EasyTave

Rafael Campos Cruz

Trabalho de Conclusão de Curso
Instituto de Matemática e Estatística
Universidade de São Paulo
para
Graduação em Ciência da Computação

Orientador: Prof. Dr. Marco Dimas Gubitoso

São Paulo, Janeiro de 2016

Agradecimentos

Dedico esta monografia à minha família pelo constante apoio e à minha namorada por estar ao meu lado ao longo desta grande jornada que foi minha graduação! Também dedico aos meus amigos de São Carlos, sem os quais não conseguiria concluir esta monografia.

Resumo

Nesta monografia trataremos do processo de desenvolvimento do easyTave: uma interface entre Java e Octave, permitindo processar modelos matemáticos e arquivos em código Matlab através de uma interface Java.

A idealização desta aplicação ocorreu ao longo do estágio do aluno Rafael Campos Cruz na empresa "Tendências - Consultoria Integrada", em que foi observado o seguinte evento: para um mesmo modelo matemático haviam pequenas variações entre diversas execuções, variações estas que só podiam vir das diferentes versões/ambientes de execução dos mesmos.

Frente a necessidade de se ter um ambiente padronizado de execução, vislumbrou-se também poder agilizar o processo de execução e extração de resultados. Processo este que, anterior a implantação da ferramenta, era usualmente feito por uma única pessoa e o tempo requisição-resposta levava em geral horas.

Não existia, até o momento, uma forma de integrar ambientes Java e Octave através do Shell do SO, sendo este o principal objetivo do pacote criado.

Constará aqui também um exemplo de aplicação do pacote: uma aplicação web que, utilizando o pacote easyTave, permite que sejam rodados diversos modelos matemáticos de forma concorrente em uma plataforma de servidor.

Explicaremos também, em detalhe, o funcionamento geral da aplicação: a forma como os recursos de servidor são compartilhados, como os usuários tem acesso aos recursos críticos e como foi feita a integração Java-Octave.

Palavras-chave: JSF, Octave, interface.

Abstract

In this monograph we will deal with the proccess of development of easyTave: an interface between Java and Octave, allowing program to process mathematical models and files in the Matlab language through Java.

The idealizations of this application came to be during the intership of the student Rafael Campos Cruz in the company "Tendências - Consultoria Integrada", there was observed the following event: for a single economical model there were small deviations between simulations in different machines, these variations could only come from difference in versions or in the enviroment of execution.

Faced with this necessity of having a standardized enviroment, there was also the idead to streamline the execution and result extraction process. This proccess, before the implantion of the tool, was usually made by a single person and generally took hours to be done.

There wasn't, to this moment, a way to integrate enviroments Java and Octave through the SO's shell, thus the main objective of the package is allowing that.

Also, we will explain, in detail, the general operation of the application: how it handles shared resources, how its users have access to critical resources and how the Java-Octave integration was done.

Keywords: JST, Octave, interface.

Sumário

Capítulo 1

Introdução

EasyTave é um pacote Java que permite integrar o interpretador de código MATLAB Octave com aplicações Java, possibilitando assim que códigos M e MOD sejam lidos e executados dentro do Octave e retornados às requisições principais do usuário.

Esta interface Java-Octave até o momento não existia e ela se torna interessante de ser criada devido ao altíssimo desempenho de execução de métodos matemáticos com relação à outras linguagens de programação e, também, por ter uma sintaxe extremamente simples e intuitiva, permitindo que sejam criados modelos matemáticos extremamente complexos, sem que o usuário precise ter grandes conhecimentos de computação para atingir esta eficiência.

A fim de possibiltiar esta integração é feita a gerência das streams de execução do shell do ambiente de execução: através da API Runtime do Java podemos ter acesso ao shell e ao Octave, podendo assim manipular suas entradas e saídas, criando assim a interface entre os dois.

É interessante fazer a ressalva que o pacote faz uso de aplicações nativas da máquina em que ela está rodando, seja num modelo Cliente-Servidor ou seja em execuções individuais por desenvolvedores, sendo assim necessário configurar o ambiente de execução para, corretamente, se adequar aos requisitos do easyTave. Necessidades como esta serão detalhadas ao longo da monografia.

Também detalharemos uma aplicação de exemplo homônica ao pacote que, através de uma interface web, possibilita que usuários forneçam códigos à aplicação e as rodem assíncronamente, competindo por recursos do servidor e obtendo de forma paralela seus resultados.

Capítulo 2

Octave

2.1 O que é o Octave?

O programa GNU Octave (E$^+$) é um simulador matemático que trabalha com uma linguagem altamente compatível com MATLAB, ou seja, na maior parte das instâncias ele é capaz de compilar e executar o mesmo código que é executado no, também simulador matemático, MATLAB.

No entanto há aí uma grande diferença: enquanto o Octave é Open Source e publicado sob a licença GPL, o MATLAB é um software proprietário que tanto não permite modificação pelos usuários quanto possui um altíssimo custo de licença (na data desta publicação está avaliada em 2,650USD para a licença individual completa, (MAT)), o que torna um tanto quanto proibitivo para a maioria das pessoas.

Outra diferença importante a ser apontada entre as duas aplicações é o fato de que há sim pequenas diferenças na linguagem processada por MATLAB e Octave: a linguagem processada por Octave permite algumas comodidades ao programador que o MATLAB não permite, como permitir o caractere "#"para iniciar comentários e a função fprintf ser usada tanto para imprimir no STDOUT quanto em um arquivo.

Vale também apontar que o MATLAB, apesar do custo, possui algumas vantagens em relação ao Octave como: a disponibilidade de algumas funções de GUI exclusivas ao mesmo (o que poderia gerar conflito entre a migração de alguns códigos entre MATLAB e Octave) e também uma série de "caixas de ferramenta"especializadas para lidar com aplicações dos mais diversos tipos (como mercado financeiro, computações físicas, etc): o fato é que muitas dessas caixas de ferramenta não estão presentes dentro do Octave, pois ainda não foram desenvolvidas pela comunidade ((FAQ)).

Contudo, para a maior parte das necessidades, o Octave é mantido e garantido pela comunidade como um compilador e interpretador de MATLAB equivalente e suficiente, sanando a maior parte das necessidades de simulação matemática dos usuários.

