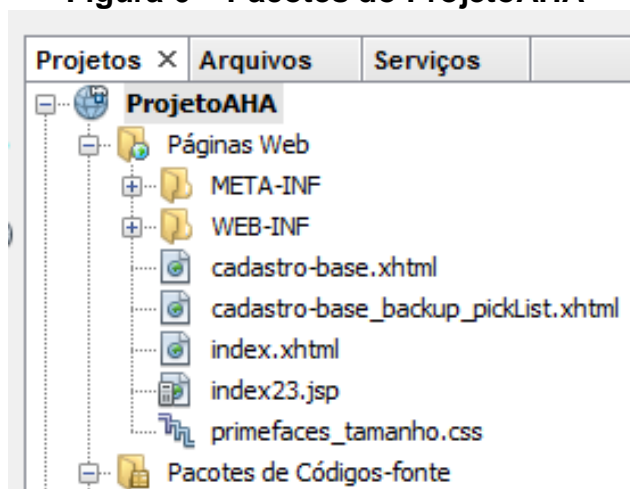
**Figura 5 – Pacote dao da Camada *Model***



Fonte: Elaborada pelos autores

Por ultimo, temos as páginas XHTML<sup>8</sup>, ilustradas pela Figura 6, que contém os códigos JSF. Essas páginas representam a *View* da camada MVC. Esta camada compõe toda interface do sistema. Mais especificamente, é na *View* que é exibida toda a informação gerada pela camada *Model*.

**Figura 6 – Pacotes do ProjetoAHA**



Fonte: Elaborada pelos autores

<sup>8</sup> O XHTML, ou *eXtensible Hypertext Markup Language*, é, antes de tudo um arquivo XML. Diversas melhorias na estrutura do HTML foram feitas na criação XHTML. (LUCKOW; MELO, 2010)

Com a utilização do padrão MVC foi possível facilitar o reaproveitamento de código, como também aumentar a manutenibilidade<sup>9</sup> e portabilidade para adição de novos recursos ao sistema.

#### 4.4 Ferramentas utilizadas

Esta seção descreve como foi preparado o ambiente de desenvolvimento utilizado no software AHA (Alocação de Horários Acadêmicos). Esse ambiente é composto pelas ferramentas: *Java*, *Apache Tomcat*, *NetBeans IDE*, *MySQL* e *MySQL Workbench*, *Hibernate* e *JSF*.

Os critérios que mais contribuíram na escolha de tais ferramentas foram a experiência dos autores no uso das ferramentas e o fato de que as ferramentas são disponibilizadas gratuitamente, além de existirem comunidades altamente atuantes na evolução e documentação dessas tecnologias.

A seguir, demonstra-se como se realizou a instalação e configuração de cada uma das ferramentas em um ambiente *Windows*.

##### 4.4.1 Instalação do Java

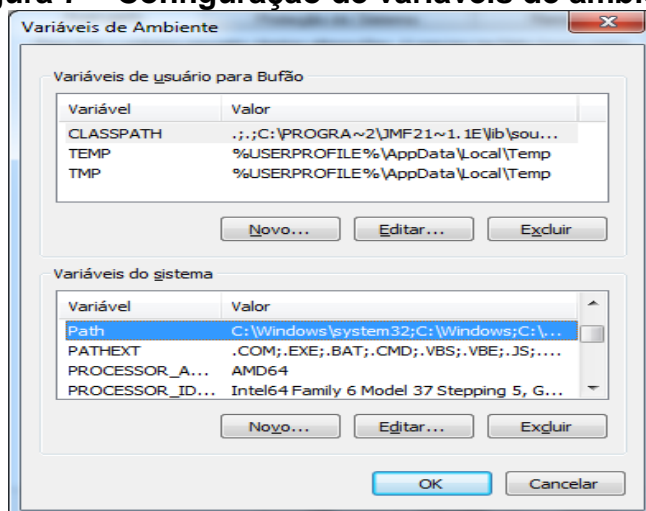
A versão do *Java* utilizada foi a 7. O arquivo para instalação do *Java* foi obtido em <http://www.oracle.com/technetwork/java/javase/downloads/index.html>, onde se escolheu o pacote JDK (*Java Development Kit*), o qual é necessário para o desenvolvimento de aplicativos *Java*.

Após a instalação do JDK, fez-se a configuração das variáveis de ambiente do *Windows* que definem o local de instalação do *Java*. Essas configurações realizaram-se pelo fato de programas como *Apache Tomcat* precisarem saber qual é o diretório onde o *Java* foi instalado. Para mudar as configuração de variáveis de ambiente é necessário acessar o menu “Variáveis de Ambiente” conforme a Figura 7: Criou-se uma variável com o nome *JAVA\_HOME* com o caminho onde o *Java* foi instalado no diretório “*C:\Arquivos de programa\Java\jdk1.7*”, e alterado o valor da variável *Path* acrescentando o texto “*%JAVA\_HOME%\bin*” ao final do caminho da variável.

---

<sup>9</sup> Manutenibilidade é um termo usado para definir que certo Software ou Hardware tem uma certa facilidade de ser mantido através de manutenções.

**Figura 7 – Configuração de variáveis de ambiente**

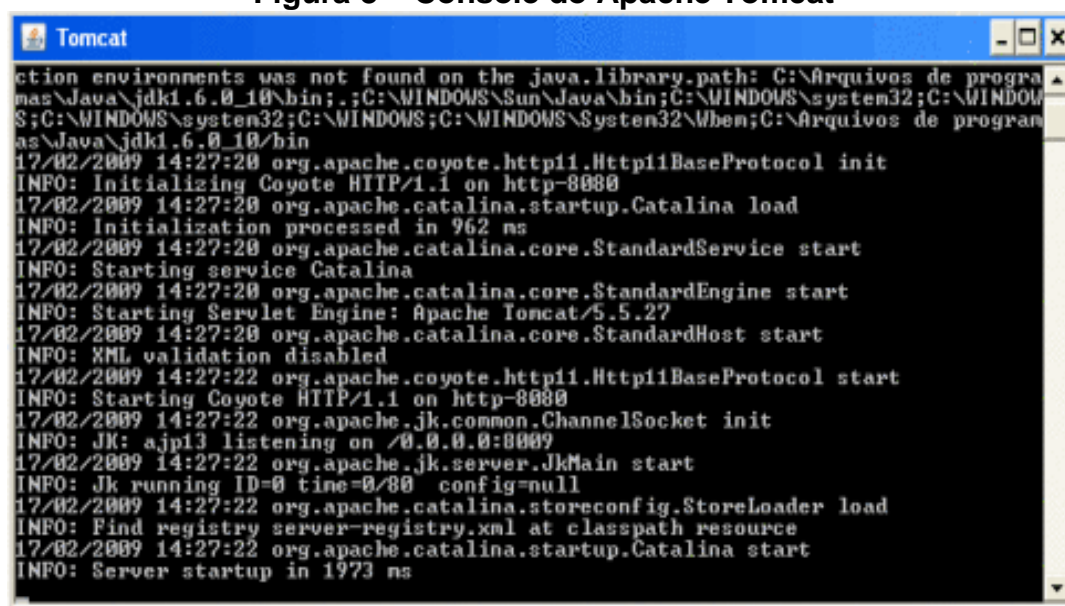


Fonte: Elaborado pelos autores

#### 4.4.2 Instalação do Apache Tomcat

O *Apache Tomcat* é um servidor *web Java*. Ele suporta a execução das tecnologias *Java Servlet*<sup>10</sup> e *JavaServer Pages*<sup>11</sup>, o que permite que o *Java* funcione em um ambiente *web*.

**Figura 8 – Console do Apache Tomcat**



Fonte: Elaborado pelos autores

<sup>10</sup> Classe Java usada para estender as funcionalidades das aplicações hospedadas em um servidor web. (LUCKOW; MELO, 2010)

<sup>11</sup> Tecnologia utilizada para criar páginas web de forma dinâmica baseadas em HTML, XML e outros tipos de documentos. (LUCKOW; MELO, 2010)

A versão do *Apache Tomcat* utilizada foi a 6, e seu pacote de instalação pode ser encontrado no site <http://tomcat.apache.org/download-60.cgi>. O pacote de instalação escolhido foi o *Windows Service Installer*, que permite a instalação do *Apache Tomcat* por meio de um assistente no *Windows*.

Na instalação do *Apache Tomcat* é necessário criar uma nova variável de ambiente com o nome de “*CATALINA\_HOME*”, tendo como valor o caminho de instalação do *Apache Tomcat* “*c:\apache-tomcat-6.0.26*”. Após a instalação foi realizado um teste para certificar se todo processo de instalação ocorreu normalmente através da execução do arquivo “*startup.bat*”, que se encontra dentro da pasta *bin* do *Apache Tomcat*, conforme a Figura 8.

Para executar a página inicial padrão do *Apache Tomcat* é preciso abrir o navegador de *internet* e digitar o endereço: <http://localhost:8080/ProjetoAHA>.

#### **4.4.3 Instalação do MySQL e MySQL Workbench**

*MySQL* é o sistema de gerenciamento de banco de dados (SGBD) de código-fonte aberto mais popular do mundo, tendo mais de 70 milhões de usuários em todo mundo. É utilizado por empresas como Amazon.com, Google, Motorola, MP3.com, NASA, Yahoo e outras. É um banco de dados que apresenta um excelente desempenho e fácil manuseio (LUCKOW; MELO, 2010).

O download do arquivo de instalação do *MySQL* fez-se através do site <http://dev.mysql.com/downloads/mysql>, onde foi escolhida a versão *Mysql Community Server*. Além do próprio *MySQL* como SGBD, também utilizou-se o *MySQL Workbench*. Essa ferramenta facilitou bastante à administração do banco de dados, permitindo criar todas as rotinas de *backup*, consultas e atualizações da base de dados de maneira gráfica. Outro recurso utilizado do *MySQL Workbench* foi o *Database Modeling*, que permitiu construir o desenho e modelagem do banco de dados através de diagramas de entidade e relacionamento. O *MySQL Workbench* foi obtido no site <http://www.mysql.com/downloads/workbench>.

