UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de São José do Rio Preto

Gabriel Covello Furlanetto

Módulo para importação de modelos GridSim pelo iSPD

São José do Rio Preto
2013

Gabriel Covello Furlanetto

Módulo para importação de modelos GridSim pelo iSPD

Monografia apresentada ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para aprovação na disciplina Projeto Final.

Orientador:

Prof. Dr. Aleardo Manacero Jr.

São José do Rio Preto
2013

Gabriel Covello Furlanetto

Módulo para importação de modelos GridSim pelo iSPD

Monografia apresentada ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para aprovação na disciplina Projeto Final.

Prof. Dr. Aleardo Manacero Jr. Gabriel Covello Furlanetto

Banca Avaliadora:

Prof.ª Dr.ª Luciana Pavani de Paula Bueno
Prof. Dr. Rodrigo Capobianco Guido

São José do Rio Preto
2013

Aos meus pais Rogerio e Ana

À minha avó Maria

Aos meus tios José Mario e Alexandre

"Se você quer ser bem sucedido, precisa ter dedicação total, buscar seu último limite e dar o melhor de si."

-Ayrton Senna

Agradecimentos

Primeiramente gostaria de agradecer aos meus pais Rogerio e Ana, que sempre me apoiaram não apenas nos estudos, mas em todos os momentos e deram condições para que pudesse seguir em frente.

Agradeço também a minha avó Maria, que sempre me incentivou e torceu por mim e a toda a minha família.

Aos meus orientadores Prof. Dr. Aleardo Manacero Jr. e Profa. Dra. Renata Spolon Lobato, pela orientação, profissionalismo, paciência e dedicação, além da amizade.

Aos meus amigos e afilhados Marcelo Rebellato e Liliam Andrade, que sempre torceram por esta conquista, comemorando cada passo até sua chegada.

Ao meu amigo Denison Menezes, que me ajudou muito ao longo do projeto, com sua calma e paciência, e dedicando seu tempo para isso.

Os amigos de escola e de Olímpia, que sempre estiveram do meu lado durante os quatro anos de graduação, dando apoio nos momentos ruins e participando sempre dos bons momentos, Ariéli Vitorasso Blanco, João Eduardo Penariol, Vitorugo Vitorasso, Priscila Minani, André Gregório, Fernando Henrique Carvalho Silva, Maria Clara Boitar Tavares, Marcela Paro, Raissa Zamperline, Carlos Eduardo Spilimbergo, Fernanda Spilimbergo, Renan Sottero, Gabriel Volpe, Alisson Silva Lopes e Alexandre Silva Lopes.

Ao pessoal do laboratório, Danilo Costa, Diogo Tavares, Cássio H. Volpatto, Leandro Moreira, Gabriel Saraiva, Leonardo Santos, Igor Butarello, Vitor Hugo Cândido, Mário Camara Neto, Arthur Jorge, Matheus Oliveira, Thiago Okada, João Antônio Magri e Lúcio Rodrigo Carvalho.

Aos amigos de sala, que sempre estudaram comigo quando necessário e passaram grande parte desses quatro anos ao meu lado, Vinícius Galhardi e Vinícius Oliveira, Heitor Rodrigues, João Marcos Visotaky, Guilherme Salim, Eduardo de Paula, William Ferreira, Gustavo Cagnin, André Morielo Caetano e a única menina da nossa sala e minha grande amiga Mariana Faleiros Penna.

Ao Rafael de Souza Stábile, que morou junto comigo durante grande parte dos anos, com quem sempre estudei para as provas e também é um grande amigo e também ao pessoal que conosco morou, Afonso Sales, Vinícius Bertucci, Bruno Sanches, Bruno Zani, Igor Lampa e Alan Vinhaes.

À minha grande amiga e "bixete"Marina Martins, e ao meu grande amigo, músico e companheiro de shows Daniel Angelotti.

Às duas pessoas muito importantes que conheci ao longo desses anos, Layla Chris R. Ferreira e Bárbara Passos Panosso, que estiveram comigo em grandes momentos e se tornaram muito especiais.

Ao Victor Aoqui e Paulo Henrique Oliveira, com quem pouco tempo passei pessoalmente, mas sempre me ajudaram não apenas com o projeto, mas também ao longo do curso, se tornando também meus grandes amigos.

Ao pessoal da turma de 2012, Maycon Cruz, Verônica Moreira, Laís Andreóli, Juliana Arvani, André Regino, Jorge Simeão e Matheus Gonçalves.

Agradeço também uma pessoa especial que conheci quando estava na parte final de meu projeto, Lígia M. Landi. Ela me deu apoio na etapa final de escrita desse e comigo vibrou a cada fase que era concluída.

E aos demais amigos que fiz na faculdade, Marcela Prado, Matheus Goyos, Ingrid Martins, Lívia Souza, Guilherme Volpe e Rafael Gombrade. Todas essas pessoas nunca serão esquecidas devido à sua importância em minha vida.

Por fim, gostaria de agradecer à UNESP pelo apoio e financiamento concedidos juntamente com o CNPq de na maior parte de minha iniciação científica com bolsas PIBIC/Reitoria à FAPESP que auxiliou o projeto, fornecendo equipamentos ao laboratório do GSPD.

Gabriel Covello Furlanetto

Resumo

O projeto desenvolvido é um módulo para importação de modelos externos que foi acoplado ao simulador de grades computacionais iSPD (*iconic Simulator of Parallel and Distributed Systems*), desenvolvido pelo GSPD (Grupo de Sistemas Paralelos e Distríbuidos). Tal módulo nada mais é que um interpretador, que deve ler um modelo do GridSim, outro simulador, e convertê-lo em um modelo do iSPD. Para isso, foi desenvolvida uma gramática com regras que sejam capazes de reconhecer a linguagem Java, usada pelo GridSim, e, a partir da gramática, usando-se a ferramenta *Java Compiler Compiler* 5.0, a qual é uma ferramenta de geração de códigos para interpretação de linguagens, o interpretador foi produzido. Parâmetros contidos no código GridSim e necessários para a simulação do iSPD, devem ser armazenados em uma estrutura de dados. A partir da qual, serão buscados pelo iSPD e carregados, para que assim a conversão seja finalizada. Desse modo, a funcionalidade desejada foi acoplada ao simulador, possibilitando que usuários do GridSim possam executar seus modelos também no iSPD, com a finalidade de verificar se eles representam o que esperavam e se não houve erro ao programá-los.

Palavras-chave: Interpretação. Gramática. Grades Computacionais

Abstract

The project developed is an module to import of external models to be coupled to the simulator grids iSPD (*i*conic Simulator of Parallel and Distributed Systems), developed by GSPD (*G*rupo de Sistemas Paralelos e Distribuídos). The interpreter should read a model GridSim, another simulator, and convert it into a model iSPD. For this, will be developed a grammar with rules that are able to recognize the Java language, used by GridSim, and from the grammar, using the tool *J*ava Compiler Compiler 5.0, which is a code generation tool for interpreting languages, the interpreter was produced. The parameters contained in GridSim code, needed for the simulation of iSPD, should be stored in a data structure. From which, will be sought by iSPD and loaded, so that the conversion is finished. Thus, the desired functionality was coupled to the simulator, enabling you GridSim users can run their models also in iSPD in order to verify whether they represent what they expected and if there was no error to program them.

Keywords: Interpretation. Grammar. Grid computing

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

GSPD Grupo de Sistemas Paralelos e distribuídos

iSPD iconic Simulator of Parallel and Distributed Systems

Capítulo 1

Introdução

Um sistema distribuído é um conjunto de máquinas ligadas através de rede, que se comunicam e fazem sincronização através de trocas de mensagens. Desse modo, para que isso seja possível, mascarando a heterogeneidade das redes, tais sistemas utilizam um software chamado *middleware*. Tal sistema deve: se sujeitar a falhas, recuperando-se sem maiores prejuízos, funcionar mesmo com ausência de sincronismo de relógios, garantir segurança mínima aos seus usuários, trabalhar com a concorrência entre seus componentes, funcionar bem diante da escalabilidade, ou seja, capacidade de não perder suas características principais diante do aumento de usuários, e heterogeneidade, ou seja, capacidade de aumento do número de máquinas sem perdas, sendo seu uso favorecido principalmente pelo compartilhamento de recursos (COULORIS; DOLLIMORE; KINDBERG, 2001).

Dessa forma, esses tipos de sistemas possuem imensa utilização, principalmente na área de pesquisas, como por exemplo para cálculos avançados feitos por físicos ou mesmo no ramo comercial. Assim, com o passar dos anos tem se tornado cada vez mais comum o uso de grades computacionais em várias áreas e por isso, a fim de otimizar a construção das grades, vem surgindo vários simuladores, como o GridSim (GRIDSIM, 2013) e o iSPD (iconic Simulator of Parallel and Distributed Systems) (MANACERO et al., 2012), sobre os quais será tratado mais profundamente ao longo deste projeto.

1.1 Motivação

A importância de tal projeto surge uma vez que o iSPD é o primeiro simulador de grades computacionais a oferecer ao usuário facilidades, já que, com sua interface icônica,

possibilita que um usuário leigo possa simular seu projeto de grade até mesmo sem saber utilizar linguagens de programação. Desse modo, a inserção do módulo de interpretação de modelos GridSim no simulador iSPD torna-se também muito importante, uma vez que esse já possuia um interpretador SimGrid (SIMGRID, 2013) e agora possui também um exportador para o GridSim. Tal importância é devida principalmente a comunicação entre simuladores que foi criada, a qual não existe em outros softwares do mesmo tipo. Desse modo, usuários, tanto do iSPD, quanto do GridSim, poderão migrar de simulador ou validar e confirmar seus modelos em outro programa do tipo.

1.2 Objetivo

O objetivo principal deste trabalho é acrescentar, na ferramenta de simulação de grades computacionais iSPD, do Grupo de Sistemas Paralelos e Distribuídos (GSPD) (GSPD, 2013), o módulo de interpretação de modelos externos provenientes do GridSim; ou seja, para adicionar ao iSPD uma funcionalidade que permita ao usuário que tenha em suas mãos um modelo criado no GridSim, também fazer uma simulação, com o mesmo modelo, no iSPD, e desse modo tenha chance não apenas de verificar se o modelo que tem está correto, mas também de validar os dados obtidos com sua simulação anterior.

1.3 Organização do texto

Além deste capítulo introdutório, o capítulo 2 faz uma análise teórica dos demais simuladores existentes, do iSPD e, por fim, apresenta fundamentações teóricas utilizadas ao longo do trabalho. No capítulo 3 é apresentado como o interpretador foi implementado, mostrando desde a fase de estudos até a fase de codificação. A fase de testes e validação aparece no capítulo 4, com simulações no GridSim e no iSPD, e apresentando também mensagens de erro que o simulador traz consigo ao interpretar um código. Por fim, no capítulo 5 aparecem as conclusões que puderam ser tiradas com o trabalho, junto com as principais dificuldades encontradas, trabalhos futuros que podem ser feitos e as publicações.

Capítulo 2

Revisão Bibliográfica

Este capítulo apresenta o iSPD (iconic Simulator of Parallel and Distributed Systems) (MENEZES et al., 2012), simulador para o qual foi acoplado o novo módulo de interpretação, o GridSim (GRIDSIM, 2013), os demais simuladores de grades computacionais existentes e disponíveis na atualidade, e por fim alguns conceitos teóricos sobre compilação necessários para realizar o restante do trabalho.

2.1 iSPD

O iSPD consiste no simulador de grades computacionais desenvolvido pelo Grupo de Sistemas Paralelos e Distribuídos (GSPD)(GSPD, 2013). É uma ferramenta desenvolvida na linguagem Java que tenta trazer ao usuário as maiores facilidades possíveis. Para isso, oferece uma interface icônica, a qual pode ser vista na figura 2.1, e agora contando com a possibilidade da comunicação, através da conversão de modelos, com mais dois simuladores (GridSim, no qual o trabalho se insere e Simgrid(SIMGRID, 2013), a qual já estava pronta), permitindo com que ele, além de validar seu modelo em outros simuladores, possa também converter um modelo de um simulador textual em um tipo mais visual e fácil de compreender.