2.2 Porque a escolha do Octave?

Ao longo do estágio do aluno Rafael Campos Cruz foi vista a necessidade de centralizar a execução de um modelo econométrico da empresa Tendências Consultoria Integrada. Este modelo de altíssima complexidade simularia as flutuações de dados econômicos da economia brasileira de acordo com decisões do Banco Central, dados estes como interferência no câmbio de dólar, divulgação de dados de consumo da população e etc.

No entanto, havia uma característica muito específica deste modelo: a intenção era que ele fosse uma ferramenta para todos os analistas da empresa testarem hipóteses de impactos na economia, fazendo assim necessário que fossem feitas diversas alterações no modelo econométrico de forma rápída e sem grandes dificuldades ao usuário.

Não só isso: o modelo deveria ser atualizado periodicamente para se adequar a novas realidades da economia, sendo assim ideal que todos os usuários referenciassem um único código fonte.

Foi então envisionado que este procedimento poderia ser feito através de uma interface web que automatizaria todas as alterações realizadas pelo usuário no código fonte, sendo assim necessário que um servidor da empresa pudesse rodar um interpretador de códigos MATLAB (linguagem do modelo) e que houvesse alguma interface entre a linguagem da interface web e o interpretador.

Havia assim a necessidade de um ambiente dedicado para o processamento destes modelos, no entanto, devido ao uso de diversas pessoas da mesma licença de forma remota, se esbarraria numa área cinza da legalidade sobre poder ou não usar o MATLAB para uma aplicação como esta. Foi assim escolhido o Octave para suportar a aplicação, dado que a licença GPL permitiria sem grandes problemas que tal uso fosse realizado.

Além da permissividade deste uso incomum da aplicação o Octave tornaria desnecessária a aquisição de novas licenças do MATLAB para realizar simulações futuras, tornando-se assim ainda mais atrativo como opção de adoção dentro da empresa. Escolhida a ferramenta, focou-se então em desenvolver a estrutura necessária para o funcionamento dela.

Figura 2.1: *Landing page.*

A linguagem escolhida para o desenvolvimento foi Java, devido a familiaridade do desenvolvedor e a simplicidade de se criar uma interface XHTML compatível com um ambiente back-end que realizaria todo o "trabalho pesado"para lidar com manipulação de dados e segurança de informação, contudo esbarrou-se no problema: como integrar as duas aplicações completamente diferentes?

Criada a interface, nasceu a aplicação SET - Simulador Econométrico Tendências - que, vem realizando diversas simulações com sucesso desde então e tendo seu escopo de atuação expandido de uso interno da empresa para clientes externos como bancos e fundos de investimento.

2.3 Vantagens do Octave

Dado que a linguagem MATLAB é só mais uma linguagem de programação, qual a vantagem dos usuários utilizarem esta linguagem em detrimento de linguagens mais estabelecidas como C e Java? Estabeleçamos aqui então um teste muito simples para avaliarmos a eficiência das linguagens para experimentos matemáticos: multiplicação de matrizes quadradas.

A multiplicação de matrizes é um problema computacional muito comum, pois se implementado de forma intuitiva possui complexidade $\mathcal{O}(n^3)$ e, esta forma, é a mais natural para qualquer pessoa expressar a multiplicação.

Façamos então o seguinte experimento: faremos multiplicações de matrizes quadradas de ordem n onde $n \in [1; 20]$; cada multiplicação será repetida 100 vezes para cada ordem de n e, cada multiplicação, sendo feita com matrizes quadradas de campos de tipo float aleatoriamente gerados no intervalo de 0 a 1. Os dados de tempo coletados tem sensibilidade máxima da ordem de $1E-7$, sendo valores inveriores a este considerados 0.

As multiplicações serão realizadas da forma mais simples e intuitiva o possível, ou seja, a forma que uma pessoa sem conhecimento especializado expressaria uma matriz e realizaria a multiplicação das mesmas. As linguagens de análise serão MATLAB, C e Java e o ambiente de simulação é o que segue:

SO Windows 10 64 bits
Processador Intel Core i7-2670QM
8GB RAM
Octave 4.0.0
Java 1.8.0_66
GCC 4.7.3

Primeiro simulamos a operação na linguagem C, que é didaticamente uma linguagem muito aceita, obtemos os seguintes dados:

Tamanho	Tempo médio (s)	Tempo médio (ms)	Tempo/simulação (s)	Tempo/simulação (ms)	Tempo acumulado (s)	Tempo acumulado (ms)
1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
2	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
3	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
4	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
5	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
6	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
7	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
8	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
9	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
10	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
11	1.50E-04	1.50E-01	1.50E-02	1.50E+01	1.50E-02	1.50E+01
12	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.50E-02	1.50E+01
13	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.50E-02	1.50E+01
14	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.50E-02	1.50E+01
15	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.50E-02	1.50E+01
16	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.50E-02	1.50E+01
17	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.50E-02	1.50E+01
18	1.60E-04	1.60E-01	1.60E-02	1.60E+01	3.10E-02	3.10E+01
19	2.80E-04	2.80E-01	2.80E-02	2.80E+01	5.90E-02	5.90E+01
20	1.60E-04	1.60E-01	1.60E-02	1.60E+01	7.50E-02	7.50E+01

Figura 2.2: *Dados das simulações em C*

Em seguida simulamos as operações em ambiente Java, obtendo:

Tamanho	Tempo médio (s)	Tempo médio (ms)	Tempo/simulação (s)	Tempo/simulação (ms)	Tempo acumulado (s)	Tempo acumulado (ms)
1	2.33E-07	2.33E-04	2.33E-05	2.33E-02	2.33E-05	2.33E-02
2	7.13E-07	7.13E-04	7.13E-05	7.13E-02	9.46E-05	9.46E-02
3	1.57E-06	1.57E-03	1.57E-04	1.57E-01	2.52E-04	2.52E-01
4	3.53E-06	3.53E-03	3.53E-04	3.53E-01	6.04E-04	6.04E-01
5	6.31E-06	6.31E-03	6.31E-04	6.31E-01	1.24E-03	1.24E+00
6	1.04E-05	1.04E-02	1.04E-03	1.04E+00	2.27E-03	2.27E+00
7	1.00E-05	1.00E-02	1.00E-03	1.00E+00	3.28E-03	3.28E+00
8	3.50E-06	3.50E-03	3.50E-04	3.50E-01	3.63E-03	3.63E+00
9	4.95E-06	4.95E-03	4.95E-04	4.95E-01	4.12E-03	4.12E+00
10	6.80E-06	6.80E-03	6.80E-04	6.80E-01	4.80E-03	4.80E+00
11	8.84E-06	8.84E-03	8.84E-04	8.84E-01	5.69E-03	5.69E+00
12	1.11E-05	1.11E-02	1.11E-03	1.11E+00	6.80E-03	6.80E+00
13	1.38E-05	1.38E-02	1.38E-03	1.38E+00	8.18E-03	8.18E+00
14	1.78E-05	1.78E-02	1.78E-03	1.78E+00	9.95E-03	9.95E+00
15	2.12E-05	2.12E-02	2.12E-03	2.12E+00	1.21E-02	1.21E+01
16	2.47E-05	2.47E-02	2.47E-03	2.47E+00	1.45E-02	1.45E+01
17	2.71E-05	2.71E-02	2.71E-03	2.71E+00	1.73E-02	1.73E+01
18	1.00E-05	1.00E-02	1.00E-03	1.00E+00	1.83E-02	1.83E+01
19	1.13E-05	1.13E-02	1.13E-03	1.13E+00	1.94E-02	1.94E+01
20	1.24E-05	1.24E-02	1.24E-03	1.24E+00	2.06E-02	2.06E+01

Figura 2.3: *Dados das simulações em Java*

Finalmente, simulamos o mesmo experimento em MATLAB, chegando aos seguintes dados:

Tamanho	Tempo médio (s)	Tempo médio (ms)	Tempo/simulação (s)	Tempo/simulação (ms)	Tempo acumulado (s)	Tempo acumulado (ms)
1	2.00E-05	2.00E-02	2.00E-03	2.00E+00	2.00E-03	2.00E+00
2	2.00E-05	2.00E-02	2.00E-03	2.00E+00	4.00E-03	4.00E+00
3	0.00E+00	0.00E+00	0.00E+00	0.00E+00	4.00E-03	4.00E+00
4	0.00E+00	0.00E+00	0.00E+00	0.00E+00	4.00E-03	4.00E+00
5	1.00E-05	1.00E-02	1.00E-03	1.00E+00	5.00E-03	5.00E+00
6	1.00E-05	1.00E-02	1.00E-03	1.00E+00	6.00E-03	6.00E+00
7	0.00E+00	0.00E+00	0.00E+00	0.00E+00	6.00E-03	6.00E+00
8	3.00E-05	3.00E-02	3.00E-03	3.00E+00	9.01E-03	9.01E+00
9	5.00E-06	5.00E-03	5.00E-04	5.00E-01	9.51E-03	9.51E+00
10	1.00E-05	1.00E-02	1.00E-03	1.00E+00	1.05E-02	1.05E+01
11	1.00E-05	1.00E-02	1.00E-03	1.00E+00	1.15E-02	1.15E+01
12	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.15E-02	1.15E+01
13	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.15E-02	1.15E+01
14	1.00E-05	1.00E-02	1.00E-03	1.00E+00	1.25E-02	1.25E+01
15	3.00E-05	3.00E-02	3.00E-03	3.00E+00	1.55E-02	1.55E+01
16	2.00E-05	2.00E-02	2.00E-03	2.00E+00	1.75E-02	1.75E+01
17	1.00E-05	1.00E-02	1.00E-03	1.00E+00	1.85E-02	1.85E+01
18	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.85E-02	1.85E+01
19	1.00E-05	1.00E-02	1.00E-03	1.00E+00	1.95E-02	1.95E+01
20	2.50E-05	2.50E-02	2.50E-03	2.50E+00	2.20E-02	2.20E+01

Figura 2.4: *Dados das simulações em MATLAB*

Gráficamente podemos claramente ver o comportamento das amostras e como elas se relacionam. Observamos 3 métricas nos gráficos: tempo médio de execução, tempo total das execuções e tempo acumulado por todas as operações.

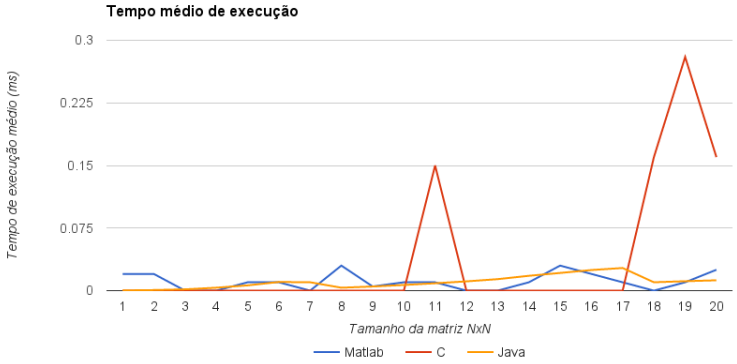

Figura 2.5: *Comparação do tempo médio das execuções entre as linguagens.*

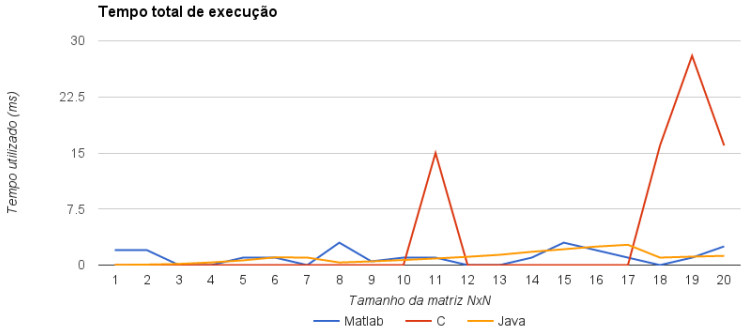

Figura 2.6: *Comparação do tempo total das execuções entre as linguagens.*

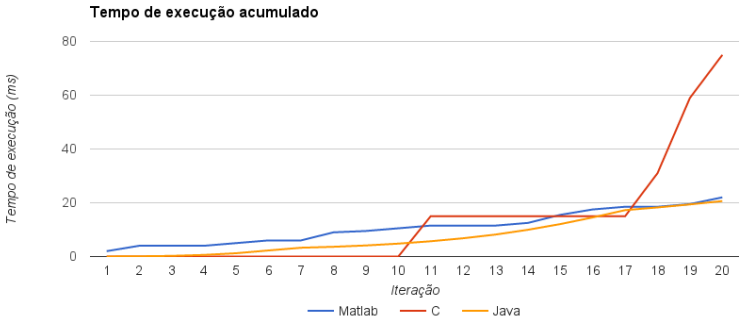

Figura 2.7: *Tempo de execução acumulado a cara iteração de N.*

Fica claro que apesar de para pequenos n o Octave se mostra ineficiente para com as outras linguagens, mas para $n's$ maiores ele mantém o custo de execução aproximadamente linear, enquanto que as outras linguagens tem um custo equivalente ou maior.