#### **4.4.4 Instalação do NetBeans**

Para a criação do código, utilizou-se a IDE *NetBeans* 7.3. A utilização dessa ferramenta foi necessária para realizar a edição do código fonte da aplicação, no



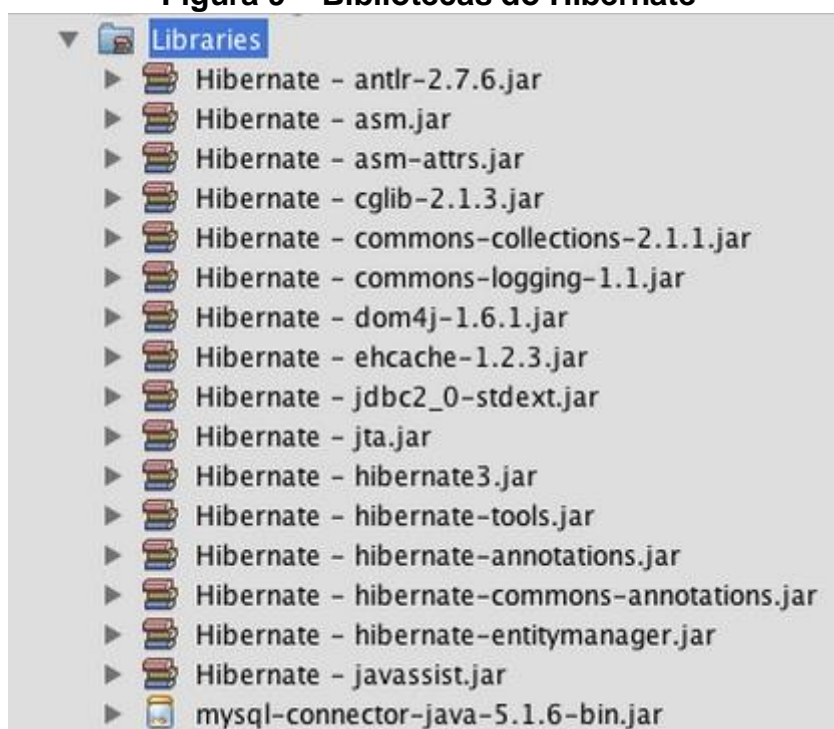
qual se obteve um desenvolvimento rápido do *software*. O *NetBeans 7.3* oferece suporte às mais recentes tecnologias utilizadas nos projetos *Java Web*, possibilitando assim integrar todas as ferramentas usadas no *software* AHA em um mesmo ambiente de desenvolvimento.

O *NetBeans* é uma ferramenta *Open Source* que pode ser encontrada para *download* no site <https://netbeans.org/downloads>.

#### 4.4.5 Instalação e configuração do Hibernate no NetBeans

Para começar a usar o *Hibernate*, foi necessário acessar o site <http://www.hibernate.org> e fazer o *download* da biblioteca complementar do *framework*. Nesta implementação utilizou-se o *Hibernate 3*. Após o *download* do *Hibernate*, foi necessário copiar as suas bibliotecas para dentro da pasta *libraries* no projeto *web* do *NetBeans 7.3*, conforme a Figura 9.

**Figura 9 – Bibliotecas do Hibernate**



Fonte: Elaborado pelos autores

O próximo passo realizado foi criar o arquivo XML de configuração do *Hibernate*, “*hibernate.cfg.xml*”, importado para dentro da raiz do projeto, conforme visto na Figura 10.

O arquivo XML de configuração contém informações sobre a conexão do banco de dados, propriedades de inicialização e os caminhos dos arquivos de mapeamento.

**Figura 10 – Arquivo *hibernate.cfg.xml* de configuração**



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.dialect">org.hibernate.dialect.DerbyDialect</property>
    <property name="hibernate.connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
    <property name="hibernate.connection.url">jdbc:derby://localhost:1527/sample</property>
    <property name="hibernate.connection.username">app</property>
    <property name="hibernate.connection.password">app</property>
  </session-factory>
</hibernate-configuration>
```

Fonte: Elaborado pelos autores

#### 4.4.6 Instalação do JSF

O *JavaServer Faces* (JSF) é a especificação para um *framework* de componentes para desenvolvimento *Web* em Java. É uma tecnologia que permite a utilização de componentes visuais sem a necessidade de se preocupar com os elementos HTML da página *Web*. Basta apenas adicionar componentes JSF na página *Web* que os mesmos serão renderizados e exibidos em formato HTML.

Existem diversas implementações do JSF, porém a utilizada no projeto do *software* AHA é a conhecida por *Mojarra*, que pode ser obtida através do *site* <https://javaserverfaces.dev.java.net>. No *site* são disponibilizados os arquivos “*jsf-api.jar*” e “*jsf-impl.jar*” de implementação do JSF que devem ser importados para a pasta da biblioteca do projeto no *NetBeans*.

Após a importação das bibliotecas, é necessário configurar o arquivo “*web.xml*”, localizado dentro da pasta “*WEB-INF*” gerada pelo *NetBeans*, conforme a Figura 11. Nesse arquivo, o servidor *Apache Tomcat* é configurado para reconhecer e repassar uma requisição para que o JSF faça o processamento.

Mesmo utilizando os componentes JSF para inserir elementos visuais na aplicação, ainda houve a necessidade de utilizar outra ferramenta para componentes visuais, pois o JSF apenas disponibiliza componentes básicos de HTML como *inputs*, botões e *CombBox*.

Para trazer componentes mais sofisticadas a aplicação utilizou-se o *framework Primefaces*. O *Primefaces* é uma biblioteca de componentes para serem usados juntamente ao JSF, permitindo trazer uma gama maior de componentes visuais para a aplicação.

**Figura 11 – Configuração do arquivo *web.xml***

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.5"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <context-param>
    <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
    <param-value>.xhtml</param-value>
  </context-param>
  <context-param>
    <param-name>facelets.DEVELOPMENT</param-name>
    <param-value>true</param-value>
  </context-param>
  <context-param>
    <param-name>com.sun.faces.validateXml</param-name>
    <param-value>true</param-value>
  </context-param>
  <context-param>
    <param-name>com.sun.faces.verifyObjects</param-name>
    <param-value>true</param-value>
  </context-param>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jsp</url-pattern>
  </servlet-mapping>
</web-app>
```

Fonte: Elaborada pelos autores

Esta biblioteca não substituiu a implementação do JSF, ou seja, foi necessário ter as bibliotecas do JSF importadas no projeto web antes de usar o *Primefaces*. Para o uso do *Primefaces* foi necessário baixar seus arquivos no site <http://primefaces.org/downloads.html> e importá-los para a biblioteca no *Netbeans*.

A única configuração necessária para utilizar os componentes do *Primefaces* fez-se através do arquivo “*index.xhtml*” visto na Figura 12.

**Figura 12 – Declaração do *Primefaces***

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:p="http://primefaces.org/ui">
  <h:head>
    <title>Primefaces</title>
  </h:head>
  <h:body>