A ferramenta é dividida em quatro módulos básicos que, ao trabalharem de maneira cooperativa, realizam as simulações e retornam para o usuário os dados estátisticos referentes a elas, como o tempo simulado, tempo de simulação, carga de cada máquina, entre outros. Tais módulos são:

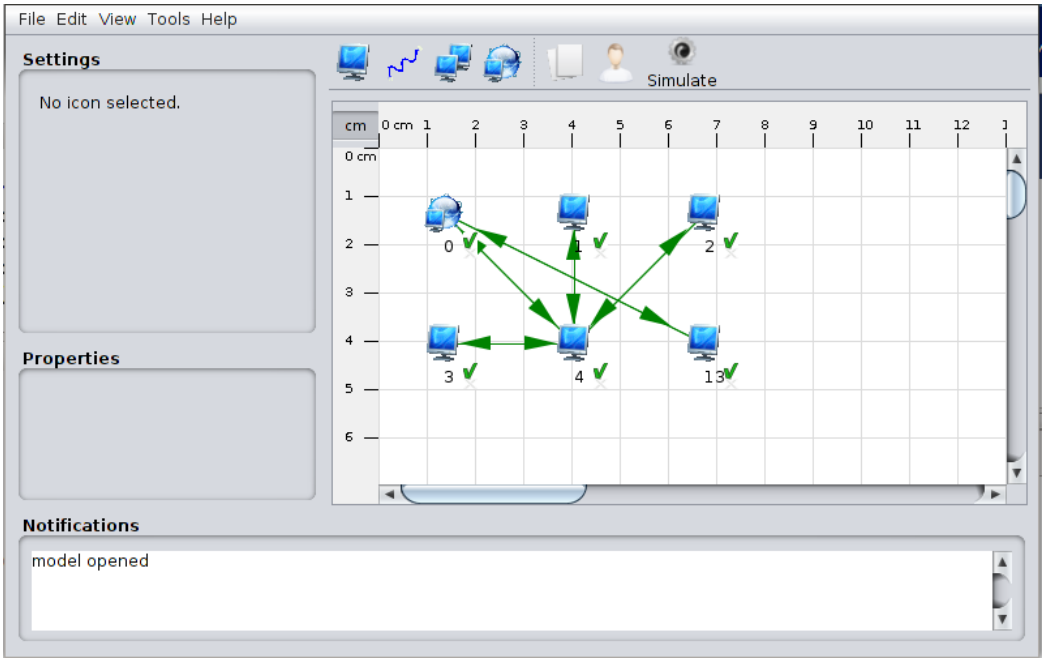

Figura 2.1: Imagem da interface icônica do iSPD que possibilita uma modelagem de forma gráfica ao usuário

- **Interpretador de modelos:** Parte do simulador que visa a conversão entre modelos usados em outras ferramentas, como o GridSim, em que este trabalho se insere, e o SimGrid, para modelos que possam ser simulados diretamente no iSPD, e desse modo há também a conversão de modelos textuais, os quais são utilizados pelos outros simuladores, para modelos gráficos, que podem ser apresentados na interface icônica, sobre a qual será falada no próximo tópico. (AOQUI et al., 2010);

- **Interface icônica:** O módulo tem como funcionalidade a interação humano-computador, visando fazer com que o usuário do simulador consiga obter maior facilidade, conforto, qualidade de acesso e fique satisfeito ao interagir com a ferramenta, não tendo tantos problemas como se precisasse, por exemplo, fazer a programação textual do modelo. (GUERRA et al., 2010);

- **Gerador e gerenciador de escalonadores:** Apesar de serem citados como um só, estes dois módulos são os responsáveis pela geração dos escalonadores do iSPD. O simulador oferece alguns tipos de escalonadores já implementados e prontos para uso, e também um ambiente gráfico em que o usuário pode digitar

seu código de escalonamento, para que esse seja feito a seu gosto (MENEZES et al., 2012).

- **Motor de simulação:** O motor é o módulo que faz a simulação propriamente dita. Para isso, ele faz leitura de um modelo de grade, converte-o em um modelo de filas e por fim realiza a simulação, disponibilizando o resultado, para que seja exibido ao usuário pela interface icônica. Tal motor pode utilizar para a simulação cargas importadas de um banco de cargas (MANACERO et al., 2012);

Desse modo, para um melhor entendimento, apresentamos abaixo na figura 2.2 um diagrama conceitual do simulador, em que é demonstrado a forma que os módulos se relacionam e garantem o funcionamento da ferramenta.

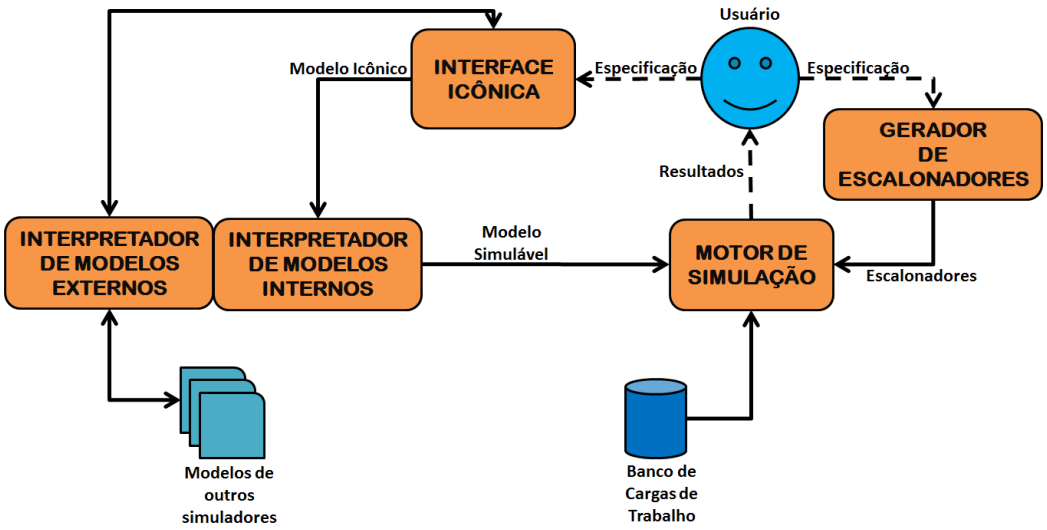

Figura 2.2: Diagrama conceitual do iSPD (MENEZES, 2012)

Assim, observamos que o iSPD consiste em um simulador de grades computacionais que busca trazer facilidades ao usuário. Nesse contexto, se encaixa o trabalho desenvolvido, que irá inserir uma nova funcionalidade a ele, a qual fará interpretação de modelos oriundos do GridSim, sobre o qual será falado na seção 2.2.

2.2 GridSim

Essa seção será utilizada para apresentar uma visão detalhada do simulador GridSim.

Atualmente, ele se encontra na versão 5.2, é desenvolvido na linguagem Java e apre-

senta funcionalidades como modelar simulações de computação paralela e distribuída. Permite fazer um estudo sobre o desempenho dos algoritmos de escalonamento usados nessas simulações, e também, que haja a ocorrência de falhas nas tarefas submetidas as grades, para que se possa observar como essa se comportaria em tal situação. Possibilita que sejam importadas cargas de trabalho de uma grade real, extensão para simulação de grades de dados, reserva de horários para utilização da grade, diminuição de trabalho aos finais de semana, entre outros (GRIDSIM, 2013).

O simulador apresenta uma arquitetura de camadas, como a apresentada na figura 2.3. Como pode ser visto, na primeira camada, há a interface Java, na segunda camada, está a parte que, utilizando os serviços disponibilizados pelas outras interfaces, possibilita a estrutura de eventos discretos do simulador. Na terceira camada, aparecem as ferramentas do GridSim, as quais possibilitam a modelagem da grade a ser simulada, seguida, na quarta camada, pelos modelos de escalonadores que podem ser utilizados para distribuir as tarefas em uma simulação. Por fim, na quinta camada, estão as aplicações da ferramenta, os usuários e as entradas e saídas utilizadas.

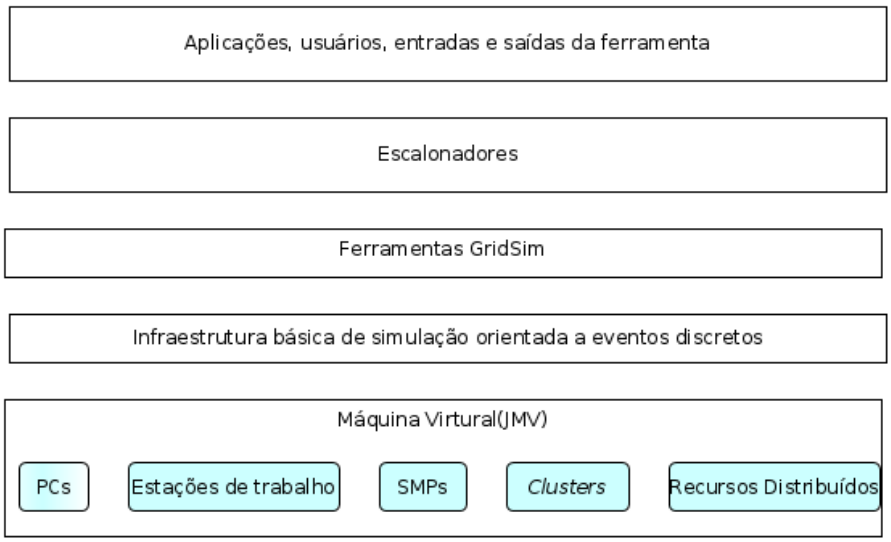

Figura 2.3: Representação gráfica da arquitetura do GridSim, adaptado de (BUYYA; MURSHED, 2002)

No entanto, a ferramenta exige que o usuário faça a modelagem da sua grade utilizando também a linguagem de programação Java, produzindo um código, do mesmo tipo do trecho visto na figura 2.4. Sendo assim, essa ferramenta acaba sendo comple-

```
public class ExemploGridSim
{
    public static void main(String[] args) //Main
    {
        System.out.println("Iniciando simulação");

        try
        {
            // Iniciar pacote GridSim
            int usuarios_grade = 2;   //
            Calendar calendario = Calendar.getInstance();
            boolean trace_flag = true;
            System.out.println("Inicializando pacote GridSim");
            GridSim.init(usuarios_grade, calendario, trace_flag);

            // Criar recursos da grade
            double taxa_transmissao = 1000; // em bits/sec
            double atraso_propagacao = 10;  // em milisegundos
            int mtu = 1500;                 // em byte
            int Recursos_totais = 4;
            ArrayList Lista_recurs = new ArrayList(Recursos_totais);
            GridResource rec = criarRecursosGrade(1, taxa_trasmissao, atraso_propagacao, mtu);
            Lista_recurs.add(recurs); // adicionar recurso na lista

            int total_tarefas = 5; //Tarefas a serem submetidas para a(s) grades
            // Criar usuários
            ArrayList Lista_usuarios = new ArrayList(usuarios_grade);
            for (int i = 0; i < 3; i++)
            {
                NetUser usuario = new NetUser(i, total_tarefas, taxa_trasmissao, atraso_propagacao, mtu, trace_flag);
                userList.add(user); //adicionar usuários a lista
            }

            // Construir topologia de rede
            Router r1 = new RIPRouter("roteador1", trace_flag);   // roteador 1
            Router r2 = new RIPRouter("roteador2", trace_flag);   // roteador 2
```

Figura 2.4: Exemplo de trecho de código utilizado para simulação no GridSim

tamente textual, e desse modo, dificultando muitas vezes o trabalho de usuários leigos no assunto.

O simulador possui vasta documentação, facilitando sua compreensão com finalidade de pesquisa e, nesse caso, com o fim do estudo a ser realizado com o intuito de realizar o projeto que foi proposto inicialmente.

Por fim, sabe-se que o projeto encontra-se em atividade na atualidade, buscando-se novas funcionalidades e maior perfeição nas simulações feitas pela ferramenta.

2.3 Outros Simuladores

Essa parte do texto será destinada à apresentação dos demais simuladores existentes no atual cenário de pesquisa em grades, entre eles, o Bricks (BRICKS, 2013), OptorSim (OPTORSIM, 2012) e SimGrid (SIMGRID, 2013).

2.3.1 SimGrid

O SimGrid também consiste em um simulador de grades computacionais. Além disso, também possui suporte para simulações P2P, MPI, e *Cloud Computing*. Ele pode ser utilizado em várias plataformas, como Linux, Mac OS e Windows, sendo uma ferra-

menta aberta para ser baixada e testada pelo usuário, sem custos. Já possui mais de 10 anos de desenvolvimento, sendo financiada por investidores e tendo suporte para mais alguns anos de pesquisa (SIMGRID, 2013).

Tal simulador possui vasta documentação disponível para o usuário poder estudá-lo, sendo desenvolvido na linguagem C. Seu usuário precisa possuir alguma habilidade com linguagens de programação, uma vez que não possui ambiente gráfico e deve ser programado também em linguagem C, para executar um modelo de simulação.

2.3.2 OptorSim

O simulador teve seu projeto iniciado em 2006. Ele também consiste em um projeto aberto, podendo qualquer usuário testá-lo sem pagar taxa nenhuma por isso. Seu principal objetivo é testar estratégias de escalonamento, as quais podem usar replicação de dados (OPTORSIM, 2012).

A ferramenta possui um modo relativamente simples de modelagem da grade e, desse modo, não obriga que o usuário tenha conhecimentos sobre linguagens de programação para fazer seus testes, apesar de oferecer a possibilidade da implementação de algoritmos também. Ela possui uma interface gráfica simples e está implementada na linguagem de programação Java, permitindo assim que possa ser usada em diferentes sistemas operacionais. (OLIVEIRA, 2007).

2.3.3 Bricks

O Bricks, como os demais simuladores de grades computacional, também realiza a simulação de diferentes topologias de rede, além de possibilitar ao usuário gerar modelos de diferentes tipos de escalonamento e assim verificar o comportamento do sistema nas mais variadas situações (BRICKS, 2013).