Dito isto, vale apontar outro ponto de vantagem do Octave como linguagem de execução de experimentos matemáticos, que é a simplicidade da sintaxe para expressar tais experimentos. Vamos analisar agora o código executado para estes experimentos.

O código do experimento em C é o que segue:

```
1 double m1[20][20];
2   double m2[20][20];
3   double m3[20][20];
4
5   int i, j, k, u, v;
6   clock_t start, stop;
7   double total;
8   double tempos[20];
9   srand(time(NULL));
10  total = 0;
11
12  for(i = 0; i < 20; i++)
13  {
14    tempos[i] = (double)0;
15
16    for(j = 0; j < 100; j++)
17    {
18      for(k = 0; k < i + 1; k++) for(u = 0; u < i + 1; u++)
19      {
20        m1[k][u] = rand()%1;
21        m2[k][u] = rand()%1;
22        m3[k][u] = 0;
23      }
24
25      start = clock();
26
27      for(k = 0; k < i + 1; k++)
28        for(u = 0; u < i + 1; u++)
29          for(v = 0; v < i + 1; v++) m3[k][u] += m1[k][v] * m2[v][u];
30
31      stop = clock();
32      tempos[i] += (stop - start) * 1.0;
33      total += (stop - start) * 1.0;
34    }
35
36    tempos[i] /= 100 * CLOCKS_PER_SEC;
37  }
```

Para o experimento em Java temos:

```
1 float[][] m1 = new float[20][20],
2            m2 = new float[20][20],
3            m3 = new float[20][20];
4     int i, j, k, u, v;
5     Random ran = new Random(System.currentTimeMillis());
6     long start, stop, total = 0;
7     long[] tempos = new long[20];
8
9     for(i = 0; i < 20; i++)
10    {
11      tempos[i] = 0;
12
13      for(j = 0; j < 100; j++)
14      {
15        for(k = 0; k < i + 1; k++) for(u = 0; u < i + 1; u++)
```

```
16          {
17            m1[k][u] = ran.nextFloat();
18            m2[k][u] = ran.nextFloat();
19            m3[k][u] = 0;
20          }
21
22          start = System.nanoTime();
23
24          for(k = 0; k < i + 1; k++)
25            for(u = 0; u < i + 1; u++)
26              for(v = 0; v < i + 1; v++) m3[k][u] += m1[k][v] * m2[v][u];
27
28          stop = System.nanoTime();
29
30          tempos[i] += stop - start;
31          total += stop - start;
32        }
33
34        tempos[i] /= 100.0;
35      }
```

E em MATLAB temos o seguinte código:

```
1 tempos = zeros(20, 1);
2 total = 0;
3
4 for i = 1:20
5
6   for j = 1:100
7     m1 = rand (i, i, "double");
8     m2 = rand (i, i, "double");
9
10    tic();
11
12    m1 = m1*m2;
13
14    atual = toc();
15    total = total + atual;
16    tempos(i) = tempos(i) + atual;
17  endfor
18
19  tempos(i) = tempos(i)/100
20 endfor
```

Dos três, claramente, o de MATLAB é mais expressivo e intuitivo para qualquer usuário, usando também da vantagem de ser já uma implementação ótimizada do nosso objetivo.

Vantagens como esta tornam a linguagem MATLAB e um interpretador/compilador sob a licença GPL como o Octave extremamente vantajosos para a comunidade acadêmica e profissional.

Capítulo 3

Interface

3.1 Motivação Inicial

Era desejado que fosse centralizada a execução de um modelo único de código MATLAB: este modelo deveria sofrer pequenas alterações em seu código e, através dela, simular vários cenários econômetricos diferentes para que analistas da empresa em que o aluno Rafael Campos Cruz realizava estágio.

Apesar de já existir uma interfave entre Java e Octave, o JavaOctave (JO2), este não possuia o escopo completo desejado: que era fazer uso da execução de um modelo matemático completo e não de pequenas funções da ferramenta.

Assim, viu-se necessário criar uma forma de comunicar a camada lógica de Java com o software Octave instalado no servidor que receberia a aplicação, a fim de sanar esta necessidade até então nunca atendida.

3.2 O JavaOctave

Como citado anteriormente, existe no momento a interface JavaOctave: o escopo dela se resume a, disponibilizando um pequeno parser de instruções simples de MATLAB, permitir o uso de algumas funções matemáticas dentro do código Java de forma extremamente explícita.

Esta ferramenta, extremamente robusta, não é capaz de tanto interpretar um modelo matemático inteiro (um arquivo de código .m) quanto lidar com variáveis internas próprias de um modelo.

Visto que, detalhes como este, tornariam a nossa necessidade atendível apenas através de se criar completamente do zero um código em Java, utilizando funções Octave para otimização, que reproduzisse o comportamento do modelo matemático, esta opção é completamente inviável do sentido prático: operações como manutenção do código da aplicação e atualização/substituição do modelo utilizado precisariam ser completamente hardcoded.

Assim, apesar de comunicar as duas aplicações, o escopo da interface atual é muito limitado e, por isso, precisamos criar uma nova ferramenta para atender a necessidade da excução íntegra de modelos matemáticos.

3.3 O easyTave

O easyTave é um pacote com uma estrutura extremamente simples: possui uma classe que vai lidar com a comunicação com o shell do SO que o recebe e uma classe que realiza o a captura das

Streams geradas pela aplicação.