  </h:body>
</html>
```

Fonte: Elaborada pelos autores

#### 4.5 Função do software AHA

A função do software AHA se resume em distribuir aulas das disciplinas das turmas de cada curso em horários disponíveis durante os dias letivos da semana sem confrontar os horários de professores, turmas e salas de aulas. Primeiramente, definiram-se quais são os horários disponíveis e também quais serão as disciplinas juntamente com seus créditos as serem distribuídos.

Levando em conta que um prédio pode conter um determinado número de salas, em cada sala é possível ter uma determinada quantidade de horários por dia e sendo que pode ter até no máximo 7 dias de aulas por semana. Chama-se de horários disponíveis todas as combinações de horários possíveis dentro dos dias, salas, e horários diário. Para saber a quantidade de Horários Disponíveis consiste em multiplicar o número de salas, horários e dias. Na Fórmula 1, tem-se que:

**Fórmula 1 – Horários Disponíveis**

$$\Rightarrow k = d \times s \times h$$

Fonte: Elaborada pelos autores

- a)  $k$  - são os horários disponíveis durante a semana;
- b)  $d$  - são os dias da semana que irão ter aulas;
- c)  $s$  - são as salas que estão à disposição para aulas;
- d)  $h$  - são os horários por dia do turno em questão.

Logo, percebe-se, que  $k$  é a quantidade de horários disponíveis nos quais se pode alocar as aulas no semestre em questão. Pode-se, portanto, afirmar que a quantidade de aulas não pode ser superior a  $k$ , o que exige que a quantidade de aulas seja conhecida para que seja possível verificar se ela pode ou não ser alocada nos horários disponíveis.

Como no IFMG-SJE não há vestibular no meio do ano, em cada semestre são oferecidas as disciplinas de períodos sortidos de cada curso. No entanto, em cada semestre, deve-se escolher os períodos de cada turma para que se possa definir as “Aulas das Disciplinas”. A quantidade de aulas a serem alocada, representada pela Fórmula 2, é o resultado da somatória da carga horária de cada disciplina dos períodos correntes de cada turma de cada curso, sendo que cada curso pode ter mais de uma turma em o mesmo período.

**Fórmula 2 – Aulas da Disciplinas**

$$x = \sum_{c \in C} \sum_{p \in P} t_{pc} \sum_{d \in D} g_{dpc}$$

**Fonte: Elaborada pelos autores**

A seguir, são apresentadas as variáveis e seus respectivos conjuntos:

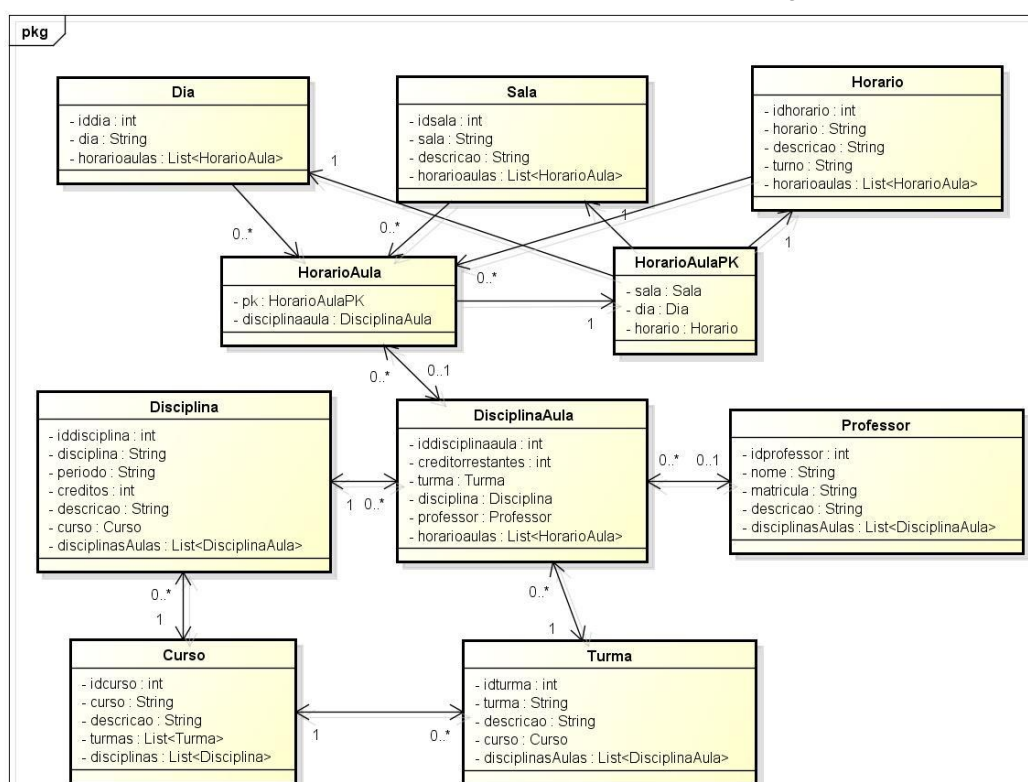
- a)  $t$  é a quantidade de turmas que cursa o período corrente do curso;
- b)  $P$  é o conjunto dos períodos de cada curso, sendo que  $p \in P$  e  $\forall p \exists t \neq 0$ ;
- c)  $g$  é a carga horária de cada disciplina;
- d)  $D$  é o conjunto de disciplinas do Cursos no período corrente, sendo que  $d \in D$  e  $\forall d \exists g \neq 0$ ;
- e)  $C$  é o conjunto de cursos que compõem a instituição, sendo que  $c \in C$  e  $\forall c \exists d \in D$ , além disso cada semestre pode possuir um  $t$  diferente no período corrente do curso em questão;

- f)  $x$  é o número de “Aulas da Disciplinas” que é nada mais que todas as aulas que serão ministradas por semana.

Logo, percebe-se que  $x$  é o somatório das aulas de cada disciplina de cada turma de cada curso. Assim, para que seja possível alocar um número  $x$  de aulas em um número  $k$  horários semanais, deve ser verdadeira a condição  $k \geq x$ .

No diagrama de classe, esboçado pela Figura 13, é mostrado as classes que formam a base da solução para o problema de Alocação de Horários Acadêmicos. Todas as classes do diagrama da Figura 13 representam suas respectivas tabelas no Banco de Dados, exceto a classe “HorarioAulaPK”. Classe esta que se faz necessária para que seja possível mapear o relacionamento muitos para muitos entre as Classes “Dia”, “Sala” e “Horario”.

**Figura 13 – Diagrama de Classe – Esboço Inicial**



Fonte: Elaborada pelos autores

Percebe-se, pelo diagrama, que o problema se resume em alocar objetos do tipo “DisciplinaAula” em objetos do tipo “HorarioAula”, ou seja, alocar Aulas das Disciplinas nos Horários Disponíveis. O *software* tem como função primordial

preparar duas lista, uma lista tipo “HorarioAula” e a outra tipo “DisciplinaAula”, e alocar todos os objetos tipo “DisciplinaAula” nos objetos da lista tipo “HorarioAula”.

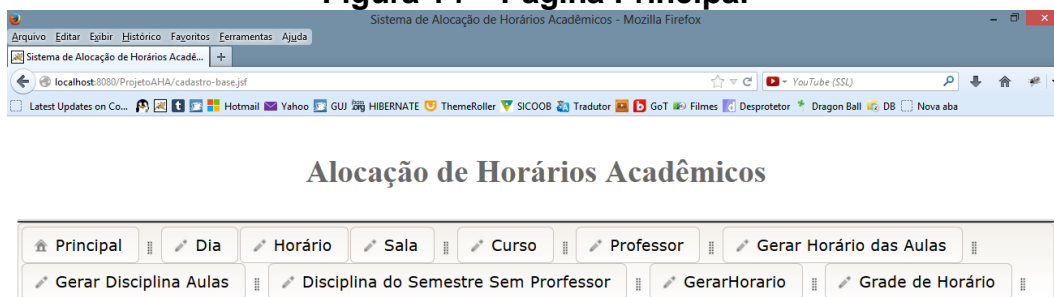
Pode-se observar pelo diagrama de classe a existência de um atributo “creditosrestantes”, ele representa a condição de que cada objeto tipo “DisciplinaAula” deve ser alocado várias vezes de acordo com o número de créditos de cada disciplina, sendo que a cada alocação deve decrementar o valor do atributo “creditosrestantes” até não restar mais créditos.

#### 4.6 Interface do CRUD<sup>12</sup>

A Interface do *software* AHA foi construída de maneira funcional, sem preocupação com as regras de *design* e usabilidade, pois não é o foco deste trabalho.

É apresentada na Figura 14 a página principal do sistema. Nota-se que ela possui apenas uma barra de botões, onde é possível encontrar todas as funcionalidades desta versão inicial do *software* AHA.

**Figura 14 – Página Principal**



**Fonte: Elaborada pelos autores**

Para a geração dos horários, fez-se necessário a criação de CRUDs para a alimentação das informações previamente necessárias. A Interface CRUD é utilizada

<sup>12</sup> CRUD (acrônimo de Create, Read, Update e Delete em língua Inglesa) para as quatro operações básicas utilizadas em bancos de dados relacionais (RDBMS) ou em interface para usuários para criação, consulta, atualização e destruição de dados.

para as quatro operações básicas utilizadas em bancos de dados relacionais ou em interface para usuários.

A Figura 15 mostra a tela de cadastro dos dias letivos da instituição, podendo escolher entre os sete dias da semana.

**Figura 15 – Cadastro dos Dias Letivos**

**Escolha os dias da semana** [X]

Domingo: ☐ Segunda: ☒ Terça: ☒ Quarta: ☒

Quinta: ☒ Sexta: ☒ Sábado: ☐

Salvar

Fonte: Elaborada pelos autores

Na Figura 16, é apresentada a tela de cadastro dos horários da instituição.

**Figura 16 – Cadastro dos Horários**

**Escolha os Horários** [X]

Adicionar Horário

(1 of 2) [1] [2] [3] [v]

Horário	Descrição	Turno
1º Horário	1º Horário 7:00	
2º Horário	2º Horário 7:40	
3º Horário	3º Horário das	

(1 of 2) [1] [2] [3] [v]

**Escolha um horário** [X]

Horário:

Descrição:

Turno:

Salvar

Fonte: Elaborada pelos autores

Na Figura 17 é possível ver a tela de cadastro das salas de aula. Após fazer o cadastro das salas, dias e horários, se faz necessário acionar o botão “Gerar HorarioAulas”, pois este irá acionar o método que cruzará as sala com horários e



com os dias, gerando todas possibilidades de alocação de horários para a estrutura da instituição. Os dados gerados serão armazenados na base de dados.

**Figura 17 – Cadastro das Salas**

The screenshot displays a web application window titled "Escolha as Salas". At the top, there is a button "Adicionar Horário". Below it, a pagination bar shows "(1 of 3)" and navigation controls. The main table has two columns: "Sala" and "Descrição". The first row contains "Sala 01" and "Sala 01 - prédio CTI", with an "Excluir" button to its right. The second row shows "Sala 02" and the third row shows "Sala 03". A modal window titled "Escolha uma Sala" is overlaid on the table, containing two text input fields labeled "Sala" and "Descrição:", and a "Salvar" button at the bottom.

**Fonte: Elaborada pelos autores**

Na Figura 18 é possível ver a tela de cadastro de cursos. Como visto, pode-se cadastrar mais de um curso, o que aparentemente supõe que o *software* não abrange especificamente ao curso de Sistemas de Informação, mas o que realmente impõe essa delimitação é que os horários a serem alocados, com as restrições e especificações referentes ao curso SI, pode não ser aceitáveis para os demais cursos do IFMG-SJE.

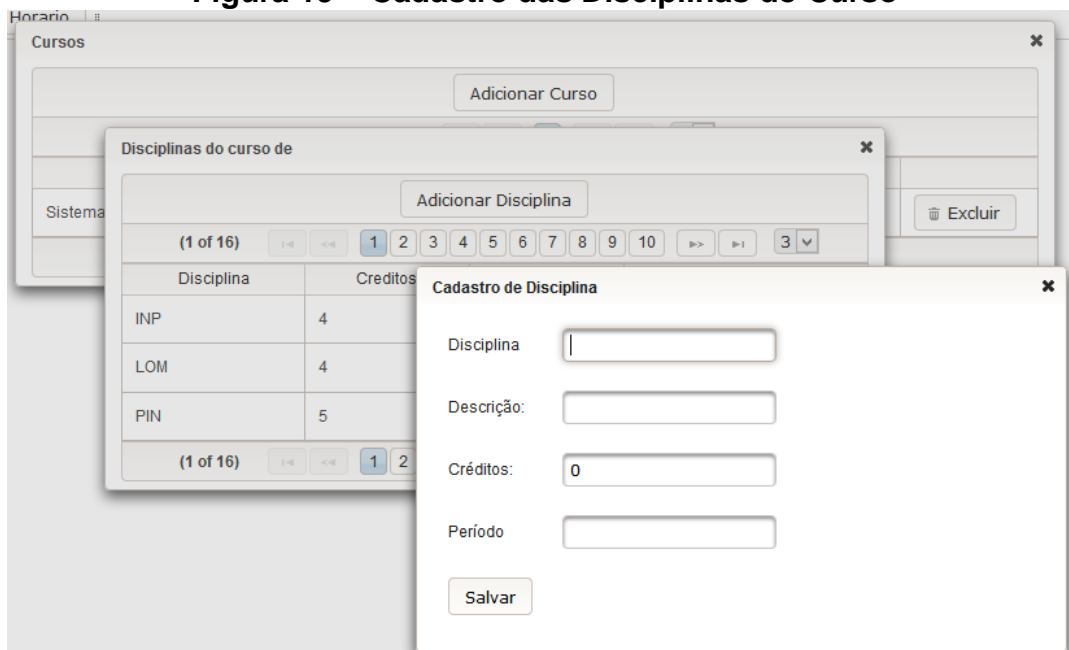
**Figura 18 – Cadastro dos Cursos**

The screenshot shows a web application window titled "Cursos". At the top, there is a button "Adicionar Curso". Below it, a pagination bar shows "(1 of 1)" and navigation controls. The main table has two columns: "Curso" and "Descrição". The first row contains "Sistemas de Informação" and "Bacharelado em Sistemas de Informação". To the right of the description, there are three buttons: "Disciplinas", "Turmas", and "Excluir". A modal window titled "Escolha um Curso" is overlaid on the table, containing two text input fields labeled "Curso" and "Descrição:", and a "Salvar" button at the bottom.