O simulador é orientado a eventos, e seu projeto encontra-se parado na atualidade, sendo implementado na linguagem de programação Java e obrigando que seu usuário domine a programação em *script* para poder construir seus modelos e simulação. (OLIVEIRA, 2007).

2.4 Conceitos de compiladores e linguagens

A seção busca apresentar alguns conceitos básicos de gramáticas, linguagens e compiladores, os quais foram muito utilizados ao longo do trabalho, uma vez que o módulo a ser inserido na ferramenta iSPD consiste em um interpretador de linguagens, o qual tem como função, no caso, interpretar a linguagem Java utilizada na construção dos modelos da ferramenta GridSim. A importância dos conceitos ditos acima também aparecem na geração de duas gramáticas, uma regular e uma livre de contexto, as quais são utilizadas na interpretação.

Desse modo, falaremos aqui sobre tais tipos de gramática, sobre as linguagens que elas geram e também das duas fases de interpretação que deveriam ser feitas, a fase de análise léxica e a fase de análise sintática.

2.4.1 Análise Léxica

A fase de análise léxica corresponde à primeira fase da compilação de um código fonte. Os dados de entrada para que ela ocorra estão presentes no código fonte, que é tratado como um texto contínuo, a partir do qual vão sendo gerados *tokens*. Tais tokens são encontrados em comparações do que está sendo lido com as palavras que formam uma linguagem de programação.

Além de separar o código em *tokens*, o analisador léxico também tem importante papel na análise, uma vez que detecta problemas léxicos nas construções que estão sendo lidas, exibindo-as ao programador.

Desse modo, o analisador léxico, deve ser capaz de reconhecer uma linguagem regular, a qual será apresentada em 2.4.4. Tal linguagem é formada por todos os símbolos utilizados por Java e, dessa forma, possibilita que a análise seja feita (SEBESTA, 2003).

2.4.2 Análise Sintática

Essa fase de compilação ocorre logo após a análise léxica e, assim, tenta encontrar erros sintáticos no código. Para isso, utiliza os *tokens* criados na fase anterior, e tenta encontrar o maior número de erros sintáticos possíveis em uma só leitura do código, retornando mensagens de erro para o programador (SEBESTA, 2003).

Para a construção desse analisador, duas abordagens são geralmente utilizadas. A *top-down*, em que constrói-se uma árvore sintática a partir da raiz para as folhas, em

que a árvore armazena todas as estruturas da linguagem presentes no código, como os *if-else* e *while*, ou uma abordagem *bottom-up*, em que tal árvore é construída a partir das folhas, até chegar na raiz (SETHI; ULLMAN; LAM, 2008).

Dessa forma, o analisador sintático deverá reconhecer uma linguagem livre de contexto, que será apresentada em 2.4.5. Tal linguagem possui nela estruturas, como por exemplo laços de repetição, condicionais ou declarações de variáveis, que formam a linguagem de programação, e assim serão lidas durante essa parte da compilação.

2.4.3 Conceitos básicos de linguagens

Antes de definir os dois tipos de linguagens que foram utilizados no projeto, serão apresentados alguns conceitos básicos sobre elas, entre eles as definições formais de alfabetos, palavras e linguagens (DELAMARO, 2004):

- **Alfabeto:** Representado pelo símbolo Σ, segundo (DELAMARO, 2004) "um alfabeto consiste em um conjunto finito e não vazio de símbolos".
- **Palavras:** As palavras de uma linguagem são representadas pela concatenação de símbolos pertencentes ao alfabeto da mesma, podendo ter comprimentos variáveis a partir de zero.
- **Linguagens:** Uma linguagem pode ser definida pelo conjunto de zero ou mais representações dos símbolos de seu alfabeto ou de palavras formadas pela concatenação de tais símbolos, como já dito na definição de palavras. Desse modo, surge uma classificação para elas, denominada Hierarquia de Chomsky, a qual faz a divisão em quatro tipos (MENEZES, 1998):
  - Tipo 0: Linguagens Recursivamente Inumeráveis
  - Tipo 1: Linguagens Sensíveis ao Contexto
  - Tipo 2: Linguagens Livres de Contexto
  - Tipo 3: Linguagens Regulares

  Assim, será tratado mais profundamente sobre as duas últimas, uma vez que são elas que aparecem no desenvolvimento do projeto sendo descrito.

2.4.4 A Linguagem Regular e sua gramática

Uma Linguagem Regular, ou do tipo 3, possui importância, uma vez que, para sua interpretação, reconhecedores simples podem ser implementados, sendo de grande facilidade a tarefa do programador. Para tal tarefa, podem ser utilizados autômatos finitos, gramáticas regulares ou expressões regulares.

Desse modo, neste projeto, o reconhecimento será feito através uma Gramática Regular (MENEZES, 1998), a qual conterá todos os símbolos utilizados na linguagem Java para importar modelos de simulação do GridSim.

2.4.5 A Linguagem Livre de Contexto e sua gramática

Linguagens Livres de Contexto, ou do tipo 2, são as responsáveis pela construção estrutural das linguagens de programação, sendo formadas pelas Gramáticas Livres de Contexto. Dessa forma, elas ganham grande importância no projeto, o qual visa fazer a interpretação de um código produzido em linguagem Java.

Assim, tais linguagens são representadas por uma quádrupla (Ω, Σ , S, P), em que:

- Ω: Representa o conjunto não vazio de símbolos terminais.
- Σ: Representa o alfabeto que forma a linguagem.
- S: Representa o símbolo de partida para as regras de produção.
- P: Representa o conjunto de produções que darão origem a linguagem.

Desse modo, como já dito, em fases posteriores do projeto, será gerada uma gramática desse tipo para o reconhecimento dos códigos desejados (DELAMARO, 2004).

2.5 Considerações finais

Neste capítulo apresentou-se a ferramenta iSPD, para a qual este trabalho irá inserir uma nova funcionalidade, a de interpretação de modelos do GridSim. Estudou-se também a ferramenta da qual será proveniente o código a ser interpretado pelo iSPD, assim como outros simuladores de grades importantes disponíveis para pesquisa na atualidade e, por fim, um pouco de teoria básica necessária para dar prosseguimento no trabalho. O capítulo 3 falará das atividades realizadas ao longo do projeto.

Capítulo 3

Metodologia de conversão entre modelos

Este capítulo traz uma apresentação de todo o trabalho realizado, desde a parte de estudos até a implementação da nova função de interpretação de modelos do GriSim (GRIDSIM, 2013) para o iSPD (MENEZES et al., 2012).

3.1 Pesquisa e estudos

Na seção 3.1 será apresentado um resumo rápido da fase de estudos precedente ao início das implementações do projeto, uma vez que os estudos aprofundados já foram descritos no capítulo 2.

Desse modo, o primeiro passo feito ao inciar-se o período de projeto foi uma pesquisa sobre a ferramenta de grades computacionais do GSPD (GSPD, 2013), seu funcionamento e em que ambiente se enquadrava o projeto a ser desenvolvido, conforme mostrado em 2.1.

Após essa pesquisa, realizou-se um estudo sobre os demais simuladores de grades existentes no ambiente de pesquisa atual, conforme mostrado em 2.2 e 2.3. Dessa forma, decidiu-se que a ferramenta a ser utilizada no projeto, que receberia um maior aprofundamento para que o iSPD tivesse uma nova função de importação, seria o GridSim, já que este também é implementado em Java, possui boa documentação e continua com seu projeto em desenvolvimento na atualidade. A partir disso, iniciou-se um estudo aprofundado da linguagem Java, para maior compreensão sobre os algoritmos de exemplo que ele apresenta em sua documentação.

Por fim, terminou-se a fase de estudos, analisando todos os parâmetros que um

código GridSim possuía que deveriam ser passados para o iSPD durante a conversão, para que assim, em um próximo passo, pudesse ser começado o desenvolvimento da função de interpretação dos códigos.

3.2 Construção das Gramáticas Regular e Livre de Contexto

O primeiro passo no desenvolvimento da nova função que seria acoplada ao iSPD foi a construção das gramáticas Regular e Livre de Contexto. Tais gramáticas seriam capazes de reconhecer completamente qualquer código do GridSim, sendo, para isso, produzidas na Forma de Backus-Naur (BNF), um tipo de formalismo para representação de linguagens formais que facilitaria, em um posterior momento, a geração do código do interpretador.

Desse modo, utilizou-se uma especificação de um interpretador Java, disponível em (JAVACC, 2013a), para fazer a gramática de tal linguagem.

Como já dito, as gramáticas são duas, uma regular e uma livre de contexto. A regular, constituída por regras de formação que dão origem a símbolos terminais, como por exemplo:

<CLASS> ::= “class”

Nela, quando <CLASS> aparecer, dará origem à palavra reservada da linguagem Java “class”, a qual, como o nome já diz, é utilizada em uma declaração de classe. Ou então:

<DECIMAL> ::= [“1- “9”] ([“0- “9”])*

<IGUALDADE> ::= “==”

Em que a regra <DECIMAL> gera os números inteiros e <IGUALDADE> gera o símbolo na linguagem que representa a igualdade entre variáveis.

Já a outra gramática, ou seja, a Livre de Contexto, é constituída por regras que geram outras regras intermediárias, como podemos ver logo abaixo, em que <importar_declaraçao> representa as regras de importação de pacotes em Java e <tipo_primitivo>

dá origem aos tipos primitivos de variáveis que podem ser declarados.

<importar_declaraçao> ::= <IMPORT> [<STATIC>] <nome> [<PONTO> “*”] <PONTO_VIRGULA>

<tipo_primitivo> ::= <BOOLEAN> | <CHAR> | <BYTE> | <SHORT> | <INT> | <LONG>| <FLOAT>| <DOUBLE>

As demais normas que geram ambas as gramáticas podem ser vistas nos apêndices A e B.

3.3 Fase de implementação

Após a especificação da gramática, foram geradas normas de exceção que estabelece se uma regra é ou não do Gridsim. Para isso, utilizou-se do código fonte disponível em (JAVACC, 2013a), que faz a interpretação de regras Java, utilizando-se dessa linguagem para implementar os condicionais que realizariam a verificação de padrões que devem estar presentes em uma norma se essa for parte do Gridsim. Assim, se um padrão importante for detectado, em um primeiro momento, uma mensagem deveria ser impressa na tela, mostrando todos os parâmetros utilizados por tal padrão, e em um próximo passo do projeto estes passariam a ser salvas em uma estrutura semelhante à uma tabela para então poderem ser utilizados pelo iSPD, como veremos mais adiante. Dessa forma, nas figuras 3.1, 3.2, 3.3 mostramos algumas mensagens sendo impressas no terminal representando os parâmetros já citados.

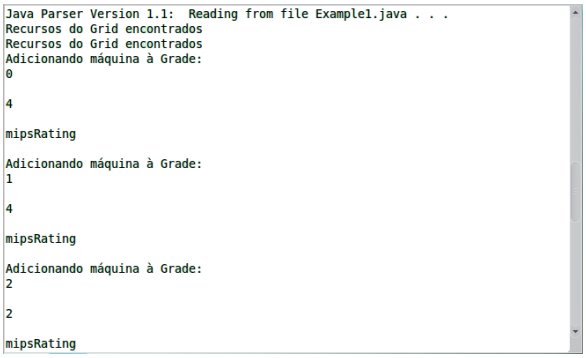

Figura 3.1: Terminal apresentando parâmetros encontrados ao adicionar uma máquina à grade

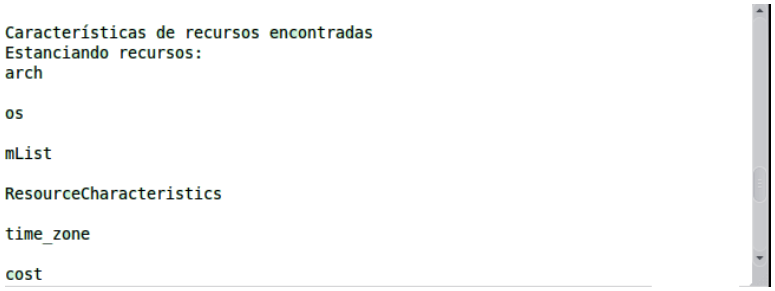

Figura 3.2: Terminal mostrando parâmetros encontrados ao instanciar recursos da grade

Figura 3.3: Terminal mostrando novamente parâmetros encontrados ao instanciar recursos da grade

A partir do momento em que o primeiro reconhecedor das estruturas GridSim ficou pronto e validado, conforme será detalhado no Capítulo 4, iniciou-se a implementação da estrutura que deveria salvar os dados a serem utilizados pelo iSPD e que estão presentes nos códigos GridSim.

Para isso, criou-se uma classe no código em que variáveis armazenam os nomes dos parâmetros junto a uma tabela de valores que armazena o módulo numérico do parâmetro a ser passado, caso esse não esteja vinculado no momento em que foi interpretado, sendo então feita uma busca e atribuindo o valor correto a variável indicada.

