UNIVERSIDADE EDUARDO MONDLANE

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE MATEMÁTICA E INFORMÁTICA

Trabalho de Licenciatura

PROGRAMAÇÃO DE HORÁRIOS DE ESCOLAS

Caso estudo

Escola Secundária Francisco Manyanga

AUTOR: Muchanga, Azarias Tomás Joaquim

Maputo, Novembro de 2007

UNIVERSIDADE EDUARDO MONDLANE

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE MATEMÁTICA E INFORMÁTICA

Trabalho de Licenciatura

PROGRAMAÇÃO DE HORÁRIOS DE ESCOLAS

Caso estudo

Escola Secundária Francisco Manyanga

AUTOR: Muchanga, Azarias Tomás Joaquim

SUPERVISORA: Prof. Dra Esselina Macome

Maputo, Novembro de 2007

DEDICATÓRIA

"O mais importante da vida não é a situação em que estamos, mas a direcção para a qual nos movemos". (Oliver Wendell Holmes)

Aos meus pais;

À minha esposa;

Ao meu filho;

Aos meus irmãos.

AGRADECIMENTOS

Agradeço à Deus, por ter me dado coragem de enfrentar o mundo da ciência e tecnologia;

À minha esposa, Neide Augusta Braga Botelho, pelo apoio, coragem e muita força durante esta luta académica e por ter me suportado.

Ao meu filho Azarias Tomás Joaquim Muchanga Júnior por ter sido a fonte de inspiração na elaboração desta dissertação.

Agradeço à minha família, aos meus pais, Tomás Simone Muchanga e Teresinha Xavier Joaquim Muchanga pelo apoio financeiro e moral.

O meu muito obrigado à minha supervisora, Prof. Doutora Esselina Macome pela paciência, atenção, críticas, ensinamentos e direccionamento ao longo do trabalho;

Agradeço aos meus colegas do DMI e amigos pelo imenso carinho que me demonstraram e pela ajuda na concretização do curso;

Um profundo agradecimento vai à todos os funcionários e estudantes do DMI, em particular aos que dedicarem parte do seu tempo para a leitura deste trabalho. Espero sinceramente contribuir de alguma forma com os conhecimentos e experiências expostas.

DECLARAÇÃO DE HONRA

Declaro por minha honra que este trabalho é fruto da minha investigação, e que o mesmo foi realizado para ser submetido à Faculdade de Ciências em cumprimento parcial para obtenção de Licenciatura em Informática na Universidade Eduardo Mondlane.

Maputo, Outubro de 2007

Azarias Tomás Joaquim Muchanga

Azarias Tomás Joaquim Muchanga

RESUMO

Um dos grandes problemas que está inserido no contexto das instituições educacionais moçambicanas ao iniciar o seu período lectivo é a programação de horários das aulas. Este problema se torna de difícil resolução devido ao grande número de possibilidades a serem analisadas e a necessidade de satisfazer um conjunto de restrições pedagógicas, administrativas e até mesmo pessoais, muitas das quais conflituantes entre si, tornando o espaço de busca vasto e altamente restrito.

A programação manual dos horários é uma tarefa árdua e normalmente requer vários dias de trabalho. Além do mais a solução obtida pode ser insatisfatória com relação a vários aspectos. Como por exemplo, um professor por ficar insatisfeito se houver muitas janelas em sua programação semanal de ensino ou pode haver aulas de uma mesma disciplina sendo ministradas em dias consecutivos, tendo como prejuízo a sedimentação da aprendizagem.

Em virtude da diversidade de regimes educacionais e das características de cada escola ou instituição de ensino o problema de programação de horários é um problema de difícil generalização. Sendo assim este trabalho foca-se na programação de horários de aulas de escolas secundárias moçambicanas, mais particularmente a Escola Secundária Francisco Manyanga.

Neste trabalho é apresentado um algoritmo híbrido baseado nas técnicas GRASP e Busca Tabu como proposta para a resolução do problema de programação de horários.

NOTAÇÕES E GLOSSÁRIO

NOTAÇÕES

Notação	Descrição
BT	Busca Tabu
GRASP	*Greedy Randomized Adaptive Search Procedure* ou Procedimento de busca adaptativa gulosa e aleatória
ESFM	Escola Secundária Francisco Manyanga
PPHE	Problema de Programação de Horários em Escolas
ESG	Ensino Secundário Geral
LCR	Lista de Candidatos Restrita
RA	Relaxação Adoptiva ou Oscilação Estratégica
DT	Director de Turma
CH	Carga Horária
AMDD	*Agile Model Driven Development* ou Desenvolvimento Ágil Orientado ao Modelo
UML	*Unified Modeling Language*
XP	*Extreme Programming*
II	Procedimento Interturmas-Intraturmas
RC	Reconexão por Caminhos
BTRC	Busca Tabu com intensificação por Reconexão por Caminhos
BTRA	Busca Tabu com intensificação por Relaxação Adoptiva

JDBC	Java Database
JSP	*Java Server Pages*
UI	*User Interface*
CRC	Classes, Responsabilidades e Colaboradores
OO	Orientada a Objectos
DB	*Database*/Base de Dados
JVM	*Java Virtual Machine*
DSL	*Domain-specific languages*
MVC	*Model-view-controller*
ASP	*Active Server Pages*
HTTP	*HyperText Transfer Protocol* (Protocolo de Transferência de Hipertexto)
HTML	*HyperText Markup Language* ou Linguagem de Marcação de Hipertexto
POJOs	*Plain Old Java Objects*

GLOSSÁRIO

Elemento	Descrição
Web browser	Um navegador (também conhecido como web browser ou simplesmente browser, termos em inglês) é um programa que habilita seus usuários a interagirem com documentos HTML (em linguagem de hipertexto) hospedados em um servidor Web, de acesso à Internet. (Wikipédia, 2007)

Container	É o ambiente em que os *servlets* são executados. Este gere as instâncias do *servlet* e provê os serviços de rede necessários para as requisições e respostas. (Oliveira, 2001)
Design pattern	Padrões de desenho de software descrevem soluções para problemas recorrentes no desenvolvimento de sistemas de software orientados a objectos. Um padrão de projecto estabelece um nome e define o problema, a solução, quando aplicar esta solução e suas consequências. Os padrões de projecto visam facilitar a reutilização de soluções de desenho - isto é, soluções na fase de projecto do software, sem considerar reutilização de código. Também acarretam um vocabulário comum de desenho, facilitando comunicação, documentação e aprendizado dos sistemas de software. (Wikipédia, 2007)
Login	O mesmo que *log on* – estrutura que permite o acesso do utilizador a um sistema, geralmente mediante autorização através de uma senha (Coelho & Fernanda, 1999)

ÍNDICE

Índice de Figuras

1 INTRODUÇÃO

Define-se Horário como sendo uma tabela que indica as horas que se fazem certos serviços (Costa & Melo) ou como sendo uma lista ou programa, usualmente visto em forma tabular, dispondo informação sobre uma série de eventos organizados: em particular, o tempo no qual esses eventos estão planeados a ocorrer.

Segundo (Wren, 1995), o problema de programação de horários e escalas pode ser definido como arranjo dentro de padrões de tempo ou espaço, no qual algumas metas são atendidas ou praticamente atendidas e onde as restrições devem ser satisfeitas ou praticamente satisfeitas.

O Problema de Programação de Horários tem atraído atenção da comunidade científica de diferentes disciplinas, tais como a Programação Linear e Inteligência Artificial há cerca de 40 anos e dentro da última década tem aumentado mais o interesse neste campo devido a sua importância e impacto económicos.

Alguns exemplos de Programação de Horários muito abordados são:

- Programação de Escala de Pessoal (*Staff Scheduling ou Rostering)* que é o processo de construir escalas de trabalho para funcionários de forma que uma organização possa satisfazer a demanda por bens e/ou serviços. Primeiramente, deve-se determinar o número de funcionários, com qualificações próprias, necessários para corresponder à demanda dos serviços. Cada funcionário será alojado a um turno (*shift*) para que, em diferentes períodos do dia, exista a satisfação da referida demanda, e jornadas (*duties*) são associadas a cada turno. Todos os acordos das convenções colectivas de trabalho devem ser observadas durante esse processo.
- Programação de Horários em Escolas consiste em alojar de uma sequência de aulas entre professores e estudantes num período prefixo de tempo (normalmente uma semana), satisfazendo uma série de restrições de vários tipos. Um enorme numero de variantes deste tipo de problemas podem ser encontrados na literatura, que diferem ente si baseando se no

tipo de instituição envolvida (Universidade ou Escola Pré universitária) e o tipo de restrições. (Shaef, 1999)

- Programação de jogos de torneios desportivos que consiste na montagem de tabela de jogos de um campeonato onde existe um determinado número de equipas que deverão defrontar entre si, satisfazendo uma série de requisitos impostos.

A construção manual do horário é uma tarefa árdua e normalmente requer vários dias de trabalho e a solução obtida pode ser insatisfatória com relação a vários aspectos. É extremamente difícil encontrar boas soluções para problemas dessa natureza por serem complexos e repletos de restrições, e mais difícil ainda é determinar a melhor solução que corresponda às preferências de todos os intervenientes.

Essa dificuldade é devida ao facto de haver grande número de combinações, o que inviabiliza uma resolução por enumeração explícita de todas as soluções possíveis. Essa intratabilidade computacional se deve ao facto de o problema de programação de pessoal ser NP-difícil (Even, e tal, 1976), uma vez que o problema de particionamento de conjuntos (*Set Partioning Problem*) e o problema de recobrimento de conjuntos (*Set Covering Problem*) que são NP-difíceis(CAREY;JONHSON,1979), podem ser reduzidos a esse tipo de problema.

Sendo assim, o uso exclusivo de técnicas de programação matemática, ditas exactas, limita-se a resolver problemas de pequenas dimensões. Para problemas de dimensões maiores, a abordagem mais comum é através de heurísticas. Em geral, as características próprias de diferentes indústrias e organizações determinarão quais os modelos matemáticos e algoritmos específicos para a programação de pessoal devem ser desenvolvidos.

Existem vários tipos de Problemas de Programação de Horários, porém este trabalho irá focalizar-se em Programação de Horários em Instituições escolares, mais particularmente os Horários dos Alunos ou Estudantes.

Sendo assim, como forma de ajudar a programação dos horários escolares e permitir uma melhor planificação do ano escolar nas instituições escolares, pretende-se desenvolver uma

ferramenta de programação de horários que permita gerar horários dos alunos em pouco tempo e satisfazendo um conjunto de restrições.

1.1 Objectivos

1.1.1 Objectivo Geral

Desenvolver um Sistema de Programação de Horários dos Alunos para a Escola Secundária Francisco Manyanga (ESFM).

1.1.2 Objectivos Específicos

- Avaliar os constrangimentos existentes no processo actual de programação de horários de alunos na ESFM;
- Conceber o modelo de Sistema de Programação de Horários de Alunos na ESFM;
- Avaliar a aplicabilidade da metodologia na concepção do modelo;
- Implementar e testar o Sistema de Programação de Horários de Alunos na ESFM.

1.2 Metodologia de Trabalho

Para avaliar os constrangimentos existentes no Sistema de Registo Académico actual, recorreu-se à:

- Entrevista a pessoa envolvida na programação de Horários de alunos na Escola Secundária Francisco Manyanga. Foi entrevistado o Professor Raúl Germano, responsável da Sala de Informática e responsável também pela programação dos horários. Foram feitas entrevistas não estruturadas que consistiam na captura de informação ao longo do desenvolvimento do sistema, tendo-se entrevistado, por vezes, mais de uma vez a mesma pessoa como forma de obter a mais correcta informação sobre a programação dos horários.

Para conceber o modelo para o Sistema de Programação de Horário de ESFM foi necessário:

- Pesquisar a bibliografia para o estudo e domínio das ferramentas a serem usadas;
- Estudar as meta-héuristicas *GRASP* e Busca Tabu usadas para a programação de horários e implementar um algoritmo híbrido na base destes, adaptável às características da ESFM;
- Estudar a metodologia *Extreme Programming/Desenvolvimento Ágil orientado ao Modelo* e, com base nessa metodologia fazer a análise e desenho do Sistema de Programação de Horários.

Para avaliar a aplicabilidade da metodologia na concepção do modelo, foi necessário:

- Fazer uma pesquisa bibliográfica, por forma a avaliar se as ferramentas, tecnologias e técnicas escolhidas para a concepção do sistema em estudo, têm vantagens e respondem às necessidades do sistema que se pretende desenvolver em relação às outras.

Para implementar e testar o Sistema de Programação de Horários de Alunos na ESFM, foi necessário:

- Organizar a plataforma e os recursos necessários (*hardware* e *software*) para o sustento do modelo construído;
- Implementar o Algoritmo Híbrido para a programação de horários em Java
- Produzir os artefactos definidos na altura da análise e desenho do sistema
- Documentar o sistema – produziu-se a documentação do sistema e um manual do utilizador para facilitar o uso e a manutenção do mesmo;
- Testar separadamente os módulos durante o desenvolvimento.

Este trabalho está divido em 6 capítulos. O primeiro faz um enquadramento lógico e contextual do trabalho, identificando os objectivos e a metodologia que será usada no trabalho para atingir tais objectivos e faz também uma abordagem ao problema de programação de horários. O segundo capítulo aborda os algoritmos usados para a criação e optimização dos horários. O terceiro capítulo aborda o processo actual de elaboração de horários na Escola Secundária Francisco Manyanga. O quarto capítulo fala de algumas tecnologias usadas para o desenvolvimento do Sistema. O quinto capítulo usa a metodologia *Extremme Programming Desenvolvimento Ágil orientado ao Modelo /*

para modelar e implementar o sistema tendo em conta o caso em estudo. As conclusões e recomendações são descritas no último capítulo.

1.3 Classificação do Problema de Programação de Horários em Escolas

(Shaef, 1999), classifica os problemas de programação de horários em 3 categorias principais, de acordo com o tipo de instituição e restrições que aparecem:

Problema de programação de horários em escolas

Este problema diz respeito à alocação de aulas de uma instituição com as características de uma escola secundária típica. Basicamente, existe um conjunto de turmas, um conjunto de professores e um conjunto de horários reservados a realização de aulas, As turmas são conjuntos disjuntos de estudantes que têm um mesmo currículo. Para cada turma há um conjunto de disciplinas, com suas respectivas cargas horárias, que devem ser cursadas. Para cada professor especifica-se a disciplina ou disciplinas, bem como as turmas para as quais o professor leccionará. O objectivo básico é fazer um quadro de horário, em geral semanal, de tal forma que:

As cargas horárias de todas as disciplinas e de todas as turmas sejam cumpridas;

Cada turma não tenha aula com mais de um professor ao mesmo tempo.

Um professor não dê aula para mais de uma turma em um mesmo horário.

Uma característica importante dos problemas desta categoria diz respeito à rigidez dos horários que são disponibilizados para a realização de aulas. Em geral, as aulas são realizadas somente em um dos turnos do dia; no entanto, quando o número de horários de um turno não suportar o número de aulas exigida pelo currículo, disponibiliza-se mais uma parte do outro turno suficiente para atender ao número requerido. Além dos mais, as aulas têm, normalmente, a mesma duração e, eventualmente, é desejável que algumas delas tenham durações diferentes. Outra característica adicional desses problemas é que, como as turmas são conjuntos disjuntos de estudantes, elas recebem as aulas em uma mesma sala (excepto para aulas de disciplinas que exigem salas especializadas como, Desenho). Desta forma, são os professores que se deslocam para leccionar em cada turma.

Problema de programação de horários de cursos

Este problema diz respeito à alocação das aulas de uma instituição com as características de uma universidade típica. Basicamente,s há um conjunto de disciplinas e, para cada curso, um certo número de aulas. Há, do mesmo modo, um conjunto de cursos. Cada curso envolve um conjunto de disciplinas. Os estudantes matriculam-se em turmas das disciplinas dos seus cursos. Há, também, um conjunto de horários disponibilizados para a realização de aulas e, em cada horário, um número limitado de salas. O problema consiste em alocar as aulas dos cursos aos horários disponibilizados, respeitando as disponibilidades e capacidades das salas existentes, de forma que nenhum estudante tenha duas ou mais aulas simultaneamente. Uma característica dos problemas desta categoria é que, regra geral, ao contrário dos problemas de horários em escolas, há maior flexibilidade em relação aos horários disponibilizados para a realização das aulas. Isto é, a princípio, uma disciplina pode ser alocada a qualquer horário de funcionamento da instituição, o que, em geral, inclui os períodos da manhã, da tarde e da noite. Outra característica diz respeito à configuração das aulas das disciplinas. Diferentemente da programação de horários em escolas, as aulas são agrupadas, de uma forma rígida, em sessões. Como o conceito de turma no problema de programação de horários de cursos é diferente do de escolas, em geral são os estudantes que se deslocam para terem as aulas.

Problema de programação de horários de exames

Este problema diz respeito à alocação dos exames de uma instituição com as características de uma universidade típica. Há um conjunto de estudantes (os quais estão matriculados em disciplinas), um conjunto de exames para cada estudante e um conjunto de horários disponibilizados para a realização dos exames. O objectivo primário é alojar cada exame a um horário, de forma que nenhum estudante tenha que fazer dois ou mais exames simultaneamente. Apesar de similar o problema de programação e horários de cursos, eles se distinguem sobretudo pela natureza das restrições envolvidas. Entre as restrições típicas que aparecem nesta categoria de problemas, destacamos: nenhum estudante pode fazer mais do que um certo número de exames por dia, exames de certas disciplinas não podem preceder a exames de outros cursos, alguns exames têm que ser realizados em um mesmo horário, certos exames devem ser realizados em horários consecutivos, etc.

Segundo (Sousa, 2000), esta classificação não é absoluta, no sentido que existem problemas que não se enquadram de forma precisa nestas categorias. Por exemplo, existem escolas secundárias europeias que dão a liberdade aos alunos para escolherem, nos dois últimos anos, as disciplinas a serem cursadas. Neste caso, o problema assemelha-se ao da programação de horários de cursos. Uma classificação mais abrangente e independente do tipo de instituição é dada por *Bardadym*. No entanto, a classificação apresentada anteriormente é a mais usada na literatura.

Desta forma segundo a classificação de (Shaef, 1999), o problema que abordaremos neste trabalho enquadra-se no Problema de Programação de Horários em Escolas.

2 ALGORITMOS USADOS

2.1 Introdução

Muitos problemas práticos são modelados da seguinte forma: dado um conjunto S de variáveis discretas s (chamadas soluções) e uma função objectivo $f: S \rightarrow R$, que associa cada solução $s \in S$ a um valor real (s), encontre a solução $s^* \in S$, óptima, para a qual $f(s)$ é mínima (ou máxima). Grande parte desses problemas é classificada como NP-difíceis, isto é, são problemas para os quais não existem algoritmos que resolvam em tempo polinomial. Tais problemas são enquadrados como "problemas de optimização combinatória".

É possível dar uma certa "inteligência" a um método de enumeração, utilizando por exemplo as técnicas "*branch and bound*"[1] ou "*branch and cut*", de forma a reduzir o número de soluções a analisar no espaço de busca. Com isto, pode ser possível resolver problemas de dimensões um pouco mais elevadas. Entretanto, certamente haverá uma dimensão acima da qual o problema se torna intratável computacionalmente.

Portanto, em problemas dessa natureza, o uso de métodos exactos se torna bastante restrito. Por outro lado, na prática, em geral, é suficiente encontrar uma "boa" solução para o problema, ao invés do óptimo global, o qual somente pode ser encontrado após um considerável esforço computacional.

[1] *Branch and bound* – Segundo Eric W. Weisssteín, é uma técnica algorítmica para uma solução óptima conservando a melhor solução distante. Se a solução parcial for melhor solução até então, ela será abandonada.

Este é o motivo pelo qual os pesquisadores têm concentrado esforços na utilização de *heurística*[2] para solucionar problemas deste nível de complexidade.

O desafio é produzir, em tempo reduzido, soluções tão próximas quanto possível da solução óptima. Muitos esforços têm sido feitos nessa direcção e heurísticas muito eficientes foram desenvolvidas para diversos problemas. Entretanto, a maioria dessas heurísticas desenvolvidas é muito específica para um problema particular, não sendo eficientes (ou mesmo aplicáveis) na resolução de uma classe mais ampla de problemas. Somente a partir da década de 80 se intensificaram os estudos no sentido de se desenvolver procedimentos heurísticos com uma certa estrutura teórica e com carácter mais geral, sem prejudicar a principal característica destes, que é a flexibilidade.

As primeiras heurísticas idealizadas para a resolução deste tipo de problemas eram heurísticas construtivas.

2.2 Formulação do Problema

O problema de programação de horários em escolas aqui abordado consiste em um conjunto de m professores de n turmas, s disciplinas e p horários semanais reservados para a realização das aulas. Os p horários semanais são distribuídos em d dias de semana de h horários diários realizados em um único turno, isto é, $p = d \times h$. As turmas, as quais estão sempre disponíveis, são conjuntos disjuntos de estudantes que cursam as mesmas disciplinas. A cada disciplina de uma turma está associando um único professor, previamente fixado. Além disso, a carga horária de cada turma é exactamente p.

Os seguintes requisitos, relativos ao quadro de horários dos professores, devem ser satisfeitos:

(a) Sobreposição de turmas

[2] *Héuristica* – Técnica que procura boas soluções (próximas da óptima) a um custo computational razoável, sem, contudo, estar capacitada a garantir a *optimalidade*, bem como garantir quão próximo uma determinada solução está da solução óptima para um determinado problema com uma determinada função objectivo.

Um professor não pode ministrar uma aula para mais de uma turma ao mesmo tempo.

(b) Sobreposição de professores

Uma turma não pode ter aula com mais de um professor em um mesmo horário.

(c) Carga horária

Todo o professor tem que cumprir a sua carga horária semanal de ensino estabelecido para cada uma das turmas.

(d) Indisponibilidade:

Um professor não pode ser alocado a um horário no qual não esteja disponível.

(e) Número máximo de aulas diárias por turma

Uma turma não pode ter mais do que 2 horários diários de aula de uma mesma disciplina. No caso de Educação Física, limita-se a uma aula diária.

(f) Número de aulas duplas

Para um dado conjunto de disciplinas, os professores requerem que, se possível, as aulas obedeçam a uma certa configuração semanal. Por exemplo, em uma disciplina de carga horária semanal de 3 horas, a configuração solicitada pode ser de uma aula dupla (ministrada em dois horários consecutivos) e outra ministrada em um único horário, ou o contrário.

(g) Complexidade do quadro de horário

A agenda dos professores deve ser tão compacta quanto possível.

Um quadro de horário de professores que viola pelo menos um dos requisitos (a),...,(e), é considerado *inviável* e não pode ser praticado pela escola.

De acordo com os requisitos (a),...,(e) podemos encontrar dois tipos de inviabilidades :

1. Inviabilidade do tipo 1 – um horário de professores se diz inviável do tipo 1 se, em pelo menos um horário, uma das condições a seguir for satisfeita:
    a. Existe uma turma tendo aulas com mais de um professor, isto é, a restrição (b) não é verificada;
    b. Existe turma sem aula.

2. Inviabilidade do tipo 2 – um horário de professores se diz inviável do tipo 2 se a restrição (e) não for satisfeita por algum professor.

2.3 Representação do problema

Um horário de professores é representado como uma matriz $Q_{m\times p}$ de valores inteiros, onde cada linha i de Q representa a alocação semanal do professor i. Cada elemento $q_{ik} \in \{-1,0,1,2,\dots,n\}$ indica a actividade do professor i no horário k. Valores negativos indicam que o professor está indisponível, enquanto valores nulos indicam inactividade no horário.

2.4 Estrutura de vizinhança

Entende-se por *vizinho* de uma solução s a solução s' que sofreu modificação m. Representa-se essa operação por $s' \leftarrow s \oplus m$.

Mais especificamente seja S o espaço de pesquisa de um problema de optimização. A função N, a qual depende da estrutura do problema tratado, associa a cada *solução* viável $s \in S$, sua *vizinhança* $(s) \subseteq S$, ou seja, todos as soluções s' geradas a partir do movimento m. Cada *solução* $s' \in (s)$ é chamada de *vizinho* de s.

Um movimento consiste na simples troca de dois valores distintos e não negativos de uma dada linha de Q. Tal movimento é identificado pela tripla $\langle i, k, k' \rangle$, onde k e k' representam os horários nos quais as actividades q_{ik} e $q_{ik'}$ do professor i serão permutadas, sendo $q_{ik} \neq q_{ik'}$ e $q_{ik} \geq 0$ e $q_{ik'} \geq 0$.

Observamos que esse tipo de movimento pode produzir inviabilidade do tipo 1 e/ou inviabilidade do tipo2, isto é, ao permutar os horários de duas actividades de um mesmo professor, permite-se que sejam geradas as seguintes inviabilidade:

a) O professor seja alocado a uma turma que já tem um outro professor previamente alocado;
b) Uma turma fique sem aula (no caso de uma das actividades representar uma turma que tem aula em um dos horários somente com o professor em questão);
c) O professor ultrapasse o limite diário de aulas da matéria sob sua responsabilidade.

Entretanto, a possibilidade de um professor ensinar simultaneamente para mais de uma turma (violação à restrição a)) é automaticamente rejeitada pela representação escolhida.

2.5 Função de Avaliação

A avaliação de um quadro de horários é também chamada de função objectivo que leva em consideração uma variedade de factores relativos aos professores, as turmas e às exigências de cada instituição de ensino, neste caso, a ESFM.

A função de avaliação é construída de forma usual, o conjunto de requisitos listados em 4.3 é dividido em subconjuntos representando uma função linear de penalidade, assegurando uma importância maior às restrições consideradas essenciais.

A função de avaliação, como o seu nome diz, é de avaliar um quadro de horários com relação a qualidade e à viabilidade, buscando medir a diferença entre um determinado quadro e outro, ideal de horário (um quadro de horário fictício que satisfaz todos os requisitos definidos em 4.3.)