**Fonte: Elaborada pelos autores**

Na Figura 19 é apresentada a tela de cadastro das disciplinas do curso, pode-se perceber que não é possível fazer o cadastro sem antes possuir um curso.

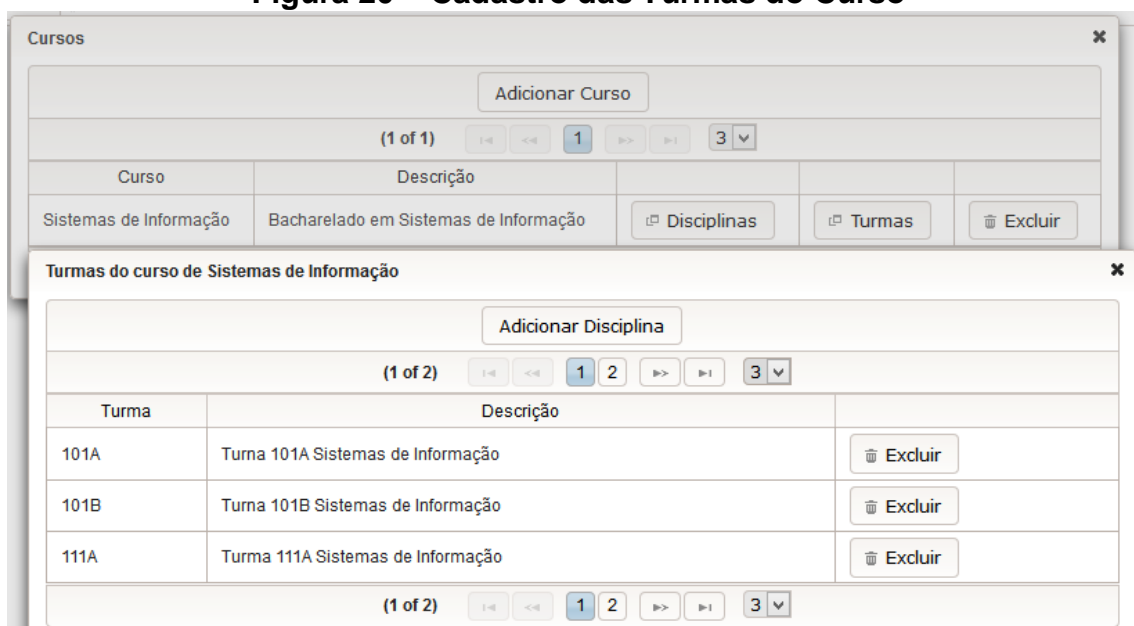
**Figura 19 – Cadastro das Disciplinas do Curso**



**Fonte: Elaborada pelos autores**

Na Figura 20 é apresentada a tela de cadastro das turmas do curso, pode-se perceber, que como no cadastro de disciplinas, não é possível fazer o cadastro sem antes existir um curso.

**Figura 20 – Cadastro das Turmas do Curso**



**Fonte: Elaborada pelos autores**

Na Figura 21 é apresentada a última tela de CRUD do sistema, que é a tela de cadastro do professor.

**Figura 21 – Cadastro dos Professores**

The screenshot shows a web application window titled 'Professores'. At the top, there is a button 'Adicionar Professor'. Below it is a pagination bar showing '(1 of 5)' and page numbers 1, 2, 3, 4, 5. The main table has two columns: 'Nome' and 'Matrícula'. It lists three professors: Bruno (Matrícula: 12345679), Sandro, and Gilmara. There is an 'Excluir' button next to Bruno's entry. A modal window titled 'Escolha um Professor' is open in the foreground, containing three input fields: 'Nome', 'Matrícula', and 'Descrição', followed by a 'Salvar' button.

Fonte: Elaborada pelos autores

Depois de criar todo o CRUD do sistema, fez-se necessária a criação de uma tela, conforme apresentada a Figura 22, que fosse possível escolher os períodos de cada turma, para distribuir as disciplinas dos cursos com as turmas de seus respectivos períodos.

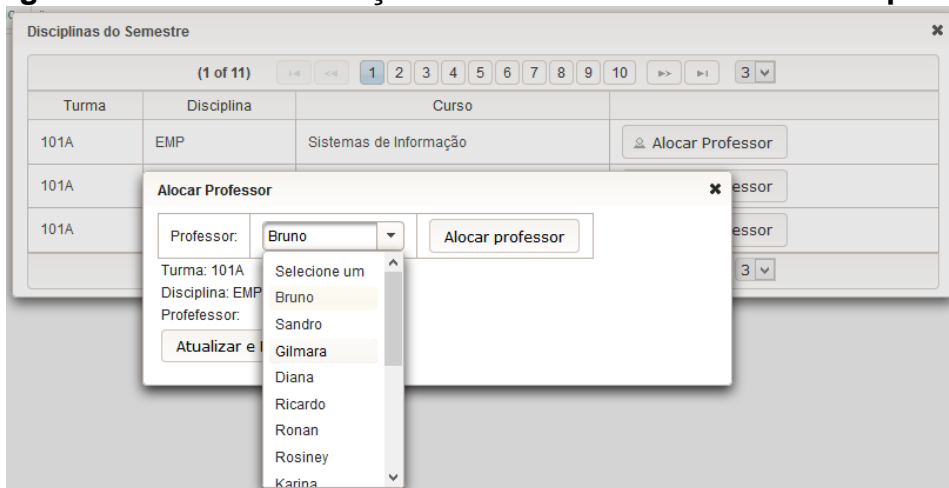
**Figura 22 – Escolha dos Períodos das Turmas**

The screenshot shows a web application window titled 'Escolha as Turmas e seus Períodos'. It features a form with two rows: 'Turma:' with a dropdown menu showing '111A' and 'Período:' with a text input showing '6'. Below the form is a button 'Adicionar Turma'. A pagination bar shows '(1 of 1)' and page numbers 1, 2, 3. The main table has two columns: 'Turma' and 'Período'. It lists one entry: 101A (Período: 8). There is an 'Excluir' button next to this entry. At the bottom, there is a button 'Gerar Disciplinas Aulas'.

Fonte: Elaborada pelos autores

Como último passo, antes de gerar a grade de horários, é necessário alocar os professores com suas respectivas disciplinas de cada curso. A Figura 23 mostra a tela que permite escolher qual professor irá ministrar determinada disciplina.

**Figura 23 – Tela de alocação de Professores em suas Disciplinas**



Fonte: Elaborada pelos autores

## 4.7 Lógica do gerador de horários

O software AHA utiliza um algoritmo desenvolvido com a finalidade de gerar uma grade de horários. Este algoritmo é formado pelos 4 métodos na classe Motor.

- a) gerarGrade();
- b) gerarHorario();
- c) buscarDisciplina();
- d) procurarVaga().

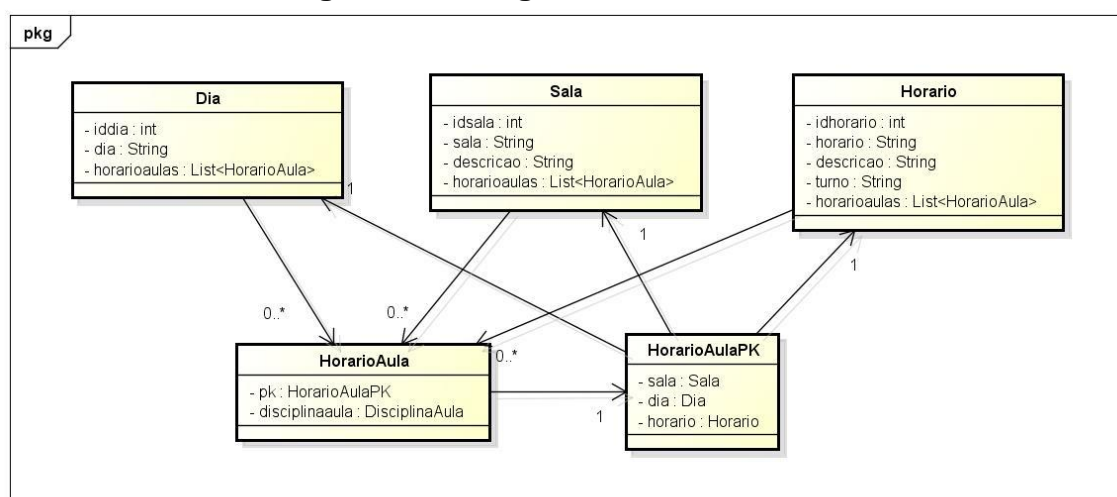
Cada um desses métodos é detalhado nas seções seguintes. Além deles, apresenta-se também o método validar() da classe RestricaoHard, que é parte importante da lógica do *software*.

### 4.7.1 O métodos gerarGrade() da classe Motor

Primeiramente é necessário entender a necessidade da criação do método gerarGrade(), para depois entender o próprio método.

Parte da base do *Software* AHA é ilustrado pelo diagrama da Figura 24. Interpretando este diagrama, em que todas as classes estão mapeadas com o *Hibernate*, tem-se uma classe “Dia” com os atributos “dia” que é do tipo *String* e refere-se à descrição do dia, e uma lista de objetos “HorarioAula”, que contém todos os horários de aulas disponíveis que possuem o objeto “Dia”. Quando obtido o elemento de qualquer posição da lista de objetos “HorarioAula”, tem-se um objeto do tipo “HorarioAula”, e dentro deste objeto, tem-se como atributo um objeto “disciplinaAula” e um objeto “pk”. O objeto “disciplinaAula” do tipo “DisciplinaAula” representa a aula que será ministrada naquele dia, já o objeto “pk” do tipo “HorarioAulaPK” possui três atributos que são objetos que fazem referência respectivamente ao dia, sala e horário que será esta aula.

**Figura 24 – Diagrama de Classe base**



powered by astah

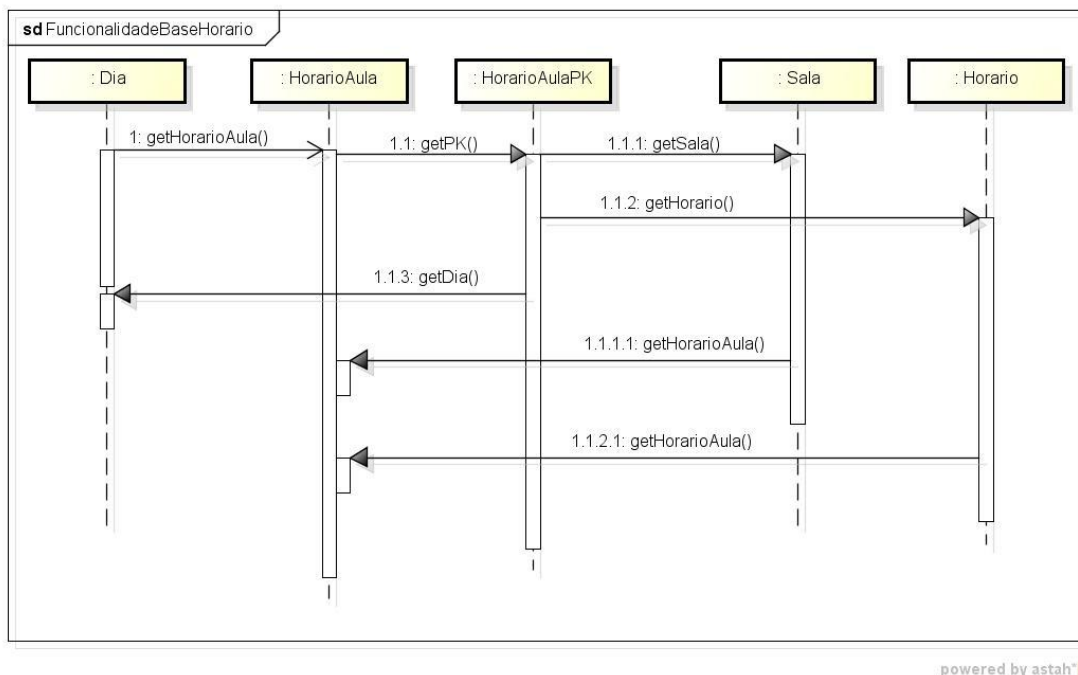
**Fonte: Elaborada pelos autores**

Pode-se notar, ao observar a Figura 25, que os objetos “sala” do tipo “Sala” e “horario” do tipo “Horario” também possuem cada um, uma lista de objetos “HorarioAula”.

Ao analisar o diagrama de sequência ilustrado pela Figura 25, nota-se que ao carregar um objeto “dia” do tipo “Dia”, tem-se acesso a uma lista de objetos “HorarioAula” nos quais se encontram todos os objetos que fazem referência ao “dia” inicial, ao acessar um objeto desta lista tem-se o acesso a um objeto “pk” do tipo “HorarioAulaPK”, que desempenha o papel de uma PK composta no MySQL e que, por sua vez, possibilita o retorno ao objeto inicial “dia”, além do acesso a um

objeto “sala” do tipo “Sala” e um objeto “horário” do tipo “Horario”. Através do objeto “sala”, pode-se acessar os atributos da sala em questão e ainda acessar uma lista de objetos “HorarioAula” que contém todos os horários disponíveis que possuem o objeto “sala”. Pelo fato da lista de objetos “HorarioAula” sempre acessar o objeto que ela representa, perde-se a referência do primeiro objeto acessado.

**Figura 25 – Diagrama de Sequência**



Fonte: Elaborada pelos autores

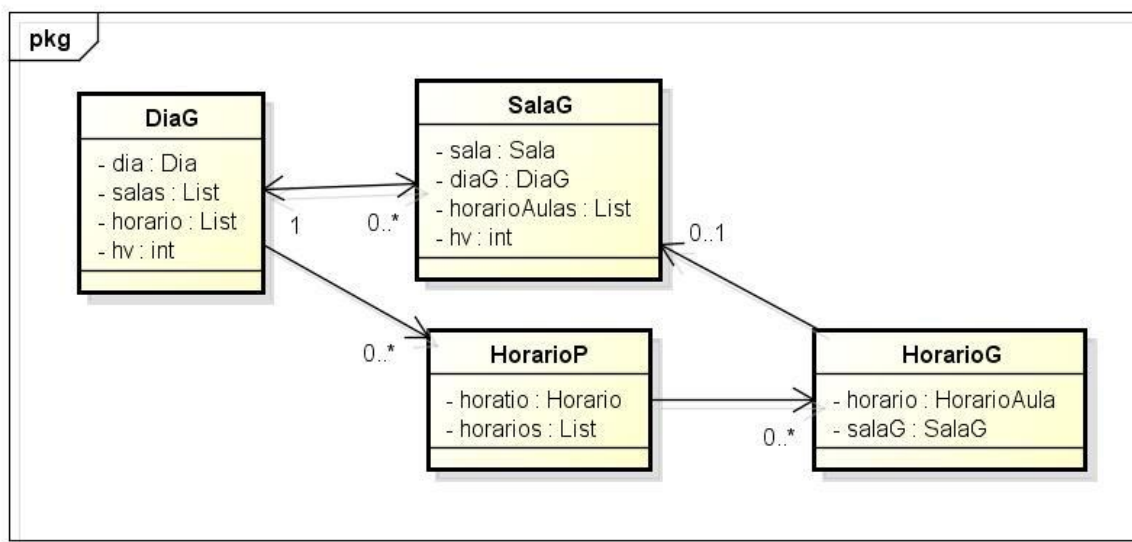
Ao acessar um objeto “Dia”, seguido de um objeto “HorarioAula”, um objeto “HorarioAulaPK”, um objeto “Sala” e, finalmente, uma lista dentro de “Sala”, tem-se os horários disponíveis que contém o objeto “Sala” selecionado. Porém, tais horários contêm todos os objetos “Dia”, e não apenas o objeto “Dia” inicial, assim torna-se impraticável o manuseio desta estrutura imposta pelo *Hibernate*, pela perda da referência ao objeto inicial.

O método gerarGrade() faz uso de 4 classes, conforme a Figura 26, ela foi criada para o único propósito de dividir todos os horários de aulas disponíveis, agrupando-os por dia. Em cada dia será agrupado por sala e também por horário (1º Horário, 2º Horário, etc).

A classe “DiaG” representa um dia da semana que tem a possibilidade de se ter aula, dentro desta classe tem-se os atributos “dia” do tipo “Dia”, que é a referência de um dia da semana; tem-se ainda o atributo “salas” que é uma lista de

objetos “SalaG”; em sequência, tem-se uma lista “horarios” de objetos “HorarioP”; e por último tem-se o atributo “hv”, que indica quantos horários vagos possui o dia em questão.

**Figura 26 – Diagrama de Classe**



powered by astah

**Fonte: Elaborada pelos autores**

A classe “SalaG” representa uma sala com um atributo “sala” tipo “Sala”, e um atributo “diaG” tipo “DiaG”, uma lista “horarioAulas” de objeto “HorarioAula” vindo do Hibernate e, por fim, um atributo “hv” que representa os horários vagos de cada sala no dia correspondente.

A classe “HorarioP” representa um horário de aula em um determinado dia, com atributo “horario” tipo “Horario” e uma lista “horarios” de objetos “HorarioG”.

Por fim, a classe “HorarioG” representa um horário disponível ou não, com o atributo “horarioAula” tipo “HorarioAula” vindo do Hibernate, e um atributo “salaG” tipo “SalaG” que referencia a sala que pertence o objeto “horarioAula”.

O método gerarGrade() cria uma lista de objetos tipo “DiaG” que, a partir do acesso subsequente, não perde a referência ao objeto raiz dia. Ele agrupa por dia e em sequência por sala, no qual a mesma possui todos os horários disponíveis naquele dia para aquela sala. Já na lista “horarios” dentro da Classe “DiaG”, é agrupado os horários disponíveis de cada dia por horário, e de cada um é guardado a referência de qual sala pertence, permitindo a procura de um horário vago partindo dos primeiros horários, e quando encontrado é possível acessar a sala que o horário

disponível pertence, e descobrindo se os próximos horários daquela sala naquele dia também estão vagos.

No Quadro 3 é apresentado o código do método gerarGrade().

**Quadro 3 - gerarGrade()**