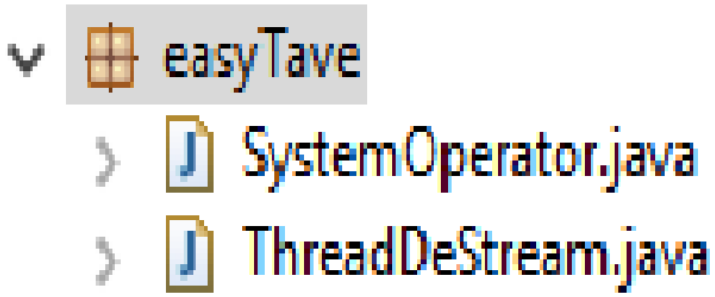

Figura 3.1: *Estrutura de Classes.*

O processo desempenhado pelas classes, a fim de realizar estas ações, merece detalhamento:

3.3.1 ThreadDeStream

Esta classe é uma sobrecarga da classe Thread, parte do conjunto básico de classes da linguagem. Nesta classe, no entanto, o método "run"irá consumir completamente uma Stream recebida, armazenando todo o seu conteúdo na variável info, privada à classe.

Esta classe só pode ser criada recebendo como parâmetro um Object do tipo Stream, o que da a característica principal de ela ser uma especialização de Thread.

3.3.2 SystemOperator

O SystemOperator realizará toda a interação com o shell do SO em que está sendo executado, fazendo uso da classe ThreadDeStream para capturar as duas Streams criadas por esta comunicação com o shell: uma ErrorStream e uma InputStream.

Através da captura destas streams temos acesso a todo log de erros do Octave: de erros de compilação a erros de execução [1], informações essências tanto para teste quanto para saber se algo no ambiente esta impedindo a execução do programa; temos também acesso ao log bem sucedido do programa que, neste caso, veríamos as alterações realizadas nas variáveis ou quaisquer mensagens que o usuário inseriu em seu código.

Como, no entanto, nossa intenção principal é capturar todas as saídas do Octave, SystemOperator também recebe a função de capturar um arquivo com o estado detalhado de todas as variáveis de sistema ao final da execução do programa e apagar quaisquer arquivos desnecessários ao que o usuário deseja recuperar: para alguns de código MATLAB o Octave passa primeiro por um interpretador que vai otimizar ainda mais o código fornecido, i.e. Dynare (DYN), e operações como estas vão gerar vários arquivos de saída além do que desejamos, estes arquivos são eliminados.

Esta classe, a fim de ser criada, deve receber como parâmetro um endereço de atuação [2], um código a ser executado e, na versão atual, se ele requerirá o uso do interpretador Dynare.

Através destes parâmetros, sua execução irá fornecer em suas variáveis o log de erros e a execução do Octave, bem como o relatório de valores finais das variáveis do programa.

Vale acrescentar também que SystemOperator é uma especialização de Runnable e, como tal, pretende-se ser usado como uma Thread independente da execução do programa principal.

[1] Subentenda-se erros obtidos devido a versionamento, incompatibilidades do ambiente, falta de memória ou interrupção.

[2] Um diretório no qual os arquivos de saída possam ser gerados com segurança e apagados.

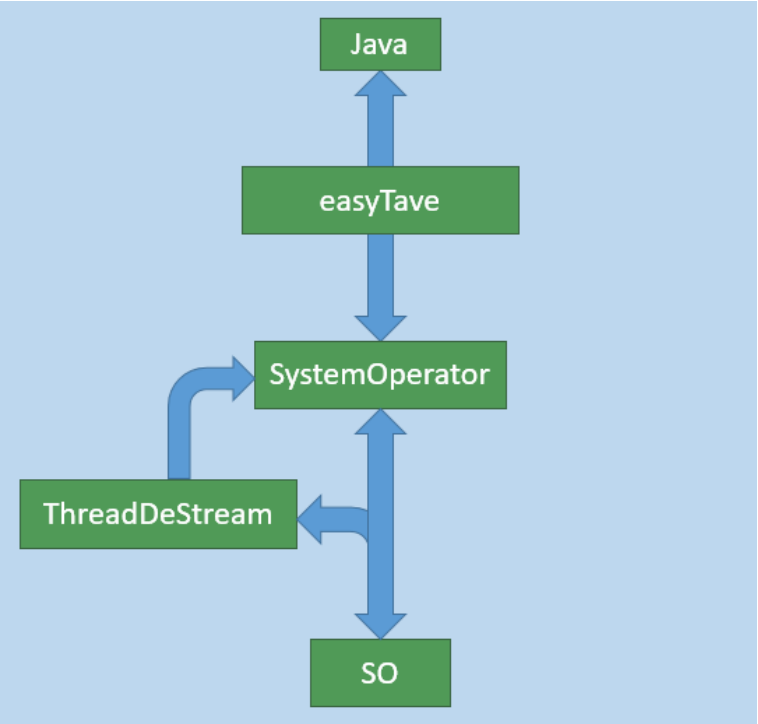

Figura 3.2: *Comunicação entre Java e SO através do easyTave.*

3.4 Requisitos do pacote

O easyTave, até o momento, é compatível com dois ambientes: Linux e Windows; havendo necessidades específicas à cada um deles para que seu uso seja possível.

No ambiente Linux, deve estar disponível no diretório /usr/bin/ o arquivo de nome "octave". Caso a instalação tenha sido realizada através do comando "apt-get", estará automaticamente disponível o arquivo na pasta. Caso sua aplicação não esteja rodando dentro do usuário "root", vale acrescentar a necessidade do acesso ao arquivo para o usuário.

No ambiente Windows você encontrará o instalador disponível na página do projeto Octave. Após a instalação você deve adicionar o diretório base do programa ao path do Windows, a fim de disponibilizar o acesso ao octave.exe .

Uma última recomendação é que seja usada versão 3.6.x ou posterior, pois consegue realizar própriamente a compatibilidade entre códigos compilados no software MATLAB em ambiente Octave.

3.5 Limitações atuais do pacote

O pacote, até o momento, contempla apenas compatibilidade com o interpretador Dynare, sendo que, caso necessite de chamada explícita para a execução de algum outro interpretador, pode ser incluída a instrução dentro do modelo ou então feita uma atualização ao pacote, que está publicado dentro do BitBucket como de livre acesso a colaboradores.