Um quadro de horário Q é avaliado com base na seguinte função objectivo, a qual deve ser minimizada:

$$f(Q) = \omega \times f_1(Q) + \delta \times f_2(Q) + \rho \times f_3(Q) \text{[Equação 2-1]}$$

Onde:

$f(Q)$ — o valor da função de avaliação para o quadro Q.

$f_1(Q)$ — o valor da função de avaliação de inviabilidade do tipo 1, chamada também função de avaliação dos requisitos essenciais.

$f_2(Q)$ — o valor da função de avaliação de inviabilidade do tipo 2.

$f_3(Q)$ — o valor da função de avaliação do nível de satisfação dos professores com relação ao atendimento de seus requisitos pessoais.

ω, δ, ρ — são pesos que reflectem a importância de f. de forma a gerar uma estrutura hierárquica, tais pesos são escolhidos satisfazendo a condição: $\omega > \delta \gg \rho$

2.5.1 Avaliação da inviabilidade do tipo 1

O nível de inviabilidade do tipo 1 do quadro Q, ou nível de requisitos essenciais atendidos do quadro Q, é mensurado somando-se, para cada horário k:

a) O número de vezes l_k que uma turma está sem actividade em k;
b) O número de vezes $s_k$que mais de um professor dá aula para uma mesma turma no horário k.

$f_1(Q) = \sum_{k=1}^{p}(l_k + s_k)$ *[Equação 2-2]*

2.5.2 Avaliação da inviabilidade do tipo 2

Com relação ao nível de inviabilidade do tipo 2, o quadro Q é avaliado somando-se o número de vezes e_i que a restrição (e) não é atendida para cada professor i, isto é:

$f_2(Q) = \sum_{i=1}^{m} e_i$*[Equação 2-3]*

2.5.3 Avaliação dos requisitos pessoais

A satisfação ao atendimento dos requisitos pessoais dos professores é medida com relação a complexidade do quadro de horários (restrição (g)), bem como ao atendimento do número de aulas duplas requeridas (restrição (f)). Mais precisamente :

$f_3(Q) = \sum_{i=1}^{m}(\alpha_i \times B_i + \beta_i \times V_i + \gamma_i \times D_i)$*[Equação 2-4]*

Onde:

$f_3(Q)$ — o valor da função de avaliação dos requisitos pessoais;

B_i — número de buracos[3];

V_i — número de dias na semana que cada professor está envolvido em alguma actividade de ensino em um mesmo turno;

D_i — diferença não negativa entre o número mínimo requerido de aulas duplas e o efectivamente existente na agenda do professor i.$\left(D_i = max\{0, duplas(Q_i^{requerido}) - duplas(Q_i^{corrente})\}\right)$.

[3] Buracos são horários ociosos entre dois horários de aula de um mesmo turno

2.6 Procedimento Intraturmas-Interturmas

O II é um procedimento baseado em caminhos mínimos que é accionado quando a solução sem inviabilidade do tipo 1 está disponível.

O II subdivide-se em 2 fases:

1. Procura restaurar a viabilidade do quadro de horário, isto é, colocar a zero a parcela f_2 da função da avaliação.
2. Se bem sucedido, tenta melhoriar os requisitos de qualidade exigidos para o quadro de horário, navegando no espaço das soluções viáveis, isto é, tenta diminuir o valor da componente f_3 respeitando as restrições (a),...(e).

O II é um procedimento melhoriado do proposto por (Alvarez-Valdes, Martin, & Tamarit, 1996).

Como a forma de actuação de II, tanto para restaurar a viabilidade quanto para melhoriar o quadro de horário viável semelhante, mostraremos o seu princípio de funcionamento apenas para o segundo caso.

2.7 Procedimento Intraturmas

Suponhamos, então, que está disponível uma solução sem inviabilidade.

Assim, dado um quadro Q de horário de professores nessas condições $(f_1(Q) = f_2(Q))$, definimos o grafo da turma j por $G_j = (V_j, A_j)$, onde V_j é o conjunto dos horários reservados para a turma j, e A_j é o conjunto de arcos orientados definido conforme a seguir:

$A_j = \{k, \bar{k}$: a aula da turma j lecciona no horário k que pode ser leccionada também no horário $\bar{k}$ sem violar os requisitos (a),...(e) descritos em 4.2$\}$

Da construção do grafo G_j deduz-se que para um arco orientado $(k, \bar{k})$ pertencer ao conjunto A_j as seguintes condições precisam de ser atendidas:

i. O professor do horário k precisa estar disponível no horário $\bar{k}$;

ii. As restrições relativas à aula do horário k precisam de estar, também, satisfeitas no horário $\bar{k}$.

A cada arco $(k, \bar{k}) \in G_j$ associados um a custo $\Delta f_i(k, \bar{k})$, o qual representa a variação do custo de se transferir o professor i do horário k para o horário k para o horário $\bar{k}$, tendo em vista a componente f_3 da função de avaliação. Desta forma, o custo é obtido calculando-se a diferença entre os valores da função objectivo, relativa ao professor, nas configurações nova e antiga, isto é:

$\Delta f_i(k, \bar{k}) = f_i(k) - f_i(\bar{k})$ *[Equação 2-5]*

Sendo $f(.)\Delta = (\rho * f_3)(.)$

A tabela 2.1 mostra um fragmento de um quadro Q_1 de horários de professores. Cada linha i representa um professor $(i = P1, P2, P3, P4)$, e cada coluna k um horário $(k = H1, H2, H3, H4, H5)$ de um mesmo dia. Cada elemento q_{ik} desta tabela representa a actividade do professor i no horário k. A, B, C e D são turmas.

Um traço (—) significa que o professor está indisponível, enquanto uma célula vazia indica que não há actividade no horário. A coluna f_i indica o valor da função de avaliação de cada professor, calculada conforme 4.1 e 4.2, com $\rho = 1, \alpha_i = 1$ e $\beta_i = \gamma_i = 0\ \forall i$. Para esse quadro de horário, tem-se $f(Q_1) = f_{P1} + f_{P2} + f_{P3} + f_{P4} = 1 + 1 + 0 + 0 = 2$.

Tabela 2-1: Quadro Q_1

	H1	H2	H3	H4	H5		f_i
P1	A		B	B			1
P2	B	C		A	A		1
P3		B	A	C	B		0
P4	C	A	C	D	—		0

A **Error! Reference source not found.** mostra o grafo G_A de horários da turma A. Cada horário representado por um vértice, ao qual está associado um professor. O arco $(H1, H5)$ de custo -1

indica, por exemplo, que se o professor $P1$ mudar sua aula do horário $H1$ para o horário $H5$, haverá uma diminuição no valor da sua função objectivo de 1 unidade ($\Delta f_{P1}(H1, H5) = f_{P1}(H5) - f_{P1}(H1) = 0 - 1 = -1$).

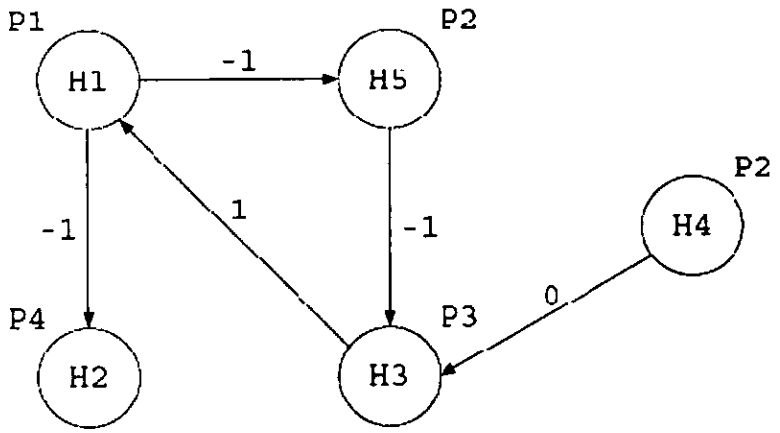


Figura 2-1: G_A da turma A

Para encontrar um quadro de horário com um valor menor para a função objectivo 4.1, é suficiente procurar um ciclo de custo negativo em G_j. Observa-se que a existência de um ciclo garante a realocação de todas as aulas nele envolvidas, preservando o atendimento aos requisitos (a),…,(d). O atendimento ao requisitos (e) será discutido mais adiante.

No exemplo em questão, a sequência de arcos $\{(H1, H5), (H5, H3), (H3, H1)\}$ forma um ciclo de custo total $-1 (= -1 + (-1) + 1)$. Tal sequência, por envolver aulas de uma mesma turma, define um conjunto de movimentos ditos *intraturmas*.

A tabela 2.2 mostra Q_1', o quadro de horários dos professores após os movimentos e a Figura 4.2 ilustra G_A', o novo grafo de horários da turma A. Em função dos movimentos promovidos em G_A , o novo quadro Q_1' tem função objectivo dada por $f(Q_1') = f_{P1} + f_{P2} + f_{P3} + f_{P4} = 0 + 0 + 1 + 0 = 1$.

	H1	**H2**	**H3**	**H4**	**H5**		f_i
P1			B	B	A		0
P2	B	C	A	A			0

P3	A	B		C	B		1
P4	C	A	C	D	—		0

Tabela 2-2:Quadro Q_1'

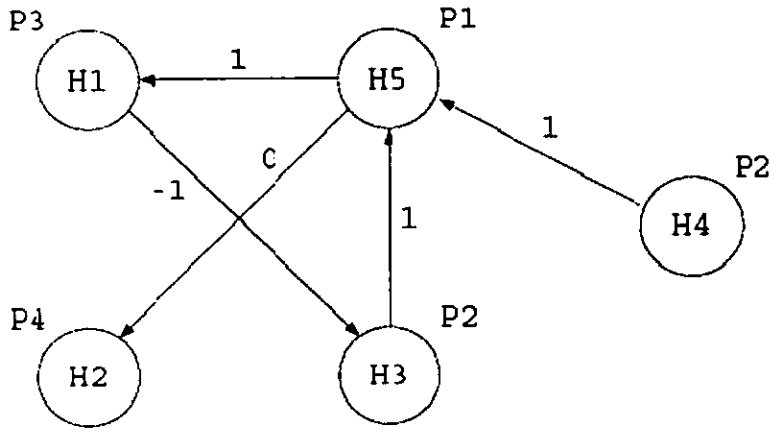


Figura 2-2: Grafo G_A' da Turma A após os movimentos

Após a actualização do quadro de horário e do grafo correspondente da turma A, procurarmos pela existência de novos ciclos de custo negativo.

Como pode ser visto pela Figura 2.2, não existem mais tais ciclos no grafo G_A' do exemplo considerado. A ideia, então, é repetir o procedimento para uma outra turma e, assim, sucessivamente, até que não seja mais possível melhoriar o quadro de horário dos professores através de movimentos intraturmas.

A existência de um ciclo de custo negativo pode, todavia, não garantir melhor no valor da função objectivo, assim como pode gerar soluções inviáveis do tipo 2, conforme se exemplifica a seguir.

A Tabela 2-3 representa o fragmento de um quadro Q_2 de horários de professores, de 2 dias com 5 horários cada um. A função objectivo f é avaliada conforme 2.1 e 2.3, com $\rho = 1, \alpha_i = 1$ e $\beta_i = \gamma_i = 0\ \forall i$.(Lembre-se que estamos partindo do pressuposto que um quadro viável está disponível, isto é, que $f_1(Q_2) = f_2(Q_2) = 0$). A este quadro de horário está associada uma função objectivo com valor $f(Q_2) = f_{P1} + f_{P2} + f_{P3} + f_{P4} = 5 + 4 + 4 + 2 = 15$.

A Figura 4.3, que ilustra o grafo de horários da turma A relativo a este quadro, mostra que existe um ciclo de custo $-1(= -1 + 0 + 0 + 0)$.

	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		f_i
P1	A		A	B		A						5
P2		A						B	B	A		4
P3				A	A		A					4
P4								A	A			2

Tabela 2-3: Quadro Q_2

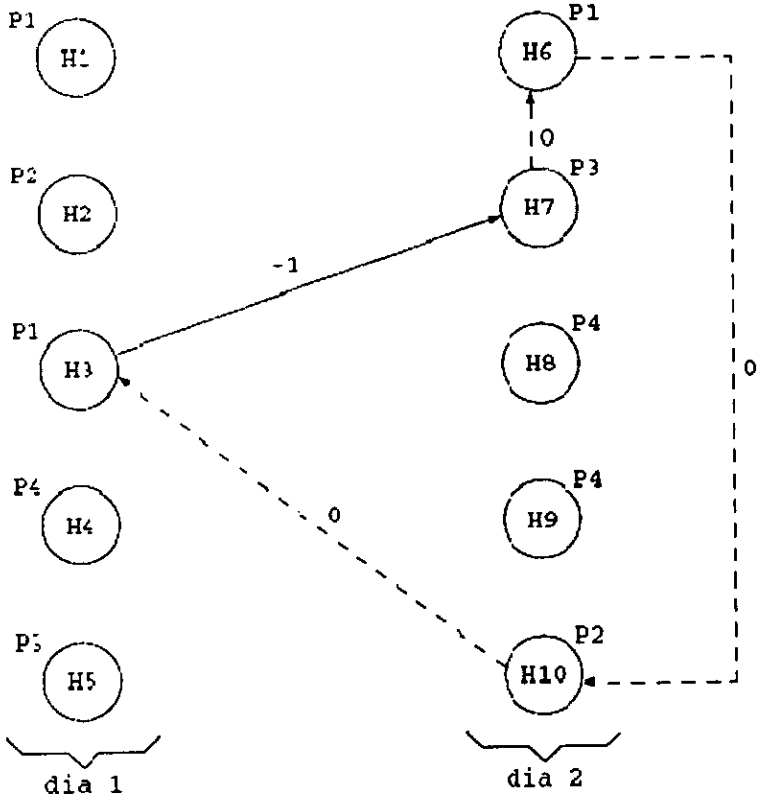


Figura 2-3: Exemplo de ciclo de custo negativo que não representa melhoria no valor da função objectivo

O quadro Q_2' resultante (Tabela 2-4) mostra que, na realidade, o valor da função objectivo aumenta em uma unidade ($f(Q_2') = f_{P1} + f_{P2} + f_{P3} + f_{P4} = 6 + 4 + 4 + 2 = 16$), ao invés de diminuir de uma unidade, conforme se esperava.

Outra situação que pode ocorrer está exemplificada na Figura 4.4. Supondo que cada professor possa ministrar, no máximo, dois horários de aula por dia para cada turma, são possíveis os movimentos das aulas dos horários $H6$ e $H11$ para o dia 1, tomados individualmente. No entanto, se os arcos correspondentes a esses movimentos fizerem parte de um mesmo ciclo de custo negativo, tal como o mostrado na Figura 2-4: Exemplo de ciclo de custo negativo que produz inviabilidade do tipo 2., a viabilidade não estará garantida.

	H1	**H2**	**H3**	**H4**	**H5**	**H6**	**H7**	**H8**	**H9**	**H10**		f_i
P1	A											6
P2		A	A									4
P3				A	A	A						4
P4												2

Tabela 2-4: Quadro Q_2'

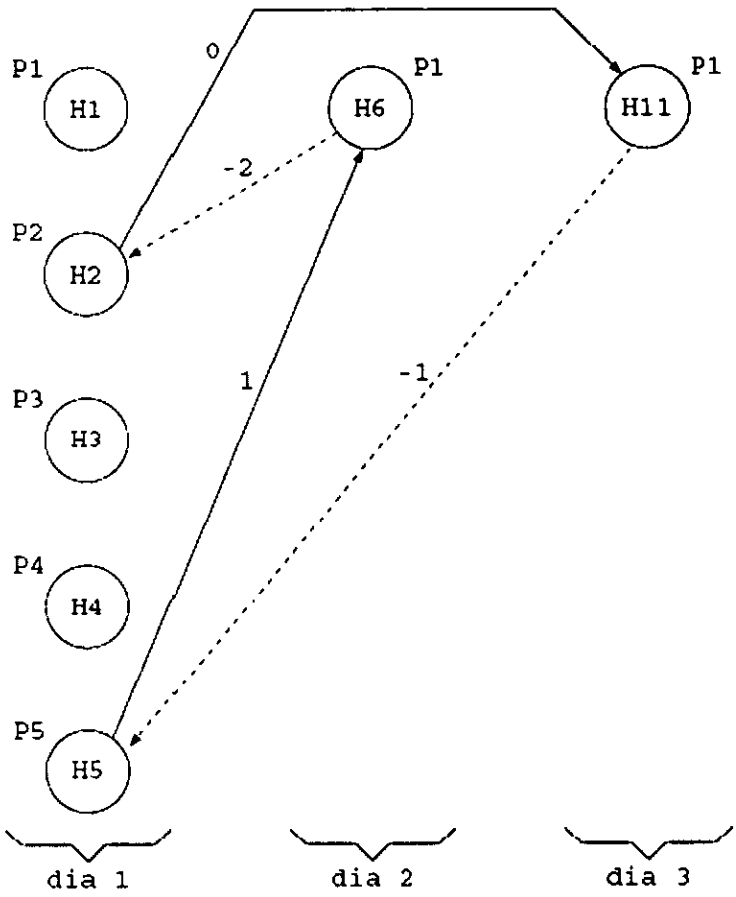


Figura 2-4: Exemplo de ciclo de custo negativo que produz inviabilidade do tipo 2.

Essas situações, entretanto, só ocorrem quando um mesmo professor participa em mais de um arco no ciclo. Assim, faz-se necessário verificar a viabilidade e o valor da função objectivo após os movimentos candidatos. De forma a encontrar outros ciclos de custo negativo em um grafo nessas condições, procedemos como se segue:

Escolhemos um arco qualquer do ciclo, $(k, \bar{k}) \in G_j$, e o inserimos em uma lista L de movimentos proibidos. A seguir, actualizamos o grafo da turma sob avaliação, excluindo de G_j os arcos pertencentes a L, e procuramos outro ciclo de custo negativo. Quando não mais for possível encontrar ciclos de custo negativo, passamos para uma outra turma e limpamos a lista L.

Podemos usar um procedimento iterativo que analisa, a cada vez, uma turma. Para cada turma j construímos seu grafo G_j e aplicamos um algoritmo para detectar ciclos de custo negativo. Enquanto houver ciclo de custo negativo que melhore o valor da função objectivo e não produza inviabilidade, realizamos os movimentos, actualizamos o quadro de horários e o grafo G_j. Quando já não existirem tais ciclos no grafo da turma j, partimos para uma nova turma. Esse procedimento é encerrado quando nenhum movimento de melhoria for possível para todas as turmas.

O procedimento *Intraturmas*, descrito na Figura 2-5:

Procedimento *Intraturmas(Q,f(.))*

1. $j \leftarrow 1$;{Número da turma}
2. *novarodada*← ***false***
3. **Enquanto** ($j \leq n$) **faça**
4. **enquanto** *(houver ciclo de custo negativo em* G_j
5. *que melhore função objectivo*
6. *e não viole viabilidade)* **faça**
7. *Actualize* G_j *e o Quadro de Horário da turma j;*
8. *novarodada*←***true****;*
9. **fim-enquanto***;*
10. **se** *(novarodada=true e* $j = n$*)* **então**
11. $j \leftarrow 1$*;*
12. *novarodada*← ***false****;*
13. **senão**
14. $j \leftarrow j + 1$*;*
15. **fim-se**
16. **fim-enquanto**

Figura 2-5: Pseudocódigo do Algoritmo Procedimento Intraturmas

2.7.1 Procedimento Interturmas

Ao final do procedimento Intraturmas podem restar ainda arcos de custo negativo nos grafos das turmas. A Figura 6.6, que considera os grafos de duas turmas, j e $\bar{j}$, ilustra tal situação.

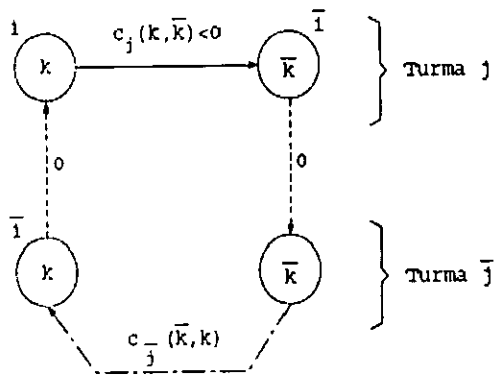


Figura 2-6: Grafos das turmas j e $\bar{j}$ *resultantes do procedimento Intraturmas.*

Nesta figura, há um arco de custo negativo na turma j, de k a $\bar{k}$, a saber $c_j(k,\bar{k})$, sinalizando que poderá haver uma melhoria no valor da função objectivo, se o professor i tiver sua aula do horário k transferida para o horário $\bar{k}$.

Esta transferência, entretanto, não pode ocorrer porque o professor $\bar{\imath}$, do horário $\bar{k}$, não está disponível no horário k (Neste horário, ele está a leccionar a turma $\bar{\jmath}$).

A ideia, então, é trocar o horário dessas aulas do professor $\bar{\imath}$, de forma a permitir a procura de um ciclo de custo negativo envolvendo os grafos das duas turmas, conectados nos horários k e $\bar{k}$.

Seja $c_{\bar{\jmath}}(k,\bar{k})$ o custo do caminho mínimo de $\bar{k}$ a k em $G_{\bar{\jmath}}$. A existência de um ciclo de custo negativo envolvendo as duas turmas pode ser observada verificando se a condição $c_j(k,\bar{k}) + c_{\bar{\jmath}}(k,\bar{k}) < 0$ é satisfeita ao transferir o professor $\bar{\imath}$, da turma j para a turma $\bar{\jmath}$ no horário $\bar{k}$, e da turma $\bar{\jmath}$ para a turma j no horário k (arcos de custo nulo da Figura 2-6, já que não há custos envolvidos na transferência de um professor de uma turma para outra em um mesmo horário).

Desta forma, para cada arco $(k,\bar{k}) \in G_j$, de custo negativo, pesquisamos a existência de um ciclo de custo negativo envolvendo esse arco de G_j, o grafo de $G_{\bar{\jmath}}$ e os arcos de custo nulo que os conectam. Esta sequência de arcos, por envolverem turmas diferentes, define os chamados movimentos *interturmas*. Assim como *Intraturmas*, faz-se necessário verificar a viabilidade e o valor da função objectivo antes e depois dos movimentos candidatos. O procedimento *Interturmas* esta descrito na Figura 2-7.

Procedimento *Interturmas(Q,f(.))*

1. $j \leftarrow 1$;{Número da turma}
2. *novarodada*← ***false***
3. **Enquanto** ($< n$) **faça**
4. **para** *(para todo arco* $(k, \bar{k}) \in G_j$ de custo negativo) **faça**
5. *Seja* $\bar{\imath}$ *o professor do horário* $\bar{k}$;
6. *Seja* $\bar{j}$, $j \neq j$,*a turma na qual* $\bar{\imath}$ *está em actividade no horário k;*
7. *Seja* $c_j(k, \bar{k})$ *o custo de se tranferir, na turma j,*
8. *a aula do horário k para o horário* $\bar{k}$;
9. *Seja* $c_{\bar{j}}(k, \bar{k})$ *o custo, na turma* $\bar{j}$, *do caminho mínimo entre* $\bar{k}$ *e k;*
10. **Se** $(c_j(k, \bar{k}) + c_{\bar{j}}(k, \bar{k}) < 0$
11. *e função objectivo melhorar*
12. *e viabilidade não for violada)* **então**
13. *Actualize* G_j *e o Quadro de horários da turma j;*
14. *Actualize* G_j *e o Quadro de horários da turma* $\bar{j}$;
15. *Chamar o Algoritmo Intraturmas;*
16. *novarodada*← ***true***
17. **fim-se**
18. **fim-para**
19. **se** *(novarodada=true e* $j = n$*)* **então**
20. $j \leftarrow 1$;
21. *novarodada*← *false;*
22. **senão**
23. $j \leftarrow j + 1$;
24. **fim-se**

Figura 2-7: Pseudocodigo do Procedimento Interturmas

Observamos que toda a vez que o procedimento Interturmas produz uma solução melhor, o procedimento Intraturmas é novamente accionado. Isso deve-se ao facto de que o quadro de horário dos professores poderá ser melhoriado com movimentos intraturmas, em função das alterações promovidas nos quadros de horário das turmas j e $\bar{j}$.

2.7.2 Procedimento II

O procedimento II consta de dois procedimentos descritos anteriormente:

1. O procedimento Intraturmas que realiza somente movimentos intraturmas e sua aplicação resulta em n grafos $G_j, \forall j = 1, \ldots, n$, com, possivelmente, arcos de custo negativo.
2. O procedimento Interturmas que realiza movimentos interturmas envolvendo esses arcos de custo negativo e acciona Intraturmas sempre que há melhoria no valor da função objectivo.

Conforme foi dito anteriormente, o procedimento II é accionado logo que haja uma solução sem inviabilidade do tipo 1. II actua em duas etapas: primeiramente, ele tenta recuperar a viabilidade. Sendo bem sucedido, ele tenta, em uma segunda etapa, melhoriar os requisitos de qualidade do quadro de horário. Na primeira etapa II, usa somente a componente f_2 da função objectivo como função avaliadora dos custos dos movimentos, isto é,. a expressão $f(Q) = \omega \times f_1(Q) + \delta \times f_2(Q) + \rho \times f_3(Q)$[Equação 2-1 é avaliada tomando-se $f(.) = (\rho \times f_3)(.)$, uma vez que $f_1(Q) = f_2(Q) = 0$.

O pseudocódigo do algoritmo é descrito na Figura 2-8.