```
public List<DiaG> gerarGrade() {
    List<DiaG> listaDiaG;
    listaDiaG = new ArrayList<DiaG>();
    for (int i = 0; i < this.dias.size(); i++) {
        DiaG diaG = new DiaG();
        diaG.setDia(this.dias.get(i));
        diaG.setHv(this.dias.get(i).getHorarioAulas().size());
        diaG.salas = new ArrayList<SalaG>();
        for (int j = 0; j < this.salas.size(); j++) {
            SalaG salaG = new SalaG();
            salaG.setSala(this.salas.get(j));
            salaG.setDiaG(diaG);
            salaG.horarioAulas = new ArrayList<HorarioAula>();
            for (int t = 0; t < this.salas.get(j).
                getHorarioAulas().size(); t++) {
                if (this.salas.get(j).getHorarioAulas().get(t).getPk().
                    getDia().getIddia() == this.dias.get(i).getIddia()) {
                    salaG.horarioAulas.add(this.salas.get(j).
                        getHorarioAulas().get(t));
                }
            }
            salaG.setHv(salaG.horarioAulas.size());
            diaG.salas.add(salaG);
        }
        listaDiaG.add(diaG);
    }
    for (int i = 0; i < listaDiaG.size(); i++) {
        HorarioCTR horarioCTR = new HorarioCTR();
        List<Horario> h = horarioCTR.listar();
        for (int j = 0; j < h.size(); j++) {
            HorarioP horarioP = new HorarioP();
            horarioP.setHorario(h.get(j));
            listaDiaG.get(i).horarios.add(horarioP);
        }
    }
    for (int i = 0; i < listaDiaG.size(); i++) {
        for (int j = 0; j < listaDiaG.get(i).salas.size(); j++) {
            for (int t = 0; t < listaDiaG.get(i).salas.get(j).
                getHorarioAulas().size(); t++) {
                for (int y = 0; y < listaDiaG.get(i).horarios.size(); y++) {
                    if (listaDiaG.get(i).salas.get(j).getHorarioAulas().
                        get(t).getPk().getHorario().equals(
                            listaDiaG.get(i).horarios.get(y).getHorario())) {
                        HorarioG horarioG = new HorarioG();
                        horarioG.setHorarioAula(listaDiaG.get(i).
                            salas.get(j).getHorarioAulas().get(t));
                        horarioG.setSalaG(listaDiaG.get(i).
                            salas.get(j));
                        listaDiaG.get(i).horarios.get(y).
                            horarios.add(horarioG);
                    }
                }
            }
        }
    }
}
```