Ele também não permite que rotinas de geração de gráficos sejam executadas dentro do seu código e, caso o sejam, incorrerá em mau funcionamento do mesmo. Isto se deve pelo alto custo de executar e capturar as Streams de criação das imagens visuais pelo Octave, sendo assim esta funcionalidade foi intencionalmente excluída da aplicação.

No mais não é também recomendado o acesso simultâneo de duas Threads diferentes ao Oc-

tave, dado que isso pode gerar simulações incorretas pelo programa dado a alteração de variáveis compartilhadas pelas várias instâncias do programa sendo rodadas.

Capítulo 4

Aplicação exemplo

4.1 Ideia inicial

Para este exemplo de uso do easyTave decidimos criar uma aplicação que poderia ser usada por diversos estudantes ao mesmo tempo para rodar códigos de MATLAB.

A aplicação, implementanda em JSF, busca, através da gerência de recursos de servidor, permitir acesso ao Octave instalado no servidor usando o pacote de interface criado e gerir dinâmicamente os usuários que requisitam acesso à aplicação.

Uma série de decisões arbitrárias foram realizadas a fim de permitir a integridade da aplicação e o não sobrecarregando do servidor: códigos podem rodar por, no máximo, 10 minutos, por exemplo.

Detalharemos a seguir o trabalho realizado.

4.2 Conceitos aplicados

Existem 3 pilares fundamentais no desenvolvimento do exemplo para o easyTave: **restrição do uso de seção crítica** (a aplicação Octave no servidor e outros recursos), **multithreading** e **modelo cliente-servidor**.

4.2.1 Restrição do Uso de Seção Crítica

Como o Octave não foi feito para múltiplos acessos simultâneos e queremos estressar o mínimo o possível o servidor, temos que dividir entre os múltiplos usuários o espaço de armazenamento, execução de seus modelos matemáticos e o processamento de máquina.

Assim foram tomadas as seguintes precauções: são criadas apenas 8 Threads de execução ao mesmo tempo e apenas uma Thread é executada por vez, garantindo o acesso exclusivo ao Octave a cada execução.

Contudo, não é perdido tempo de execução: as threads efetuam tarefas assim que são criadas e assim que encerram sua execução, estando constantemente realizando tarefas e se desperdiçando o mínimo possível de atenção da CPU. Os diversos ambiente de execução são isolados uns dos outros, garantindo uma execução rápida, eficiente e com uma carga razoável à máquina.

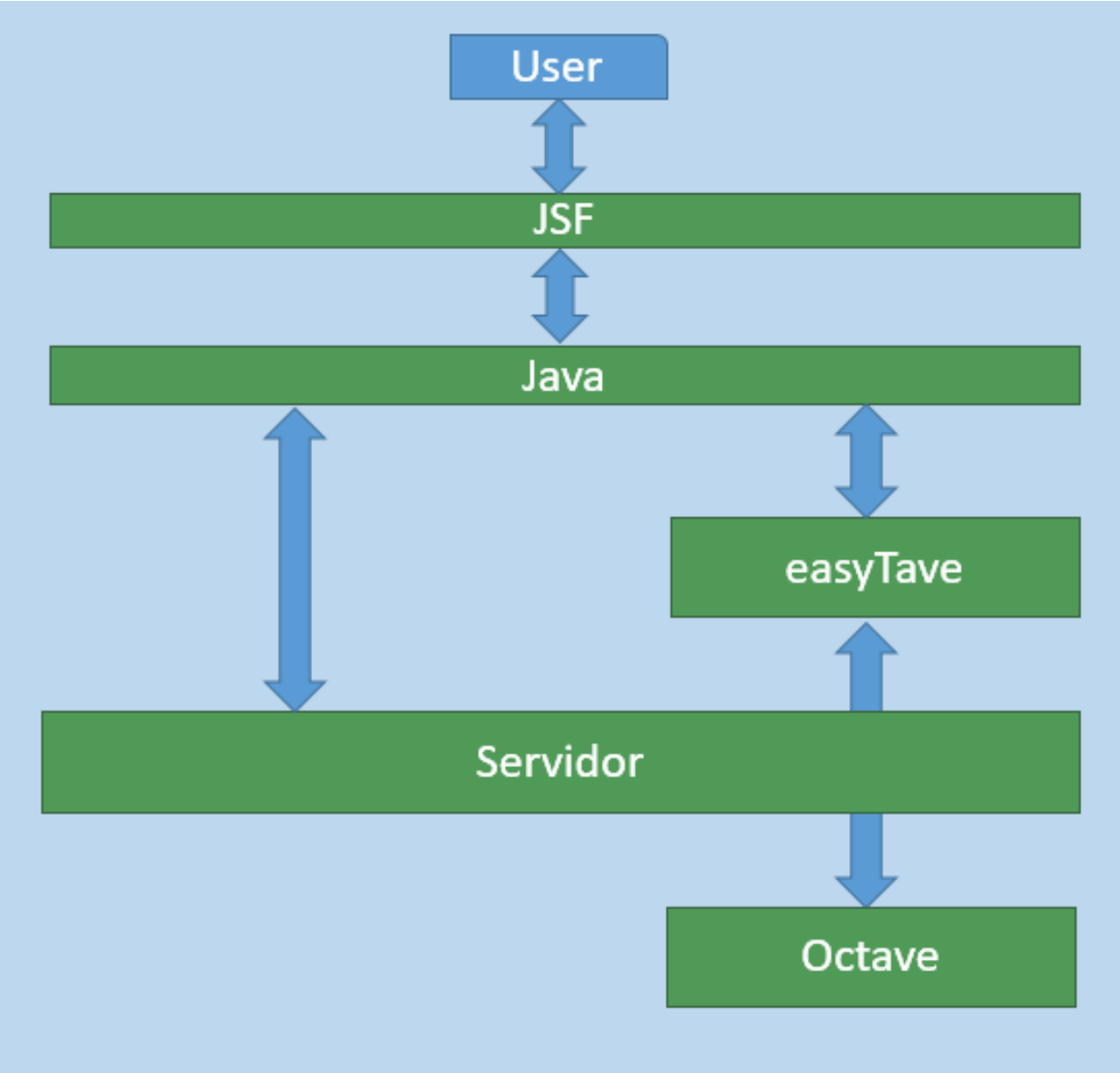

Figura 4.1: *Diagrama do fluxo da informação dentro da aplicação.*

4.2.2 Multithreading

A fim de garantir o bom uso da CPU, temos sempre 8 Threads criadas por vez em 3 estados diferentes: executando, apenas uma delas; criando ou aguardando execução (para todos quesitos, agem como o mesmo estado); e encerrando sua execução.