Procedimento II*(Q,f(.))*

1. *Seja Q solução inicial satisfazendo* $f_1(Q) = 0$;
2. *Seja* Q^* *a melhor solução resultante de II;*
3. ***Se*** $(f_2(Q) \neq 0)$ ***então***
4. $Q \leftarrow$ *Intraturmas*(Q, f_2);
5. ***Se*** $(f_2(Q) \neq 0)$ *então* $Q \leftarrow$ *Interturmas*(Q, f_2);
6. ***fim-se;***
7. **se** $(f_2(Q) = 0)$ **então**
8. $Q \leftarrow$ *Intraturmas*(Q, f_3);
9. $Q \leftarrow$ *Intraturmas*(Q, f_3);
10. **fim-se;**
11. $Q^* \leftarrow Q$;
12. Retorne Q^*;

Figura 2-8: Pseudócodigo do Procedimento II

2.8 Geração da Solução Inicial

A solução inicial é gerada de forma construtiva, via um procedimento parcialmente guloso, conforme a seguir se descreve.

Inicialmente, determinamos os horários mais críticos, isto é, aqueles que têm o menor número de professores disponíveis. A seguir, enumeramos e ordenamos todas as aulas, dando prioridade aquelas mais difíceis de serem alocadas, ou seja, aquelas que têm os professores com maior carga horária e maior número de horários indisponíveis. Formamos, então, uma lista restrita de candidatos dessas aulas, de tamanho $|LCR|$, e seleccionamos, aleatoriamente, uma delas. A seguir, procuramos alojá-la (usando a ordem de horários críticos) de forma que não haja, a princípio, nenhum tipo de inviabilidade (no caso, violação às restrições (b) e (e) descritas em 2.2). Não sendo possível, admitimos, agora, violação somente à restrição (e). Se ainda assim a alocação for possível, a aula é alocada admitindo-se também à restrição (b). Toda a vez que uma aula é alocada, actualizamos os horários críticos, bem como as aulas mais difíceis remanescentes.

Observamos que, como todas as aulas são alocadas, ainda que com algum tipo de inviabilidade, a restrição (c) descrita em 2.2 é automaticamente verificada.

A Figura 2-10 apresenta o pseudocódigo correspondente.

Podemos ver também que o tamanho $|LCR|$ da lista de candidatos restrita e o valor inicial t permitido para o tipo de alocação $TipoAloc$ são os únicos parâmetros do procedimento $ConstrucaoSolucaoInicial$. Se tivermos $|LCR| = 1$ e $t = 3$, gera-se uma solução gulosa; se $|LCR|$ for igual ao número de aulas ainda a alocar e $t = 1$, será gerada uma solução aleatória.

2.9 Algoritmo Busca Tabu

A Busca Tabu é um procedimento adaptativo que utiliza uma estrutura de memória para guiar um método de descida a continuar a exploração do espaço de soluções mesmo na ausência de movimentos de melhoria, evitando que haja a formação de ciclos, isto é, o retorno a um óptimo local previamente visitado.

Mais especificamente, começando com uma solução inicial s_0, um algoritmo BT explora, a cada iteração, um subconjunto V da vizinhança $N(s)$ da solução corrente s. O membro s' de V com melhor valor nessa região segundo a função $(.)$ torna-se a nova solução corrente mesmo que s' seja pior que s, isto é, que $f(s') > f(s)$ para um problema de minimização. (Sousa, 2000)

O critério de escolha do melhor vizinho é utilizado para escapar de um mínimo local. Esta Estratégica, entretanto, pode fazer com que o algoritmo crie ciclos, isto é, que retorne a uma solução já gerada anteriormente.

De forma a evitar que isto ocorra, existe uma lista tabu T, que é uma lista de movimentos proibidos. A lista tabu clássica contém os movimentos reversos aos últimos $|T|$ movimentos realizados (onde $|T|$ é um parâmetro do método) e funciona como uma fila de tamanho fixo, isto é, quando um novo movimento é adicionado à lista, o mais antigo sai. Assim, na exploração do subconjunto V da vizinhança (s) da solução corrente s, ficam excluídos da busca os vizinhos s' que são obtidos de s por movimentos m que constam na lista tabu.

A lista tabu, se por um lado reduz o risco de formação de ciclos (uma vez que ela garante o não retorno, por $|T|$ iterações, a uma solução já visitada anteriormente), por outro lado também pode proibir movimentos que ainda não foram visitados (Sousa, 2000) citando (de Werra, 1989).

Assim, existe também uma *função de aspiração*, que é um mecanismo que retira, sob certas circunstâncias, o *status* tabu de um movimento. Mais precisamente, para cada possível valor v da função objectivo existe um nível de aspiração (v): uma solução s' em V pode ser gerada se

$f(s') < A(f(s))$, mesmo que o movimento m esteja na lista tabu. A função de aspiração A é tal que, para cada valor v da função objectivo, retorna outro valor (v), que representa o valor que o algoritmo aspira ao chegar à v. Um exemplo simples de aplicação desta ideia é considerar $A\big(f(s)\big) = f(s^*) - 1$, onde s^* é a melhor solução encontrada até então. Neste caso, aceita-se um movimento tabu somente se ele conduzir a um vizinho melhor que s^*. Esta é a chamada aspiração por objectivo. Esse critério fundamenta-se no facto de que soluções melhores que a solução s^* corrente, ainda que geradas por movimentos tabu, não foram visitadas anteriormente, evidenciando que a lista e movimentos tabu podem impedir, não somente o retorno a uma solução já gerada anteriormente. mas também a outras soluções ainda não geradas.

Duas regras são normalmente utilizadas de forma a interromper o procedimento. Pela primeira, pára-se quando é atingido um certo número máximo de iterações sem melhoria no valor da melhor solução. Pela segunda, quando o valor da melhor solução chega a um limite inferior conhecido (ou próximo dele). Esse segundo critério evita a execução desnecessária do algoritmo quando uma solução óptima é encontrada ou quando uma solução é julgada suficientemente boa.

Os parâmetros principais de controle do método de Busca Tabu são a cardinalidade $|T|$ da lista tabu, a função de aspiração A, a cardinalidade do conjunto V de soluções vizinhas testadas em cada iteração e $BTmax$, o número máximo de iterações sem melhoria no valor da melhor solução.

Apresenta-se, pela Figura 2-9, o pseudocódigo de um algoritmo de BT básico, segundo (Sousa, 2000). Neste procedimento, f_{min}é o valor mínimo conhecido da função f.

Procedimento $BT(f(.), N(.), A(.), |V|, f_{min}, |T|, BTmax, s)$;

1. $s^* \leftarrow s$; {Melhor solução obtida até então}
2. $Iter \leftarrow 0$; {Contador do número de iterações}
3. $MelhorIter \leftarrow 0$; {Iteração mais recente que forneceu s^*}
4. $T \leftarrow \emptyset$; {Lista Tabu}
5. Inicialize a função de aspiração A;
6. **enquanto** $(f(s) > f_{min}$ e $Iter - MelhorIter < BTmax)$ **faça**
7. $Iter \leftarrow Iter + 1$;
8. Seja $s' \leftarrow s \oplus m$ o melhor elemento de $V \subseteq N(s)$ tal que
9. o movimento m não seja tabu $(m \notin T)$ **ou**
10. s' atenda a condição de aspiração $(f(s') < A(f(s)))$;
11. Actualize a lista tabu T;
12. $s \leftarrow s'$;
13. **se** $(f(s) < (s^*))$ **então**
14. $s^* \leftarrow s$;
15. $MelhorIter \leftarrow Iter$;
16. **fim-se**;
17. Actualize a função de aspiração A;
18. **fim-enquanto**;
19. $s \leftarrow s^*$
20. Retorne s;

fim *BT*

Figura 2-9: Algortimo Busca Tabu

É comum em métodos de BT incluir estratégicas de intensificação, as quais visam concentrar a pesquisa em determinadas regiões consideradas promissoras. Uma Estratégica típica é retornar a soluções já visitadas para explorar a sua vizinhança de forma mais efectiva. Outra Estratégica consiste em incorporar atributos das melhores soluções já encontradas durante o progresso da pesquisa, e estimular componentes dessas soluções a tornar parte da solução corrente. Nesse caso, são consideradas livres no procedimento de busca local apenas as componentes não associadas às boas soluções, permanecendo as demais componentes fixas. Um critério de término, tal como um número fixo de iterações, é utilizado para encerrar o período de intensificação.

Métodos baseados em BT incluem, também, Estratégicas de diversificação. O objectivo dessas Estratégicas, que tipicamente utilizam uma memória de longo prazo, é redireccionar a pesquisa para regiões ainda não suficientemente exploradas do espaço de soluções. Estas Estratégicas procuram, ao contrário das Estratégicas de intensificação, gerar soluções que têm atributos significativamente diferentes daqueles encontrados nas melhores soluções obtidas. A diversificação, em geral, é utilizada somente em determinadas situações, como, por exemplo, quando a uma solução s, não existem movimentos m de melhoria para ela, indicando que o algoritmo já exauriu a análise naquela região. Para escapar dessa região, a ideia é estabelecer uma penalidade (s, m) para uso desses movimentos. Um número fixo de iterações sem melhoria no valor da solução óptima corrente é, em geral, utilizado para accionar essas Estratégicas.

Métodos de BT incluem, também, listas tabu dinâmicas (Sousa, 2000) citando (Dammeyer & Voß, 1993) e (Schaefer, 1996), muitas das quais actualizadas de acordo com o progresso da pesquisa (Sousa, 2000).

Segundo (Sousa, 2000) , provou-se a convergência finita de alguns métodos de BT baseados em memória por recenticidade e por frequência.

Para uso neste problema, usaremos uma variante do algoritmo BT usando o procedimento II descrito em 2.9.

Esta variante é uma sofisticação do algoritmo BT, idealizada em vista do bom desempenho de II no algoritmo básico de BT, e é usada em um procedimento GRASP (Feo & Resende, 1995), conforme será descrito mais adiante em 2.11.

Partindo de uma solução inicial, gerada pelo procedimento construtivo, a meta-heurística[4] BT segue, iterativamente, explorando toda a vizinhança (Q) da solução corrente Q, através de

[4] As *meta-heurísticas* são procedimentos destinados a encontrar uma boa solução, eventualmente a *óptima*, consistindo na aplicação, em cada passo, de uma heurística subordinada, a qual tem que ser modelada para cada problema específico (Ribeiro ,1996).

Contrariamente às heurísticas convencionais, as *meta-heurísticas* são de carácter geral e têm condições de escapar de *óptimos locais*.

movimentos definidos em 2.4 e guiados pela função objectivo definida pela expressão 2.2. O algoritmo segue, então, para o vizinho Q' que produzir o menor valor $f(Q')$, independentemente de ele ser pior que o valor $f(Q)$ corrente.

As *meta-heurísticas*, assim como os métodos de busca local tradicionais, diferenciam-se entre si basicamente pelas seguintes características:

a. Critério de escolha de solução inicial;
b. Definição de vizinhança *N(s)* de uma solução *s;*
c. Critério de selecção de uma solução vizinha dentro de *N(s)*, mecanismo usado para escapar de *óptimos locais*;
d. Critério de término.

Procedimento ConstrucaoSolucaoInicial

1. *Seja* (h_k) *o número de professores indisponíveis no horário* h_k*;*
2. *Seja HC o conjunto de* p *horários* h_k *ordenados segundo a sua*
3. *criticidade:*
4. $HC \leftarrow \{h_1, h_2, \dots, h_p\}$
5. $h_k \in HC\ satisfaz\ I(h_k) \geq I(h_{k+1})\ \forall k = 1, \dots, p-1;$
6. Seja (a_j) a carga horária do professor da j-ésima aula a_j
7. Somada ao número de seus horários indisponíveis;
8. Seja AD o conjunto das q aulas a_j ordenadas segundo sua
9. dificuldade de alocação:
10. $AD \leftarrow \{a_1, a_2, \dots, a_q\}$
11. $a_j \in AD$ satisfaz $CH(a_j) \geq CH(a_{j+1})\ \forall k = 1, \dots, q-1;$
12. Seja $Q^0 \leftarrow \emptyset$ o quadro de horário dos professores;
13. *Seja* $TipoAloc = \begin{cases} 3\ se\ a\ aula\ é\ alocada\ sem\ inviabilidade \\ 2\ se\ a\ aula\ é\ alocada\ com\ inviabilidade\ do\ tipo\ 2 \\ 1\ se\ a\ aula\ é\ alocada\ com\ inviabilidade\ do\ tipo\ 1 \end{cases}$
14. **enquanto**$(AD \neq \emptyset)$ **faca**
15. seja a_j uma das $|LCR|$ aulas mais difíceis
16. $t = 3;$
17. **repita**
18. $k = 1;$
19. **repita**
20. *se*(a_j *pode ser alocada ao horário* h_k *com TipoAloc t)*
21. **então** $Q^0 \leftarrow Q^0 \cup \{a_j \rightarrow h_k\}$
22. **senão** $k \leftarrow k + 1;$
23. **até** *(*$k > p$ **ou** a_j *seja alocada com Tipo Aloc t);*
24. **se** (a_j *não foi alocada com TipoAloc t*)
25. **então** $t \leftarrow t - 1;$
26. **até**(*que* a_j *seja alocada*);
27. $AD \leftarrow AD - \{a_j\};$
28. *Actualize e reordene os conjuntos AD e HC;*

Figura 2-10: Procedimento ConstrucaoSolucaoInicial

Segundo (Sousa, 2000) sempre que uma solução sem sobreposições é gerada (isto é, $f_1(Q) = 0$), accionamos o procedimento II, com a condição de que uma solução de um mesmo valor não é necessariamente igual. Entretanto, esta condição foi imposta por dois motivos:

1. Pela eficiência do procedimento II
2. Para não elevar desnecessariamente o tempo de processamento do algoritmo

(Sousa, 2000) considera duas variantes deste algoritmo, porém neste trabalho usaremos uma das variantes, BT-IIm, que a seguir descrevemos, pelo facto de se ter mostrado melhor desempenho.

Procedimento *BT-IIm(Q)*

1. *Obtenha uma solução inicial Q;*
2. $Q^* \leftarrow Q$; {Melhor solução obtida até então}
3. $Iter \leftarrow 0$; {Contador do número de iterações}
4. $MelhorIter \leftarrow 0$; {Iteração mais recente que forneceu Q^*}
5. $T \leftarrow \emptyset$; {Lista Tabu}
6. $BTmax \leftarrow$*Número máximo permitido de iterações*
7. *consecutivas sem melhora em* (Q^*);
8. Inicialize a função A de aspiração
9. **enquanto** $(Iter - MelhorIter < BTmax)$**faça**
10. $Iter \leftarrow Iter + 1$;
11. *Seja* $Q' \leftarrow Q \oplus m$ *o melhor elemento de* $N(Q)$ *tal que*
12. o movimento m não seja tabu $(m \notin T)$ **ou**
13. Q' atenda a condição de aspiração $(f(Q') < A(f(Q)))$;
14. Actualize a lista tabu T;
15. Actualize a função de aspiração A;
16. $Q \leftarrow Q'$;
17. **se** $(f_1(Q') = 0)$ **então**
18. $Q \leftarrow II(Q', f)$;
19. Actualize a lista tabu T com os movimentos reversos de II;
20. **fim-se**;
21. **se** $(f(Q) < (Q^*))$ **então**
22. $Q^* \leftarrow Q$;
23. $Melhor \leftarrow Iter$;
24. **fim-se**;
25. **fim-enquanto**
26. Retorne Q^*;

fim *BT-IIm*

Figura 2-11:Procedimento BT-IIm

O procedimento BT, chamado nesta versão por BT-IIm continua sua busca a partir da solução produzida por II. Neste caso, tendo em vista que II faz mudar a região pesquisada, incluímos na lista tabu T os movimentos reversos àqueles realizados por II caso, evidentemente, a busca tenha logrado êxito, isto é, os movimentos tenham sido de melhoria relativamente ao valor da solução corrente e nenhuma inviabilidade tenha sido gerada.

A Figura 2-11 mostra o pseudocódigo do algoritmo BT-IIm.

2.10 GRASP(Greedy Randomized Adaptive Search Procedure)

GRASP(Procedimento de busca adaptativa gulosa e aleatória) é um método iterativo, proposto por (Feo & Resende, 1995) , que consiste de duas fases: uma fase de construção, na qual a solução é gerada, elemento a elemento, e de uma fase de busca local, na qual um óptimo local na vizinhança da solução construída é pesquisado. A melhor solução encontrada ao longo de todas as iterações GRASP realizadas é retornada como resultado. O pseudo-código descrito pela Figura 2-12 ilustra procedimento GRASP

Procedimento $GRASP(f(.), g(.), N(.), GRASPMax, s)$;

1. $f^* \leftarrow \infty$;
2. **para** $(Iter = 1,2, \ldots, GRASPmax)$ **faça**
3. $Construcao(g(.), \alpha, s)$;
4. $BuscaLocal(f(.), N(.), s)$;
5. **se** $(f(s) < f^*)$ **então**
6. $s^* \leftarrow s$;
7. $f^* \leftarrow f(s)$;
8. **fim-se**;
9. **fim-para**
10. $s \leftarrow s^*$;
11. Retorne s;

fim GRASP;

Figura 2-12: Pseudo-código do algoritmo GRASP para um problema de minimização

Na fase de construção, uma solução é iterativamente construída, elemento por elemento. A cada iteração dessa fase, os próximos elementos candidatos a serem incluídos na solução são colocados em um lista C de candidatos, seguindo um critério de ordenação predeterminado. Esse processo de selecção é baseado em uma função adaptativa gulosa $g: C \mapsto \Re$, que estima o benefício da selecção de cada um dos elementos. A heurística é adaptativa porque os benefícios associados com a escolha de cada elemento são actualizados em cada iteração da fase de construção para reflectir as mudanças oriundas da selecção do elemento anterior. A componente probabilística do procedimento reside no facto de que cada elemento é seleccionado de forma aleatória a partir de um subconjunto restrito formado pelos melhores elementos que compõem a lista de candidatos. Este subconjunto recebe o nome de lista de candidatos restrita (LCR). Esta técnica de escolha permite que diferentes soluções sejam geradas em cada iteração GRASP. Seja $\alpha \in [0,1]$ um dado parâmetro. O pseudo-código representado pela Figura 2-13 descreve a fase de construção GRASP.

Procedimento *Construcao*$(g(.), \alpha, s)$;

1. $s \leftarrow \emptyset$;
2. Inicialize o conjunto C de candidatos;
3. **enquanto** $(C \neq \emptyset)$ **faça**
4. $g(t_{max}) = \min\{g(t) | t \in C\}$;
5. $g(t_{max}) = \max\{g(t) | t \in C\}$;
6. $LCR = \{t \in C | g(t) \leq g(t_{min}) + \alpha(g(t_{max}) - g(t_{min}))\}$;
7. Selecione, aleatoriamente, um element $t \in: LCR$;
8. $s \leftarrow s \cup \{t\}$;
9. Actualize o conjunto C de candidatos;
10. **fim-enquanto**
11. Retorne s;

Figura 2-13: Fase de construção de um algoritmo GRASP

Procedimento *BuscaLocal* $(f(.), N(.), s)$;

1. $V = \{s' \in N(s) | f(s') < f(s)\}$;
2. **enquanto** $(|V| > 0)$ **faça**
3. Seleccione $s' \in V$;
4. $s \leftarrow s'$;
5. $V = \{s' \in N(s) | f(s') < f(s)\}$;
6. **fim-enquanto**
7. Retorne s;

Figura 2-14: Fase de Busca Local de um algoritmo GRASP

do procedimento *Construção*. Um valor $\alpha = 0$ faz gerar soluções puramente gulosas, enquanto $\alpha = 1$ faz gerar soluções totalmente aleatórias.

Assim como em muitas técnicas determinísticas, as soluções geradas pela fase de construção do GRASP provavelmente não são localmente óptimas com respeito à definição de vizinhança adoptada. Daí a importância da fase de busca local, a qual objectiva melhoriar a solução construída. A Figura descreve o pseudocódigo de um procedimento básico de busca local com respeito a uma certa vizinhança $(.)$ de s.

A eficiência da busca local depende da qualidade da solução construída. O procedimento de construção tem então um papel importante na busca local, uma vez que as soluções construídas constituem bons pontos de partida para a busca local, permitindo assim acelerá-la.

O parâmetro α, que determina o tamanho da lista de candidatos restrita, é basicamente o único parâmetro a ser ajustado na implementação de um procedimento GRASP. Em (Feo & Resende, 1995) discute-se o efeito do valor de α na qualidade da solução e na diversidade das soluções geradas durante a fase de construção. Valores de α levam a uma lista de candidatos restrita de tamanho muito limitado (ou seja, valor de α próximo da escolha gulosa) implicam em soluções finais de qualidade muito próxima daquela obtida de forma puramente gulosa, obtidas com um esforço computacional. Entretanto uma escolha de α próxima da selecção puramente aleatória leva a uma grande diversidade de soluções construídas, por outro lado, muitas das soluções construídas são de qualidade inferior, tornando mais lento o processo de busca local.

O procedimento GRASP procura, portanto, conjugar bons aspectos dos algoritmos puramente gulosos, com aqueles dos procedimentos aleatórios de construção de soluções.

Procedimentos GRASP mais sofisticados incluem Estratégicas adaptativas para o parâmetro α. O ajusto desse parâmetro ao longo das iterações anteriores , produz soluções melhores do que aquelas obtidas considerando-o fixo (Prais & Reibeiro, 1998) (Prais & Ribeiro, Parameter Variation in GRASP Implementations, 1999) (Prais & Ribeiro, Variação de Parâmetros em Procedimentos GRASP, 1999).

2.11 Algoritmo GBT-II

O Algoritmo GRASP-Busca Tabu – Interturmas-Intraturmas (GBT-II) é algoritmo principal para a resolução do problema em causa e incorpora os algoritmos GRASP, BT e II descritos anteriormente.

O algoritmo GBT-II é um procedimento GRASP descrito em 2.10, onde a solução inicial é gerada por um procedimento construtivo parcialmente guloso e o refinamento é feito através de um método BT segundo 2.9.

Quando uma solução sem inviabilidade do tipo 1 é gerada, o método BT-IIm acciona o procedimento II descrito em 2.7. Esta sequência de construção e refinamento é repetida até que uma determinada condição de paragem seja satisfeita.

A apresenta o pseudo-código do algoritmo GBT-II.

Procedimento *GBT-II(Q)*

1. Seja Q^* a melhor solução e f^* o valor associado;
2. $f^* \leftarrow \infty$;
3. **enquanto** (*condição de paragem não for satisfeita*) **faça**
4. $Q^0 \leftarrow ConstrucaoSolucaoInicial;$
5. $Q \leftarrow BT\text{-}II(Q^0);$
6. **se** $(f(Q) < f^*)$ **então**
7. $Q^* \leftarrow Q;$
8. $f^* \leftarrow f(Q);$
9. **fim-se**
10. **fim-enquanto**
11. Retorne Q^*;

Figura 2-15: Pseudo-código do algoritmo GBT-II.

3 SISTEMA ACTUAL

3.1 Introdução

A instituição considerada para a análise do PPHE é a Escola Secundária Francisco Manyanga. Neste capítulo, será detalhado o método manual de elaboração de horários. Apresentar-se à também o levantamento de dados efectuado na ESFM para o desenvolvimento deste trabalho.

3.2 Levantamento de Dados

A fase de levantamento dos dados relativos a instituição é de grande importância para a construção do modelo. É necessário entender o funcionamento da instituição e conhecer a sua estrutura, com esse intuito foi aplicado um questionário de levantamento das características da instituição.

A ESFM lecciona I e II ciclos do ESG nos períodos de manhã, tarde e noite. No período da manhã é seccionado o II ciclo, no período da tarde o I ciclo e no da noite ambos.

No período da manhã, são leccionadas as 11ª e 12ª Classes, as aulas têm início as 7:00 H e terminam as 12:30 H, em média vinte(20) turmas por classe. As turmas estão subdivididas em grupos de AI, AII, AIII, BI, BII e C, (vide Tabela 3-1).

No período da tarde são leccionadas as 8ª, 9ª e 10ª Classes das 13:00 H as 17:30 H. As turmas da 10ª classe são constituídos por alunos que fazem todas as cadeiras, os que fazem ciências[5] e os que só fazem só letras[6]. Aqueles que fazem ciências é porque reprovam a um ou mais disciplinas das ciências no ano anterior e o mesmo acontece com as letras. Geralmente, a escola enquadra-os em turmas que fazem todas as disciplinas ou em turmas que fazem ou ciências ou letras dependendo do caso; um dos grupos que tiver número menor de turmas é enquadrado nas turmas que fazem todas as disciplinas.

No período da noite, são leccionadas todas as classes do I e II ciclo; do ESG, das 18:00 H as 22:30 H totalizando 5 aulas diárias, o que perfazer 25 horas semanais.

As do turno da manhã têm aulas de Educação Física no período a tarde e as turmas da tarde têm de manhã, enquanto as turmas da noite não têm. O mesmo acontece com ás aulas do Director de Turma (DT), as turmas da manhã e da tarde têm nas quartas-feiras no terceiro tempo, enquanto as turmas da Noite não tem devido número de aulas por dia que é menor.

Uma aula simples tem a duração de 45 minutos. Uma aula dupla é um conjunto de duas(2) aulas simples de quarenta e cinco(45) minutos da mesma disciplina com intervalo de cinco(5) minutos ou quinze (15) minutos (o chamado intervalo maior). No período da manhã e da tarde, são leccionadas seis (6) aulas diárias das diferentes cadeiras, enquanto que no período da noite são leccionadas cinco(5) aulas diárias.

Os horários dos diferentes turnos são independentes, embora haja professores do mesmo ciclo que leccionam em 2 turnos mas sem influência directa na elaboração dos horários.