```

        }
    }
    return listaDiaG;
}

```

Fonte: Elaborada pelos autores

#### 4.7.2 O método gerarHorario() da classe Motor

O método gerarHorario() da classe Motor tem por finalidade escolher qual dia e qual turma serão alocados. Na escolha do dia ele percorre a lista de objetos tipo “DiaG” gerada pelo método gerarGrade(). Em cada dia ele percorre uma lista das turmas a serem alocadas. Ao selecionar uma turma naquele dia, ele faz uma requisição ao método buscarDisciplina() que retorna uma disciplina daquela turma que possui créditos que ainda não foram utilizados. Após este procedimentos, é feito uma chamada ao método procurarVaga() que finaliza a alocação.

O método gerarHorario() repete estes procedimentos até que não haja mais créditos restantes nas disciplinas de todas as turmas. Como característica principal, o método gerarHorario() se preocupa em sempre alocar todas as turmas em todos os dias letivos, garantindo que uma turma nunca fique sem ter aula em algum dia da semana, pois isso iria contra o regulamento que diz que toda turma deve ter no mínimo 200 dias letivos ao ano.

No Quadro 4 é apresentado o código do método gerarHorario().

#### Quadro 4 - gerarHorario()

```

public void gerarHorario(List<DiaG> diaGs) {
    TurmaCTR turmaCTR = new TurmaCTR();
    List<Turma> turmas = turmaCTR.listar();
    Random r = new Random();
    boolean sair = true;
    int contD = 0;
    int contDS = 0;
    int contS = 0;
    int contPara = 0;
    while (sair) {
        try {
            for (int i = 0; i < diaGs.size(); i++) {
                for (int j = 0; j < turmas.size(); j++) {
                    DisciplinaAula disciplinaAula = buscarDisciplina(
                        turmas.get(j), diaGs.get(i).getDia(), diaGs);
                    if (disciplinaAula != null) {
                        procurarVaga(diaGs.get(i).getDia(), diaGs,
                            disciplinaAula);
                    } else {
                        System.out.println("Terminou de gerar o

```

```

        horario");
    }
}
contD++;
contPara++;
if (contD == 5) {
    DisciplinaAulaCTR cTR = new DisciplinaAulaCTR();
    List<DisciplinaAula> testeDisciplinaVazia =
        cTR.listar();
    sair = false;
    for (int x = 0; x < testeDisciplinaVazia.size(); x++){
        if (testeDisciplinaVazia.get(x).
            getCreditosrestantes() > 0) {
            sair = true;
        }
    }
    contD = 0;
}
if (contPara == 5) {
    sair = false;
}
} catch (Exception e) {
    System.out.println(e);
}
}
}

```

Fonte: Elaborada pelos autores

#### 4.7.3 O método *buscarDisciplina()* da classe *Motor*

O método *bucarDisiplina()* é o mais simples entre os métodos da classe *Motor*. Através da turma passada por parâmetro, ele busca uma disciplina com créditos restantes e retorna esta disciplina. Este é o método que, em trabalhos futuros, terá prioridade na modificação, pois através dele é que será possível escolher as disciplinas de acordo com a preferência de cada professor.

Atualmente, o método tem como característica principal a ordenação das disciplinas das turmas pela quantidade de créditos, pois quanto maior essa quantidade, maior a preferência dessa disciplina. Esse cuidado é necessário para evitar a situação em que não seja possível alocar todas as disciplinas devido às restrições *hard*.

No Quadro 5 é apresentado o código do método *bucarDisiplina()*.

**Quadro 5 - *buscarDisciplina()***