A partir de tal classe, o iSPD busca as variáveis para a simulação e assim preenche os dados necessários para que ela possa ser realizada.

Dessa maneira, foi incluída na versão mais atual do simulador do grupo a funcionalidade de interpretação de modelos externos, a qual pode ser acessada diretamente a partir da interface gráfica, conforme observa-se na figura 3.4.

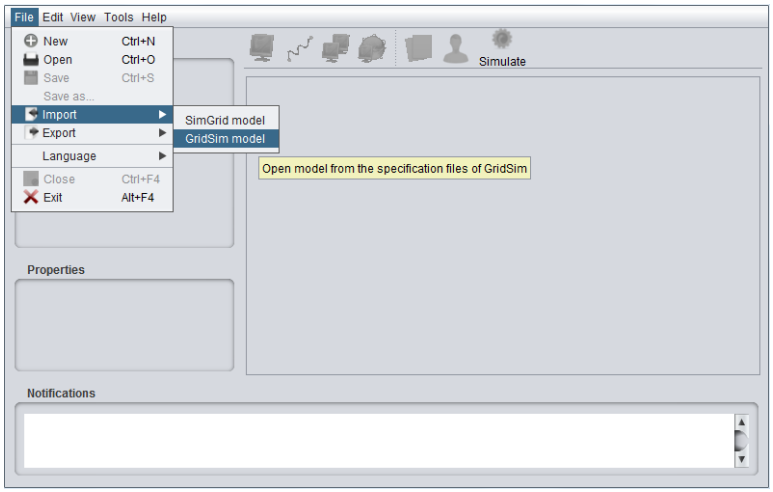

Figura 3.4: Modo de acesso à interpretação de modelos externos pela interface

3.4 Codificação do interpretador

Com o fim de realizar uma conversão de um modelo GridSim para um modelo gráfico do iSPD, como pode ser visto em 3.5, o interpretador lê um código em Java com um modelo GridSim, salva os parâmetros relevantes para o iSPD em uma classe Java e por fim, o simulador de grades do GSPD, ao gerar o modelo icônico, busca por tais dados e os carrega em sua interface.

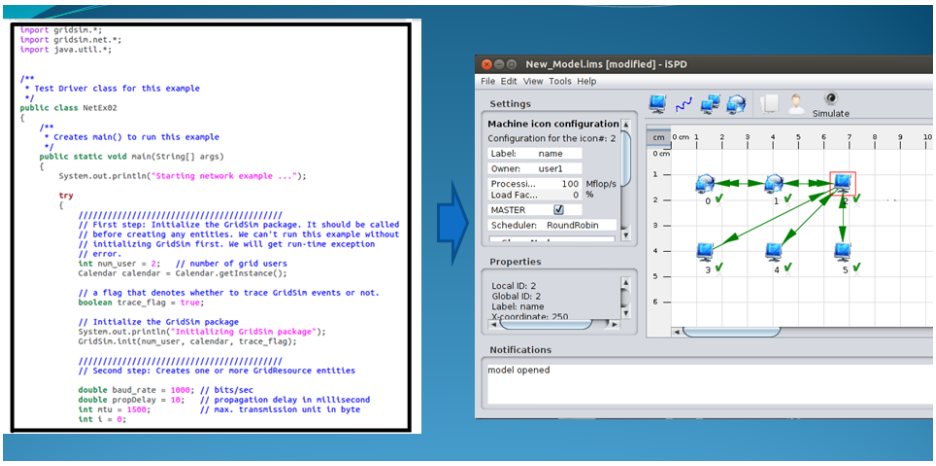

Figura 3.5: Ilustração de uma conversão de um modelo GridSim (código) para um modelo gráfico iSPD

Desse modo, implementou-se o código do interpretador, conforme pode-se observar na figura 3.6, a qual é um diagrama de classes desse código. Nele, vemos que o módulo de interpretação possui três classes, sendo elas a InterpretadorGridSim, a JavaParser e, por fim, a ResourceChar.

A classe ResourceChar tem como finalidade ser a classe armazenadora dos parâmetros úteis que vão sendo lidos ao longo do código interpretado. Dessa forma, pode-se ver que nela existem apenas atributos, sem chamadas de métodos, os quais são responsáveis por guardar informações de cada dado. Por exemplo, *gridletLength*, que armazena o tamanho de uma tarefa ou *propDelay* que armazena dados relativos aos atrasos de propagação.

Já a classe JavaParser é quem se encarrega da análise sintática propriamente dita. Assim, utiliza seus métodos, como *create_for_expression()*, para obter valores a serem armazenados. E, por fim, a classe InterpretadorGridSim, que possui um único método interpreta(), o qual é responsável por passar as variáveis lidas e armazenadas ao iSPD, para que esse faça a conversão e exiba na tela o modelo icônico.

Por fim, algumas classes utilizadas não foram detalhadas, uma vez que eram geradas pela ferramenta JavaCC (JAVACC, 2013b), o que aconteceu também com os métodos *set* e *get*. O código fonte dessas classes citadas foi gerado a partir da especificação das gramáticas, conforme mostradas em 3.2 diretamente na ferramenta JavaCC, a qual por se tratar de uma ferramenta de geração de código, produziu os analisadores léxico e sintático, para os quais foram implementadas regras de exceção capazes de capturar os dados necessários para uma simulação, e desse modo, após o acoplamento da função ao simulador do GSPD, a fase de codificação chegou ao fim.

3.5 Considerações finais

Este capítulo trouxe informações sobre os estudos realizados, de modo a aumentar a base teórica para o início do projeto. Após isso, houve a apresentação das gramáticas construídas para a implementação do interpretador e do método usado em tal construção, seguido de uma rápida apresentação do passo a passo adotado ao longo do projeto para que o interpretador fosse implementado. Por fim, foi feito um detalhamento maior sobre a codificação desse.

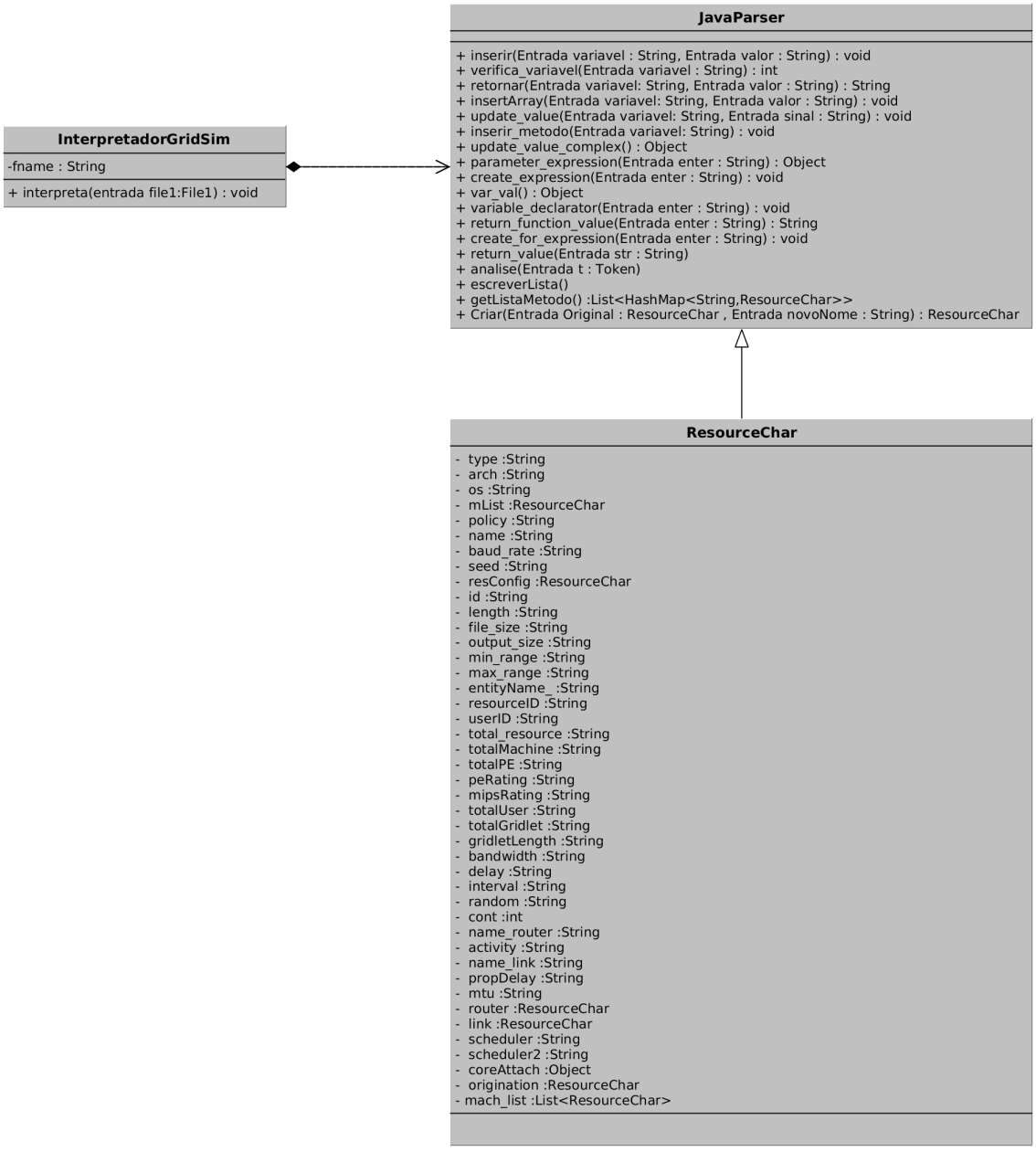

Figura 3.6: Diagrama de classes do interpretador

Capítulo 4

Testes e validação do interpretador de modelos GridSim para o iSPD

O capítulo tem como finalidade apresentar os testes e resultados obtidos como forma de validação do projeto desenvolvido e aqui apresentado, desde os testes realizados durante a codificação até os realizados com o interpretador já pronto.

4.1 Testes durante o período de codificação

Ao longo do trabalho, procurou-se tornar cada vez mais simultâneas as atividades de validação e a de continuação da codificação do protótipo. Desse modo, assim que os módulos para recolher dados necessários ao iSPD para realizar a simulação eram implementados, preocupava-se em verificar se esses dados estavam corretos, executando-se testes estruturais, em que requisitos do próprio código eram utilizados nas verificações.

Dessa maneira, primeiramente faziam-se testes de unidade, e cada um dos parâmetros que deviam ser coletados eram verificados. Após isso, ocorriam testes de integração, verificando se, quando colocados para ser recolhidos em conjunto, os parâmetros ainda satisfaziam os valores que eram desejados. Por fim, fazia-se análise dos dados que compunham a classe armazenadora e comparava-os aos dados impressos na tela, terminando a primeira avaliação se todos eles batessem. Também realizava-se uma verificação manual do código GridSim e comparava-se com os dados apresentados por tal classe. Assim, se fosse observado que os dados apresentados eram os dados neces-

sários e os que deveriam ter sido lidos, concluía-se que as novas codificações estavam funcionando corretamente e poderia ser dada continuidade no projeto.

Por fim, após o término do período de codificação realizou-se a integração do protótipo à versão mais atual do simulador do GSPD e, com o intuito de testar se com a integração o simulador continuava a produzir dados consistentes, realizou-se a última série de validações. Esta será mais detalhada em 4.2.

4.2 Testes e validações realizados em comparação do iSPD com o GridSim

Após a integração do protótipo ao simulador e a fase de testes realizados em sua codificação, uma nova etapa de validação se iniciou. Nela, modelos GridSim eram simulados na própria ferramenta e comparados com modelos completos desse simulador, que eram interpretados pela ferramenta construída e geravam um modelo icônico para o iSPD o qual era simulado na ferramenta do GSPD.

Todos os testes foram realizados em um Intel Core i5-3210M 2.5 - 3.1GHz, com 4 GB de memória ram DDR 3 e HD de 500 GB. E, desse modo, eles serão apresentados a seguir.

4.2.1 Exemplo de conversão 1

Os testes comparativos seguem todos o mesmo procedimento. Neles, um código GridSim é interpretado na ferramenta criada ao longo desse trabalho e convertido em um modelo icônico para o iSPD. Após isso, tal modelo é simulado no *software* do GSPD e o tempo resultante é comparado com uma simulação do código inicial, em Java, feita no GridSim e também com o mesmo modelo de grade criado diretamente no iSPD. Desse modo, se os tempos forem semelhantes, significa que o interpretador está funcionando corretamente. A afirmação se comprova, pois grades iguais, em simuladores diferentes se comportaram de forma semelhante.