As cargas horárias das disciplinas estão distribuídas na tabela abaixo:

[5] Ciências — Matemática, Biologia, Desenho, Física, Química

[6] Letras — Português, História, Geografia, Inglês

II Ciclo								I Ciclo			
Grupo	Disciplinas	Dir	Noc	Grupo	Disciplinas	Diu	Noc	Secção	Disciplinas	Diu	Noc
		CH	CH			CH	CH			CH	CH
AI	Português	4	4	BI	Português	4	—	Letras	Português	5	4
	Inglês	4	4		Inglês	4	—		Inglês	3	3
	Francês	4	4		Geografia	4	—		História	2	2
	História	4	4		Física	4	—		Geografia	2	2
	Geografia	4	4		Química	4	—	Ciências	Biologia	3	2
	Filosofia	4	4		Matemática	5	—		Física	3	3
AIII	Português	4	—	BII	Português	4	—		Química	3	3
	Inglês	4	—		Inglês	4	—		Matemática	5	4
	Francês	4	—		Biologia	4	—		Desenho	2	2
	História	4	—		Física	4	—				
	Geografia	4	—		Química	4	—				
	Filosofia	4	—		Matemática	5	—				
	Biologia	4	—	C	Português	4	4				
AIV	Português	4	—		Inglês	4	4				
	Inglês	4	—		Física	4	4				
	Francês	4	—		Química	4	4				
	História	4	—		Matemática	5	5				
	Geografia	4	—		Desenho	3	3				
	Filosofia	4	—								
	Matemática	5	—								

Tabela 3-1: Cargas horárias das disciplinas

No ESG nacional existe um limite estabelecido de carga horária semanal máxima dos professores por disciplina que se encontra abaixo, representado na Tabela 3-2 agrupado por disciplina e ciclo. Isto é, um professor de Português do I Ciclo do ESG, por exemplo pode ter a carga horária máxima de 25 horas semanais, e um professor de uma disciplina que não seja Matemática ou Português do II Ciclo pode ter uma carga horária máxima de 20 horas semanais.

Disciplina	I Ciclo	II Ciclo
Português	25	20
Matemática	25	20
Restantes	24	20

Tabela 3-2: Carga Horária semanal máxima por professor

3.3 Descrição do Sistema Actual

Os passos usados para a elaboração dos Horários pelo entrevistado, Professor Germano Raul, o chefe da Sala de Informática da ESFM e responsável pelo processamento de horários, são:

Para um determinado turno em função do número de turmas existentes, encontrar o número de professores necessários para cada disciplina tendo em conta a carga horária máxima do professor.

Disciplina:	**Português**	
Carga Horária por Prof.	25 h	Veja a Tabela 3-2
Turno:	Tarde	
Classes:	8ª,9ª,10ª	
Turmas:	15 turmas da 8ª	
	13 turmas da 9ª	
	14 turmas da 10ª(2 turmas de Ciências)	
Carga Horária da Disciplina	5 h	Veja a Tabela 3-1
Total Carga Horária:	15+13+12=40	Soma das Turmas que assistem às aulas da Disciplina (12 turmas da 10ª classe porque tem 2 turmas de ciências que não assistem aulas da

Em seguida, distribui-se o número de professores encontrados pelas respectivas turmas, obedecendo a carga horária máxima prevista por cada professor segundo a Tabela 3-2. É neste momento que também são codificados os professores e associados a professores específicos. Cada professor é associado a um número de turmas e o código atribuído é composto pelo código da disciplina e um número de ordem dos professores na disciplina em causa, como mostramos na :

Codigo	**Nome do Professor**	**Carga Horária**	**Turmas**
P1	Professor de Português 1	25	8ª 1,2,3,4,5
P2	Professor de Português 2	25	8ª 6,7,8,9,10
P3	Professor de Português 3	25	8ª 11,12,13,14,15
P4	Professor de Português 4	25	9ª 1,2,3,4,5
P5	Professor de Português 5	25	9ª 6,7,8,9,10

Agora que temos os professores distribuídos construímos um quadro de horários com os tempos a partir de segunda à sexta-feira, de um lado, e as turmas, de outro lado. Esse quadro será posteriormente preenchido com os códigos dos professores indicando que um determinado professor lecciona uma determinada turma em um determinado tempo.

Grupo	**turma**	**2ªFeira**						**3ªFeira**						**4ªFeira**						**turma**
		1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6	
8ªClasse Curso Diurno	1	P1	P1	M1	M2													**DT**	**DT**	1
	2																	**DT**	**DT**	2
	3																	**DT**	**DT**	3
	4																	**DT**	**DT**	4

	5																**DT**	**DT**	5
	6																**DT**	**DT**	6
	7																**DT**	**DT**	7
	8																**DT**	**DT**	8
	9																**DT**	**DT**	9
	10																**DT**	**DT**	10
	11																**DT**	**DT**	11
	12																**DT**	**DT**	12
	13																**DT**	**DT**	13
	14																**DT**	**DT**	14
	15																**DT**	**DT**	15

Tabela 3-5: Representação de um Quadro de horário em construção

Na Tabela 3-5 representamos um quadro de horário em construção que contém 15 turmas da 8ª Classe e os respectivos horários de segunda a quarta-feira. Notamos que, nas quartas-feiras, no quinto e sexto tempos as aulas são reservadas ao Director de Turma (DT).

No caso real, o quadro incluirá todas as turmas de um determinado turno e os respectivos horários de segunda-feira a sexta-feira, na Tabela 3-5 optamos por mostrar menos, por uma questão de espaço.

O processo de elaboração de horário é pseudo-aleatório, visto que já temos os professores e os respectivos códigos e turmas que leccionam, inicia-se o preenchimento do quadro de horário. Como podemos ver na Tabela 3-5, na segunda-feira no primeiro tempo e segundo tempo, a turma, 1 da 8ª Classe tem aulas de Português com o professor P1. Temos que ter em conta que o P1 já foi escolhido anteriormente para leccionar a turma 8ª1, então, só poderemos preencher as células com professores que foram escolhidos antes para leccionar as turmas das células que preenchemos.

O processo continua manualmente e "aleatoriamente" mas sempre obedecendo aos seguintes critérios:

- Numa determinada linha não deverá existir código repetido, isto é, nenhum professor poderá estar a leccionar duas turmas no mesmo horário.
- Criar, sempre que possível, aulas duplas.
- Uma turma não poderá ter aulas da mesma disciplina durante 3 dias seguidos.
- Incluir as horas de planificação dos professores.
- Evitar que uma turma não poderá ter uma aula no último tempo de um determinado dia e no dia seguinte ter a mesma aula no primeiro tempo.
- Evitar buracos nos horários das turmas.

Este processo será repetitivo efectuando se trocas entre determinadas aulas de modo a satisfazer as condições acima. E, por fim, teremos um quadro de horário completo para um determinado turno.

4 TECNOLOGIAS USADAS

Este trabalho combina tecnologias diversas muitas das quais *open-source*. E ao logo deste capitulo vamos descrever as mais importantes justificando sempre que possível as opções de escolha; dando também alternativas possíveis de outras tecnologias que poderiam ser usadas em substituição.

Este trabalho foca mais no desenvolvimento e menos na infra-estrutura, so desta forma escolhemos tecnologias que possa facilitar a colocação da aplicação em produção, e pequenas em tamanho.

4.1 Wicket[7]

Existe uma gama de *web frameworks* hoje em dia de variados estilos. As tecnologias mais conhecidas tais como Java Server Faces, ASP.NET, GWT são baseadas em eventos, com desenho orientado a componentes, o que se assemelham à programação tradicional do GUI. Esta ideia faz sentido dado que as aplicações Web se tornaram como aplicações Desktop na sofisticação e funcionalidade. Entretanto, muitas desses *frameworks* podem tornar-se insustentáveis, exigindo

[7] http://wicket.apache.org/

sustentação de ferramentas pesadas e curvas de aprendizagem penosas. Outro facto é a mistura de código e *markup* dificultam frequentemente os testes, *reflactorings* e a separação dos conteúdos (*separations of concerns*). Dai que surge um novo Web *framework* chamada Wicket que vem resolver estes e outros problemas desafios.

Wicket é uma *web-application-framework* Java orientado a componentes. É muito diferente de outras frameworks baseadas em requisição/acção tais como *Struts, WebWork* or Spring MVC em que a submissão de um formulário traduz-se uma única acção. Em Wicket uma acção de um utilizador provoca tipicamente um evento em um dos componentes do formulário, que responde por sua vez ao evento. Alguns dos outros *frameworks* que falham nesse aspecto são *Taperstry*, *JSF* e *ASP.NET.* essencialmente *framework* como *Struts* deram nascimento a um conceito de MVC para Web que encapsulam acções *coarse-grained* em contraste com as acções *fine-grained* que os *developers* estão habituados a usar nas programação de aplicações *desktop.* As *frameworks* orientadas a components tais como Wicket trazem esta mais familiar experiência de programação para a Web. (Gurumurthy, 2006)

Neste trabalho usaremos o Wicket como *framework* de apresentação por possuir as seguintes caracteristicas:

- Modelo orientado a componentes usando Orientação a objectos tal como *Swing* – as paginas e objectos em wicket são objectos reais que suportam encapsulamento, herança e eventos.
- Facilidade de Desenvolvimento – Porque o Wicket é Java e HTML podemos aplicar os nossos conhecimentos de Java e o usar um bom editor de HTML para fazer o resto.
- *Separations of Concerns* – Wicket não mistura *markup* com o codigo Java e não adiciona nenhuma sintaxe especial aos ficheiros de *markup.* O codigo HTML e Java estão em paralelo e são associados somente por IDs, que são atributos em HTML e propriedades dos componentes em Java. Sendo o HTML do Wicket apenas HTML e o Java apenas Java, os programadores e *designers* podem trabalhar grande parte do tempo independentemente sem precisar de ferramentas especiais.
- Segurança – Wicket é seguro por natureza. Os URLs não expõem informação sensível e todos os trajectos dos componentes são relativos a sessão. Passos explícitos devem ser tomadas de modo a partilhar informação entre as sessões.
- Suporte transparente ao botão de retrocesso – Wicket contém um sistema de gestão das versões das paginas perfeitamente confuravel. Quando um utilizador submete a pagina

em um formulário ou clica em um *link* de uma página que ele acessou com o buttão de retrocesso do seu navegador, o Wicket pode reverter todo objecto página com o seu estado tal como estava inicialmente.

- Componentes Recusáveis – Os componentes reusáveis do Wicket são particularmente fáceis de criar.
- Validação Simples, Flexível e configurável dependendo da língua
- Suporte a todas características básicas do HTML

Para mais informações sobre Wicket sugerimos o livro (Gurumurthy, 2006) e o site (Apache Software Foundation, 2007)

4.2 Test-NG[8]

O processo de teste de um software é uma maneira de verificar se este está correto, completo e qual o seu nível de qualidade. Por isso, este é um procedimento indispensável no desenvolvimento de qualquer software.

Para que o desenvolvimento dirigido por testes fosse possível, era necessária uma ferramenta básica que permitisse escrever e executar casos de teste. A primeira implementação desse tipo de ferramenta foi feita em Smalltalk e foi chamada de SUnit. A partir dela, abstraiu-se uma arquitetura conhecida como "*framework* xUnit". "Esta arquitetura foi implementada em várias linguagens de programação e para diferentes plataformas".

O JUnit é uma implementação do framework xUnit feito na linguagem Java que permite escrever testes de unidade e executá-los repetidas vezes. Entre as suas funcionalidades destacam-se: o uso de assertivas para testar resultados esperados; a possibilidade de criação de objetos comuns que são compartilhados por todos os testes; e os suites de teste para organização e execução conjunta de vários testes. (Hemrajani, 2006)

Mais do que qualquer outro framework de teste, o JUnit levou os desenvolvedores a entenderem a necessidade de testar ou, mais especificamente, de fazer testes de unidade. Porém, com a evolução da linguagem Java, o JUnit se tornou simples demais. O TestNG foi feito para aproveitar as novas

8

capacidades da linguagem Java. Inspirado no JUnit e também no NUnit, um framework para testes de unidade em .Net, o TestNG introduziu novas características aos testes de unidade, dentre elas:

- Suporte a anotações em Java.
- Configurações de teste em arquivos XML.
- Não requer a extensão de uma classe.
- Permite definir grupos de teste.
- Execução de testes em paralelo.
- Aceita parâmetros nos métodos de teste

4.3 Hibernate[9]

Para fazer o mapeamento da base de dados foi usado o *framework* Hibernate de modo a facilitar o desenvolvimento da camada de persistência.

O Hibernate é um framework para o mapeamento objeto-relacional escrito na linguagem Java, mas também é disponível em .Net como o nome NHibernate. Este programa facilita o mapeamento dos atributos entre uma base tradicional de dados relacionais e o modelo objeto de uma aplicação, mediante o uso de arquivos (XML) para estabelecer esta relação.

O objetivo do Hibernate é diminuir a complexidade entre os programas Java, baseado no modelo orientado a objeto, que precisam trabalhar com um banco de dados do modelo relacional (presente na maioria dos SGDBs). Em especial, no desenvolvimento de consultas e atualizações dos dados.

Sua principal característica é a transformação das classes em Java para tabelas de dados (e dos tipos de dados Java para os da SQL). O Hibernate gera as chamadas SQL e libera o desenvolvedor do trabalho manual da conversão dos dados resultante, mantendo o programa portável para quaisquer bancos de dados SQL, porém causando um pequeno aumento no tempo de execução.

Nas questões relacionadas para o gerenciamento de transações e na tecnologia de acesso à base de dados são de responsabilidade de outros elementos na infrestrutura do programa. Apesar de existirem API no Hibernate para possuir operações de controle transacional, ele simplesmente delegará estas funções para a infrestrutura na qual foi instalada.

[9] http://www.hibernate.org/

No caso de aplicações construídas para serem executadas em servidores de aplicação, o gerenciamento das transações é realizado segundo o padrão JTA. Já nas aplicações standalone, o programa delega o tratamento transacional ao driver JDBC.

4.4 Spring[10]

O Spring é um framework open source criado por Rod Johnson e descrito em seu livro "Expert One-on-One: J2EE Design e Development". Trata-se de um framework não intrusivo, baseado nos padrões de projeto inversão de controle (IoC) e injeção de dependência.

No Spring o container se encarrega de "instanciar" classes de uma aplicação Java e definir as dependências entre elas através de um arquivo de configuração em formato XML. Dessa forma o Spring permite o baixo acoplamento entre classes de uma aplicação orientada a objetos.

O Spring possui uma arquitetura baseada em interfaces e POJOs (Plain Old Java Objects), oferecendo aos POJOs características como mecanismos de segurança e controle de transações. Também facilita testes unitários e surge como uma alternativa à complexidade existente no uso de EJBs.

Este framework oferece diversos módulos que podem ser utilizados de acordo com as necessidades do projeto, como módulos voltados para desenvolvimento Web, persitência, acesso remoto e programação orientada a aspectos.

5 MODELAÇÃO DO SISTEMA

Para criar um projecto bem sucedido, é sempre necessário um envolvimento directo dos usuários, simplicidade (desenho, ferramentas, documentação), senso comum e um básico/mínimo processo de desenvolvimento de software.

[10] http://www.springframework.org/

Uma boa solução é a combinação das melhores práticas e técnicas recomendadas por Agile Model Driven Development (Ambler, 2007) e Extreme Programming (Wells, 2006).

Todo o projecto, quer pequeno ou grande que seja, deve seguir uma básica estrutura ou processo (metodologia), que pode ser uma simples página *checklist* ou um processo mais formal. Não ter nenhum processo é extremamente mau, mas ter em excesso é também igualmente mau. Para encontrarmos um balanço entre esses dois extremos depende do tamanho do projecto e dos requisitos do cliente.

Para o desenvolvimento deste projecto usaremos AMDD e XP como metodologias de desenvolvimento de software.

XP é uma metodologia de desenvolvimento ágil de software baseada nos valores simplicidade, comunicação, feedback e coragem. Para implementar XP não é preciso usar diagramas ou processos formais. É preciso fazer uma equipe unir-se em torno de algumas práticas simples, obter feedback suficiente e ajustar as práticas para a sua situação particular. (Beck, 2000)

XP e AMDD dispõem de alguns guiões fundamentais para criar aplicações eficientes e rapidamente. XP e AMDD são métodos complementares porque XP dispõe de uma disciplinada visão de desenvolvimento de software com ciclo de vida completo que vai de acordo com a satisfação do cliente. Por outro lado, AMDD geralmente dispõe de práticas efectivas para a modelação e documentação, mas vai ainda mais longe dispondo um conjunto de melhores e saudáveis práticas que podem facilitar um projecto de desenvolvimento de software. (Hemrajani, 2006). Para mais informações sobre metodologias agil no geral recomendo a leitura do (Pedro, 2007)

Nas próximas secções aplicaremos algumas partes das metodologias neste projecto.

5.1 Requisitos[11] do Negócio

[11]**Requisito** num sistema é uma funcionalidade ou característica considerada relevante na óptica do utilizador. Normalmente representa o comportamento esperado do sistema que, na prática consiste, num serviço que deve ser disponibilizado a um utilizador. (Nunes, 2001)

Em 3.3 encontramos a definição do problema. À definição do problema acrescentamos o seguinte para complementarmos:

- O utilizador terá que ser capaz, através de um *interface* gráfico de introduzir os professores de um determinado turno, turmas leccionadas, gerar um quadro de horários atráves do programa.
- O utilizador poderá exportar o quadro gerado pelo programa para um formato usado geralmente na Escola.

5.2 Concepção do Modelo

5.2.1 *Domain Model* ou Modelo Principal

Contém essencialmente todas as entidades e as relações, mas sem atributos. Isto irá ajudar-nos a definir alguns conceitos iniciais e as respectivas relações entre eles.

O modelo principal é tipicamente captado no contacto com utilizadores com experiência nas regras do negócio como intervenientes ou analistas.

A figura abaixo mostra o Modelo Principal para o Sistema em estudo e podemos ver que contém um simples modelo para podemos começar a modelar o Sistema.

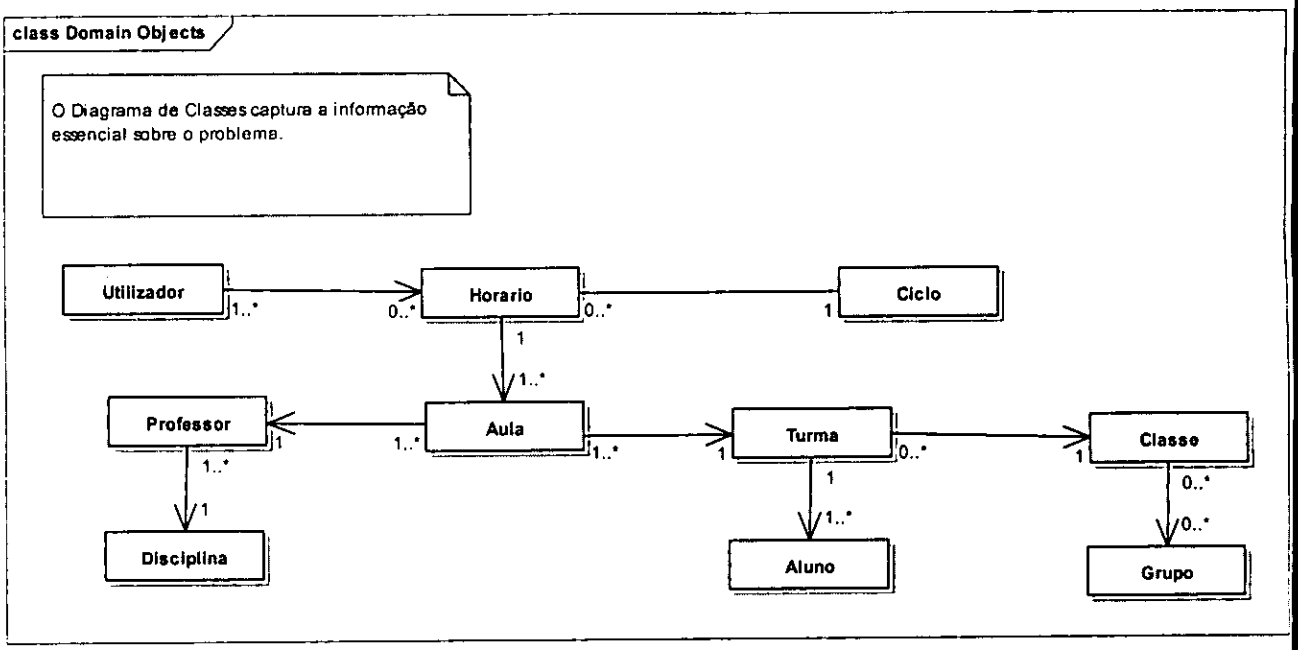


Figura 5-1:Modelo Principal

5.2.2 Protótipo - User interface(UI)

É um técnica iterativa da análise em que os utilizadores são envolvidos activamente do desenho do Interface do Sistema. E tem diversas finalidades:

- Permite explorar o domínio do problema com os intervenientes no sistema por ser um resultado da Analise;
- Permite explorar o domínio da solução do sistema por ser um resultado do desenho;
- É um veículo que permite mostrar os *User Interfaces* do sistema.
- Permite continuar a desenvolver o sistema.

(Ambler, 2006)

O protótipo do UI é construído cedo antes que o sistema inteiro seja analisado, projectado ou executado porque permite dar uma cara ao Sistema o que coloca as pessoas envolvidas no Sistema motivadas no sentido de concluírem o desenvolvimento do sistema.

A finalidade principal de criar um protótipo de UI é poder expor e testar a funcionalidade e a usabilidade do sistema antes que o desenho real ou implementação do sistema inicie. Desta forma, pode assegurar que se está a desenvolver correctamente o sistema antes que se dispenda demasiado tempo e recursos no desenvolvimento.

As figuras 5.2 a 5.4 mostram como as várias janelas do Sistema vão ficar.

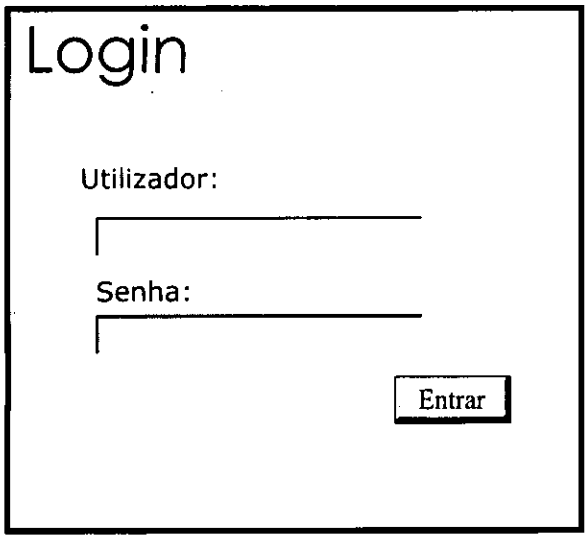


Figura 5-2:Interface de Login

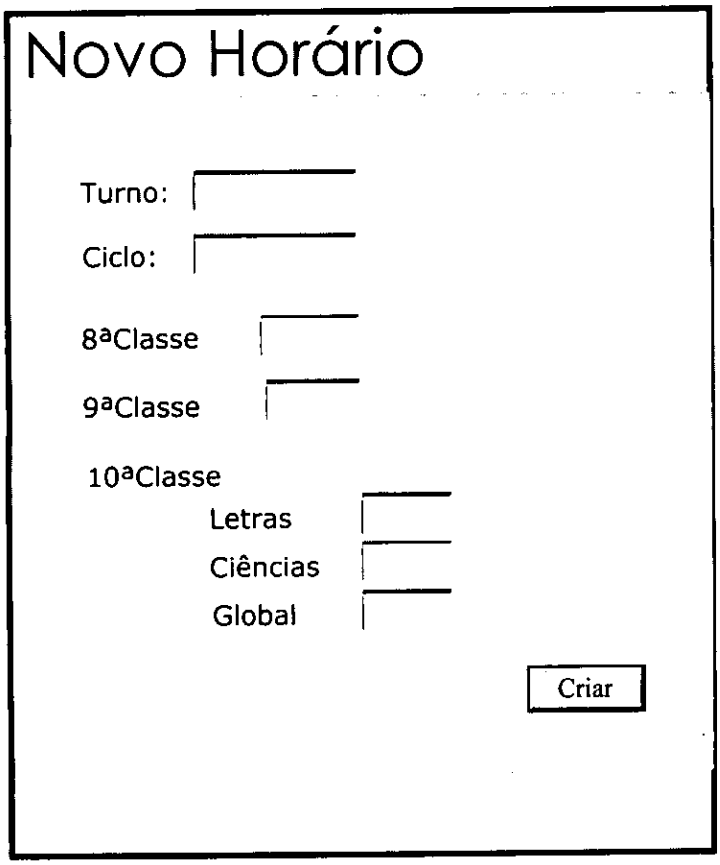


Figura 5-3:Interface de Criação de Novo Horário

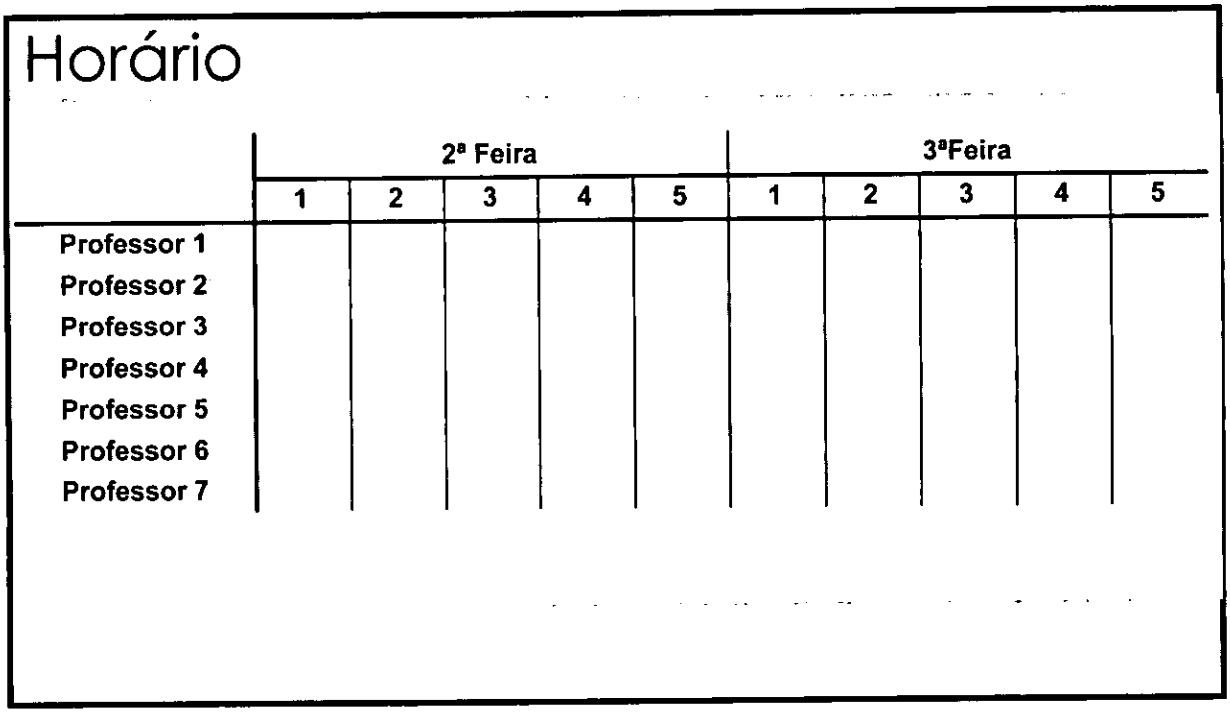


Figura 5-4:Interface de Listagem do Quadro de Horário contendo os horários de segunda e terça feiras, os restantes dias apresentam o mesmo formato.