```

public DisciplinaAula buscarDisciplina(Turma turma, Dia dia,
    List<DiaG> diasG) {

```

```

Random gerador = new Random();
int cont = 0;
List<DisciplinaAula> lista = turma.getDisciplinaAulas();
Collections.sort(lista);
while (true) {
    if (cont != lista.size()) {
        if (lista.get(cont).getCreditosrestantes() != 0) {
            return turma.getDisciplinaAulas().get(cont);
        }
        cont++;
    } else {
        return null;
    }
}
}

```

Fonte: Elaborada pelos autores

#### 4.7.4 O método procurarVaga() da classe Motor

O método procurarVaga() é o mais complexo. A partir do dia e da disciplina, ele faz a busca por horários vagos seguidos em uma mesma sala.

Como foi definido que sempre deve ser alocado duas ou mais aulas de uma disciplina em sequência, o método percorre os horários do dia em questão, sempre buscando alocar a partir dos primeiros horários e evitando horários fragmentados.

Primeiramente, o método procurarVaga() busca o número de créditos a ser alocado da disciplina através do método disciplinaNumVagas(). Em seguida ele procura na lista gerada pelo método gerarGrade(), o objeto tipo DiaG referente ao dia passado por parâmetro. Após testar se esse dia possui créditos suficientes para alocar a disciplina, é pesquisado qual menor horário disponível no dia para ser alocado. Com o horário escolhido, é verificado se existem horários seguidos e livres na sala referente ao horário escolhido. Se a sala possui horários suficientemente livres, faz-se a verificação através do método validar() da classe RestricaoHard, fazendo a alocação caso seja possível ou passando para o próximo horário, caso contrário.

No Quadro 6 é apresentado o código do método procurarVaga().

#### Quadro 6 - procurarVaga()

```

public String procurarVaga(Dia dia, List<DiaG> diasG, DisciplinaAula
disciplinaAula) {
    try {
        int credito = disciplinaNumVagas(disciplinaAula);
        for (int i = 0; i < diasG.size(); i++) {

```

```

        if (diasG.get(i).getDia().equals(dia)) {
            if (diasG.get(i).getHv() < credito) {
                return "DiaSemHorarioSuficiente";
            }
            for (int horP = 0; horP < diasG.get(i).horarios.size();
                horP++) {
                for (int horG = 0; horG < diasG.get(i).horarios.
                    get(horP).horarios.size(); horG++) {
                    if (diasG.get(i).horarios.get(horP).horarios.
                        get(horG).getSalaG().getHv() >= credito
                        && diasG.get(i).horarios.get(horP).
                            horarios.get(horG).getHorarioAula().
                                getDisciplinaAula() == null) {
                        SalaG sg = diasG.get(i).horarios.get(horP).
                            horarios.get(horG).getSalaG();
                        for (int haVal = 0; haVal < sg.horarioAulas.
                            size(); haVal++) {
                            if (sg.horarioAulas.get(haVal).equals(
                                diasG.get(i).horarios.get(horP).
                                    horarios.get(horG).getHorarioAula())) {
                                int haTrue = haVal;
                                if (haVal + credito <= sg.
                                    horarioAulas.size()) {
                                    for (int haVal2 = haVal; haVal2 <
                                        haVal + credito; haVal2++) {
                                        if (!this.restricaoHard.
                                            validar(sg.horarioAulas.
                                                get(haVal2),
                                                    disciplinaAula)) {
                                            haTrue = -1;
                                        }
                                    }
                                }
                                if (haTrue != -1) {
                                    for (int haVal2 = haTrue;
                                        haVal2 < haTrue + credito;
                                        haVal2++) {
                                        sg.getHorarioAulas().
                                            get(haVal2).
                                                setDisciplinaAula
                                                    (disciplinaAula);
                                        int prov = sg.getHv();
                                        --prov;
                                        sg.setHv(prov);
                                        prov = diasG.get(i).
                                            getHv();
                                        --prov;
                                        diasG.get(i).setHv(prov);
                                        prov = disciplinaAula.
                                            getCreditosrestantes();
                                        --prov;
                                        disciplinaAula.
                                            setCreditosrestantes(prov);
                                    }
                                }
                                return "AlocadoComSucesso";
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
}
} catch (Exception e) {
    System.out.println(e);
}
return null;
}

```

Fonte: Elaborada pelos autores

#### 4.7.5 O método validar() da classe RestricaoHard

O método validar() tem a função de impedir que um professor possua mais de uma aula no mesmo dia e mesmo horário. Ele também não permite que sejam alocadas mais de uma aula de uma turma no mesmo horário e mesmo dia. Ele utiliza de um objeto tipo “HorarioAula”, passado por parâmetro, que com as características do *Hibernate* é possível percorrer as informações do próprio objeto com muita rapidez, fazendo assim as devidas comparações para que se obtenha o resultado desejado.

No Quadro 7 é apresentado o código do método validar().

#### Quadro 7 - validar()

```

public Boolean validar(HorarioAula horarioAula, DisciplinaAula
    disciplinaAula) {

    for (int i = 0; i < horarioAula.getPk().getDia().
        getHorarioAulas().size(); i++) {

        if (horarioAula.getPk().getDia().getHorarioAulas().
            get(i).getDisciplinaAula() != null) {
            if (horarioAula.getPk().getDia().getHorarioAulas().
                get(i).getDisciplinaAula().equals(disciplinaAula)) {

                return false;

            }
        }

        if (horarioAula.getPk().getDia().getHorarioAulas().
            get(i).getPk().getHorario().equals(
                horarioAula.getPk().getHorario())
            && !horarioAula.getPk().getDia().getHorarioAulas().
                get(i).equals(horarioAula)
            && horarioAula.getPk().getDia().getHorarioAulas().
                get(i).getDisciplinaAula() != null) {

            if (horarioAula.getPk().getDia().getHorarioAulas().
                get(i).getDisciplinaAula().getProfessor().
                    equals(disciplinaAula.getProfessor())
                || horarioAula.getPk().getDia().

```

```

        getHorarioAulas().get(i).getDisciplinaAula().
        getTurma().equals(disciplinaAula.getTurma())) {

            return false;