Dessa maneira, no primeiro teste comparativo, o código interpretado, do qual podemos ver um pequeno trecho na figura 4.1, vindo do GridSim, gerava um modelo, que é mostrado na figura 4.2. Nele, um mestre principal envia suas tarefas para serem executadas em uma grade com outro mestre e 3 escravos. Este modelo foi executado, e seu tempo de simulação foi comparado com o de um modelo criado diretamente no iSPD, que aparece em 4.3, e com o modelo original, executado no GridSim.

```
package network.example02;

/* Description: A simple program to demonstrate of how to use GridSim
 *              network extension package.
 *              This example shows how to create user and resource
 *              entities connected via a network topology, using link
 *              and router.
 *
 */

import gridsim.*;
import gridsim.net.*;
import java.util.*;


/**
public class NetEx02
{
    /**
    public static void main(String[] args)
    {

    /**
    private static GridResource createGridResource(String name,
                double baud_rate, double delay, int MTU)
    {
        System.out.println();
        System.out.println("Starting to create one Grid resource with " +
                "3 Machines");

        // Here are the steps needed to create a Grid resource:
        // 1. We need to create an object of MachineList to store one or more
        //    Machines
        MachineList mList = new MachineList();

        // 2. Create one Machine with its id, number of PEs and MIPS rating per PE
        //    In this example, we are using a resource from
        //    hpc420.hpcc.jp, AIST, Tokyo, Japan
        //    Note: these data are taken the from GridSim paper, page 25.
        //          In this example, all PEs has the same MIPS (Millions
        //          Instruction Per Second) Rating for a Machine.
        int mipsRating = 377;
        Machine machine = new Machine(0, 4, mipsRating);
        mList.add(machine);   // First Machine
```

Figura 4.1: Trecho do código do GridSim para o qual foi feita a conversão e simulação de teste

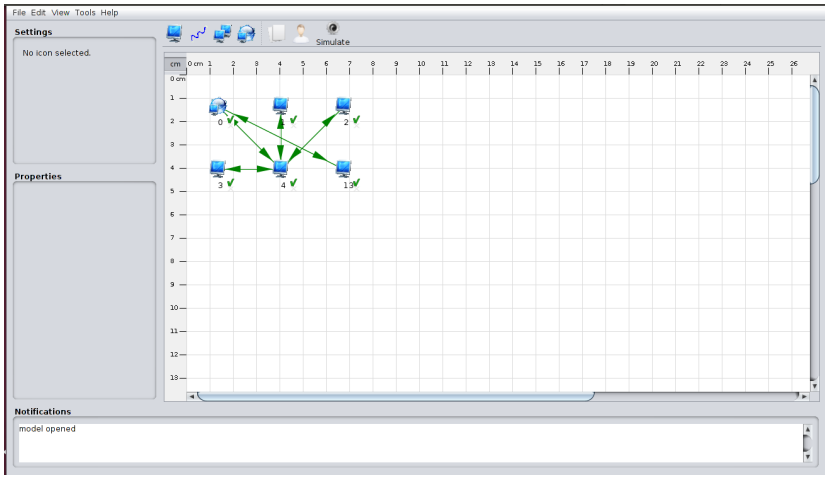

Figura 4.2: Modelo gerado pela conversão a partir do código do GridSim

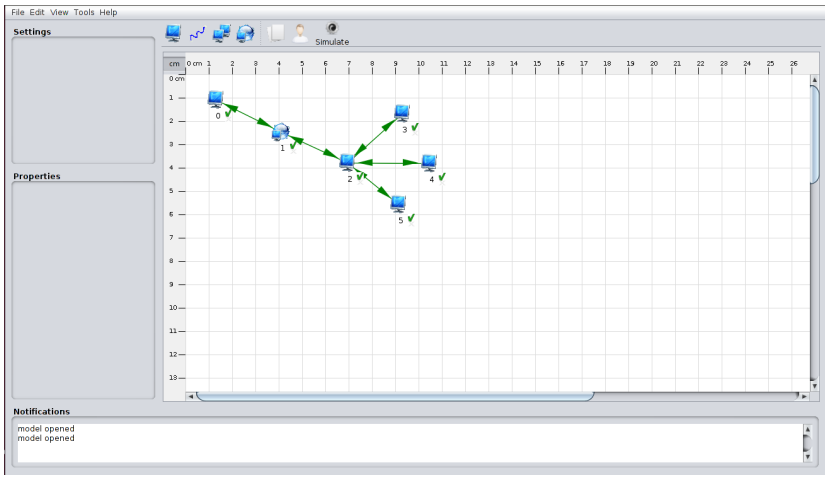

Figura 4.3: Modelo gerado diretamente no iSPD

Observa-se que apesar dos modelos não estarem dispostos no espaço de forma idêntica, ambos representam a mesma grade.

Desse modo, procurou-se representar, em forma de gráfico e também com uma tabela de tempos, os resultados obtidos com os primeiros testes. O gráfico aparece na figura 4.4. Nela, vemos os resultados das simulações para 5 testes, o primeiro com 1000, o segundo com 2000, o terceiro com 4000, o quarto com 8000 e o quinto com 16000 tarefas. Em tal gráfico, podemos perceber que o equilíbrio é tão grande que as

barras possuem quase todas a mesma altura, demonstrando assim que tanto o modelo convertido, quanto o criado diretamente no iSPD são compatíveis e estão validados junto ao simulador GridSim.

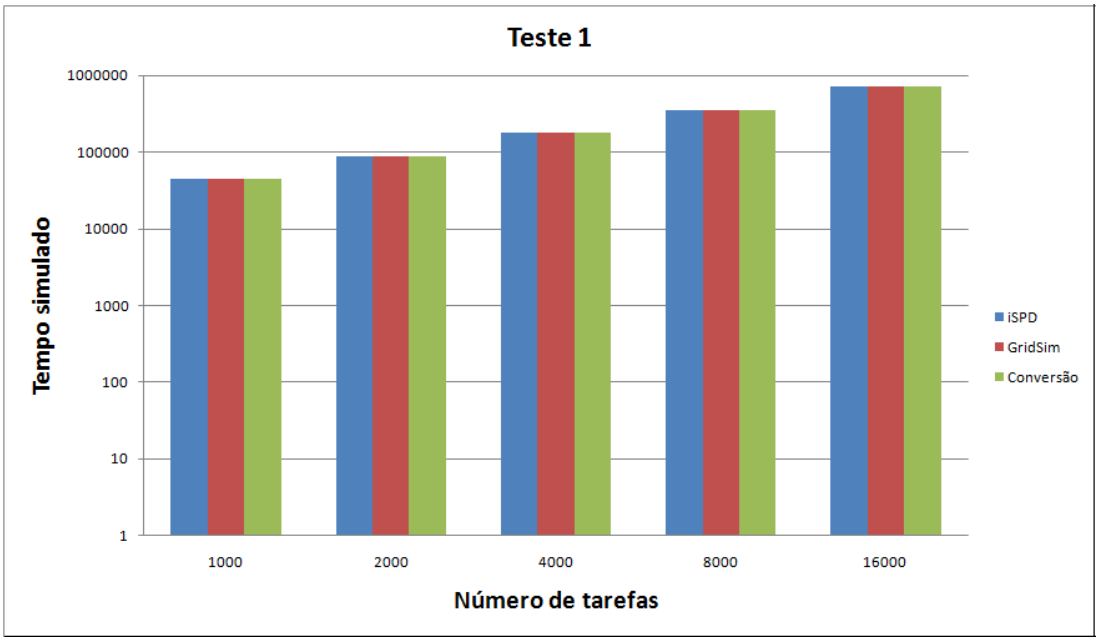

Figura 4.4: Gráfico comparativo entre os testes realizados

Tal semelhança entre os tempos simulados também pode ser percebido na tabela 4.1, em que cada linha indica o número de tarefas da execução do modelo, seguido de seus respectivos tempos.

Número de tarefas	iSPD	Conversão	GridSim
1000	44318	44419	44327
2000	88482	88769	88491
4000	176944	177391	176953
8000	353735	354757	353744
16000	707448	709368	707457

Tabela 4.1: Tempos teste 1

4.2.2 Exemplo de conversão 2

O teste comparativo seguiu o mesmo procedimento do anterior. Desse modo, o código interpretado vindo do GridSim gerava um modelo, o qual é mostrado na figura 4.5. Esse modelo é muito parecido com o da primeira conversão. Dessa forma, um mestre principal envia suas tarefas para serem executadas em uma grade, no entanto, esta possui 2 outros mestres com 5 escravos cada. Este modelo também foi executado, e seu tempo simulado foi comparado com o de um modelo criado diretamente no iSPD, como aparece na figura 4.6, e com o modelo original, executado no GridSim.

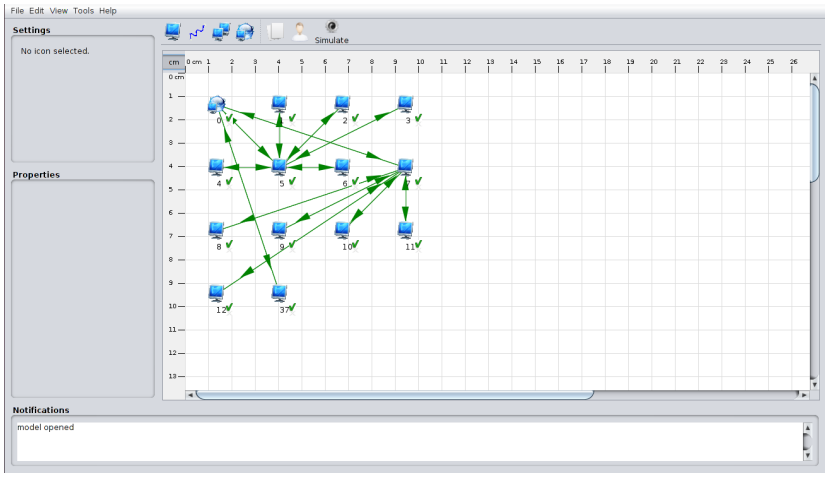

Figura 4.5: Modelo de grade gerado pela conversão a partir do código do GridSim

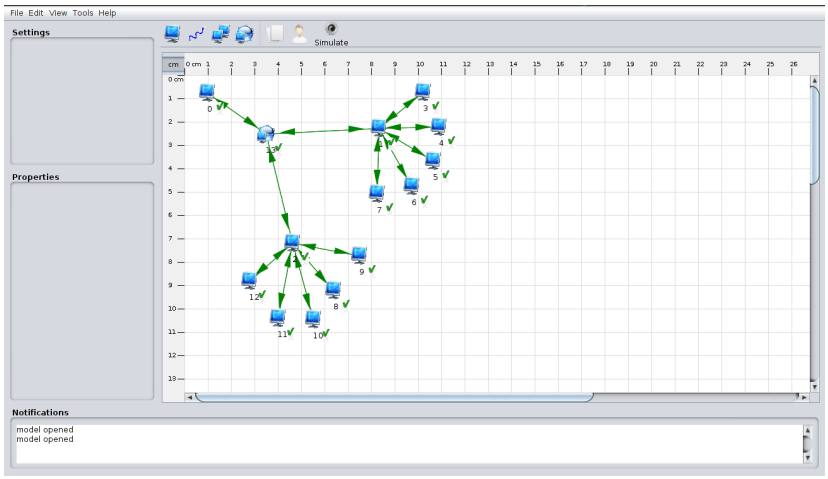

Figura 4.6: Modelo do teste 2 gerado diretamente no iSPD

Por fim, conforme feito no exemplo anterior, também foram representados todos os dados obtidos na forma de gráfico e tabela, nos mesmos padrões. E o equilíbrio foi novamente notado, como pode-se ver na tabela 4.2 e na figura 4.7.

Número de tarefas	iSPD	Conversão	GridSim
1000	13292	13490	13414
2000	26555	26875	26677
4000	53080	53645	53202
8000	106131	107186	106253
16000	212232	214269	212354

Tabela 4.2: Tempos teste 2

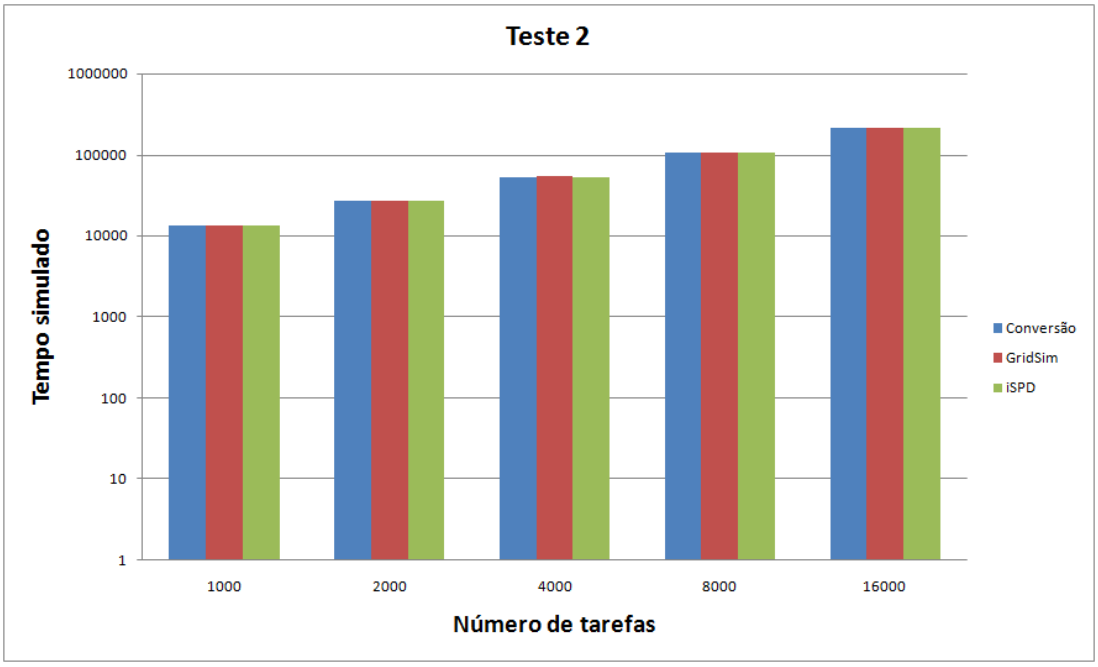

Figura 4.7: Gráfico comparativo entre os testes realizados

4.2.3 Exemplo de conversão 3

Por fim, no último teste de validação do interpretador, também foram seguidos os mesmos procedimentos. Nele, gerou-se o modelo mostrado na figura 4.8, o qual possui um mestre principal ligado a um roteador que se conecta a outros 2, os quais fazem uma nova ligação, cada qual, com um mestre e cada um desses possui 3 máquinas. O

modelo foi executado, e seu tempo simulado foi comparado com o de um modelo criado diretamente no iSPD (figura 4.9) e com o modelo original, executado no GridSim, conforme ocorreu também nos demais testes.