5.2.3 Storyboard

Um *Storyboard* , também chamado diagrama de Fluxo do UI é essencial para mostrar um mapa de navegação entre as várias telas do Sistema. A figura 6.5 mostra o *storyboard* do Sistema.

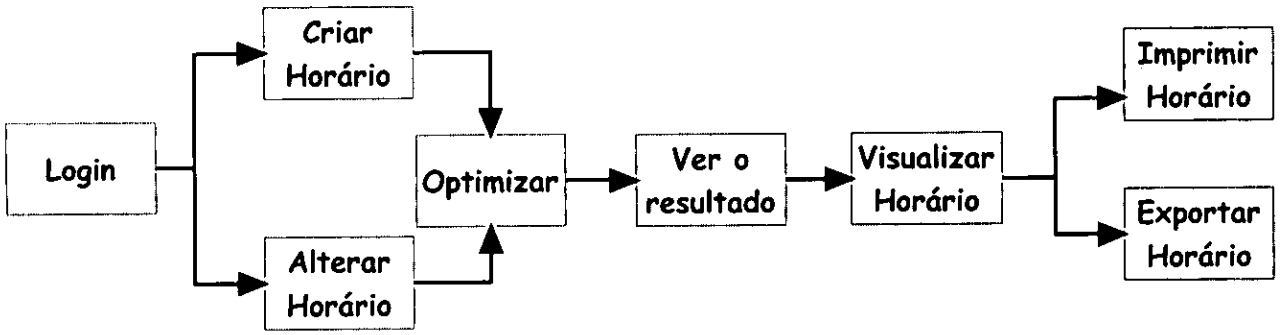


Figura 5-5:Storyboard

5.2.4 User Stories

Depois dos requisitos de negócio e protótipos de UI elaborados para o Sistema construímos os *User Stories*.

User Stories são o resultado preliminar do desenvolvimento para as equipes de um projecto XP. Um *User Story* é uma definição de nível extremamente elevado de um requisito, contendo informação suficiente para que os *Developers* possam produzir uma estimativa razoável do esforço para o executar. Uma boa forma de imaginar um *User Story* é recordar-se das entrevistas feitas aos Clientes (Partes interessadas ou *Stakeholders*[12]). São relativamente mais pequenos que os *Use Cases*[13]. (Ambler, Agile Modeling: Effective Practices for Modeling and Documentation, 2006).

[12] ***stakeholder*** é qualquer um que seja utilizador directo/indirecto, gestor de utilizadores, gestor sénior, pessoal do nível operativo, pessoal do suporte, desenvolvedores que trabalham no outro sistema que integra ou interage com o

Por se tratar da metodologia XP/AMDD, usaremos o termo *User Story* em vez de *Use Case*.

Segundo (Hemrajani, 2006), estes termos, mesmo servindo para a mesma finalidade, o *User Story* tende a ser mais pequeno como referimos anteriormente na proporção de um para três frases. Os restantes detalhes são discutidos entre o *Developer* e o Cliente quando o *Developer* inicia um determinado *User Story*, daí o termo "*Participação Activa dos Stakeholders*".

Usaremos *User Storiers* não só para definir os requisitos mas também para dar nome as classes Java, usando *Story name/Tag* e para criar testes de aceitação.

A tabela 6-1 mostra os *User Stories* com as prioridades e as estimativas iniciais de desenvolvimento de cada um deles.

#	Story Name/Tag	Descrição	Prioridade	Pontos (Estimativa)
1	Criar Horário	O Utilizador poderá introduzir os dados necessários para criação de um Quadro do Horário, validá-lo e salvá-lo.	1	2
2	Listar Horários	Listar os Quadros de Horários existentes com a possibilidade de seleccionar um deles, ver seus detalhes e alterá-los.	1	1
3	Entrar	O Utilizador poderá entrar no Sistema usando um nome e chave válidos	2	1
4	Sair	O Utilizador poderá sair do sistema abandonando a secção actual	2	1
5	Optimizar Horário	O Utilizador poderá optimizar um determinado horário com a possibilidade de programar a execução para uma determinada hora.	3	5

software em desenvolvimento, ou profissionais de manutenção afectados pelo desenvolvimento de um projecto de software. (Ambler, 2006)

[13] ***Use Cases*** constituem a técnica em UML que serve para representar o levantamento de requisitos[13] de um sistema, e, para assegurar que tanto o utilizador final como o perito numa determinada área possuam um entendimento comum dos requisitos. (Nunes, 2001)

6	Notificação via e-mail	O Utilizador poderá escolher a opção de ser notificado via e-mail no final da execução da optimização.	3	1
7	Ver o Resultado	O Utilizador poderá ver o resultado da execução da optimização.	4	1
8	Imprimir Horário	O Utilizador poderá imprimir o Quadro de Horários	4	1
9	Exportar Horário	Exportar o Quadro de Horários Completo para excel, pdf etc	4	2
			Total de Pontos	**15**

Tabela 5-1: User Stories, Prioridades e Estimativas

As estimativas mostradas na tabala 5-1 são iniciais. O *Developer* fornece as estimativas mais exactas no início de iteração quando o *User Story* está a ser desenvolvido.

Os pontos são alguma unidade de medida relativa ao desenvolvimento do Sistema. Por exemplo um ponto pode significar um dia útil de trabalho regular, uma semana ou outro período que os clientes concordam. Sendo isso medida usada para indicar uma estimativa ao dono do Sistema.

5.3 Implementação do Modelo

Esta secção corresponde a implementação do Sistema com Desenho e Arquitectura baseados em XP/AMDD, e tem os seguintes objectivos:

- Desenvolver Diagramas Simples de Arquitectura;
- Explorar os objectos usando *CRC-Cards;*
- Desenvolver diagramas de classe e pacotes;
- Criar uma estrutura de directórios para o desenvolvimento do Sistema;
- Listar todos os detalhes a considerar no desenvolvimento do Sistema.

Na sessão 5.2 definimos os requisitos baseando-nos na metodologia XP. Nesta secção iremos criar uma arquitectura e desenho de modo a ajudar-nos a desenvolver o Sistema usado tecnologias como:

- Wicket;
- Spring Framework;
- Hibernate;
- Quartz;
- Maven;
- Test-NG

A figura 5.6 mostra alguns dos possíveis artefactos que podem ser produzidos nos níveis de lançamento ou intermédios de desenvolvimento do Sistema. Segundo (Hemrajani, 2006), os artefactos de lançamento são aqueles que são produzidos antes do lançamento de uma versão do Sistema; os artefactos de nível intermédio são aqueles que são produzidos em cada iteração do desenvolvimento. Estes artefactos não são todos obrigatórios para qualquer projecto de desenvolvimento de software, por isso podemos escolher aqueles que precisamos.

A grande diferença da metodologia XP é que a arquitectura e desenho aparecem durante o ciclo de desenvolvimento do Sistema e não repentinamente. Ou seja a aplicação continua a evoluir em várias iterações. O beneficio desta visão é que o desenho é realmente aplicável a aquilo que se esta a desenvolver e não depois de seis meses quando os requisitos já tenham sido alterados o que é possível num mundo de constantes mudanças em que nos encontramos. (Hemrajani, 2006)

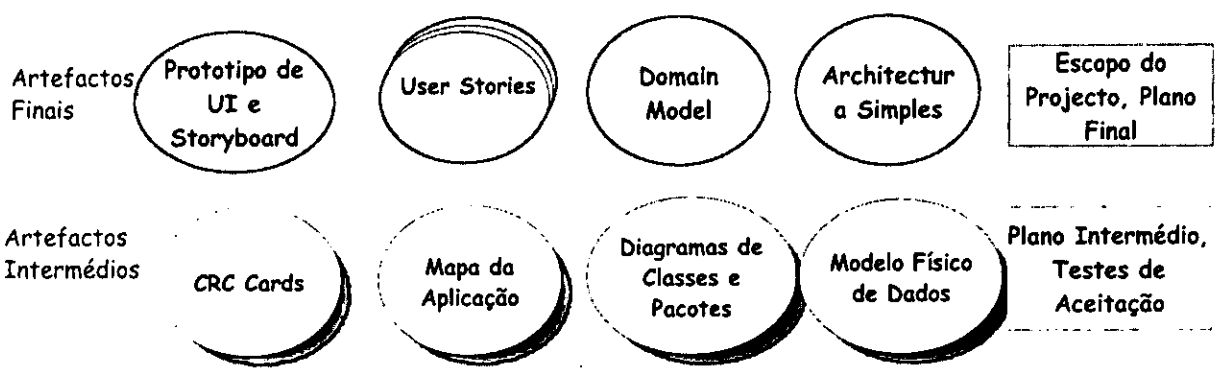


Figura 5-6: Escolhas dos artifactos que serão produzidos nos processos finais e intermediários segundo a XP/AMDD. (Hemrajani, 2006)

5.4 Diagrama de Arquitectura de Forma Livre

A figura 6.7 mostra a Arquitectura de alto nível do Sistema baseada no modelo de três camadas, camada do cliente (*web browser)*, camada média (Servidor Aplicacional) e Camada de Dados (Base de Dados)

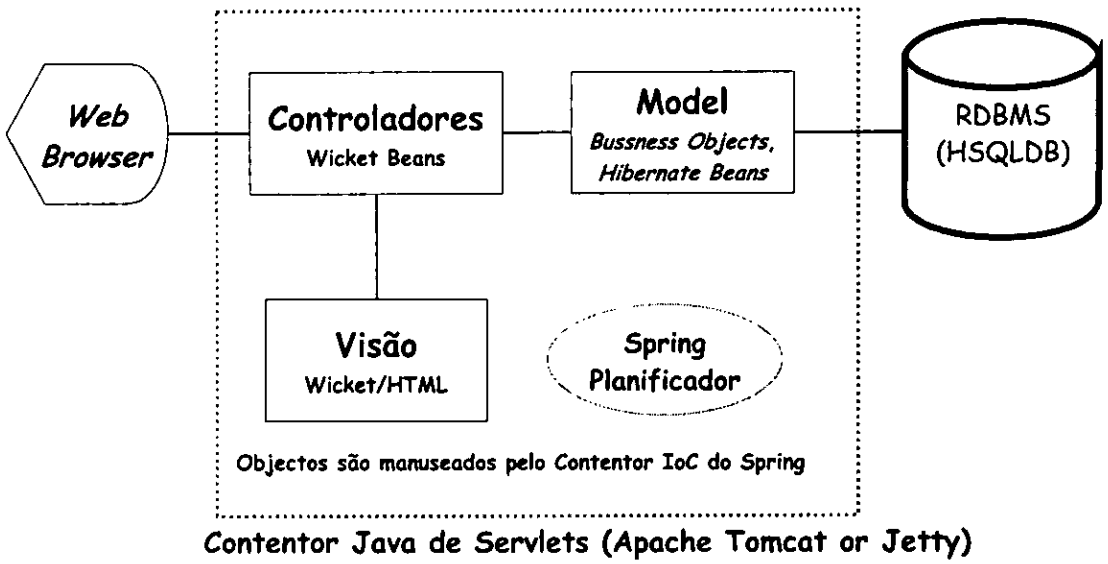


Figura 5-7:Arquitectura de Alto Nível do Sistema

Podemos ver que a arquitectura usa o *design pattern* (padrão de desenho) *model-view-controller*[14](MVC), o controlador é o ponto de entrada duma requisição HTTP/web; o modelo de dados e a apresentação. O modelo de dados é responsável pelos dados, que são obtidos pelo controlador e passados para a apresentação para poderem ser apresentados. (Hemrajani, 2006).

[14] Model-view-controller (MVC) é um padrão de arquitectura utilizado em engenharia de software. Em aplicações complexas, que enviam uma série de dados para o usuário, o *Developer* frequentemente necessita de separar os dados (*Model*) da interface (*View*). Desta forma, alterações feitas na interface não afectarão a manipulação dos dados, e estes poderão ser reorganizados sem alterar a interface do usuário. O MVC resolve este problema através da separação das tarefas de acesso aos dados e lógica do negócio da apresentação e interacção com o utilizador, introduzindo um componente entre os dois: o *Controller*. (Wikipédia, 2007)

Para apresentação, foi usado o *framework* baseado em componentes Wicket, beneficiando de um modelo de programação puramente orientado a objecto. O *framework* Wicket foi descrito em 4.1.

Toda a camada média é gerida pelas classes do Wicket e pelo *Spring Framework* para a inversão de controlo e o *Hibernate* para a camada de persistência.

5.5 CRC Cards

O *Domain Model* (figura 6.1) permite nos explorar os objectos do domínio do problema ou do negócio; os *User Stories* (Tabela 6.1) permitem encontrar as classes Controladoras do UI. Associando os dois primeiros ao UI (figura 6.2) e tendo em conta que usaremos o *design pattern* MVC podemos fazer o desenho inical das classes usando *CRC Cards.*

Para a parte do modelo do *design pattern* MVC já temos algumas nomes das entidades extraídas do *Domain Model.* Para a parte do Controlador temos os *User Stories.* (Hemrajani, 2006)

Os *CRC Cards* (Cartões de Classe, Responsabilidade e Colaboração) fornecem uma técnica informar OO para descobrir as interacções entre as classes. *CRC Cards* também podem ser usados para criar diagramas de classes caso seja necessário.

A tabela 6.2 mostra a disposição de um simples *CRC Card* com algumas explicações dos três componentes que contem.

Nome da Classe (Substantivo)	
Responsabilidades (obrigações desta classe, tais como métodos de negócio, excepções, métodos da segurança, atributos/variáveis)	Colaboradores (outras classes exigidas para fornecer uma solução completa a uma exigência de nível elevado)

Tabela 5-2:Disposição de um Simples CRC Card (Hemrajani, 2006)

Horario	
Representa o Quadro de Horário	Classe
Guarda o turno	Professor
Guarda o ciclo	Disciplina
Guarda as Classes	
Guarda os dados dos Professores e Turmas e as Aulas entre eles	

Tabela 5-3: Exemplo de *CRC Card* para a classe *Horario*

HorarioManager	
Busca os Horarios na DB	Horario
Grava o Horario na Base de Dados	
Optimiza o Horario usando o algoritmo descrito no Capito Anteriror	

*Tabela 5-4:*Exemplo de *CRC Card* para a classe HorarioManager

HorarioListController	
Controlador (em MVC) para retornar uma lista de Horários	HorarioManager

Tabela 5-5: Exemplo de *CRC Card* para a classe HorarioManager

5.6 Mapa de Fluxo da Aplicação

Mapa de Fluxo da Aplicação mostra como o UI irá funcionar (ou fluir) de um ponto ao outro. Esta técnica serve também mapeia os *User Stories* a camada de apresentação (V do MVC), que por sua vez mapeia para o Controlador e este por fim para o objectos do modelo. (Hemrajani, 2006)

Story Tag	Visão	Classe Controladora	Colaboradores	Tabelas Afectadas
Criar Horário	criarhorario	HorarioControler	HorarioManager	Horario
Listar Horário	listarhorario	HorarioListControler	HorarioManager	Horario

Tabela 5-6:Exemplo de um Mapa de Fluxo da Aplicação

5.7 Diagrama de Classes

Vejamos agora o rudimentar diagrama de classes. Segundo (Hemrajani, 2006) este é um passo opcional porque os *CRC Cards* e o Mapa de Fluxo da Aplicação disponibilizam informação suficiente para começamos a programação.

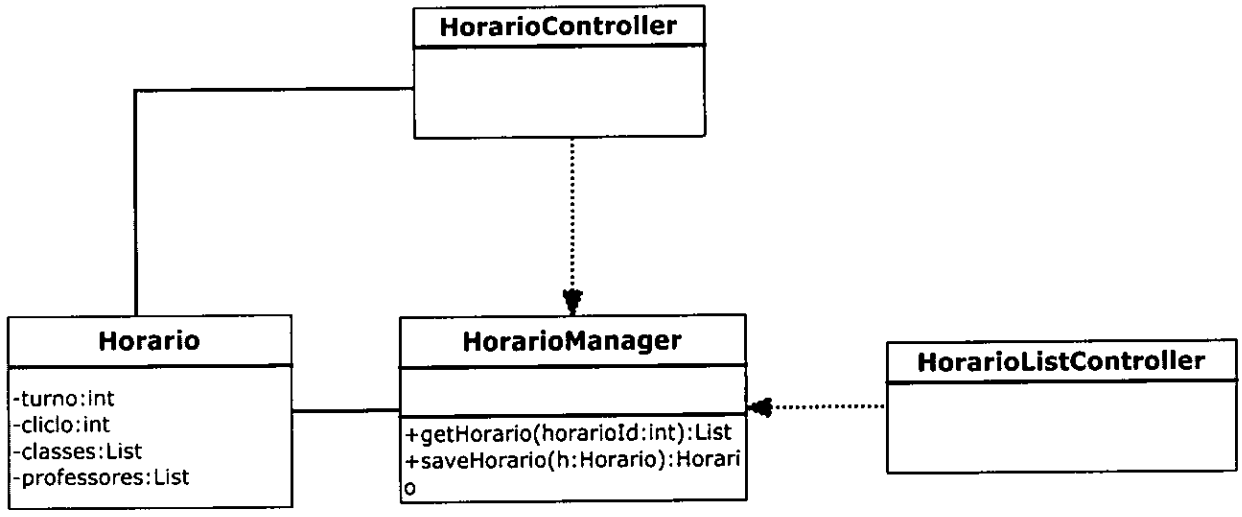


Tabela 5-7:Exemplo de um simples e mínimo Diagrama de Classes da Aplicação

5.8 Diagrama de Pacotes

São uma forma dos diagramas de classes. A diferença é que o digrama de pacotes mostra a relação entre os pacotes individuais. Podemos considerar o diagrama de pacotes uma visão abrangente do Sistema. (Knoernschild, 2001)

A figura 6.8 mostra o diagrama de pacotes do sistema.

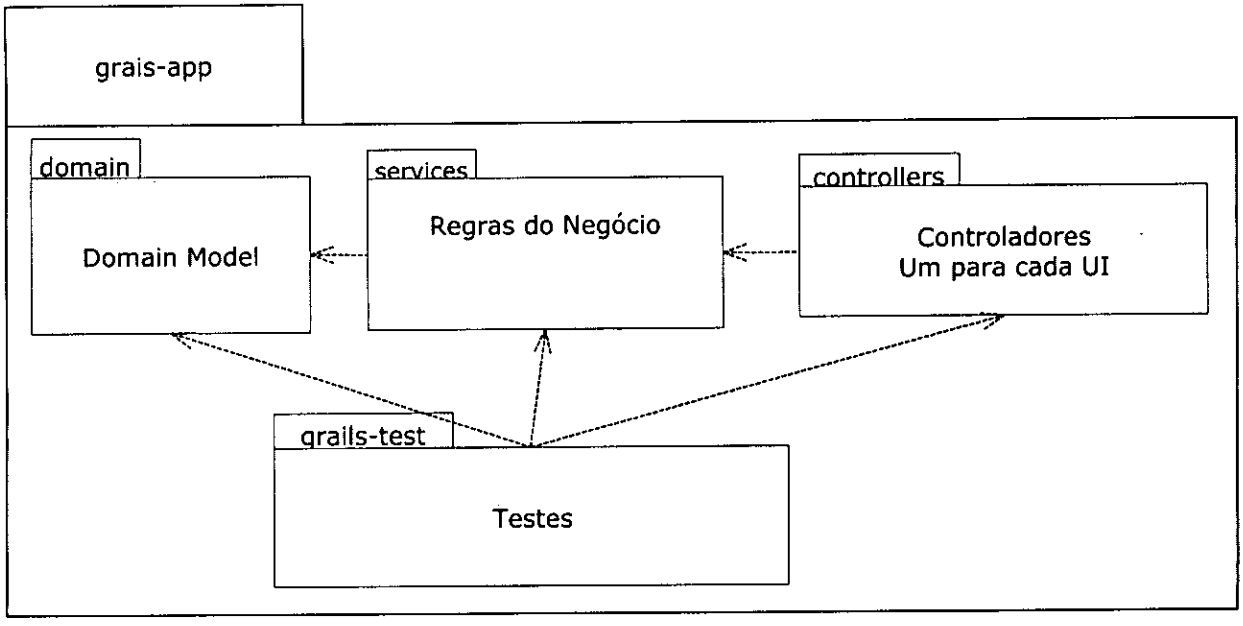


Figura 5-8:Diagrama de Pacotes do Sistema

5.9 Estrutura do Directório

Dado que no desenvolvimento do projecto foi usado o Apache Maven 2 como *buid tool* foi adoptada a estrutura de directórios padrão do Maven.A figura 6.9 apresenta a estrutura do directório da Aplicação e a Tabela 6.8 a respectiva descrição.

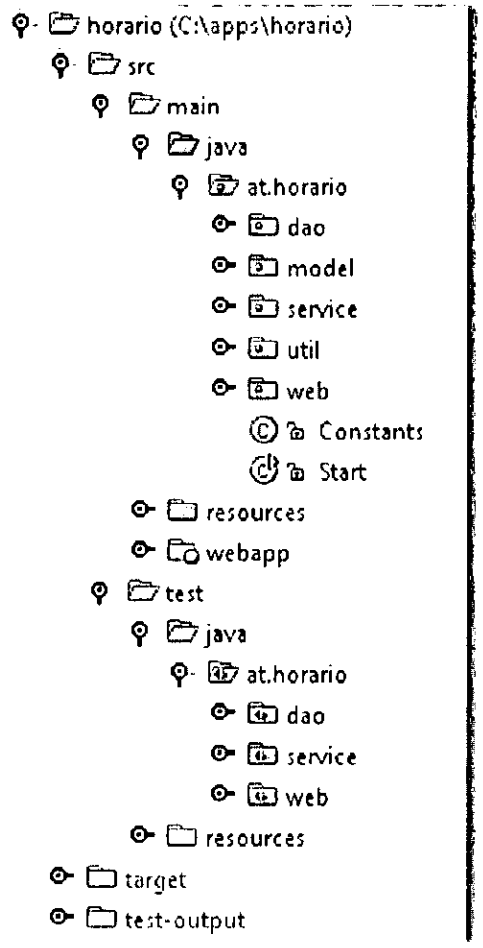


Figura 5-9:Estrutura do Directórios

5.10 Testes de Aceitação

Os testes de Aceitação podem servir como detalhe dos requisitos tal com o muitos projectos *Agile*. Um exemplo é uma lista de operações validas que podem ser executadas numa determinada tela da Aplicação. A ideia de user testes de aceitação como requisitos é útil porque estes testes são algo que o *Cliente* espera da aplicação. (Hemrajani, 2006)

Exemplo de teste de aceitação para a nossa aplicação é:

Login

- O nome do Utilizador deverá contém 10 caracteres. A palavra-chave deve ter entre oito a 10 caracteres
- Só utilizadores validos podem fazer o *Login*

6 CONCLUSÕES E RECOMENDAÇÕES

Este trabalho foi escrito para facilitar a programação de horários referentes aos alunos nas Escolas Secundárias Moçambicanas através de processos automatizados desenvolvidos em ambiente *Web*. E disponibiliza-lo em formato digital de modo a ser integrado com outros Sistemas existentes nas referidas Escolas. Pelo volume de estudantes que a Escola Secundária Francisco Manyanga acolhe e por ter feito os meu estudos pré-universitários lá e ter vivido de perto as dificuldades inerentes a programação de horários, foi escolhida como ponto de partida para a resolução do problema.

O sistema actual de programação de horários na Escola Secundária Francisco é manual e penoso exigindo bastante tempo e atenção por parte dos seus intervenientes. Isto faz com que este não esteja isento de constrangimentos (não respeito as cargas horárias limite dos professores, dispersão irregular das aulas não permitindo a segmentação da materia, etc) ou mesmo disponibilização de informação a tempo e hora, devido a necessidade de satisfazer um conjunto de restrições pedagógicas, administrativas e até mesmo pessoais, muitas das quais conflituantes entre si.

A proposta de implementação do modelo que funciona em ambiente *Web*, tem em vista solucionar estes constrangimentos pelo uso do sistema, dentre eles a perda de tempo e a falta de informação disponível em tempo real, entre outros recursos adicionais.