        }

    }

    return true;
}

```

**Fonte: Elaborada pelos autores**

#### 4.8 Tratamento dos dados

Usou-se, para popular a base de dados do software, as informações do Anexo A. Esse anexo possui informações referentes à grade curricular atualizada do curso SI do IFMG-SJE. Também foram cadastrados todos os professores do curso de SI, conforme o portal SI<sup>13</sup> e as turmas que compõem o curso atualmente. Como estrutura física, foram cadastradas 10 salas de aulas que representam as salas de aulas e os laboratórios, como objetos indistinguíveis.

As únicas informações diferentes da atual grade curricular no Anexo A, que foi utilizada como teste, foram as seguintes:

- a) A mudança de período da disciplina de “Gestão de Processos” do 6º período para o 7º período. Esta mudança aconteceu para diminuir o número de créditos que a professora Diana teria de ministrar, pois ela estaria com uma carga de 28 aulas que poderiam ser alocadas em apenas 30 possibilidades, dificultando a alocação;
- b) Também não foram cadastradas as disciplinas optativas, pois as mesmas não são suportadas pela versão inicial do software AHA, pelo fato das disciplinas não serem ofertadas em um mesmo turno.

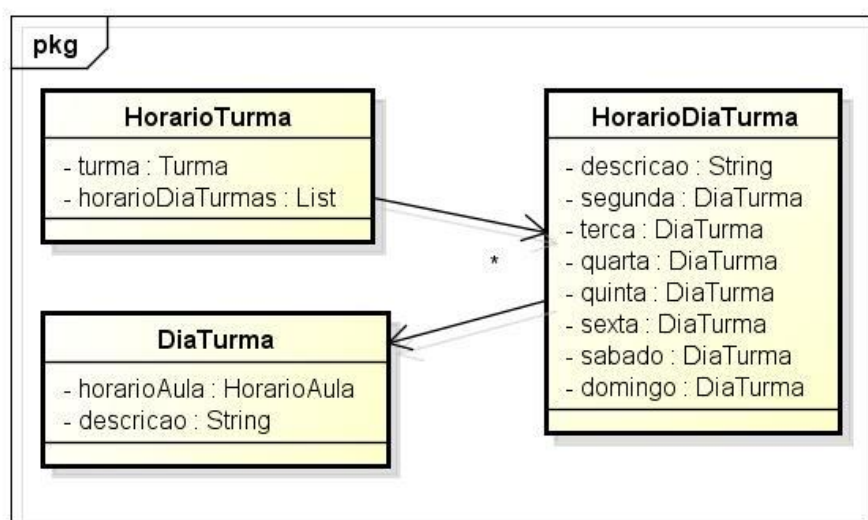
Teve-se a necessidade da criação de um algoritmo que agrupasse os resultados dos horários gerados em turmas e em horários, para a utilização do componente *SummaryRow*<sup>14</sup>, no qual foi possível uma melhor visualização da grade de horário pronta.

<sup>13</sup> <http://portal.sje.ifmg.edu.br/si/index.php/2012-11-02-16-28-39/corpo-doscente>

<sup>14</sup> *SummaryRow* é um componente auxiliar de tabela de dados utilizado para agrupar no *PrimeFaces*.

Para a implementação desse algoritmo, foi necessária a criação de 3 novas classes ilustradas pela Figura 27. O algoritmo tem a finalidade de criar uma lista de objetos “HorarioTurma”, onde tem-se um objeto para cada turma, e cada objeto possui um atributo “turma” tipo “Turma” que representa uma turma em questão, e um atributo “horarioDiaTurmas”, que é uma lista de objetos tipo “HorarioDiaTurma”, sendo que a lista “horarioDiaTurmas” representa o agrupamento em horários dos Horários Alocados que, acessada a partir do 1º Horário, possui todos os respectivos horários de cada dia da semana.

**Figura 27 – Diagrama de Classe**



powered by astah®

Fonte: Elaborada pelos autores

## 5 ANÁLISE DOS RESULTADOS

O *software* AHA foi desenvolvido sem nenhuma referência algorítmica, ou seja, não foi utilizado nenhum trabalho existente para se chegar à lógica encontrada no *software* AHA. Mas como visto em trabalhos correlatos, há uma dificuldade muito grande em se obter uma solução imediata e ótima, sendo que a maioria dos autores citados neste trabalho preferem buscar um resultado aleatório e, a partir dele, gerar modificações, conforme as necessidades, para encontrar uma solução desejada.

Considera-se como principal resultado deste trabalho o *software* AHA, no qual as informações previamente cadastradas gera um horário básico que não colide horários de turmas e professores. As informações previamente cadastradas são o: quadro de professores; o curso e suas respectivas disciplinas de sua grade curricular; a estrutura física e seus horários de apenas um único turno.

Baseando em um ambiente de teste com 1 curso com 6 turmas, 32 disciplinas, 112 aulas a serem alocadas dessas disciplinas, e uma estrutura com 10 salas de aulas, 5 dias na semana com 6 horários por dia, somando 270 horários disponíveis semanais, foi possível chegar aos seguintes tempos, apresentado pela Tabela 1.

**Tabela 1 – Testes de alocação**

Teste de Alocação	Tempo (milissegundos)
1ª	2283
2ª	1542
3ª	1273
4ª	1207
5ª	1020
6ª	1068
7ª	1109
8ª	2099
Média	1450,13 (1,45 segundos)

**Fonte: Elaborada pelos autores**

O algoritmo desenvolvido como motor do *software* AHA apresentou um desempenho satisfatório, levando em conta os tempos necessários para a geração dos horários, conforme os resultados experimentais apresentados na Tabela 1.



Na coluna Teste de Alocação, é apresentada a ordem em que se fez a operação de criação da grade de horários automática pelo *software* AHA. Já na coluna Tempo, é apresentado o tempo gasto em cada criação da grade. Uma curiosidade no teste realizado, foi que nas 6 primeiras tentativas houve um decremento em cada tentativa onde apenas na 7ª e 8ª que o tempo foi maior que o anterior. Mas em média a criação de uma grade de horário demora 1,45 segundos. A cada teste realizado, obteve-se uma grade de horário, na qual não houve colisão de horários e também não ficaram créditos sem alocar, conforme mostrado através das Figuras (28, 29 e 30), a grade de horário gerada no 8º teste.

**Figura 28 – Tela da Grade de Horários**

Grade de Horário					
Horários	Dias da Semana				
	Segunda	Terça	Quarta	Quinta	Sexta
<b>101A</b>					
1º Horário	ETL Gilmará   Sala 01	TCC II TCC Bruno   Sala 01	EMP Diana   Sala 01	EMP Diana   Sala 01	MUL Ronan   Sala 01
2º Horário	ETL Gilmará   Sala 01	TCC II TCC Bruno   Sala 01	EMP Diana   Sala 01	EMP Diana   Sala 01	MUL Ronan   Sala 01
3º Horário	MUL Ronan   Sala 03	SAS Ricardo   Sala 03	SAS Ricardo   Sala 02	<< Vago >> 	<< Vago >> 
4º Horário	MUL Ronan   Sala 03	SAS Ricardo   Sala 03	SAS Ricardo   Sala 02	<< Vago >> 	<< Vago >> 
5º Horário	<< Vago >> 	<< Vago >> 	<< Vago >> 	<< Vago >> 	<< Vago >> 
6º Horário	<< Vago >> 	<< Vago >> 	<< Vago >> 	<< Vago >> 	<< Vago >> 
<b>101B</b>					
1º Horário	MUL Ronan   Sala 05	SAS Ricardo   Sala 05	SAS Ricardo   Sala 04	<< Vago >> 	<< Vago >> 
2º Horário	MUL Ronan   Sala 05	SAS Ricardo   Sala 05	SAS Ricardo   Sala 04	<< Vago >> 	<< Vago >> 
3º Horário	ETL Gilmará   Sala 01	TCC II TCC Bruno   Sala 01	EMP Diana   Sala 01	EMP Diana   Sala 01	MUL Ronan   Sala 01
4º Horário	ETL Gilmará   Sala 01	TCC II TCC Bruno   Sala 01	EMP Diana   Sala 01	EMP Diana   Sala 01	MUL Ronan   Sala 01
5º Horário	<< Vago >> 	<< Vago >> 	<< Vago >> 	<< Vago >> 	<< Vago >> 
6º Horário	<< Vago >> 	<< Vago >> 	<< Vago >> 	<< Vago >> 	<< Vago >> 

Fonte: Elaborada pelos autores

Como dito anteriormente, foram alocados 112 créditos de 32 disciplinas de 6 turmas diferentes. As turmas 101A e 101B do 8º período tinham 16 créditos cada

uma, já as turmas 111A e 111B do 6º período tinham 17 créditos cada uma, a turma 121A do 4º período tinha 24 créditos e a turma 131A do 2º período tinha 22 créditos.

**Figura 29 – Tela da Grade de Horários**

Grade de Horario						
<div> <span>1</span> <span>2</span> <span>3</span> </div>						
Horários	Dias da Semana					
	Segunda	Terça	Quarta	Quinta	Sexta	
<b>111A</b>						
1º Horário	CSN Bruno   Sala 02	GIN Diana   Sala 02	<< Vago >> 	RCO III Ricardo   Sala 04	LPR II Edson Santos   Sala 02	
2º Horário	CSN Bruno   Sala 02	GIN Diana   Sala 02	<< Vago >> 	RCO III Ricardo   Sala 04	LPR II Edson Santos   Sala 02	
3º Horário	LPR II Edson Santos   Sala 04	GIN Diana   Sala 02	<< Vago >> 	<< Vago >> 	<< Vago >> 	
4º Horário	LPR II Edson Santos   Sala 04	<< Vago >> 	<< Vago >> 	<< Vago >> 	<< Vago >> 	
5º Horário	<< Vago >> 	RCO III Ricardo   Sala 01	GPE Diana   Sala 01	GPE Diana   Sala 01	<< Vago >> 	
6º Horário	<< Vago >> 	RCO III Ricardo   Sala 01	GPE Diana   Sala 01	GPE Diana   Sala 01	<< Vago >> 	
<b>111B</b>						
1º Horário	GPE Diana   Sala 06	LPR II Edson Santos   Sala 06	LPR II Edson Santos   Sala 05	<< Vago >> 	GPE Diana   Sala 03	
2º Horário	GPE Diana   Sala 06	LPR II Edson Santos   Sala 06	LPR II Edson Santos   Sala 05	<< Vago >> 	GPE Diana   Sala 03	
3º Horário	CSN Bruno   Sala 02	<< Vago >> 	<< Vago >> 	RCO III Ricardo   Sala 02	RCO III Ricardo   Sala 02	
4º Horário	CSN Bruno   Sala 02	GIN Diana   Sala 02	<< Vago >> 	RCO III Ricardo   Sala 02	RCO III Ricardo   Sala 02	
5º Horário	<< Vago >> 	GIN Diana   Sala 02	<< Vago >> 	<< Vago >> 	<< Vago >> 	
6º Horário	<< Vago >> 	GIN Diana   Sala 02	<< Vago >> 	<< Vago >> 	<< Vago >> 	
<div> <span>1</span> <span>2</span> <span>3</span> </div>						

Fonte: Elaborada pelos autores

As Figuras (28, 29 e 30), ilustram a janela de exibição da grade de horário pronta. Essa janela possui uma *dataTable*<sup>15</sup> que apresenta os horários semanais das turmas, a *dataTable* foi dividida em 6 colunas, sendo que a primeira indica o horário (ex.: 1º Horário, 2º Horário) agrupado pelas turmas (ex.: 101A, 101B), já as outras colunas representam as aulas que serão ministradas nos respectivos horários em cada dia da semana em que se enquadra na coluna em questão. Nos horários que possuem aulas aparecerão a sigla da disciplina e o nome do professor com a sala

<sup>15</sup> *DataTable* é um componente do PrimeFaces, ele é uma versão aprimorada do padrão *Datatable* que fornece embutido soluções para muitos casos de uso comuns como paginação, classificação, seleção, lazy loading, filtragem e mais.

que está agendada a aula. Já nos horários que não possuem aula aparece a descrição de <<Vago>>.

**Figura 30 – Tela da Grade de Horários**

Grade de Horário					
<div> <span>1</span> <span>2</span> <span>3</span> </div>					
Horários	Dias da Semana				
	Segunda	Terça	Quarta	Quinta	Sexta
<b>121A</b>					
1º Horário	AED III Rosiney   Sala 03	AED III Rosiney   Sala 03	EST Kêny   Sala 02	EST Kêny   Sala 02	RCO I Mafra   Sala 04
2º Horário	AED III Rosiney   Sala 03	AED III Rosiney   Sala 03	EST Kêny   Sala 02	EST Kêny   Sala 02	RCO I Mafra   Sala 04
3º Horário	RCO I Mafra   Sala 05	ESO I Edson Santos   Sala 04	ESO I Edson Santos   Sala 03	BDA II Julio   Sala 03	BDA II Julio   Sala 03
4º Horário	RCO I Mafra   Sala 05	ESO I Edson Santos   Sala 04	ESO I Edson Santos   Sala 03	BDA II Julio   Sala 03	BDA II Julio   Sala 03
5º Horário	SOP Bruno   Sala 01	SOP Bruno   Sala 03	<< Vago >> 	<< Vago >> 	<< Vago >> 
6º Horário	SOP Bruno   Sala 01	SOP Bruno   Sala 03	<< Vago >> 	<< Vago >> 	<< Vago >> 
<b>131A</b>					
1º Horário	SOC Marcos Murta   Sala 04	ING Penha   Sala 04	CDI I Sandro   Sala 03	CDI I Sandro   Sala 03	ALI Kêny   Sala 05
2º Horário	SOC Marcos Murta   Sala 04	ING Penha   Sala 04	CDI I Sandro   Sala 03	CDI I Sandro   Sala 03	ALI Kêny   Sala 05
3º Horário	ALI Kêny   Sala 06	FSI Mafra   Sala 05	FSI Mafra   Sala 04	AED I Karina   Sala 04	AED I Karina   Sala 04
4º Horário	ALI Kêny   Sala 06	FSI Mafra   Sala 05	FSI Mafra   Sala 04	AED I Karina   Sala 04	AED I Karina   Sala 04
5º Horário	<< Vago >> 	<< Vago >> 	<< Vago >> 	AED I Karina   Sala 04	AED I Karina   Sala 04
6º Horário	<< Vago >> 	<< Vago >> 	<< Vago >> 	<< Vago >> 	<< Vago >> 

**Fonte: Elaborada pelos autores**

Pode ser visto pelas Figuras (28, 29 e 30) que não houve colisão e que todas as disciplinas estão alocadas de acordo com o especificado, ou seja, estão alocadas sequenciadas de acordo com a quantidade de créditos de cada uma. Também se pode notar que apenas dois horários vagos entre aulas de duas turmas foi relatado. Horários este da professora Diana, que ministra 22 aulas em 30 possibilidades, o que torna extremamente difícil de encaixar todas as aulas sem colisão.

## 6 CONSIDERAÇÕES FINAIS

Como se estipulou que o presente trabalho delimita-se apenas ao curso de Sistemas de Informação, pelos motivos já expostos anteriormente, pode-se concluir que todos objetivos foram alcançados com sucesso. Obteve-se uma solução Web para o problema de Alocação de Horários Acadêmicos, mesmo que ainda básica, mas bem estruturada e de maneira automática e rápida e, que devido às tecnologias e padrões utilizados no desenvolvimento, é permite facilmente adequações de novas funcionalidades ao *software*.

Na implementação, seguiu-se o padrão MVC com orientação a objeto, o que permite uma fácil manipulação e alteração em suas funcionalidades. E como sua parte lógica também está separada por funções, o *software* em geral ficou com uma adaptabilidade consideravelmente boa, o que permite modificar as funções independentemente uma das outras.

Como limitação do sistema, pode-se citar o fato de não abranger os demais cursos, o que impossibilita o uso do mesmo na instituição, pois os professores do curso SI ministram aulas para os demais cursos. Uma adequação a esta realidade é bem complexa pelo fato do IFMG-SJE ser uma instituição com cursos Superiores e Ensino Médio integrado com o Técnico.

Sendo assim, considera-se este trabalho como um marco inicial para resolução para o problema de Alocação de Horários do IFMG-SJE, onde se pretende desenvolver novas versões para o *software* AHA, a fim de deixa-lo usual. Também se espera inspirar novos pesquisadores a explorar este vasto campo de pesquisa, em particular, no estudo de caso baseado no IFMG-SJE.

Como dito no decorrer do texto, Java é considerada por muitos como a maior curva de aprendizagem na atualidade, mas mesmo assim é uma linguagem robusta com inúmeras APIs que ameniza sua complexidade na implementação.

O desenvolvimento do presente trabalho se tornou de grande valia no aprendizado pessoal dos autores, em que contribuiu muito para o crescimento pessoal, profissional e principalmente acadêmico dos mesmos.

## REFERÊNCIAS BIBLIOGRÁFICAS

BARDADYM, V. A. **Computer-Aided School and University Timetabling: The New Wave**. Lecture Notes in Computer Science Volume 1153, pp 22-45, 1996.

BITTENCOURT, G. **Inteligência Artificial: Ferramentas e Teorias**. Florianópolis: UFSC, 2006.

BRAZ JÚNIOR, O. O. **Otimização de horários em instituições de ensino superior através de algoritmos genéticos**. Dissertação (Mestrado). Universidade Federal de Santa Catarina Programa de Pós-Graduação em Engenharia de Produção. Florianópolis 2000.

CONSTANTINO, A. A. **Algoritmos heurísticos construtivos para agrupamento de alunos em turmas**. Pesquisa Operacional na Gestão do Conhecimento, p. 310-321. XLI SBPO 2009.

DEITEL, H. M. ; AND DEITEL, P. J. **Java Como Programar**. Prentice Hall, 2008.

EVEN, S.; ITAI, A. E.; SHAMIR, A. **On the complexity of timetabling and multicommodity flow problems**. SIAM Journal of Computation, 1976.

FAYAD, M. E.; SCHMIDT, D. C. **Object-oriented Application frameworks**. Communications of the ACM. Vol. 40, 1997.

FERNANDES, A. M. **Inteligência Artificial: I noções gerais**. Florianópolis: VisualBooks, 2005.

FREITAS , M. J.; COSTA, F. P.; FERNANDES, I. G. G. **Um Algoritmo Evolutivo Híbrido para o Problema de Programação de Horários em Escolas**. Disponível em: <<http://www.decom.ufop.br/prof/marcone/Publicacoes/ENEGEP-2002-PHE.pdf>> Acesso em 17/10/2013. ENEGEP 2006.

HOLZNER, Steven. **Desvendando XML**. Rio de Janeiro: Campus, 2001.

KOTSKO, E. G. S.; MACHADO, A. L. F.; SANTOS, E. M. **Otimização na alocação de professores na construção de uma grade horária escolar**. Ambiência v.1 n.1 p. 31-45, Guarapuava-PR 2005.

LUCKOW, D. H.; MELO, A. A. **Programação Java para Web**. São Paulo: novatec 2010.

MARQUES NETO, M. C.; BARBOSA, F.; FREITAS, C.; BAGANO, P. **Uma Ferramenta Baseada em Algoritmos Genéticos para a Geração de Tabela de Horário Escolar**. In: Sétima Escola Regional de Computação Bahia-Sergipe, Vitória da Conquista, 2007.

**Obtenha Informações sobre a Tecnologia Java**. Disponível em: <[http://www.java.com/pt\\_BR/about/](http://www.java.com/pt_BR/about/)>. Acesso em: 11/04/2013.

PAIM, A. D.; GREIS, I. C. **Abordagens para elaboração automatizada de tabela de horários acadêmicos**. XI Seminário Intermunicipal de Pesquisa, 2008.

PRODANOV, C. C.; FREITAS, E. C. **Metodologia do trabalho científico: Métodos e Técnicas da Pesquisa e do Trabalho Acadêmico**. Rio Grande do Sul: Universidade Feevale, 2013.

ROUSSEAU, Jean-Jacques. **Emílio ou da educação**. Tradução de Sérgio Milliet, Difel, São Paulo, 1968.

SCHAERF, A. **A survey of automated timetabling** A. Schaerf. Artificial Intelligence Review, 1999.

SOARES FILHO, O. H. **Utilização do Framework Hibernate Para Mapeamento Objeto/Relacional na Construção de Um sistema de Informação**. Disponível em <http://campeche.inf.furb.br/tccs/2006-I/2006-1odilonherculanosoesfilhovf.pdf>> Acesso em: 17 out. de 2013. Blumenau 2006.

SOUZA, M. J. F. **Programação de horários em escolas: uma aproximação por metaheurísticas**. Tese (Doutorado em Engenharia de sistemas e Computação) – UFRJ, Rio de Janeiro 2000.

TIMÓTEO, G. T. **Desenvolvimento de um algoritmo genético para a resolução de timetabling**. 2002.

## ANEXO A

CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO  
MATRIZ CURRICULAR – 2013

1º PERÍODO					
Código	Disciplina	Nº aulas	Nº horas	Créditos semanais	Pré-requisito(s)
INP	Introdução à Programação	80	60h	4	-----
LOM	Lógica Matemática	80	60h	4	-----
PIN	Português Instrumental	100	75h	5	-----
FMA	Fundamentos de Matemática	80	60h	4	-----
ICO	Introdução à Computação	80	60h	4	-----
FIL	Filosofia	40	30h	2	-----
<b>Total</b>		<b>460</b>	<b>345</b>	<b>23</b>	-----
2º PERÍODO					
Código	Disciplina	Nº aulas	Nº horas	Créditos semanais	Pré-requisito(s)
AED I	Algoritmos e Estruturas de Dados I	120	90h	6	INP
CDI I	Cálculo Diferencial e Integral I	80	60h	4	FMA
SOC	Sociologia	40	30h	2	-----
ALI	Álgebra Linear	80	60h	4	-----
FSI	Fundamentos de Sist. de Informação	80	60h	4	-----
ING	Inglês Instrumental	40	30h	2	-----
<b>Total</b>		<b>440</b>	<b>330</b>	<b>22</b>	-----
3º PERÍODO					
Código	Disciplina	Nº aulas	Nº horas	Créditos semanais	Pré-requisito(s)
AED II	Algoritmos e Estruturas de Dados II	80	60h	4	AED I
IUM	Interfaces Usuário-Máquina	80	60h	4	-----
CDI II	Cálculo Diferencial e Integral II	80	60h	4	CDI I
MCI	Metodologia Científica	80	60h	4	-----
ACO	Arquitetura de computadores	80	60h	4	ICO
BDA I	Bancos de Dados I	80	60h	4	-----
<b>Total</b>		<b>480</b>	<b>360</b>	<b>24</b>	-----
4º PERÍODO					
Código	Disciplina	Nº aulas	Nº horas	Créditos semanais	Pré-requisito(s)

AED III	Algoritmos e Estruturas de Dados III	80	60h	4	AED II
EST	Estatística Geral	80	60h	4	-----
RCO I	Redes de Computadores I	80	60h	4	ACO
ESO I	Engenharia de Software I	80	60h	4	-----
BDA II	Bancos de Dados II	80	60h	4	BDA I e INP
SOP	Sistemas Operacionais	80	60h	4	ACO e AED I
<b>Total</b>		<b>480</b>	<b>360</b>	<b>24</b>	-----

### 5º PERÍODO

Código	Disciplina	Nº aulas	Nº horas	Créditos semanais	Pré-requisito(s)
FAD	Fundamentos da Administração	40	30h	2	-----
LPR I	Linguagem de Programação I	100	75h	5	BDA II
RCO II	Redes de Computadores II	80	60h	4	RCO I
ESO II	Engenharia de Software II	120	90h	6	ESO I
IED	Informática na Educação	60	45h	3	-----
	<b>Subtotal</b>	<b>400</b>	<b>300</b>	<b>20</b>	
	Optativa	40	30h	2	
<b>Total</b>		<b>440</b>	<b>330</b>	<b>22</b>	-----

### 6º PERÍODO

Código	Disciplina	Nº aulas	Nº horas	Créditos semanais	Pré-requisito(s)
GPE	Gestão de Pessoas	80	60h	4	FAD
LPR II	Linguagem de Programação II	80	60h	4	BDA II
GIN	Gestão da Informação	60	45h	3	-----
CSO	Computadores e Sociedade	40	30h	2	-----
GPR	Gestão de Processos	60	45h	3	-----
RCO III	Redes de Computadores III	80	60h	4	RCO II
	<b>Subtotal</b>	<b>400</b>	<b>300</b>	<b>20</b>	
	Optativa	40	30h	2	
<b>Total</b>		<b>440</b>	<b>320</b>	<b>22</b>	-----

### 7º PERÍODO

Código	Disciplina	Nº aulas	Nº horas	Créditos semanais	Pré-requisito(s)
GPJ	Gerência de Projetos	60	45h	3	ESO II
LPW	Linguagem de Programação Web	80	60h	4	INP



COC	Contabilidade e Custos	60	45h	3	-----
PPR	Padrões de Projeto	60	45h	3	ESO I e LPR II
IAR	Inteligência Artificial	60	45h	3	AED III e LOM
TCC I	Trabalho de Conclusão de Curso I	80	60h	4	-----
<b>Total</b>		<b>400</b>	<b>300</b>	<b>20</b>	-----
<b>8º PERÍODO</b>					
Código	Disciplina	Nº aulas	Nº horas	Créditos semanais	Pré-requisito(s)
EMP	Empreendedorismo	80	60h	4	-----
MUL	Multimídia	80	60h	4	-----
SAS	Segurança e Auditoria de Sistemas	80	60h	4	RCO III
ETL	Ética e Legislação	40	30h	2	-----
TCC II	Trabalho de Conclusão de Curso II	80	60h	4	TCC I
<b>Total</b>		<b>440</b>	<b>270</b>	<b>18</b>	-----
<b>Disciplinas obrigatórias (sem o TCC e sem optativa)</b>					<b>2.445 h</b>
<b>Disciplinas optativas</b>					<b>60 h</b>
<b>Estágio Supervisionado</b>					<b>300 h</b>
<b>Atividades Complementares</b>					<b>100 h</b>
<b>Trabalho de Conclusão de Curso – TCC</b>					<b>120 h</b>
<b>CARGA HORÁRIA TOTAL</b>					<b>3025 h</b>

**Bruno de Souza Toledo**  
Coord. Curso Bacharelado em  
Sistemas Informação

**Paulo do Nascimento**  
Coord. Geral de Grad. Pós-Graduação