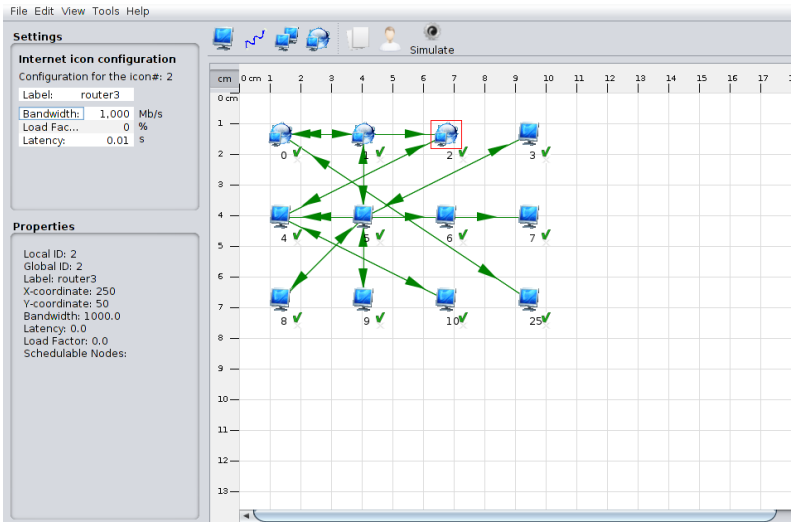

Figura 4.8: Modelo de grade gerado pela conversão a partir do código do GridSim

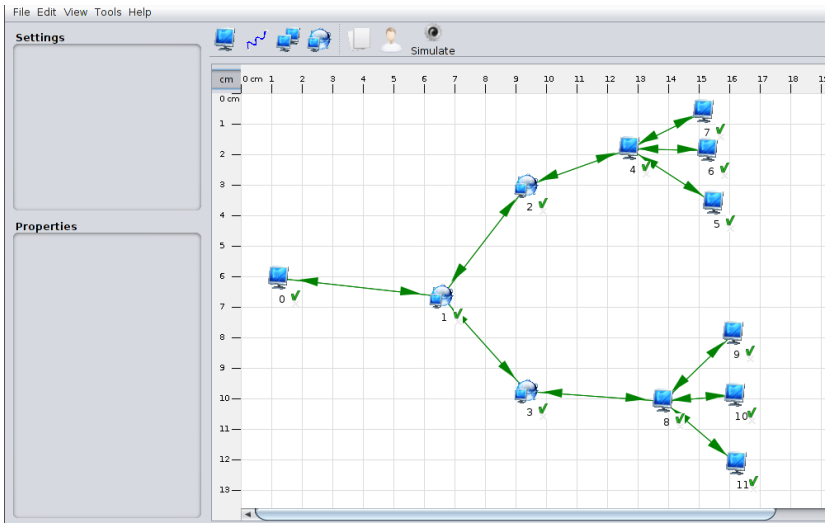

Figura 4.9: Modelo do teste 3 gerado diretamente no iSPD

E, desse modo, o mesmo equilíbrio notado nos testes anteriores também foi obtido aqui, conforme mostrado na figura 4.10 e na tabela 4.3, que seguem os mesmos padrões das já apresentadas até aqui.

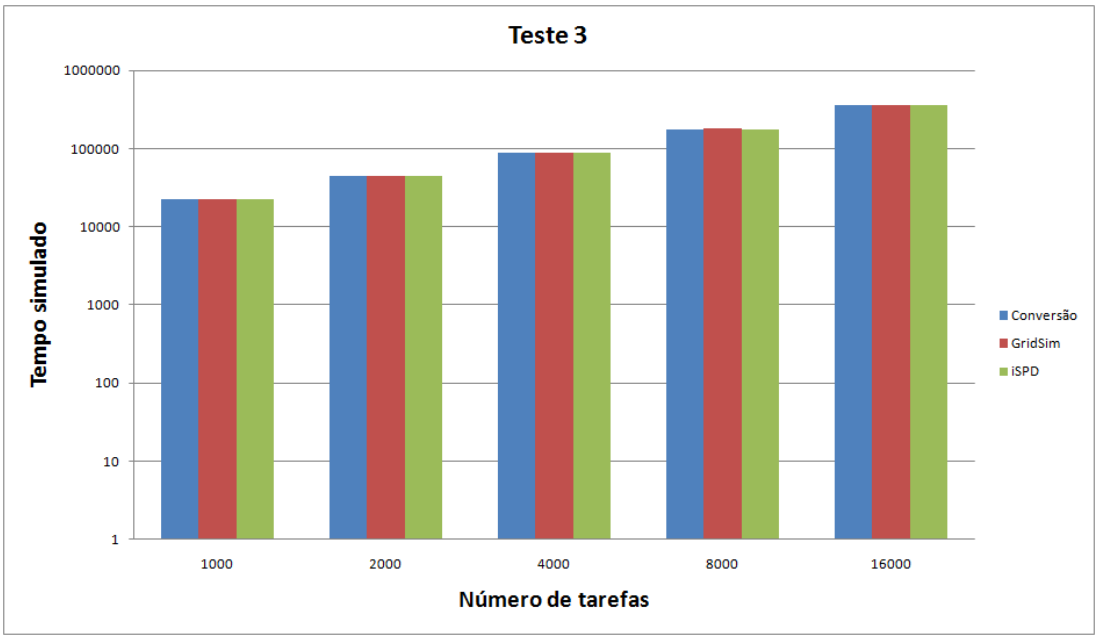

Figura 4.10: Gráfico comparativo entre os testes realizados

Número de tarefas	iSPD	Conversão	GridSim
1000	22268	22333	22238
2000	44417	44523	44367
4000	88581	89014	88551
8000	177043	177883	176993
16000	353834	355738	353804

Tabela 4.3: Tempos teste 3

Após os testes, pode-se observar que apesar da semelhança entre os dados obtidos, percebe-se, após todas as séries de testes, mais uma vantagem do iSPD com relação ao GridSim. O simulador de grades do GSPD, independente do tamanho das tarefas e do número delas, não possuía tempo de resposta elevado, tempo esse, na casa de segundos, enquanto o GridSim, teve crescimento exponencial no tempo de resposta, devido ao modo de implementação do modelo, sendo eles na casa de horas quando realizou-se o teste com 16000 tarefas. Tais diferenças podem ser vistas na figura 4.11 e na tabela 4.4,

em que mostra-se os tempos de resposta de cada uma das simulações do terceiro teste para o GridSim e o iSPD em segundos.

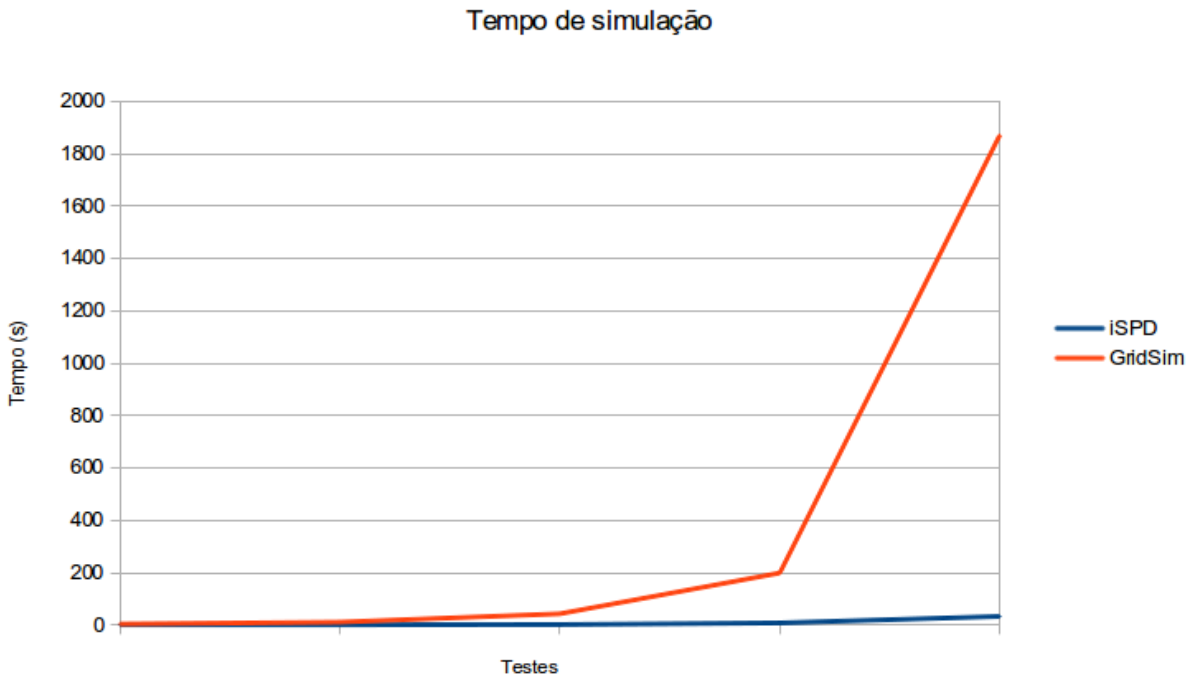

Figura 4.11: Gráfico comparativo entre os tempos de resposta dos testes realizados

Número do teste	iSPD	GridSim
1	0,8	4
2	1	11
3	2	43
4	8	199
5	33	1871

Tabela 4.4: Tempos de resposta

4.2.4 Exemplo de conversão com erros

Além das conversões de modelos que validaram o simulador, também foram realizadas algumas interpretações com erros para mostrar as mensagens de erro que são passadas aos usuários quando tal tipo de falha acontece.

Desse modo, nessa parte do capítulo, serão mostrados alguns trechos de código, seguidos de suas mensagens de alerta e também as mensagens de erro que são exibidas.

No primeiro caso, vemos abaixo o trecho de código de um *for*, em que a variável k não foi declarada:

```
for (int i = 0; i < k; i++)
{
    GridResource res = createGridResource(1, baud_rate, propDelay, mtu);
}
```

Dessa forma, o usuário recebe na tela a mensagem vista na figura 4.12.

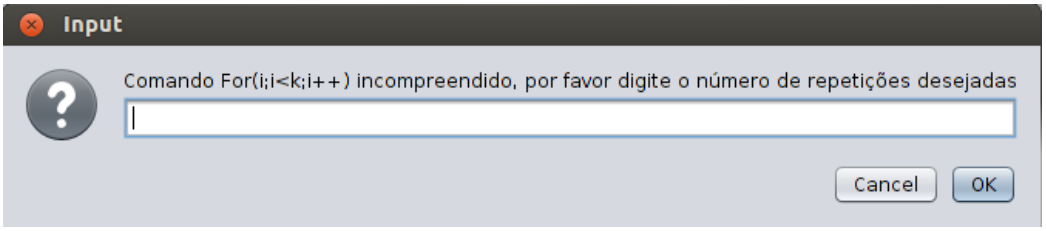

Figura 4.12: Janela de aviso exibida ao usuário quando um *for* com erros é encontrado

Para os demais erros encontrados, janelas seguindo o modelo da figura 4.13 são mostrados ao usuário, esperando que ele arrume o código que está sendo interpretado.