Neste contexto concluiu-se que:

- A necessidade de automatização dos processos de trabalho e acesso à informação em tempo real faz com que os desenvolvedores de sistemas adoptem tecnologias que disponibilizem interface *Web*. Para tal, foram desenvolvidas páginas lógicas e dinâmicas que permitam com que os utilizadores possam aceder remotamente as funcionalidades do sistema;
- A tecnologia usada para implementação deste modelo depende de certos factores tais como: dimensão da organização, número de usuários, natureza da informação que irá tratar, vantagens e desvantagens do seu uso, que não implique custos não suportáveis da aplicação desenvolvida e que seja fundamentalmente de fácil manutenção e segura. Assim sendo, foi usada a tecnologia Wicket para a camada de apresentação e Controle, o Hibernate para

camada de persistência, e o Spring para a inversão de controle para o desenvolvimento das páginas *Web* para o modelo proposto;

- Wicket é uma tecnologia para desenvolvimento de páginas *Web,* que é orientada a componentes e que se destaca das demais, por apresentar uma solução baseada em somente puro código Java e HTML isto é com distinção entre lógica e código HTML, um modelo de dados em POJOs e livre de XML. Simplificando deste modo o desenvolvimento de aplicações. O Wicket corre sobre qualquer Sistema Operativo, desde que possua um servidor que a suporte. Para o efeito, usou-se o Jetty, que se mostrou o servidor mais adequado para suportá-lo pelas suas caracteristicas (Simplicadade, Escalabilidade, Pode ser embebido em qualquer aplicação, facilmente usado em forma de *plugin)* e por ter a possibilidades de ser obtido gratuitamente na *Internet*;
- O SGBD usado, HSQLDB, é o mais adequado para este Sistema visto ele:
    - Fornecer recursos simplificados e apropriados para as suas aplicações, tendo um custo extremamente reduzido;
    - Fornecer um completo acesso ao código fonte, assegurando uma maior liberdade para as diversas plataformas existentes;
    - Ser suportado por uma série de plataformas como o Linux, MacOS X, UNIX e Microsoft Windows;
    - É 100% Java um SGBD completo com as capacidades do Java em relação a objectos
    - Em relação a outros poder ser adquirido gratuitamente na *Internet.*

- Após diversos testes de usabilidade com alguns dos utilizadores do sistema conclui-se também que, a implementação deste modelo, trouxe soluções automatizadas para transacções que dizem respeito à Programação de Horários para Alunos na ESFM.

Recomendações:

De modo a resolver os actuais problemas do Sistema actual de Programação de Horários na ESFM, com recurso aos resultados do presente trabalho recomenda-se:

- O treinamento do pessoal envolvido;

- A instalação e uso do Sistema proposto para ESFM e outras escolas Secundárias moçambicanas com sistema de Ensino similar a ESFM para programação de horários dos alunos;
- O estudo e extensão do presente trabalho para que possa abranger a programação de horários nas Escolas não só de alunos assim como de Provas, Exames, etc.
- O estudo e estensão do presente trabalho para que possa abranger a programação de horários não só de Escolas assim como de outras instituições.

7 BIBLIOGRAFIA

7.1 Bibliografia Referenciada

Alvarez-Valdes, R., Martin, G., & Tamarit, J. (1996). *Constructing Good Solutions for the Spanish School Timetabling Problem. Journal of the Operational Research Society.*

Ambler, S. W. (2006). (Ambysoft Inc.) Consultado em 11 de Junho de 2007, de http://www.agilemodeling.com/artifacts/uiPrototype.htm

Ambler, S. W. (2007). (Ambysoft Inc.) Consultado em 11 de Junho de 2007, de http://agilemodeling.com/

Ambler, S. W. (2006). *Agile Modeling: Effective Practices for Modeling and Documentation.* (Ambysoft Inc.) Consultado em 14 de Junho de 2007, de User Stories: http://www.agilemodeling.com/artifacts/userStory.htm

Apache Software Foundation. (2007). *Apache Wicket.* Consultado em 16 de Outubro de 2007, de Apache Wicket: http://wicket.apache.org/

Barclay, K., & Savage, J. (2007). *Groovy Programming : An Introduction for Java Developers.* Elsevier Inc.

Barclay, K., & Savage, J. (2006). *Groovy Programming: An Introduction for Java Developers.* Morgan Kaufmann.

Bastos, M., & Ribeiro, C. (1999). *Reactive Tabu Search with Path Relinking for the Steiner Problem in Graphs. Third Metaheuristics Interancional Conference* (pp. 31-36). Brazil: Angra dos Reis.

Beck, K. (2000). *Extreme Programming Explained.* Addison-Wesley.

Booch, G., Maksimchuk, R. A., Engle, M. W., & Young, B. J. (2007). *Object-Oriented Analysis and Design with Applications* (3ª Edição ed.). Massachusetts: Addison-Wesley.

Coelho, C., & Fernanda, M. (1999). *Dicionário breve de informática e multimédia.* Lisboa: Editorial Presença.

Costa, J., & Melo, A. S. *Dicionário de Português* (10ª Edição ed.). 2000: Porto Editora.

Dammeyer, F., & Voß, S. (1993). *Dynamic tabu list management using the reverse elimination method. In Hammer, P.L (editor), Tabu Search, v41, Annals of Operations* Research. Amsterdan: Baltzer Science Publishers.

de Werra, D. (1989). "*Tabu Search Techniques : A Tutorial and an Application to Neural Networks.*" OR Spektrum.

Donwsland, K. (1993). *Simulated Anneling. In Reeves, CR(Editor), Modern Heuristic Techniques for Combinatorial Problems, Advanced Topics in Computer Science Series.* London: Black Scientific Publications.

Feo, T. A., & Resende, M. (1995). *Greedy randomized adaptiva search procedures. Journal of Global Optimization* , 109-133.

Glover, F. (1996). *Tabu Search and adaptive memory programming - advances, aplications and challenges. In R. Barr, R. Helgason, and J. Kennigton, editors, Interfaces in Computer Sciences and Operations Research.* Kluwer Academic Publishers.

Glover, F., & Hanafi, S. *Tabu Search and Finite Convergence. To apper in Discrete Applied Mathematics.*

Glover, F., & M., L. (1997). *Tabu Search.* Boston: Kluwer Academic Publishers.

Gurumurthy, K. (2006). *Pro Wicket.* Apress.

Hanafi, S. *On the Convergenc of Tabu Search. To appear in Jounal of Heuristics.*

Hemrajani, A. (2006). *Agile Java Development with Spring, Hibernate and Eclipse.* Sams.

Hertz, A. (1992). *Finding a feasible course shedule using tabu search, Discrete Applied Mathematics.*

Kirkpatrick, S., Gellat, D., & Vecchi, M. (1983). *Optimization by Simulated Annealing.* 671-680. Science, v. 220.

Knoernschild, K. (2001). *Java™ Design: Objects, UML, and Process.* Addison Wesley.

König, D., Glover, A., King, P., Laforge, G., & Skeet, J. (2007). *Groovy in Action.* Manning Publications.

Nunes, M. E. (2001). *Fundamental de UML.* Lisboa: Editora de Informática FCA.

Oliveira, A. P. (2001). *Apostila Servlet/JSP,.* Universidade Federal de Viçosa.

Pedro, P. (2007). *Metodologia Agile Modeling, Caso de estudo: Emissão de Declarações e Certificados da UEM* . Maputo: UEM.

Prais, M., & Ribeiro, C. C. (1999). *Variação de Parâmetros em Procedimentos GRASP.* Investigación Operativa.

Ribeiro, C., E, U., & F., W. R. (2002). A hybrid GRASP with pertubations for the Seiner problem in graphs. *INFORMS Jornal on Computing* , 14:228-246.

Rocher, G. K. (2006). *The Definitive Guide to Grails.* Apress.

Rosseti, L. C. (2003). *Estratégias sequenciais e paralelas de GRASP com reconexão por caminhos para o problema de síntese de redes a 2-caminhos.* Rio de Janeiro: Pontifícia Universidade Católica do Rio de Janeiro.

Schaefer, A. (1996). *Tabu Search techniques for large high-school timetabling problems. 30th National Conference on Artificial Intelligence,* (pp. 363-368).

Shaef, A. (1999). *A survey of automated timetabling, Artificial Intelligence Review.*

Sousa, M. . J. (2000). *Programação de Horários em Escolas: Uma aproximação por Metaheurísticas.* Thesis, Rio de Janeiro.

Wells, D. (2006). (Don Wells) Consultado em 11 de Junho de 2007, http://extremeprogramming.org/

Wikipédia. (2007). Consultado em 16 de Junho de 2007, http://pt.wikipedia.org/wiki/Web_browser

Wikipédia. (2007). Consultado em 16 de Junho de 2007, http://pt.wikipedia.org/wiki/Design_pattern

Wikipédia. (2007). Consultado em 16 de Junho de 2007, http://pt.wikipedia.org/wiki/Mvc

Wren, A. (1995). *Scheduling, timetabling and rostering – a special relationship?, ICPTAT'95 – Proceeding of the Internacional Conference on the Practice and Theory of Automate Timetabling.*

7.2 Bibliografia Consultada

Amaral, W. (1999), Guia Para Apresentação de Teses, Dissertações, Trabalhos de Graduação, 2ª Edição, Livraria Universitária, UEM

Rudolph, Jason (2006), *Getting Started with Grails*, ISBN 978-1-4303-0782-2, C4Media Inc, InfoQ.

8 ANEXO A – GUIÃO DE ENTREVISTAS

Entrevista - Responsável pela Programação de Horários – Prof. Raúl Germano

13 de Novembor de 2006

Objectivo da Entrevista:

Obter conhecimento acerca do funcionamento do sistema em foco e a opinião relativa à ao sistema.

As questões a serem submetidas serão:

1. Quanto à estrutura da Escola?
    a. Quantas salas têm?
    b. Quantos laboratórios têm?
    c. Qual é a hora do intervalo maior?
    d. Qual o número de funcionários?
    e. Quais são os horários de início e termino das aulas?
2. Quanto ás Turmas?
    a. Quantas aulas cada turma tem?
    b. Quantos horários são disponibilizados por cada turma?
    c. Quantos turnos têm?

d. Quantas turmas têm por turno?

e. Qual é o número máximo de aulas de uma mesma disciplina para mesma turma?

3. Quanto aos professores?

a. Como é feita a escolha de professor por disciplina?

b. Quantos professores estão associados por matéria a cada turma?

c. Quantos professores são efectivos e quantos são contratados?

d. Qual o número mínimo de horas/aulas por professor?

e. Qual o número máximo de horas/aulas por professor?

f. Os professores têm algum tipo de folga?

g. Como é tratada as preferências e as indisponibilidades dos professores por horário?

h. Existem casos de professores que necessitam dar aulas em um mesmo horário para turmas distintas?

4. Quanto as aulas

a. Existem aulas duplas, triplas? É obrigatório?

b. Uma aula que inicia antes do intervalo maior e outra que continua após o intervalo maior é considerada aula dupla?

c. Aulas com janelas são permitidas?

d. Há relação de precedência entre as disciplinas?

e. Qual o número de aulas por disciplina?

5. Outros

a. Quanto tempo demora a programar um horário?

b. O que gostaria que fosse o *output* desse sistema (computarizado)?

c. Mais alguma coisa para reforçar?

9 ANEXO B – ALGUMAS PAGÍNAS DO SISTEMA

Nesta secção do documento, são apresentadas algumas páginas do Sistema incluindo a descrição das principais funcionalidades do sistema.

O Sistema foi concebido para facilitar a programação de horários através de um interface *web*. Com a este sistema poderá se criar horários dos estudantes de uma forma rápida e eficiente. O tempo de programação de horários pode levar dias quando processando manualmente e neste sistema diminuimos drasticamente para poucos minutos (normalmente menos que 1 hora de processamento). E de modo que o utilizador não espere tanto tempo na tela disponibilizamos um mecanimos de notificações e da escolha da hora de processamento.

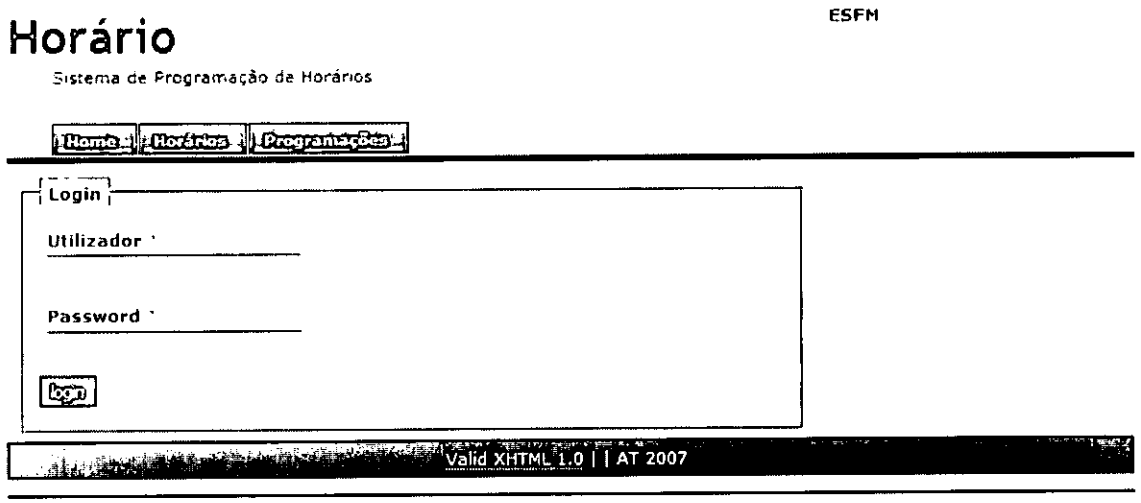
O sistema foi desenvolvido usando a ferramente da *build* Maven usando as tecnologias Wicket para apresentação, Spring para inversão de controle para servir de ponte no uso de outras bibliotecas, Hibernate para a camanda de persitência, Test-NG para testes. Desta forma, o sistema pode facilmente ser continuado por outros programadores.

Estrutura do Sistema

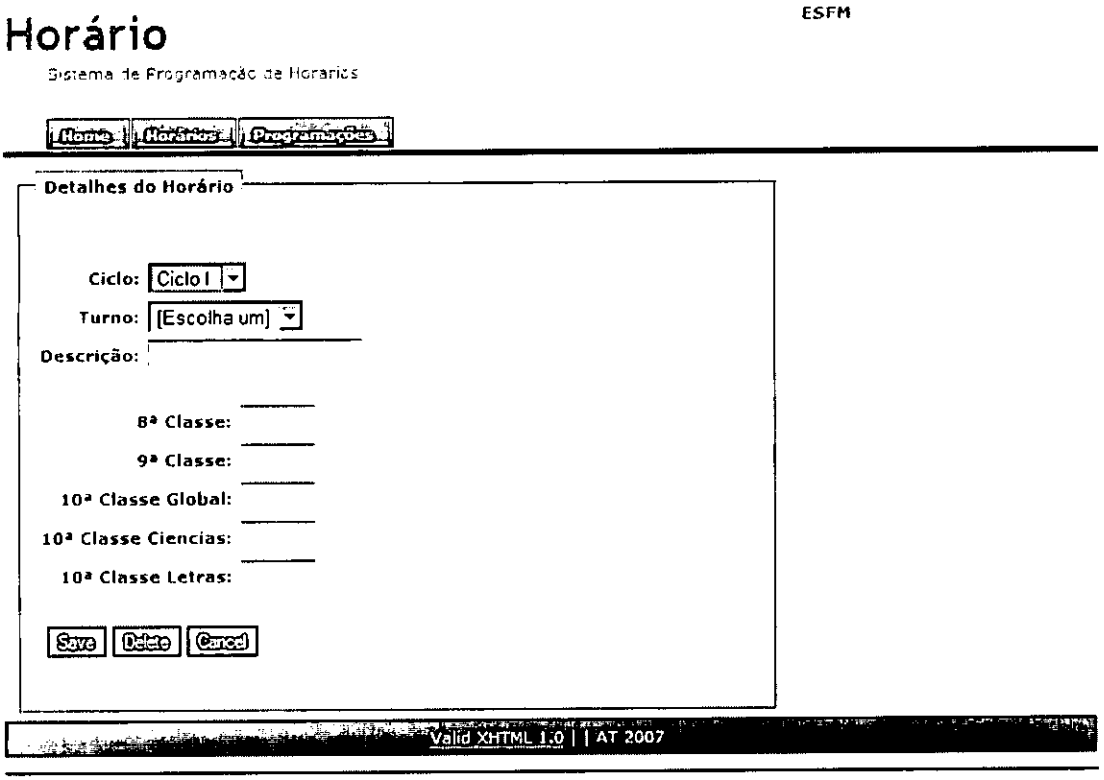
O Sistema foi concebido para ser usado pelos seguintes profiles: *Administrador do Sistema* e *Utilizador Simples*.

O administrador do sistema é o responsável pela manutenção da aplicação, ele não possui acesso a todas telas, por razões de segurança. E o utilizador simples apenas criar um horário, exporta-lo e optimiza-lo.

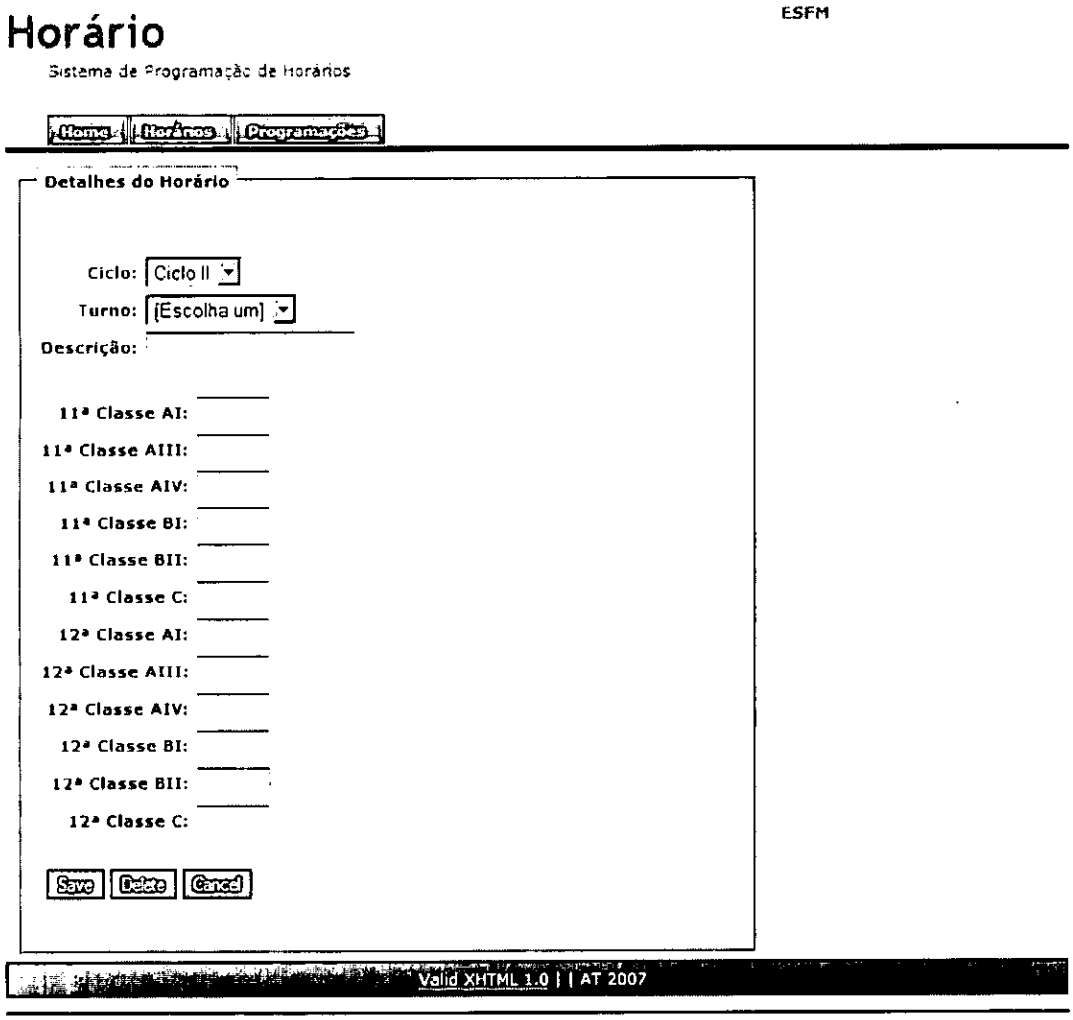
Página Inicial



Criação do Horário (Ciclo I)



Criação do Horário (Ciclo II)


ESFM
Horário
Sistema de Programação de Horários
Home
Horários
Programações
Detalhes do Horário
Ciclo:
Ciclo II
Turno:
[Escolha um]
Descrição:
11ª Classe AI:
11ª Classe AIII:
11ª Classe AIV:
11ª Classe BI:
11ª Classe BII:
11ª Classe C:
12ª Classe AI:
12ª Classe AIII:
12ª Classe AIV:
12ª Classe BI:
12ª Classe BII:
12ª Classe C:
Save
Delete
Cancel
Valid XHTML 1.0 | | AT 2007

Lista de Horários

Entrar | Horários | Detalhes do Horário | Programações | Sair

Lista de Horários

Novo Horario

A mostrar 1 à 3 de 5 << < 1 2 > >>

Horário ID	Ciclo	Turno	Descrição	
1	Ciclo I	Diurno	sdfsdfdf	
8	Ciclo I	Diurno	fgsdfgsdfg	
9	Ciclo I	Noturno	adfasdfafd	

Quadro de Horários

excel

	2ªFeira						3ªFeira						4ªFeira	
	1	2	3	4	5	6	1	2	3	4	5	6	1	2
P1	8.1	8.2	8.1	8.5	8.3	8.4	8.2	8.5	8.5	8.2	8.4	8.4	8.5	8.5
P2	9.4	9.5	9.1	9.4	9.2	9.3	9.3	9.5	9.4	9.4	9.5	9.3	9.3	9.4
P3	10.3	10.2	10.2	10.4	10.1	10.5	10.2	10.2	10.1	10.3	10.1	10.3	10.3	10.4
P4	10.14	10.14	10.13	10.11	10.11	10.13	10.14	10.12	10.15	10.13	10.11	10.15	10.12	10.11
M1	8.2	8.5	8.2	8.4	8.4	8.5	8.1	8.3	8.3	8.5	8.2	8.5	8.3	8.1
M2	9.3	9.2	9.4	9.3	9.5	9.1	9.4	9.4	9.2	9.3	9.1	9.5	9.4	9.1
M3	10.4	10.4	10.3	10.3	10.2	10.1	10.3	10.3	10.4	10.4	10.2	10.5	10.5	10.3
M4	10.9	10.6	10.6	10.6	10.6	10.7	10.6	10.7	10.10	10.6	10.9	10.9	10.6	10.7
H1	9.1	8.1	8.4	10.1		9.5	9.5	9.3	10.2	8.3		10.1	9.2	9.2
H2	10.5	10.13	10.14	10.15	10.13	10.11	10.12	10.11		10.14	10.12		10.15	
G1	10.2	10.1	9.5	9.5	9.5	9.4	10.1	8.2	8.1	9.2	8.1	8.3		10.2
G2	10.12	10.12	10.11	10.13	10.14	10.15	10.15	10.14	10.13	10.5	10.5		10.11	
B1	8.5	8.4	8.5	9.1	8.2	9.3	9.2	8.4	9.3	9.2	9.2	8.2	9.1	8.4
B2	10.1	9.4	10.1	10.5	10.4	10.4	10.2		9.5		10.3			
F1	8.4	9.1	8.5	9.2	9.1	8.2	8.4	8.3	9.3	8.4	8.3	9.1	8.1	
F2	9.5	10.3	10.6	9.4	10.6	10.2	10.1	10.5		10.2		10.6	10.1	10.5
F3	10.8	10.10	10.7	10.7	10.10	10.8	10.10	10.9	10.9	10.7			10.8	
I1	8.5	8.3	9.3	8.3	9.3	8.1	8.3	8.1	8.2	8.1	8.5	9.2	8.4	8.2
I2	10.11	10.11		9.5	10.3		10.11	10.1	10.5		10.4	9.4	10.4	10.1
I3	10.13	10.15	10.15	10.12	10.12	10.14	10.13		10.14	10.15	10.14	10.12	10.13	
Q1	9.2	9.3	8.3	8.1	8.1	8.3	9.1	9.2	8.4	9.1	9.3	8.1	8.2	9.3
Q2	10.6	10.3	10.4	10.2	10.5	10.3	10.5	10.4	10.3	9.5	9.4	10.4	10.2	9.5
Q3	10.7	10.9	10.9	10.8	10.7	10.10	10.6	10.8		10.10	10.7	10.10	10.9	
D1	8.3	9.4	9.2	8.2	9.4	9.2	8.5	9.1	9.1	10.1		10.2	9.5	8.3
D2	10.10	10.5	10.5	10.10	10.9	10.6	10.4	10.6	10.7	10.5		10.8	10.7	10.9

File - C:\apps\horario\src\main\java\at\horario\model\Horario.java

```
 1  package at.horario.model;
 2
 3  import static javax.persistence.CascadeType.ALL;
 4  import javax.persistence.*;
 5  import static javax.persistence.FetchType.EAGER;
 6  import static javax.persistence.FetchType.LAZY;
 7  import static javax.persistence.GenerationType.AUTO;
 8  import java.io.Serializable;
 9  import java.util.Date;
10  import java.util.Set;
11
12  @Entity
13  public class Horario extends GenericObject<Long> implements Serializable {
14     @ManyToOne(fetch = EAGER)
15     @JoinColumn(name = "ciclo")
16     private Ciclo ciclo;
17     @Column(precision = 1, scale = 0)
18     private Turno turno;
19     @Column(length = 50)
20     private String descricao;
21     @Column(length = 23)
22     private Date dataCriacao;
23     @Column(length = 1)
24     private Boolean processado;
25     @Id
26     @GeneratedValue(strategy = AUTO)
27     private Long horarioId;
28     private User user;
29
30     @OneToMany(fetch = LAZY, mappedBy = "horario", cascade = ALL)
31     private Set<HorarioClasseGrupo> horarioClasseGrupos;
32     @OneToMany(fetch = LAZY, mappedBy = "horario", cascade = ALL)
33     private Set<Turma> turmas;
34     @OneToMany(fetch = LAZY, mappedBy = "horario", cascade = ALL)
35     private Set<Professor> professores;
36     @OneToMany(fetch = LAZY, mappedBy = "horario", cascade = ALL)
37     private Set<Celula> celulas;
38     @OneToMany(fetch = LAZY, mappedBy = "horario", cascade = ALL)
39     private Set<HorarioSchedule> horarioSchedules;
40
41     public Set<HorarioSchedule> getHorarioSchedules() {
42        return horarioSchedules;
43     }
44
45     public void setHorarioSchedules(Set<HorarioSchedule> horarioSchedules) {
46        this.horarioSchedules = horarioSchedules;
47     }
48
49     public Set<Celula> getCelulas() {
50        return celulas;
51     }
52
53     public void setCelulas(Set<Celula> celulas) {
54        this.celulas = celulas;
55     }
56
57     public Set<Professor> getProfessores() {
58        return professores;
59     }
60
```