Criação das Threads

Na criação das Threads são copiados e configurados seus mini-ambientes [1] de execução paralelamente, espaço este onde os arquivos gerados pela simulação do modelo matemático do Octave serão gerados, armazenados e interpretados. Desta forma este ambiente único é de exclusivo uso da Thread, criando informações de estado fixo que não podem ser acessadas por outras threads.

[1] Por mini-ambiente subentendo um diretório dentro do servidor de acesso exclusivo ao usuário da sessão ativa.

Execução das Threads

Durante a execução, o código fornecido pelo usuário é compilado dentro do Octave e executado: tanto a Stream de saída, quanto a Stream de erros são capturadas nesta fase e fornecidas posteriormente ao usuário.

É nesta etapa também que o Octave gera, internamente, um log do estado de todas as variáveis envolvidas na execução. Este log também é capturado durante a execução e salvo no ambiente local da Thread.

Morte das Threads

Ao final de cada execução toda informação necessária é extraída dos ambientes de execução assincronamente e entregue aos seus usuários. Todas as informações destes espaços são destruídas, evitando qualquer interferência com outras execuções.

4.2.3 Modelo Cliente-Servidor

A base do modelo Cliente-Servidor é poder fornecer, assincronamente, o mesmo serviço a vários clientes diferentes, através da troca de mensagens e envio de pacotes entre servidor e cliente (eMS94).

Através do uso do JSF [2] e de um servidor de aplicações Java, somos capazes de fornecer o serviço para vários usuários simultâneos, contudo, precisamos tomar alguns cuidados quanto a implementação a fim de garantir o isolamento dos usuários, ou seja, dar a garantia de que a experiência de uso será uniforme entre todas as partes.

Utilizamos duas ferramentas poderosíssimas para este isolamento: os JavaBeans e o funcionamento nativo do JSF!

Os JavaBeans são classes que, através do JSF, operam como "Static"para cada um dos usuários, tendo seu escopo e ciclo de vida variando de acordo com o estabelecido no desenvolvimento (eRMH06). Os beans são responsáveis por encapsular a interação do usuário com a parte lógica do programa, fornecendo à interface XHTML poderosas ferramentas e simples ferramentas para tratar e as entradas dos usuários e retornar o resultado destas.

Nesta aplicação são utilizados dois tipos de Beans: ApplicationScoped e SessionScoped. O primeiro tem sua vida iniciada no momento em que a aplicação é colocada no ar no servidor de aplicações, morrendo somente quando a mesma é retirada de operação. Já o segundo tipo tem sua vida iniciada no momento em que o usuário acessa a aplicação e morre no momento em que ele deixa de usar a aplicação.

A capacidade de operar de tal forma é essencial em um ambiente como o nosso, em que temos disputa por um recurso crítico finito: o Bean SessionScoped permite que cada um dos usuários tenha um ambiente próprio isolado dos outros, guardando os seus dados adequadamente; em paralelo, o Bean ApplicationScoped é capaz de armazenar informações críticas da aplicação, como quais usuários estão esperando para ter acesso à seção crítica, o acesso exclusivo ao processamento de dados do servidor e retornar corretamente as saídas da aplicação para cada um dos requisitantes.

4.3 Decisões de implementação

Ao longo da criação da aplicação, buscou-se seguir 2 valores principais: tornar o uso da aplicação o mais natural o possível para os usuários e o mais confiável dentro do que fosse viável. Para manter

[2] JSF - JavaServer Faces - é um padrão de interfaces server-side para o usuário, intregrando de maneira fácil a interface XHTML com o código Java. (Ora)

estes ideais algumas decisões específicas foram tomadas.

O layout escolhido para o programa é o mais simples e funcional o possível: a primeira página possui apenas 2 botões - um para fornecer um arquivo e o outro para enviar o arquivo para ser processado.

Figura 4.2: *Pagina inicial da aplicação.*

Assim eliminamos totalmente uma das maiores barreiras do uso do Octave: não é necessário configurar o ambiente, saber instalar o programa, saber usar a UI dele ou os seus comandos específicos no shell - qualquer usuário que saiba programar na linguagem MATLAB pode facilmente rodar seus programas.

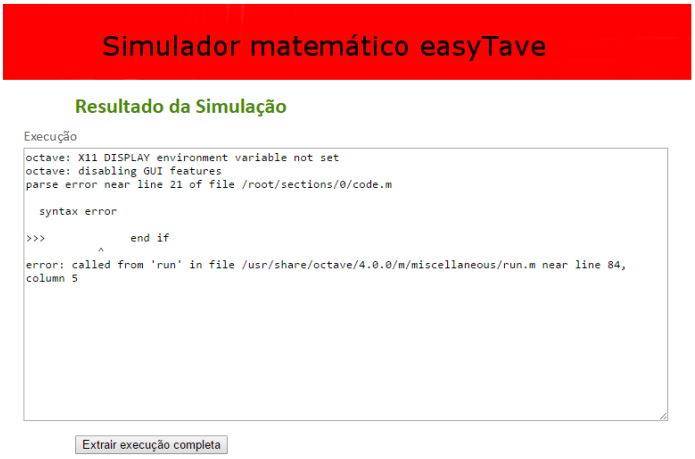

Figura 4.3: *Pagina de resultado mostrando um erro de compilação.*

Contudo, premeditando a possibilidade de falha humana pelos futuros usuários, devemos sempre contar com um caso crucial: e se os programas não terminarem? E se houver um loop infinito ou uma condição de término nunca atingível? Os outros usuários tem seu uso impedido?