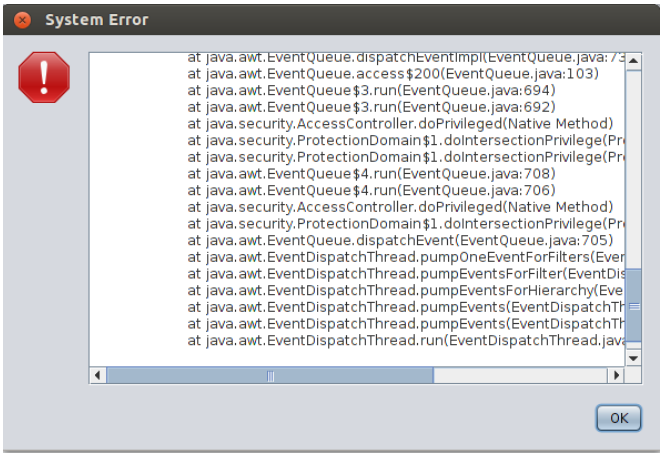

Figura 4.13: Janela de erro exibida ao usuário ao fazer-se qualquer tipo de leitura com problemas

4.3 Considerações finais

Dessa forma, os testes realizados e apresentados neste capítulo visam mostrar como funcionou a validação do interpretador e como seus resultados foram extremamente satisfatórios. Por fim, o capítulo 5 apresentará as conclusões tiradas com o trabalho e os próximos passos a serem tomados no projeto.

Capítulo 5

Conclusões

Ao longo do desenvolvimento deste trabalho, procurou-se apresentar uma visão geral sobre as ferramentas de simulação de grades computacionais mais importantes e, em seguida, uma visão sobre o simulador de grades do GSPD, o iSPD, sobre o qual seria inserida a nova função de interpretação de modelos externos. Após isso, buscou-se demonstrar todos os passos seguidos durante a implementação da função, assim como sua codificação e, por fim, sua validação.

5.1 Considerações finais

Notou-se que o iSPD tem como finalidade facilitar com que o usuário possa simular seu modelo de grades computacionais. Para isso, entre suas vantagens, aparecem uma interface icônica e a conversão entre modelos, podendo tal modelo vir do GridSim, no qual o trabalho foi feito, ou do SimGrid, e até mesmo exportar um modelo para esses.

Por fim, a nova função que foi integrada à ferramenta tem como principal vantagem a possibilidade de fazer com que um usuário leigo que possua em suas mãos um modelo GridSim possa simular seu modelo em outra ferramenta. E, desse modo, validar sua simulação.

5.2 Problemas encontrados

Entre os problemas encontrados enquanto o trabalho era realizado aparecem principalmente:

- Dificuldade na elaboração das gramáticas capazes de reconhecer uma linguagem de programação.
- Dificuldade de se fazer um interpretador para a linguagem Java, uma vez que essa é muito abrangente e complexa.
- Dificuldades na fase de integração do protótipo a versão final do iSPD.

5.3 Direções futuras

Como trabalhos futuros para o projeto em desenvolvimento pode-se ver:

- Inclusão da importação de modelos provindos do simulador de *Cloud Computing*, CloudSim (CLOUDSIM, 2013), o qual possui modelos de simulação muito parecidos com os do GridSim, uma vez que implementa uma função com finalidade específica de simular *cloud* para este.
- Aprimoração do interpretador GridSim, uma vez que Java é uma linguagem muito complexa e abrangente, o que dificulta a especificação de um interpretador completo em uma primeira versão.

5.4 Publicações

Este trabalho obteve algumas publicações durante seu período de desenvolvimento. Dentre elas estão:

1. *Desenvolvimento de Plataforma de Simulação de Grades Flexível e Orientada à Avaliação de Desempenho: Módulo de Interpretação de Modelos Externos e Módulo de Exportação de Modelos Icônicos. Apresentado na III Escola Regional de Alto Desempenho de São Paulo (ERAD)., 2012, Campinas-SP. Anais da III Escola Regional de Alto Desempenho de São Paulo (ERAD), 2012. p. 1-4.* (FURLANETTO et al., 2012)
2. *Desenvolvimento do módulo de Interpretação de Modelos externos 2013 Apresentado no Congresso de Iniciação Científica de 2013 (Cic - Unesp).*
3. *Desenvolvimento do módulo de Interpretação de Modelos externos 2012 Apresentado no Congresso de Iniciação Científica de 2012 (Cic - Unesp).*

Referências Bibliográficas

AOQUI, V. et al. Interpretador de modelos externos para simulador de grades computacionais. In: UNIVERSIDADE PRESBITERIANA MACKENZIE. *I Escola Regional de Alto Desempenho de São Paulo*. São Carlos, 2010. CD-ROM, p. 1–2.

BRICKS. Bricks: A performance evaluation system for grid computing scheduling algorithms. Disponível em <http://ninf.apgrid.org/bricks/>, acessado em 17 de dezembro de 2013. 2013.

BUYYA, R.; MURSHED, M. Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Pract. and Exper.*, v. 14, p. 1175–1220, 2002. ISSN 0038-0644.

CLOUDSIM. Cloudsim: A framework for modeling and simulation of cloud computing infrastructures and services. Disponível em <http://www.cloudbus.org/cloudsim/>, acessado em 17 de dezembro de 2013. 2013.

COULORIS, G.; DOLLIMORE, J.; KINDBERG, T. *Distributed Systems concepts and design*. 3rd. ed. [S.l.]: Addison Wesley, 2001.

DELAMARO, M. *Como construir um compilador utilizando ferramentas Java*. [S.l.]: Novatec, 2004.

FURLANETTO, C. G. et al. Desenvolvimento de plataforma de simulação de grades flexível e orientada à avaliação de desempenho: Módulo de interpretação de modelos externos e módulo de exportação de modelos icônicos. In: *III Escola Regional de Alto Desempenho de São Paulo*. Campinas: [s.n.], 2012. CD-ROM, p. 1–4.

GRIDSIM. Gridsim: A grid simulation toolkit for resource modelling and application scheduling for parallel and distributed computing. Disponível em <http://www.buyya.com/GridSim/>, acessado em 17 de dezembro de 2013. 2013.

GSPD. Gspd's homepage. Disponível em <http://www.dcce.ibilce.unesp.br/spd/>, acessado em 17 de dezembro de 2013. 2013.

GUERRA, A. I. et al. Plataforma de simulação de grades computacionais: Interface icônica. In: UNIVERSIDADE PRESBITERIANA MACKENZIE. *I Escola Regional de Alto Desempenho de São Paulo*. [S.l.], 2010. CD-ROM, p. 1–2.

JAVACC. Java compiler compiler(javacc) - the java parser generator. Disponível em <https://java.net/projects/javacc/downloads?page=2&path%5B%5D=contrib&path%5B%5D=grammars&theme=java.net>, acessado em 17 de dezembro de 2013. 2013.

JAVACC. Java compiler compiler(javacc) - the java parser generator. Disponível em <https://javacc.java.net/>, acessado em 17 de dezembro de 2013. 2013.

MANACERO, A. et al. ispd: an iconic-based modeling simulator for distributed grids. In: *Annals of 45th Annual Simulation Symposium*. Orlando, USA: [s.n.], 2012. (ANSS12, CDROM), p. 1–8.

MENEZES, D. *Geração de Algoritmos de Escalonamento para Simulação de Grades Computacionais*. Dissertação (Mestrado) — Instituto de Biociências, Letras e Ciências Exatas (IBILCE), Universidade Estadual Paulista "Júlio de Mesquita Filho"(UNESP), Câmpus de São José do Rio Preto, São José do Rio Preto, 2012.

MENEZES, D. et al. Scheduler simulation using ispd, an iconic-based computer grid simulator. In: *Computers and Communications (ISCC), 2012 IEEE Symposium on*. [S.l.: s.n.], 2012. p. 637 –642. ISSN 1530-1346.

MENEZES, P. B. *Linguagens formais e autômatos*. [S.l.]: Sagra Luzzato, 1998.

OLIVEIRA, L. J. de. *Comparação de ferramentas de simulação de grades computacionais*. Tese (Monografia) — Instituto de Biociências, Letras e Ciências Exatas (IBILCE), Universidade Estadual Paulista "Júlio de Mesquita Filho"(UNESP), Câmpus de São José do Rio Preto, São José do Rio Preto, 2007.

OPTORSIM. Simulating data access optimization algorithms - optorsim. Disponível em <http://optorsim.sourceforge.net/>, acessado em 17 de dezembro de 2013. 2012.

SEBESTA, R. W. *Conceitos de linguagens de programação*. [S.l.]: Bookman, 2003.

SETHI, R.; ULLMAN, J. D.; LAM, M. S. *Compiladores: princípios, técnicas e ferramentas*. [S.l.]: Pearson Addison Wesley, 2008.

SIMGRID. Welcome to the simgrid project! Disponível em <http://SimGrid.gforge.inria.fr/>, acessado em 17 de dezembro de 2013. 2013.

Apêndice A

Gramática Regular

<Comentários> ::= “//” <qualquer_caracter> | ”/*” <qualquer_caracter> “*/”

<ABSTRACT> ::= “abstract”

<BOOLEAN> ::= “boolean”

<BREAK> ::= “break”

<BYTE> ::= “byte”

<CASE> ::= “case”

<CATCH> ::= “catch”

<CHAR> ::= “char”

<CLASS> ::= “class”

<CONST> ::= “const”

<CONTINUE> ::= “continue”

<DEFAULT> ::= “default”

<DO> ::= “do”

<DOUBLE> ::= “double”

<ELSE> ::= “else”

<ENUM> ::= “enum”

<EXTENDS> ::= “extends”

<FALSE> ::= “false”

<FINAL> ::= “final”

<FINALLY> ::= “finally”

<FLOAT> ::= “float”

<FOR> ::= “for”

<GOTO> ::= “goto”

<IF> ::= “if”

<IMPLEMENTS> ::= “implements”

<IMPORT> ::= “import”

<INSTANCEOF> ::= “instanceof”

<INT> ::= “int”

<INTERFACE> ::= “interface”

<LONG> ::= “long”

<NATIVE> ::= “native”

<NEW> ::= “new”

<NULL> ::= “null”

<PACKAGE> ::= “package”

<PRIVATE> ::= “private”

<PROTECTED> ::= “protected”

<PUBLIC> ::= “public”

<RETURN> ::= “return”

<SHORT> ::= “short”

<STATIC> ::= “static”

<STRICTFP> ::= “strictfp”

<SUPER> ::= “super”

<SWITCH> ::= “switch”

<SYNCHRONIZED> ::= “synchronized”

<THIS> ::= “this”

<THROW> ::= “throw”

<THROWS> ::= “throws”

<TRANSIENT> ::= “transient”

<TRUE> ::= “true”

<TRY> ::= “try”

<VOID> ::= “void”

<VOLATILE> ::= “volatile”

<WHILE> ::= “while”

<DECIMAL> ::= [“1-“9”] ([“0-“9”])*

<HEX_LITERAL> ::= “0”[“x”,“X”] ([“0-“9”,“a-“f”,“A-“F”])+

<OCTAL_LITERAL> ::= “0”([“0-“7”])*

<PONTO_FLUTUANTE> ::= ([“0-“9”])+ ”.”([“0-“9”])* (<EXPONENT>)? ([“f”,“F”,“d”,“D”])? | “.”([“0-“9”])+ (<EXPONENT>)? ([“f”,“F”,“d”,“D”])? | ([“0-“9”])+ <EXPONENT> ([“f”,“F”,“d”,“D”])? | ([“0-“9”])+ (<EXPONENT>)? [“f”,“F”,“d”,“D”]

<ABRE_PAR> ::= “(“

<FECHA_PAR> ::= “)”

<ABRE_CHAVE> ::= “{“

<FECHA_CHAVE> ::= “}”

<ABRE_COLCHETE> ::= “[“

<FECHA_COLCHETE> ::= “]”

<PONTO_VIRGULA> ::= “;”

<VIRGULA> ::= “,”

<PONTO> ::= “.”

<ARROBA> ::= “@”

<IGUAL> ::= “=”

<MENOR> ::= “<”

<MAIOR> ::= “>”

<EXCLAMAÇAO> ::= “!”

<TIL> ::= “~”

<INTERROGAÇAO> ::= “?”

<DOIS_PONTOS> ::= “:”

<IGUALDADE> ::= “==”

<MENOR_IGUAL> ::= “<=”

<MAIOR_IGUAL> ::= “>=”

<DIFERENTE> ::= “!=”

<OU> ::= “||”

<E> ::= “&&”

<INCREMENTO> ::= “++”

<DECREMENTO> ::= “–”

<MAIS> ::= “+”

<MENOS> ::= “-”

<VEZES> ::= “*”

<DIVIDIDO> ::= “/”

<E_COMERCIAL> ::= “&”

<PIPE> ::= “|”

<XOR> ::= “^”

<PORCENTO> ::= ”%”

<MENOR_MENOR> ::= “< <”

<MAIS_IGUAL> ::= ”+=”

<MENOS_IGUAL> ::= “-=”

<VEZES_IGUAL> ::= “*=”

<DIVIDIDO_IGUAL> ::= “/ =”

<E_IGUAL> ::= “& =”

<PIPE_IGUAL> ::= “| =”

<XOR_IGUAL> ::= “^=”

<PORCENTO_IGUAL> ::= “%=”

<MENOR_MENOR_IGUAL> ::= “< <=”

<MAIOR_MAIOR_IGUAL> ::= “> >=”

<MAIOR_MAIOR_MAIOR_IGUAL> ::= “> > > =”

<TRES_PONTOS> ::= “...”