File - C:\apps\horario\src\main\java\at\horario\model\Horario.java

```
 61    public void setProfessores(Set<Professor> professores) {
 62        this.professores = professores;
 63    }
 64
 65    public Set<Turma> getTurmas() {
 66        return turmas;
 67    }
 68
 69    public void setTurmas(Set<Turma> turmas) {
 70        this.turmas = turmas;
 71    }
 72
 73
 74    public Ciclo getCiclo() {
 75        return ciclo;
 76    }
 77
 78    public void setCiclo(Ciclo ciclo) {
 79        this.ciclo = ciclo;
 80    }
 81
 82    public Turno getTurno() {
 83        return turno;
 84    }
 85
 86    public void setTurno(Turno turno) {
 87        this.turno = turno;
 88    }
 89
 90    public String getDescricao() {
 91        return descricao;
 92    }
 93
 94    public void setDescricao(String descricao) {
 95        this.descricao = descricao;
 96    }
 97
 98    public Date getDataCriacao() {
 99        return dataCriacao;
100    }
101
102    public void setDataCriacao(Date dataCriacao) {
103        this.dataCriacao = dataCriacao;
104    }
105
106    public Boolean getProcessado() {
107        return processado;
108    }
109
110    public void setProcessado(Boolean processado) {
111        this.processado = processado;
112    }
113
114    public Long getHorarioId() {
115        return horarioId;
116    }
117
118    public void setHorarioId(Long horarioId) {
119        this.horarioId = horarioId;
120    }
```

File - C:\apps\horario\src\main\java\at\horario\model\Horario.java

```
121
122
123     public String toString() {
124         return "Horario{" +
125                 "ciclo=" + ciclo +
126                 ", turno=" + turno +
127                 ", descricao='" + descricao + '\'' +
128                 ", dataCriacao=" + dataCriacao +
129                 ", processado=" + processado +
130                 ", horarioId=" + horarioId +
131                 ", user=" + user +
132                 '}';
133     }
134
135     public boolean equals(Object o) {
136         if (this == o) return true;
137         if (!(o instanceof Horario)) return false;
138
139         Horario horario = (Horario) o;
140
141         return !(horarioId != null ? !horarioId.equals(horario.horarioId) : horario.horarioId != null);
142
143     }
144
145     public int hashCode() {
146         int result;
147         result = (ciclo != null ? ciclo.hashCode() : 0);
148         result = 31 * result + (turno != null ? turno.hashCode() : 0);
149         result = 31 * result + (descricao != null ? descricao.hashCode() : 0);
150         result = 31 * result + (dataCriacao != null ? dataCriacao.hashCode() : 0);
151         result = 31 * result + (processado != null ? processado.hashCode() : 0);
152         result = 31 * result + (horarioId != null ? horarioId.hashCode() : 0);
153         return result;
154     }
155
156     public Long getId() {
157         return horarioId;
158     }
159
160
161     public Set<HorarioClasseGrupo> getHorarioClasseGrupos() {
162         return horarioClasseGrupos;
163     }
164
165     public void setHorarioClasseGrupos(Set<HorarioClasseGrupo> horarioClasseGrupos) {
166         this.horarioClasseGrupos = horarioClasseGrupos;
167     }
168 }
169
```

File - C:\apps\horario\src\test\java\at\horario\dao\ListDataTest.java

```
 1 package at.horario.dao;
 2
 3 import at.horario.model.*;
 4 import org.hibernate.criterion.DetachedCriteria;
 5 import static org.hibernate.criterion.Restrictions.eq;
 6 import org.testng.annotations.Test;
 7 import static org.unitils.reflectionassert.ReflectionAssert.assertRefEquals;
 8 import org.unitils.spring.annotation.SpringApplicationContext;
 9 import org.unitils.spring.annotation.SpringBeanByName;
10
11 import java.util.List;
12
13 @SpringApplicationContext({"classpath*:/applicationContext*.xml"})
14 public class ListDataTest extends BaseUnitilsTestNG {
15     @SpringBeanByName
16     private GenericDao<Ciclo, Integer> cicloDao;
17     @SpringBeanByName
18     private ClasseDao classeDao;
19     @SpringBeanByName
20     private GenericDao<Grupo, String> grupoDao;
21     @SpringBeanByName
22     private GenericDao<Disciplina, String> disciplinaDao;
23     @SpringBeanByName
24     private ChProfessorDao chProfessorDao;
25     @SpringBeanByName
26     private ChDisciplinaDao chDisciplinaDao;
27
28
29     @Test
30     public void testCiclos() {
31         List<Ciclo> result = cicloDao.getAll();
32         System.out.println("Lista de Ciclos:" + result);
33         assertRefEquals("Lista de Ciclos vazia", result.isEmpty(), false);
34         assertRefEquals("Lista de Ciclos diferente de 2", result.size(), 2);
35     }
36
37     @Test
38     public void testDisciplinas() {
39         List<Disciplina> result = disciplinaDao.getAll();
40         System.out.println("Lista de Disciplinas:" + result);
41         assertRefEquals("Lista de Disciplinas vazia", result.isEmpty(), false);
42         assertRefEquals("Lista de Disciplinas diferente de 11", result.size(), 11);
43     }
44
45     @Test
46     public void testClasses() {
47         List<Classe> result = classeDao.getAll();
48         for (int i = 0; i < result.size(); i++) {
49             Classe grupo = result.get(i);
50             //System.out.println((i+1)+"="+grupo);
51         }
52         assertRefEquals("Lista de Classes vazia", result.isEmpty(), false);
53         assertRefEquals("Lista de Classes diferente de 5", result.size(), 5);
54     }
55
56     @Test
57     public void testGrupos() {
58         List<Grupo> result = grupoDao.getAll();
59         for (int i = 0; i < result.size(); i++) {
60             Grupo grupo = result.get(i);
```

File - C:\apps\horario\src\test\java\at\horario\dao\ListDataTest.java

```
61          //System.out.println((i+1)+"="+grupo);
62      }
63      assertRefEquals("Lista de Grupos vazia", result.isEmpty(), false);
64      assertRefEquals("Lista de Grupos diferente de 9", 9, result.size());
65  }
66
67  @Test
68  public void testGroupsII() {
69      DetachedCriteria criteria = grupoDao.criteria();
70      criteria.createCriteria("classes").add(eq("classeId", 11));
71      List<Grupo> result = grupoDao.getAll(criteria);
72      for (int i = 0; i < result.size(); i++) {
73          Grupo grupo = result.get(i);
74          System.out.println(grupo.getGrupoNome());
75      }
76      int size = 6;
77      assertRefEquals("Lista de Grupos diferente de " + size, size, result.size());
78  }
79
80  @Test
81  public void testClasseGrupos() {
82      List<Classe> result = classeDao.getAll();
83      int numGrupos = 0;
84      for (int i = 0; i < result.size(); i++) {
85          Classe classe = result.get(i);
86          DetachedCriteria criteria = grupoDao.criteria();
87          criteria.createCriteria("classes").add(eq("classeId", classe.getClasseId()));
88          List<Grupo> grupos = grupoDao.getAll(criteria);
89          numGrupos += grupos.size();
90          System.out.println(classe.getClasseId() + " ---" + grupos.size());
91      }
92      int expectedResult = 17;
93      assertRefEquals("Lista de Grupos diferente de " + expectedResult, expectedResult, numGrupos);
94  }
95
96  @Test
97  public void testChProfessor() {
98      List<ChProfessor> result = chProfessorDao.getAll();
99      for (int i = 0; i < result.size(); i++) {
100         ChProfessor grupo = result.get(i);
101         System.out.println((i + 1) + "=" + grupo);
102     }
103
104     int expResult = disciplinaDao.getAll().size() * 2;
105     assertRefEquals("Lista de ChProfessor vazia", result.isEmpty(), false);
106     assertRefEquals("Lista de ChProfessor diferente de " + expResult, expResult, result.size());
107 }
108
109
110 @Test
111 public void testChDisciplina() {
112     List<ChDisciplina> result = chDisciplinaDao.getAll();
113     for (int i = 0; i < result.size(); i++) {
114         ChDisciplina grupo = result.get(i);
115         System.out.println((i + 1) + "=" + grupo);
116     }
117
118     int expResult = 0;
119     List<Disciplina> disciplinaList = disciplinaDao.getAll();
120     for (int i = 0; i < disciplinaList.size(); i++) {
```

File - C:\apps\horario\src\test\java\at\horario\dao\ListDataTest.java

```
121         Disciplina disciplina = disciplinaList.get(i);
122         DetachedCriteria criteria = grupoDao.criteria();
123         criteria.createCriteria("disciplinas").add(eq("classeId", disciplina.getDisciplinaID()));
124         List<Grupo> l = grupoDao.getAll(criteria);
125         expResult += l.size() * 2;
126       }
127
128       assertRefEquals("Lista de ChDisciplina vazia", result.isEmpty(), false);
129       //TODO: to remove
130       //assertRefEquals("Lista de ChDisciplina diferente de "+expResult, expResult, result.size());
131     }
132
133
134 }
135
```

File - C:\apps\horario\src\main\java\at\horario\service\AlgoritmosManager2.java

```
 1 package at.horario.service;
 2
 3 //~--- non-JDK imports --------------------------------------------------------
 4
 5 import at.horario.model.algo.*;
 6 import static at.horario.model.algo.Quadro.*;
 7
 8 import java.util.ArrayList;
 9 import java.util.Arrays;
10 import java.util.LinkedList;
11 import java.util.Random;
12
13 /**
14  * Class description
15  *
16  * @author ATD
17  * @version 1.0, 2007.05.12 at 10:34:08 CAT
18  */
19 public class AlgoritmosManager2 {
20     public static final String[] NOMES_DIAS_SEMANA = {"Segunda Feira", "Terca Feira", "Quarta Feira", "Quinta Feira", "Sexta
    Feira"};
21     private HorarioAlgo horarioAlgo;
22     private int tamanho_LRC;
23     private int buscaTabuMax;
24     private int tamanhoListaTabu;
25     private final static int DIAS_SEMANA = 5;
26     public final static int TEMPOS_DIA = 6;
27
28     public final static int HORARIOS_SEMANA = TEMPOS_DIA * DIAS_SEMANA;
29     private int pesoGap;
30     private int pesoExcessoAula;
31     private int pesoSobreposicao;
32     private int graspMAX;
33
34
35     public void setGraspMAX(int graspMAX) {
36         this.graspMAX = graspMAX;
37     }
38
39     public AlgoritmosManager2() {
40     }
41
42
43     public int getPesoGap() {
44         return pesoGap;
45     }
46
47     public void setPesoGap(int pesoGap) {
48         this.pesoGap = pesoGap;
49     }
50
51     public int getPesoExcessoAula() {
52         return pesoExcessoAula;
53     }
54
55     public void setPesoExcessoAula(int pesoExcessoAula) {
56         this.pesoExcessoAula = pesoExcessoAula;
57     }
58
59     public int getPesoSobreposicao() {
```

File - C:\apps\horario\src\main\java\at\horario\service\AlgoritmosManager2.java

```
60      return pesoSobreposicao;
61    }
62
63    public void setPesoSobreposicao(int pesoSobreposicao) {
64      this.pesoSobreposicao = pesoSobreposicao;
65    }
66
67    public void setTamanhoListaTabu(int tamanhoListaTabu) {
68      this.tamanhoListaTabu = tamanhoListaTabu;
69    }
70
71    public void setTamanho_LRC(int tamanho_LRC) {
72      this.tamanho_LRC = tamanho_LRC;
73    }
74
75
76    public void setBuscaTabuMax(int buscaTabuMax) {
77      this.buscaTabuMax = buscaTabuMax;
78    }
79
80    /**
81     * Method description
82     *
83     * @param horarioSettings
84     */
85
86    /**
87     * Method description
88     *
89     * @param horarioAlgo
90     */
91    public void setHorario(HorarioAlgo horarioAlgo) {
92      this.horarioAlgo = horarioAlgo;
93    }
94
95    /**
96     * Method description
97     *
98     * @return
99     */
100   public Quadro construaSolucaoInicial() {
101     Quadro s = new Quadro(horarioAlgo.getNumProfessores(), HORARIOS_SEMANA TEMPOS_DIA);
102     int[] horarioCriticidades = new int[HORARIOS_SEMANA];   // criticidade dos horarios
103
104     // em principio todos professores estÃo disponiveis por isso , o numÃro de professores indisponiveis Ã igual a 0
105     int[] ordenaHC = new int[HORARIOS_SEMANA];   // ordena horarios segundo a criticidade, ordem decrescente
106
107     for (int i = 0; i < ordenaHC.length; i++) {
108       ordenaHC[i] = i;   // ordem normal dado que os horarios tem a mesma criticidade
109     }
110
111     ProfCH[] aulas = new ProfCH[horarioAlgo.getNumProfessores()];
112     int numAulas = horarioAlgo.getNumProfessores();
113     int total = 0;
114
115     for (int i = 0; i < horarioAlgo.getNumProfessores(); i++) {
116       ProfessorAlgo pa = horarioAlgo.getProfessores()[i];
117       ProfCH profCH = new ProfCH();
118
119
```

File - C:\apps\horario\src\main\java\at\horario\service\AlgoritmosManager2.java

```
120         for (int j = 0; j < pa.getIndisponibilidades().length; j++) {
121             int h = pa.getIndisponibilidades()[j];
122             s.getMatriz()[pa.getNum() - 1][h] = -20;
123         }
124
125
126         profCH.setCargaHoraria(pa.getCh());
127         profCH.setProfessor(pa.getNum());
128         profCH.setTurmas(new ArrayList<TurmaCH>());
129
130         int chProfessor = 0;
131
132         for (int j = 0; j < pa.getTurmas().length; j++) {
133             int turma = pa.getTurmas()[j];
134             int ch = pa.getCargaHorariaTurma(j);
135
136             profCH.getTurmas().add(new TurmaCH(turma, ch));
137             chProfessor += ch;
138         }
139         profCH.setCargaHoraria(chProfessor);
140
141         aulas[i] = profCH;
142         total += profCH.getCargaHoraria();
143     }
144
145     Arrays.sort(aulas);   // ordenar os Professores segundo a carga horÁria
146
147     Random oRandom = new Random();
148
149     while (total-- > 0) {
150         int virtualEscolhida = oRandom.nextInt((tamanho_LRC <= numAulas)
151                 ? tamanho_LRC
152                 : numAulas);                    // tamanho dinamico depende do tamanho da lista
153         int escolhida = 0;
154         for (int j = 0; j <= virtualEscolhida; escolhida++) {
155             if (!aulas[escolhida].getTurmas().isEmpty()) {
156                 j++;
157             }
158         }
159         escolhida--;
160         ProfCH profCH = aulas[escolhida];
161         int professor = profCH.getProfessor() - 1;
162         //System.out.println(profCH.getTurmas().size());
163         int innerEscolhida = oRandom.nextInt(profCH.getTurmas().size());
164         TurmaCH turmaCH = profCH.getTurmas().get(innerEscolhida);
165
166         profCH.addCargaHoraria();
167         turmaCH.subCargaHoraria();
168
169         if (turmaCH.getCargaHoraria() == 0) {          // se já tiver alocado todas as numeroTurmas
170             profCH.getTurmas().remove(innerEscolhida);   // vamos remover
171             if (profCH.getTurmas().isEmpty()) {
172                 //aulas[escolhida] = null;
173                 numAulas--;
174             }
175         }
176         ordernarAulas(aulas, professor);
177
178         // agora vamos tentar encaixar em um dos horÁrios
179         int t = 3;                              /*
```

File - C:\apps\horario\src\main\java\at\horario\service\AlgoritmosManager2.java

```
180                                         * o valor de t representa o tipo de alocação
181                                         * 3 se a aula for alocada sem inviabilidade
182                                         * 2 se a aula for alocada com inviabilidade do tipo 2
183                                         * 1 se a aula for alocada com inviabilidade do tipo 1
184                                         */
185         boolean alocada = false;
186
187         while (!alocada) {
188           for (int i = 0; (i < ordenaHC.length) && !alocada; i++) {   // do horario + critico ao menos critico
189             int h = ordenaHC[i];                    // horário par alocar a turma
190
191             if (s.getMatriz()[professor][h] == 0) {//se estiver disponivel
192               s.getMatriz()[professor][h] = turmaCH.getTurma();
193
194               switch (t) {
195                 case 3:// nao adminitimos nenhum restrição
196                   int sobreposicao = calculaSobreposicao(s, professor, h);
197
198                   if (sobreposicao > 0) {
199                     break;
200                   }
201
202                   int excessoAulas = calculaExcessoAulas(s, professor, h);
203
204                   if (excessoAulas > 0) {
205                     break;
206                   }
207
208                   alocada = true;
209
210                   break;
211
212                 case 2:                          // admitimos so excesso de aulas
213                   sobreposicao = calculaSobreposicao(s, professor, h);
214
215                   if (sobreposicao > 0) {
216                     break;
217                   }
218
219                   alocada = true;
220                 default:                         // admitimos a sobreposicao e o excesso de aulas
221                   alocada = true;
222               }// switch
223
224               if (!alocada) {
225                 s.getMatriz()[professor][h] = 0;
226               } else {
227                 horarioCriticidades[h]++;//horário aumentou
228                 ordernarHC(ordenaHC, horarioCriticidades, h);
229
230               }
231             }
232           }// for
233           t--;
234         } // while
235       }
236
237       return s;
238     }
239
```

File - C:\apps\horario\src\main\java\at\horario\service\AlgoritmosManager2.java

```
240    public void avaliar(Quadro s) {
241
242        /*
243         * Criar um array um para guardar as colunas e ordena-las
244         */
245        int[] colunas = new int[s.matriz.length];
246
247        s.setSobreposicao(0);
248        s.setExcessoaulas(0);
249        s.setGaps(0);
250        s.setValor(0);
251
252        /*
253         * Sobreposicao
254         */
255        for (int h = 0; h < s.matriz[0].length; h++) {    // percorrer os horarios
256            s.sobreposicao += calculaSobreposicao(s, h);
257        }
258
259        /*
260         * Excesso dia,
261         */
262        for (int i = 0; i < s.matriz.length; i++) {    // percoerr as linhas=professores
263            calculaGapsAndExcessoaulas(s, i);
264
265            // if (tempGaps > 0) System.out.println("Gap:"+i);
266            // if (tempExcessoaulas > 0) System.out.println("TempExcessoaulas:"+i);
267            s.gaps += s.tempGaps;
268            s.excessoaulas += s.tempExcessoaulas;
269        }
270
271        s.valor = PESO_GAP * s.getGaps() + PESO_EXCESSO_AULA * s.getExcessoaulas() + PESO_SOBREPOSICAO * s.getSobreposicao();
272    }
273
274
275    private void ordernarHC(int[] ordenaHC, int[] horarioCriticidades, int h) {
276        for (int i = h; i > 0; i--) {
277            if (horarioCriticidades[i] > horarioCriticidades[i - 1]) {
278                int temp = ordenaHC[i];
279                ordenaHC[i] = ordenaHC[(i - 1)];
280                ordenaHC[(i - 1)] = temp;
281            }
282        }
283    }
284
285    private void ordernarAulas(ProfCH[] aulas, int professor) {
286        for (int i = professor; i > 0; i--) {
287            if (aulas[i].getCargaHoraria() > aulas[i - 1].getCargaHoraria()) {
288                ProfCH temp = aulas[i];
289                aulas[i] = aulas[(i - 1)];
290                aulas[(i - 1)] = temp;
291            }
292        }
293    }
294
295    /**
296     * Method description
297     *
298     * @param s
299     * @param horario
```

File - C:\apps\horario\src\main\java\at\horario\service\AlgoritmosManager2.java

```
300     * @return
301     */
302    private int calculaSobreposicao(Quadro s, int horario) {
303      int _sobreposicao = 0;
304
305      for (int p = 0; p < s.matriz.length; p++) {   // percorrer os professores para saber se ha sobreposicao
306        s.colunas[p] = s.matriz[p][horario];      // copiar a coluna
307      }
308
309      Arrays.sort(s.colunas);
310
311      int temp = 0;   // so tem direito a 1 sobreposicao por dia, supomos q 0 -1
312
313      for (int p = 1; p < s.matriz.length; p++) {       // comecamos da segunda turma
314        if (s.colunas[p] > 0) {                  // se tiver uma turma
315          if (s.colunas[p] != s.colunas[p - 1]) {   // se for diferente da anterior
316            _sobreposicao += temp;               // se for zero 0 pk so tem 1 e nao precisa contar
317            temp = 0;
318          } else {                          // aumenta a contagem
319            ++temp;
320          }
321        }
322      }
323
324      return _sobreposicao;
325    }
326
327    /**
328     * Method description
329     *
330     * @param s
331     * @param professor
332     * @param horario
333     * @return
334     */
335    private int calculaSobreposicao(Quadro s, int professor, int horario) {
336      int _sobreposicao = 0;
337      int turma = s.matriz[professor][horario];
338
339      for (int p = 0; p < s.matriz.length; p++) {   // percorrer os professores para saber se ha sobreposicao
340        if (s.matriz[p][horario] == turma) {
341          _sobreposicao++;
342        }
343      }
344
345      return _sobreposicao - 1;
346    }
347
348    /**
349     * Method description
350     *
351     * @param s
352     * @param professor
353     * @param horario
354     * @return
355     */
356    public int calculaExcessoAulas(Quadro s, int professor, int horario) {
357      int _excessoAula = 0;
358      int startHorario = (horario / TEMPOS_DIA) * TEMPOS_DIA;
359      int turma = s.matriz[professor][horario];
```

File - C:\apps\horario\src\main\java\at\horario\service\AlgoritmosManager2.java

```
360
361     for (int i = startHorario; i < startHorario + TEMPOS_DIA; i++) {
362       if (s.matriz[professor][i] == turma) {
363         _excessoAula++;
364       }
365     }
366
367     return (_excessoAula > 2)
368         ? _excessoAula - 2
369         : 0;
370   }
371
372
373   public String toString(Quadro quadro) {
374     int dia = 0;
375     String diasStr = "  |";
376
377     for (int i = 0; i < NOMES_DIAS_SEMANA.length; i++) {
378       String d = NOMES_DIAS_SEMANA[i];
379
380       while (d.length() < 18) {
381         if (d.length() % 2 == 0) {
382           d = "-" + d;
383         } else {
384           d = d + "-";
385         }
386       }
387
388       // System.out.println(""+d.length());
389       diasStr += d + "|";
390     }
391
392     String horarioStr = "  |";
393
394     for (int j = 1; j <= TEMPOS_DIA * NOMES_DIAS_SEMANA.length; j++, dia++) {
395       String h = "" + j;
396
397       while (h.length() < 3) {
398         h = " " + h;
399       }
400
401       horarioStr += h;
402
403       if (j % TEMPOS_DIA == 0) {
404         horarioStr += "|";
405       }
406     }
407
408     String s = "";
409
410     s += diasStr + "\n";
411     s += horarioStr + "\n";
412
413     for (int i = 0; i < horarioStr.length(); i++) {
414       s += "-";
415     }
416
417     s += "\n";
418
419     for (int i = 0; i < quadro.getMatriz().length; i++) {
```

File - C:\apps\horario\src\main\java\at\horario\service\AlgoritmosManager2.java

```
420         s += (i < 10)
421                 ? i + " "
422                 : "" + i;
423         s += "|";
424
425         for (int j = 0; j < quadro.getMatriz()[i].length; j++) {
426           String v = " " + quadro.getMatriz()[i][j];
427
428           if (quadro.getMatriz()[i][j] == 0) {
429             v = "##";
430           } else if (quadro.getMatriz()[i][j] < 0) {
431             v = "--";
432           }
433
434           while (v.length() < 3) {
435             v = " " + v;
436           }
437
438           s += v;
439
440           if (j % TEMPOS_DIA == TEMPOS_DIA - 1) {
441             s += "|";
442           }
443         }
444
445         s += "\n";
446       }
447
448       s += "Avalicao:" + quadro.getValor() + "\n";
449       s += "Sobreposicao:" + quadro.getSobreposicao() + "\n";
450       s += "Excesso Aulas:" + quadro.getExcessoaulas() + "\n";
451       s += "Gaps:" + quadro.getGaps() + "\n";
452
453       return s;
454     }
455
456     private Quadro interturmas(Quadro q) {
457       return null;  //To change body of created methods use File | Settings | File Templates.
458     }
459
460
461     private Quadro intraturmas(Quadro q) {
462       return null;
463     }
464
465     private Quadro intraturmasInterturmas(Quadro q) {
466       if (q.getExcessoaulas() != 0) {
467         q = intraturmas(q);
468         if (q.getExcessoaulas() == 0) q = interturmas(q);
469       }
470       if (q.getExcessoaulas() == 0) {
471         q = intraturmas(q);
472         q = interturmas(q);
473       }
474       return q;
475     }
476
477     public Quadro BT_Ilm(Quadro q) {
478       //melhor solucao obtida ate entao
479       Quadro melhor = q.cloneSolucao();
```