Aí entra outro detalhe do desenvolvimento: um tempo máximo arbitrário 20 segundos de execução é garantido a cada um dos usuários. PAra o contexto de alunos realizando pequenas simulações e vide os experimentos feitos em que fizemos 20000 cálculos de complexidade $\mathcal{O}(n^3)$ seguidos com custo aproximado de $2.2E-2s$, então, para todos os casos de bom uso este tempo é mais do que suficiente e garante que nenhum usuário espere muito, mantendo todos bem atendidos.

Outro ponto é que a sessão de cada usuário tem apenas 10 minutos de vida: devemos sempre nos prevenir de não sobrecarregar o servidor com diversos usuários ociosos que, apesar de não utilizarem a maioria dos recursos da aplicação, ainda sim incorrem em custo de processamento pela aplicação ficar escutando por ações do usuário. Estabelecendo um tempo de uso máximo de 10 minutos damos uma margem generosa aos usuários, levando ainda em conta o experimento anteriormente

realizado, mantendo a salvaguarda de garantir a operação regular do serviço à todos os usuários, não importando a carga.

Outro detalhe importante é que a plataforma no qual é implementada a aplicação: ela em si será o agente limitante do desempenho das operações matemáticas. Assim, se preve que, no mínimo, a instância que receberá o servidor de aplicações tenha a seguinte configuração:

> 1GB RAM Intel Xeon E5-2676 2.4GHZ 1GB de memória física Octave 3.8 ou posterior Java 1.7 ou posterior

A aplicação, apesar de pouco custoza em termos de recurso de máquina e da arquitetura desenvolvida, opera, intuitivamente, melhor com recursos tão bons quantos ou melhores para lidar com requisições mais complexas.

Há também arbitrariedades da implementação: operam-se 8 threads por vês pois, empiricamente, verificou-se que é um número adequado de processos para que não se gerem atrasos em outras aplicações disputando pelo servidor.

4.4 Limitações

O easyTave, atualmente possui algumas limitações: ele em si não oferece um compilador com recursos além do que o próprio Octave oferece, não permite edição do código dentro da aplicação.

Estas melhorias seriam sim interessantes, contudo não há como efetuar estes avanços sem que haja o prejuízo de aumentar o tempo de uso de existência das sessões dos usuários e, consequentemente, o custo para o servidor.

Outra possivel melhoria de usabilidade seria um acesso separado à aplicação que permitisse atualizar o ambiente do servidor para que, sem necessidade de intervenção de um sysadmin, mantenha a aplicação compatível com avanços do uso da linguagem Matlab, no entanto isso geraria riscos tão grandes à presença do software em um sistema (exporia o servidor para código malicioso, uma porta de entrada de dados a mais na aplicação, etc), que esta facilidade seria, em verdade, uma grande perda.

Uma área, no entanto, que merece melhorias é a visual: a aplicação muito se beneficiaria de um design mais sofisticado, tornando a experiência de uso mais agradável ao usuário, correções estas que podem vir em uma versão posterior.

Contudo, dado o escopo de exemplo de uso da interface proposta e construida nesta monografia, mostra-se suficente para tal fim.

Capítulo 5

Recepção da Interface e Futuro

A interface fora desenvolvida com o objetivo de integrar ambientes Octave e Java através do shell do SO que a executa. Ela é extremamente bem sucedida nesta tarefa e possui já largas horas de testes em produção como prova do fato.

Uma possível melhoria para ela, no entanto, seria integra-la ao JavaOctave, tornando assim muito mais acessível a toda a comunidade de desenvolvedores.

O fato é que, no entanto, tal integração pode sair do escopo de implementação do JavaOctave atualmente que, segundo a equipe, é fornecer "alguns calculos do Octave dentro da sua aplicação Java"((JO2)) e não todos os recursos do programa de forma integral.

Algo necessário é melhorar a documentação do pacote, a fim de tornar futuras alterações pela comunidade mais fáceis, ato que será realizado ainda ao longo deste ano.

Capítulo 6

Considerações Finais

O easyTave visa integrar com facilidade o ambiente de execução Java com o software Octave e o faz com grande sucesso: o pacote é capaz de, através de ferramentas elementares da linguagem, realizar comunicação com a aplicação através do Shell do SO e executar o programa fornecendo dados ao mesmo.

No entanto a aplicação encontra-se muito sensível à configuração do ambiente em que opera, sendo extremamente adequado que em futuras versões seja blindada a aplicação disso ou, até mesmo, tornar a execução do Octave independente do SO que recebe a sua execução.

Ainda sim, o pacote agrega grande usabilidade ao GNU Octave ao remove o peso de realizar grandes processamentos matemáticos do Java que, como é de comum conhecimento da comunidade, não é a linguagem mais eficiente em termos de consumo de memória. Esta capacidade torna ambas ferramentas de utilidade muito maior e permite uma grande gama de possibilidades. Exemplos dessas possibilidades são o programa exemplo fornecido junto a esta monografia na página do estudando Rafael Campos Cruz e a supracitada, ao longo do texto, SET: ambos especializados em tarefas completamente distintas, mas sendo extremamente úteis aos seus usuários.

Em suma, o easyTave é uma grande nova ferramenta a ser inclusa ao conjunto de pacotes Java, possibilitando que tanto ambientes web quanto aplicações únicas tenham acesso a grande gama de ferramentas matemáticas do Octave, sem altos custos de aprendizado dos usuários das aplicações que usam o easyTave ou de desenvolvedores que farão uso da ferramenta.

Referências Bibliográficas

Dynare what is it? http://www.dynare.org/. Accessed: 2016-01-30.

John W. Eaton et al. Gnu octave.

David Garlan e Mary Shaw. An introduction to software architecture. *Advances in Software Engineering and Knowledge Engineering*, 1:14–15, 1994.

Bill Burke e Richard Monson-Haefel. *Enterprise Java Beans 3.0*. O'REILLY, 5ł edição, 2006.

FAQ how is octave different from matlab? http://wiki.octave.org/FAQ#How_is_Octave_different_from_Matlab.3F. Accessed: 2016-01-25.

Java Octave. https://kenai.com/projects/javaoctave/pages/Home. Accessed: 2016-01-29.

Pricing mathworks pricing and licensing. http://www.mathworks.com/pricing-licensing/. Accessed: 2016-01-25.

JSF javaserver faces technology. http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html. Accessed: 2016-01-29.