File - C:\apps\horario\src\main\java\at\horario\service\AlgoritmosManager2.java

```
480     //contador de iteracoes
481     int iter = 0;
482     //iteracao mais recente que forneceu q*
483     int melhorIter = 0;
484     //Lista Tabu
485     LinkedList<Movimento> listaTabu = new LinkedList<Movimento>();
486
487     int aspiracao = melhor.getValor();
488
489     while (iter - melhorIter <= buscaTabuMax) {
490       //System.out.println("Iteracao" + iter);
491       iter++;
492       //se ja m o melhor elemento de N(0) tal que o movimento m não seja tabu ou 0' atenda a conti□□o de aspiracão
493       Movimento m = melhorVizinho(q, listaTabu, aspiracao);
494       //System.out.println("Movimento" + m);
495       listaTabu.addLast(m);
496       if (listaTabu.size() > getTamanhoListaTabu())
497         listaTabu.removeFirst();
498
499       q.actualizar(m); //0<-0'
500 //      if (q.getSobreposicao() == 0) {//se encontrar um movimento sem sobreposicão faz
501 //        q = intraturmasInterturmas(q);
502 //      }
503
504       //System.out.println("Movimento"+m);
505
506       if (q.getValor() < melhor.getValor()) {
507         System.out.println("\t" + iter + "\t" + q.getValor() + "----------" + melhor.getValor());
508         System.out.println("melhorIter" + melhorIter + "|" + iter);
509         melhor = q.cloneSolucao();
510         aspiracao = melhor.getValor(); //actualizar a funcão de aspiracão
511         melhorIter = iter;
512       }
513     }
514     System.out.println("Iterator:" + iter);
515     return melhor;
516   }
517
518   public Quadro graspBuscaTabu() {
519     Quadro melhor = null;
520     int melhorValor = Integer.MAX_VALUE;
521     Random oRandom = new Random();
522
523     System.out.println("i\t alpha \tmelhorValor\t Valor actual");
524
525     for (int i = 0; i < graspMAX; i++) {
526       Quadro s = construaSolucaoInicial();
527       avaliar(s);
528       System.out.println("Descendo....\n");
529       s = BT_IIm(s);
530       System.out.println("...." + s.getValor());
531       if (melhorValor > s.getValor()) {
532         System.out.println("Iteracão i=\t" + i + "\t melhor =" + melhorValor + "\t actual=" + s.getValor());
533         melhor = s;
534         melhorValor = s.getValor();
535       }
536     }
537     return melhor;
538   }
539
```

File - C:\apps\horario\src\main\java\at\horario\service\AlgoritmosManager2.java

```
540    public Movimento melhorVizinho(Quadro q, LinkedList<Movimento> movimentosTabu, int aspiracao) {
541      Movimento melhor = new Movimento();
542      melhor.valor = Integer.MAX_VALUE;
543      Movimento vizinho = new Movimento();
544      for (int p = 0; p < q.matriz.length; p++) {//todos os professores. linhas = professores
545        vizinho.professor = p;
546        for (int i = 0; i < q.matriz[0].length; i++) {//colunas horarios
547          if (q.matriz[p][i] < 0) continue;//se for um lugar ocupado
548          vizinho.horario1 = i;
549          for (int j = i + 1; j < q.matriz[0].length; j++) {//colunas horarios
550            if (q.matriz[p][j] < 0) continue;//se for um lugar ocupado
551            vizinho.horario2 = j;
552
553            vizinho.sobreposicao = q.sobreposicao;
554            vizinho.gaps = q.gaps;
555            vizinho.excessoaulas = q.excessoaulas;
556
557            //calcuando as penalidades das colunas e linha mexidas
558            int sobreposicao1 = calculaSobreposicao(q, i);
559            int sobreposicaoJ = calculaSobreposicao(q, j);
560            calculaGapsAndExcessoaulas(q, p);
561            int excessoaulasProfessor = q.tempExcessoaulas;
562            int gapsProfessor = q.tempGaps;
563
564            //trocando
565            q.troca(p, i, j);
566
567            //actalizando as penalidades
568
569            calculaGapsAndExcessoaulas(q, p);
570
571
572            vizinho.sobreposicao = vizinho.sobreposicao - sobreposicao1 - sobreposicaoJ + calculaSobreposicao(q, i) + calculaSobr
    eposicao(q, j);
573            vizinho.gaps = vizinho.gaps - gapsProfessor + q.tempGaps;
574            vizinho.excessoaulas = vizinho.excessoaulas - excessoaulasProfessor + q.tempExcessoaulas;
575
576            vizinho.valor = getPesoGap() * vizinho.gaps + getPesoExcessoAula() * vizinho.excessoaulas + getPesoSobreposicao() * v
    izinho.sobreposicao;
577
578            q.troca(p, i, j);//pondo no estado inicial
579
580
581            if (melhor.valor > vizinho.valor) {
582              if (vizinho.valor < aspiracao || !movimentosTabu.contains(vizinho)) {
583                //se nao estiver na tabu ou nao for a nossa aspiracao
584                melhor.valor = vizinho.valor;
585                melhor.sobreposicao = vizinho.sobreposicao;
586                melhor.gaps = vizinho.gaps;
587                melhor.excessoaulas = vizinho.excessoaulas;
588
589                melhor.professor = p;
590                melhor.horario1 = i;
591                melhor.horario2 = j;
592              }
593            }
594          }
595        }
596      }
597
```

File - C:\apps\horario\src\main\java\at\horario\service\AlgoritmosManager2.java

```
598       return melhor;
599
600
601     }
602
603     public void calculaGapsAndExcessoaulas(Quadro q, int professor) {
604       //Systemelhor.out.println("Professor"+i);
605       q.tempGaps = 0;
606       q.tempExcessoaulas = 0;
607       int[] turmas = new int[horarioAlgo.getNumTurmas() + 1];
608       for (int primeirodia = 0; primeirodia < q.matriz[0].length; primeirodia += DIAS_SEMANA) {
609         int j;
610         int dia = primeirodia;
611         for (j = 0, dia = primeirodia; j < DIAS_SEMANA; j++, dia++) {
612           if (q.matriz[professor][dia] > 0)
613             turmas[q.matriz[professor][dia]] = 1;
614           //q.oHashSet.add(q.matriz[professor][dia]);
615         }
616
617         //for (Integer t : q.oHashSet) {
618         for (int i = 0; i < turmas.length; i++) {
619           if (turmas[i] == 0) continue;
620           //System.out.println(""+t);
621           int _quantos = -2;
622           int _gaps = -1;
623           for (j = 0, dia = primeirodia; j < DIAS_SEMANA; j++, dia++) {
624             if (q.matriz[professor][dia] == i) {
625               ++_quantos;
626               if (dia == primeirodia)
627                 ++_gaps;
628               else if (q.matriz[professor][dia - 1] != i)
629                 ++_gaps;
630             }
631           }
632           if (_quantos > 0)
633             q.tempExcessoaulas += _quantos; //numero de horas a +
634           if (_gaps > 0)
635             q.tempGaps += _gaps;
636
637           //this.setExcessoaulas(this.getExcessoaulas() + quantos);
638           //this.setGaps(this.getGaps() + gaps);
639
640         }
641       }
642
643     }
644
645 //   public static void main(String[] args) {
646 //
647 //     HorarioSettings horarioSettings = new HorarioSettings();
648 //     horarioSettings.setDIAS_SEMANA(5);
649 //     horarioSettings.setTemposDia(6);
650 //     Quadro s = new Quadro(17, horarioSettings.getHorariosSemana(), horarioSettings.getTemposDia());
651 //     AlgoritmosManager2 algoritmosManager2 = new AlgoritmosManager2(horarioSettings);
652 //     //algoritmosManager2.setHorarioSettings(horarioSettings);
653 //
654 //     for (int i = 0; i < s.getMatriz().length; i++) {
655 //       for (int j = 0; j < s.getMatriz()[0].length; j++) {
656 //         s.getMatriz()[i][j] = 15;
657 //       }
```

File - C:\apps\horario\src\main\java\at\horario\service\AlgoritmosManager2.java

```
658 //
659 //     }
660 //     System.out.println(algoritmosManager2.toString(s));
661
662     // }
663
664     public int getTamanhoListaTabu() {
665         return tamanhoListaTabu;
666     }
667
668
669 }
670
671
```

File - C:\apps\horario\src\main\java\at\horario\dao\GenericDao.java

```
 1  package at.horario.dao;
 2
 3  import org.hibernate.criterion.DetachedCriteria;
 4
 5  import java.io.Serializable;
 6  import java.util.Collection;
 7  import java.util.List;
 8
 9  import at.horario.model.HorarioSchedule;
10
11
12  /**
13   * Generic DAO (Data Access Object) with common methods to CRUD POJOs.
14   * <p/>
15   * <p>Extend this interface if you want typesafe (no casting necessary) DAO's for your
16   * domain objects.
17   *
18   * @author <a href="mailto:bwnoll@gmail.com">Bryan Noll</a>
19   * @param <T> a type variable
20   * @param <PK> the primary key for that type
21   */
22  public interface GenericDao<T, PK extends Serializable> {
23
24      /**
25       * Generic method used to get all objects of a particular type. This
26       * is the same as lookup up all rows in a table.
27       *
28       * @return List of populated objects
29       */
30      List<T> getAll();
31
32      /**
33       * Generic method to get an object based on class and identifier. An
34       * ObjectRetrievalFailureException Runtime Exception is thrown if
35       * nothing is found.
36       *
37       * @param id the identifier (primary key) of the object to get
38       * @return a populated object
39       * @see org.springframework.orm.ObjectRetrievalFailureException
40       */
41      T get(PK id);
42
43      /**
44       * Checks for existence of an object of type T using the id arg.
45       *
46       * @param id the id of the entity
47       * @return - true if it exists, false if it doesn't
48       */
49      boolean exists(PK id);
50
51      /**
52       * Generic method to save an object - handles both update and insert.
53       *
54       * @param object the object to save
55       * @return the persisted object
56       */
57      T save(T object);
58
59      /**
60       * Generic method to merge an object
```

File - C:\apps\horario\src\main\java\at\horario\dao\GenericDao.java

```
61      *
62      * @param object the object to merge
63      * @return the persisted object
64      */
65     T merge(T object);
66
67     /**
68      * Generic method to delete an object based on class and id
69      *
70      * @param id the identifier (primary key) of the object to remove
71      */
72     void remove(PK id);
73
74     void saveOrUpdateAll(Collection<T> objects);
75
76     List<T> getAll(DetachedCriteria criteria);
77
78     long count();
79
80     void update(T object);
81
82     void saveOrUpdate(T object);
83
84     DetachedCriteria criteria();
85
86     List<T> getAll(DetachedCriteria criteria, int firstResult, int maxResults);
87
88     long count(DetachedCriteria criteria);
89
90     DetachedCriteria buildCriteria(T object);
91
92     T refresh(T hs);
93 }
94
```

File - C:\apps\horario\src\main\java\at\horario\web\pages\PageTemplate.java

```
 1 package at.horario.web.pages;
 2
 3 import at.horario.model.User;
 4 import at.horario.web.HorarioSession;
 5 import at.horario.web.component.FeedbackViewPanel;
 6 import at.horario.web.component.MenuPanel;
 7 import static at.horario.web.component.MenuPanel.LINK_ID;
 8 import org.apache.commons.logging.Log;
 9 import org.apache.commons.logging.LogFactory;
10 import org.apache.wicket.IPageMap;
11 import org.apache.wicket.PageParameters;
12 import org.apache.wicket.ResourceReference;
13 import org.apache.wicket.markup.html.WebPage;
14 import org.apache.wicket.markup.html.basic.Label;
15 import org.apache.wicket.markup.html.link.Link;
16 import org.apache.wicket.markup.html.resources.StyleSheetReference;
17 import org.apache.wicket.model.AbstractReadOnlyModel;
18 import org.apache.wicket.model.IModel;
19 import org.apache.wicket.model.Model;
20 import org.apache.wicket.model.ResourceModel;
21
22 import java.util.Random;
23
24 public abstract class PageTemplate extends WebPage {
25    String[] temas = {"andreas01", "andreasat", "andreasat2", "deliciouslyblue", "deliciouslygreen"};
26    transient protected Log log = LogFactory.getLog(getClass());
27    protected FeedbackViewPanel feedback;
28    protected StyleSheetReference themeCss;
29    protected StyleSheetReference printCss;
30
31    private void init() {
32       final String className = getClass().getSimpleName().toLowerCase();
33       add(new Label("pageTitle", getLocalizer().getString(className + ".title", this) + " | " + getLocalizer().getString("webapp.
   name", this)));
34       add(new Label("heading", new ResourceModel(className + ".heading")));
35       if (getSession().getTema() == null) {
36          getSession().setTema("andreas01");
37       }
38       //final ResourceReference theme = new ResourceReference(PageTemplate.class, "styles/"+getSession().getTema()+"/theme.
   css");
39       themeCss = new StyleSheetReference("themeCss", new AbstractReadOnlyModel() {
40
41          public Object getObject() {
42             return new ResourceReference(PageTemplate.class, "styles/" + getSession().getTema() + "/theme.css");
43          }
44       });
45       printCss = new StyleSheetReference("printCss", new AbstractReadOnlyModel() {
46
47          public Object getObject() {
48             return new ResourceReference(PageTemplate.class, "styles/" + getSession().getTema() + "/print.css");
49          }
50       });
51       add(themeCss);
52       add(printCss);
53       themeCss.setOutputMarkupId(true);
54       printCss.setOutputMarkupId(true);
55
56
57       MenuPanel menuPanel = new MenuPanel("menuBar");
58
```

File - C:\apps\horario\src\main\java\at\horario\web\pages\PageTemplate.java

```
59        menuPanel.addMenu(Index.class);
60        if (!getSession().isSignedIn())
61           menuPanel.addMenu(Login.class);
62        MenuPanel.MenuItem users = menuPanel.addMenu(UserList.class, "users");
63        if (users != null) {
64           users.addMenu(UserList.class);
65           users.addMenu(UserForm.class);
66        }
67        MenuPanel.MenuItem horarios = menuPanel.addMenu(HorarioList.class, "horarios");
68        if (horarios != null) {
69           horarios.addMenu(HorarioList.class);
70           horarios.addMenu(HorarioForm.class);
71           horarios.addMenu(HorarioScheduleList.class);
72        }
73        if (getSession().isSignedIn()) {
74           MenuPanel.MenuItem temasMenu = menuPanel.addMenu(new TemaLink(LINK_ID), "Temas");
75           for (int i = 0; i < temas.length; i++) {
76              String tema = temas[i];
77              if (!tema.equals(getSession().getTema()))
78                 temasMenu.addMenu(new TemaLink(LINK_ID, tema), tema);
79
80           }
81        }
82        if (getSession().isSignedIn())
83           menuPanel.addMenu(Logout.class);
84
85        add(menuPanel);
86
87
88        feedback = new FeedbackViewPanel("feedback");
89        feedback.setOutputMarkupPlaceholderTag(true);
90        add(feedback);
91  //       IndicatingAjaxLink indicatingAjaxLink = new IndicatingAjaxLink("indicatingAjaxLink", new Model("link")) {
92  //          public void onClick(AjaxRequestTarget target) {
93  //             System.out.println(target);
94  //          }
95  //       };
96  //       add(indicatingAjaxLink);
97
98        Label labelEntrou;
99        User user = getSession().getUser();
100       if (user != null) {
101          String entrou = "Entrou como " + user.getUsername();
102          labelEntrou = new Label("entrou", new Model(entrou));
103          labelEntrou.setVisible(true);
104       } else {
105          String entrou = "";
106          labelEntrou = new Label("entrou", new Model(entrou));
107          labelEntrou.setVisible(false);
108       }
109       add(labelEntrou);
110    }
111
112
113    public PageTemplate() {
114       init();
115    }
116
117    public PageTemplate(PageParameters parameters) {
118       super(parameters);
```

File - C:\apps\horario\src\main\java\at\horario\web\pages\PageTemplate.java

```
119         init();
120     }
121
122     public PageTemplate(IModel model) {
123         super(model);
124         init();
125     }
126
127     public PageTemplate(IPageMap pageMap) {
128         super(pageMap);
129         init();
130     }
131
132     public PageTemplate(IPageMap pageMap, IModel model) {
133         super(pageMap, model);
134         init();
135     }
136
137
138     public HorarioSession getSession() {
139         return (HorarioSession) super.getSession();
140     }
141
142
143     public class TemaLink extends Link {
144         transient protected Log log = LogFactory.getLog(getClass());
145         private static final long serialVersionUID = 1L;
146         private String tema;
147
148         public TemaLink(String id) {
149             super(id);
150         }
151
152         public TemaLink(String id, String s) {
153             super(id);
154             tema = s;
155         }
156
157         public void onClick() {
158             if (tema == null) {
159                 Random random = new Random();
160                 tema = temas[Math.abs(random.nextInt()) % temas.length];
161             }
162             log.info("Tema Escolhido:" + tema);
163             ((HorarioSession) getSession()).setTema(tema);
164         }
165
166
167     }
168 }
169
```

File - C:\apps\horario\src\main\java\at\horario\web\HorarioApplication.java

```java
package at.horario.web;

import at.horario.service.LoadDataManager;
import at.horario.web.pages.*;
import org.apache.wicket.authentication.AuthenticatedWebApplication;
import org.apache.wicket.authentication.AuthenticatedWebSession;
import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.spring.injection.annot.SpringComponentInjector;
import static org.springframework.web.context.support.WebApplicationContextUtils.getRequiredWebApplicationContext;


public class HorarioApplication extends AuthenticatedWebApplication {

    public void init() {

        super.init();
        getDebugSettings().setAjaxDebugModeEnabled(true);
        getMarkupSettings().setStripWicketTags(true);
        getResourceSettings().setThrowExceptionOnMissingResource(true);


        springInjection();
        // make bookmarkable pages for easy linking from Menu/SiteMesh
        mountBookmarkablePage("/login.html", Login.class);
        mountBookmarkablePage("/users.html", UserList.class);
        mountBookmarkablePage("/userform.html", UserForm.class);
        mountBookmarkablePage("/horarioform.html", HorarioForm.class);
        mountBookmarkablePage("/horarios.html", HorarioList.class);
        mountBookmarkablePage("/horarioschedules.html", HorarioScheduleList.class);
        mountBookmarkablePage("/logout.html", Logout.class);


        LoadDataManager loadDataManager = (LoadDataManager) getRequiredWebApplicationContext(this.getServletContext()).getBean()
        loadDataManager.addAll();
    }

    protected void springInjection() {
        addComponentInstantiationListener(new SpringComponentInjector(this));
    }

    @Override
    protected Class<? extends AuthenticatedWebSession> getWebSessionClass() {
        return HorarioSession.class;
    }

    @Override
    protected Class<? extends WebPage> getSignInPageClass() {
        return Login.class;
    }


    @Override
    public Class getHomePage() {
        return Index.class;
    }

    //TODO: Mensagens

}
```

File - C:\apps\horario\src\main\java\at\horario\web\pages\PageTemplate.html

```
<!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" xml:lang="en" lang="en">
<head>
    <title wicket:id="pageTitle">[page title]</title>
    <meta http-equiv="Cache-Control" content="no-store"/>
    <meta http-equiv="Pragma" content="no-cache"/>
    <meta http-equiv="Expires" content="0"/>
    <link rel="shortcut icon" href="images/iconCalendar.gif" type="image/x-icon"/>
    <link wicket:id="themeCss" media="all" rel="Stylesheet" type="text/css"/>
    <link wicket:id="printCss" media="print" rel="Stylesheet" type="text/css"/>
    <script type="text/javascript" src="scripts/jquery-1.2.3.js"></script>
    <script type="text/javascript" src="scripts/jq-corner.js"></script>
    <script type="text/javascript" src="scripts/global.js"></script>
</head>
<body>
<div id="page">
  <div id="header" class="clearfix">
    <div id="branding">
      <h1><wicket:message key="webapp.name"/></h1>

      <p><wicket:message key="webapp.tagline"/></p>
    </div>
    <hr/>
  </div>

  <div id="content" class="clearfix">
    <div id="main">
      <h1 wicket:id="heading"/>

      <div id="fieldset">

        <div wicket:id="feedback">[[ feedback ]]</div>
        <!--<a href="#" wicket:id="indicatingAjaxLink"/> -->
        <wicket:child/>
      </div>
    </div>

    <div id="nav">
      <div class="wrapper">
        <h2 class="accessibility">Navigation</h2>

        <div wicket:id="menuBar"/>
      </div>
      <hr>
    </div>
  </div>

  <div id="footer" class="clearfix">
    <div id="divider">
      <div></div>
    </div>
    <span class="left">
      <span wicket:id="entrou"/>| &copy;
      <a href="#">
        <wicket:message key="company.name"/></a>
      <wicket:message key="copyright.year"/>
    </span>
```

File - C:\apps\horario\src\main\java\at\horario\web\pages\PageTemplate.html

```
61
62     </div>
63 </div>
64
65
66 </body>
67 </html>
68
```

File - C:\apps\horario\src\main\java\at\horario\web\pages\Login.java

```
 1  package at.horario.web.pages;
 2
 3  import org.apache.wicket.authentication.AuthenticatedWebSession;
 4  import org.apache.wicket.markup.html.form.CheckBox;
 5  import org.apache.wicket.markup.html.form.Form;
 6  import org.apache.wicket.markup.html.form.PasswordTextField;
 7  import org.apache.wicket.markup.html.form.TextField;
 8  import org.apache.wicket.model.PropertyModel;
 9
10
11  /**
12   * Login page constituents  are the same as Login.html except that ist is made up of equivalent Wicket components
13   */
14  public class Login extends PageTemplate {
15     private Form form;
16     private String username;
17     private String password;
18     private String rememberMe;
19     private static final String ADMIN = "admin";
20
21     public void setUsername(String username) {
22        this.username = username;
23     }
24
25     public void setPassword(String password) {
26        this.password = password;
27     }
28
29
30     public Login() {
31
32        TextField userIdField = new TextField("username", new PropertyModel(this, "username"));
33        PasswordTextField passwordField = new PasswordTextField("password", new PropertyModel(this, "password"));
34        CheckBox rememberMe = new CheckBox("rememberMe", new PropertyModel(this, "rememberMe"));
35
36
37        passwordField.setResetPassword(false);
38
39        form = new LoginForm("loginForm");
40        form.setMarkupId("loginForm");
41        form.add(userIdField);
42        form.add(passwordField);
43        form.add(rememberMe);
44
45        add(form);
46
47     }
48
49
50     public String getUsername() {
51        return username;
52     }
53
54     public String getPassword() {
55        return password;
56     }
57
58     public String getRememberMe() {
59        return rememberMe;
60     }
```

File - C:\apps\horario\src\main\java\at\horario\web\pages\Login.java

```
 61
 62    public void setRememberMe(String rememberMe) {
 63       this.rememberMe = rememberMe;
 64    }
 65
 66    public boolean signIn(String username, String password) {
 67       return AuthenticatedWebSession.get().signIn(username, password);
 68    }
 69
 70    protected void onSignInFailed() {
 71       // Try the component based localizer first. If not found try the
 72       // application localizer. Else use the default
 73       String errMsg = getLocalizer().getString("errors.password.mismatch", Login.this);
 74       log.error(errMsg);
 75       error(errMsg);
 76    }
 77
 78    protected void onSignInSucceeded() {
 79       // If login has been called because the user was not yet
 80       // logged in, than continue to the original destination.
 81       // otherwise to the Home page
 82       if (!continueToOriginalDestination()) {
 83          setResponsePage(getApplication().getSessionSettings().getPageFactory().newPage(
 84                getApplication().getHomePage(), null));
 85       }
 86    }
 87
 88
 89    class LoginForm extends Form {
 90       public LoginForm(String id) {
 91          super(id);
 92       }
 93
 94       @Override
 95       protected void onSubmit() {
 96          String userId = Login.this.getUsername();
 97          String password = Login.this.getPassword();
 98
 99          if (signIn(userId, password)) onSignInSucceeded();
100          else onSignInFailed();
101
102       }
103    }
104
105
106 }
107
```

File - C:\apps\horario\src\main\java\at\horario\web\pages\Login.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd" >
<html>
<wicket:extend>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <link rel="stylesheet" type="text/css" media="all" href="styles/deliciouslyblue/layout-1col.css"/>
  </head>
  <body id="login">

  <form wicket:id="loginForm">
    <fieldset style="padding-bottom: 0">
      <ul>
        <li>
          <label for="username" class="required desc">
            <wicket:message key="label.username"/> <span class="req">*</span>
          </label>
          <input type="text" class="text medium" name="username" wicket:id="username" tabindex="1"/>
        </li>

        <li>
          <label for="password" class="required desc">
            <wicket:message key="label.password"/> <span class="req">*</span>
          </label>
          <input type="password" class="text medium" name="password" wicket:id="password" tabindex="2"/>
        </li>

        <li>
          <input type="checkbox" class="checkbox" name="rememberMe" wicket:id="rememberMe" tabindex="3"/>
          <label for="rememberMe" class="choice"><wicket:message key="login.rememberMe"/></label>
          <hr/>
        </li>
        <li>
          <input type="submit" class="button" name="login" value="Login" tabindex="4"/>
        </li>
      </ul>
      <!--<li>-->
      <!--<label for="username" class="required desc">-->
      <!--<wicket:message key="label.username">Username</wicket:message>-->
      <!--<span class="req">*</span>-->
      <!--</label>-->
      <!--<input type="text" class="text medium" wicket:id="username" tabindex="1"/>-->
      <!--</li>-->

      <!--<li>-->
      <!--<label for="password" class="required desc">-->
      <!--<wicket:message key="label.password">Password</wicket:message>-->
      <!--<span class="req">*</span>-->
      <!--</label>-->
      <!--<input type="password" class="text medium" wicket:id="password" tabindex="2"/>-->
      <!--</li>-->
    </fieldset>
  </form>
  </body>
</wicket:extend>
</html>
```