Apêndice B

Gramática Livre de Contexto

<unidade_compilaçao> ::= [<declaraçao_pacote>] (<importar_declaraçao>)* (<declaraçao_tipo>)* <EOF>

<declaracao_pacote> ::= <PACKAGE> <nome> <PONTO_VIRGULA>

<importar_declaraçao> ::= <IMPORT> [<STATIC>] <nome> [<PONTO> "*"] <PONTO_VIRGULA>

<declaraçao_tipo> ::= <PONTO_VIRGULA> | <classe_interface_declaraçao> | <enum_declaraçao> | <tipo_declaraçao_anotaçao>

<classe_interface_declaraçao> ::= (<CLASS> | <INTERFACE>) <identificador> [<tipo_parametros>] [<extendslist>] [<implementslist>] <classe_interface_corpo>

<extendslist> ::= <EXTENDS> <classe_interface_tipo> (<VIRGULA> <classe_interface_tipo>)*

<implementslist> ::= <IMPLEMENTS> <classe_interface_tipo> (<VIRGULA> <classe_interface_tipo>)*

<enum_declaraçao> ::= <ENUM> <identificador> [<implementslist>] <corpo_enum>

<corpo_enum> ::= <ABRE_CHAVE> <enumconstante> (<VIRGULA> enumconstant())* [<PONTO _VIRGULA> (classe_declaraçao_interface_corpo>)*] <FECHA_CHAVE>

<enumconstante> ::= <identificador> [<argumentos>] [<classe_interface_corpo>]

<tipo_parametros>::= <MENOR> <tipo_parametro> (<VIRGULA> <tipo_parametro>)*
<MAIOR>

<tipo_parametro> ::= <identificador> [<tipo_amarrado_variavel>]

<tipo_amarrado_variavel> ::= <EXTENDS> <classe_interface_tipo> (<E_COMERCIAL>
<classe_ interface_tipo>)*

<classe_interface_corpo> ::= <ABRE_CHAVE> (<classe_declaraçao_interface_corpo>)*
<FECHA_ CHAVE>

<classe_declaraçao_interface_corpo> ::= <Inicializador> | (<classe_interface_declaraçao>
| <enum_ declaraçao> | [<tipo_ parametros>] <identificador> <ABRE_PAR> <cons-
trutor_declaraçao> | <tipo> <identificador> (<ABRE_COLCHETE> <FECHA_COLCHETE>)*
(<VIRGULA> | <IGUAL> | <PONTO_ VIRGULA>) <campo_declaraçao> | <me-
todo_declaraçao>) | <PONTO_ VIRGULA>

<campo_declaraçao> ::= <tipo> <declarador_variavel>(<VIRGULA> <declarador_variavel>)*
<PONTO_VIRGULA>

<declarador_variavel> ::= <declarador_variavel_id> [<IGUAL> <inicializador_variavel>]

<declarador_variavel_id> ::= <identificador> (<ABRE_COLCHETE> <FECHA_COLCHETE>)*

<inicializador_variavel> ::= <inicializador_vetor> | <expressao>

<inicializador_vetor> ::= <ABRE_CHAVE> [<inicializador_variavel> (<VIRGULA>
<inicializador_variavel>)*] [<VIRGULA>] <FECHA_CHAVE>

<metodo_declaraçao> ::= [<tipo_parametros>] <tipo_resultado> <declarador_metodo>
[<THROWS> <Nome_Lista>](<Bloco> | <PONTO_VIRGULA>)

<declarador_metodo> ::= <identificador> <parametros_formais> (<ABRE_COLCHETE>

<FECHA_ COLCHETE>)*

<parametros_formais> ::= <ABRE_PAR> [<parametro_formal>) (<VIRGULA> <parametro_formal>)*] <FECHA_PAR>

<parametro_formal> ::= [<FINAL>] <tipo> [<TRES_PONTOS>] <declarador_variavel_id>

<declaraçao_construtor> ::= [<tipo_parametros>] <identificador> <parametros_formais> [< THROWS> <Nome_Lista>] <ABRE_CHAVE> [<invocaçao_explicita_construtor>](<estado_bloco>)* <FECHA_CHAVE>

<invocaçao_explicita_construtor> ::= (<THIS> <argumentos> <PONTO_VIRGULA>) | [<expressao_primaria> <PONTO>] <SUPER> <argumentos> <PONTO_VIRGULA>

<Inicializador> ::= [<STATIC>] <Bloco>

<tipo> ::= <tipo_referencia> | <tipo_primitivo>

<tipo_referencia> ::= <tipo_primitivo>(<ABRE_COLCHETE> <FECHA_COLCHETE>)+ | (<TipoClasseouInterface>) (<ABRE_COLCHETE><FECHA_COLCHETE>)*

<TipoClasseouInterface> ::= <identificador> [<tipo_argumentos>] (<PONTO><identificador> [<tipo_argumentos>])*

<tipo_argumentos> ::= <MENOR><tipo_argumento> (<VIRGULA> tipo_argumento)* <MAIOR>

<tipo_argumento> ::= <tipo_referencia> | <INTERROGAÇAO> [<WildcardBounds>]

<WildcardBounds> ::= <EXTENDS> <tipo_referencia> | <SUPER> <tipo_referencia>

<tipo_primitivo> ::= <BOOLEAN> | <CHAR> | <BYTE> | <SHORT> | <INT> | <LONG> | <FLOAT>| <DOUBLE>

<tipo_resultado> ::= <VOID> | <tipo>

<nome> ::= <identificador> (<PONTO> <identificador>)*

<Nome_Lista> ::= <nome> (<VIRGULA> <nome>)*

<expressao> ::= <expressao_condicional> [<operador_atribuiçao <expressao>]

<operador_atribuiçao> ::= <IGUAL> | <VEZES_IGUAL> | <DIVIDIDO_IGUAL> | <PORCENTO_IGUAL> | <MAIS_IGUAL> | <MENOS_IGUAL> | <MENOR_MENOR_IGUAL> | <MAIOR_MAIOR_IGUAL> | <MAIOR_MAIOR_MAIOR_IGUAL> | <E_IGUAL> | <XOR_IGUAL> | <PIPE_IGUAL>

<expressao_condicional> ::= <condicional_ou_expressao> [<INTERROGAÇAO> <expressao> <DOIS_PONTOS> <expressao>]

<condicional_ou_expressao> ::= <condicional_e_expressao> (<OU> <condicional_e_expressao>)*

<condicional_e_expressao> ::= <inclusivo_ou_expressao> (<E> <inclusivo_ou_expressao>)*

<inclusivo_ou_expressao> ::= <exclusivo_ou_expressao> (<PIPE> <exclusivo_ou_expressao>)*

<exclusivo_ou_expressao> ::= <e_expressao> (<XOR> <e_expressao>)*

<e_expressao> ::= <expressao_igualdade> (<E_COMERCIAL> <expressao_igualdade>)*

<expressao_igualdade> ::= <instancia_de_expressao> ((<IGUALDADE> | <DIFERENTE>) <instancia_de_expressao>)*

<instancia_de_expressao> ::= <expressao_relacional> [<INSTANCEOF> <tipo>]

<expressao_relacional> ::= <expressao_mudanca> ((<MENOR> | <MAIOR> | <MENOR_IGUAL> | <MAIOR_IGUAL>) <expressao_mudanca>)*

<expressao_mudanca> ::= <expressao_aditiva> ((<MENOR_MENOR> | <RSIGNEDSHIFT> | <RUNSIGNEDSHIFT>) <expressao_aditiva>)*

<expressao_aditiva> ::= <expressao_multiplicativa> ((<MAIS> | <MENOS>) <expressao_ multiplicativa>)*

<expressao_multiplicativa> ::= <expressao_unaria> ((<VEZES> | <DIVIDIDO> | <PORCENTO>) <expressao_unaria>)*

<expressao_unaria> ::= (<MAIS> | <MENOS>) <expressao_unaria> | <expressao_pre_incremento> | <expressao_pre_decremento> | <expressao_unaria_nao_mais_menos>

<expressao_pre_incremento> ::= <INCREMENTO> <expressao_primaria>

<expressao_pre_decremento> ::= <DECREMENTO> <expressao_primaria>

<expressao_unaria_nao_mais_menos> ::= (<TIL> | <EXCLAMAÇAO>) <expressao_unaria> | <CastLookahead> <expressao_cast> | <expressao_posfixa>

<CastLookahead> ::= <ABRE_PAR> (<tipo_primitivo> | <tipo>(<ABRE_COLCHETE> <FECHA_ COLCHETE>| <tipo> <FECHA_PAR> (<TIL> |<EXCLAMAÇAO> <ABRE_PAR> | <identificador> | <THIS> | <SUPER> | <NEW> | <Literal>))

<expressao_posfixa> ::= <expressao_primaria> [<INCREMENTO> | <DECREMENTO>]

<expressao_cast> ::= <ABRE_PAR> <tipo_primitivo> <ABRE_PAR> <tipo> <FECHA_PAR>(<expressao_unaria> | <expressao_unaria_nao_mais_menos>)

<expressao_primaria> ::= <prefixo_primario> (<sufixo_primario>)*

<membro_seletor> ::= <PONTO> <tipo_argumentos> <identificador>

<prefixo_primario> ::= <Literal> | <THIS> | <SUPER> <PONTO> <identificador> | <ABRE_ PAR> <expressao> <FECHA_PAR> | <expressao_alocaçao> | <tipo_resultado> <PONTO> <CLASS> | <nome>

<sufixo_primario> ::= (<PONTO> (<THIS> | <expressao_alocaçao> | <identificador>)) | <membro_seletor> | <ABRE_COLCHETE> <expressao> <FECHA_COLCHETE> | <argumentos>

<literal> ::= <literal_inteiro> | <literal_float> | <literal_caracter> | <literal_string> | <literal_boleano> | <literal_null>

<literal_boleano> ::= “true” | “false”

<literal_null> ::= ”null”

<argumentos> ::= <ABRE_PAR> [<lista_argumentos>] <FECHA_PAR>

<lista_argumentos> ::= <expressao> (<VIRGULA> <expressao>)*

<expressao_alocaçao> ::= (<NEW>(<tipo_primitivo> <vetor_dimensao_inicio> | <TipoClasseouInterface> [tipo_argumentos>])) (<vetor_dimensao_inicio> | <argumentos> [<classe_interface_ corpo>])

<vetor_dimensao_inicio> ::= (<abre_colchete> <expressao> <fecha_colchete>)+ (<abre_colchete> <fecha_colchete>)* | (<abre_colchete> <fecha_colchete>)+ <inicializador_vetor>

<declaraçao> ::= <declaraçao_rotulada> | <confirma_declaraçao>| <bloco> | <declaraçao_vazia> | <declaraçao_expressao> <PONTO_VIRGULA> | <declaraçao_switch> | <declaraçao_if> | <declaraçao _while> | <declaraçao_do> | <declaraçao_for> | <declaraçao_break> | <declaraçao_continue> | <declaraçao_return> | <declaraçao_throw | <declaraçao_synchronized> | <declaraçao_try>

<confirma_declaraçao> ::= “assert” <expressao> [<DOIS_PONTOS> <expressao>] <PONTO_ VIRGULA>

<declaraçao_rotulada> ::= <identificador> <DOIS_PONTOS> <declaraçao>

<bloco> ::= <ABRE_CHAVE> (<declaraçao_bloco>)* <FECHA_CHAVE>

<declaraçao_bloco> ::= ([<FINAL>] <tipo> <identificador>) <declaraçao_variavel_local> <PONTO_ VIRGULA> | <rotulo> | <classe_interface_declaraçao>

<declaraçao_variavel_local> ::= [<FINAL>] <tipo> <declarador_variavel> (<VIRGULA> <declarador_variavel>)*

<declaraçao_vazia> ::= <PONTO_VIRGULA>

<declaraçao_expressao> ::= <expressao_pre_incremento> |<expressao_pre_decremento> | <expressao_primaria>[<INCREMENTO> | <DECREMENTO> |<operador_atribuiçao> <expressao>]

<declaraçao_switch> ::= <SWITCH> <ABRE_PAR> <expressao> <FECHA_PAR> <ABRE_ CHAVE> (<rotulo_switch> (<declaraçao_bloco>)*)* <FECHA_CHAVE>

<rotulo_switch> ::= <CASE> <expressao> <DOIS_PONTOS> | <DEFAULT> <DOIS_ PONTOS>

<declaraçao_if> ::= <IF> <ABRE_PAR> <expressao> <FECHA_PAR> <declaraçao> [<ELSE> <declaraçao>]

<declaraçao_while> ::= <WHILE> <ABRE_PAR> <expressao> <FECHA_PAR> <declaraçao>

<declaraçao_do> ::= <DO> <declaraçao> <WHILE> <ABRE_PAR> <expressao> <FECHA_PAR> <PONTO_VIRGULA>

<declaraçao_for> ::= <FOR> <ABRE_PAR> (<tipo> <identificador> <DOIS_PONTOS> <expressao> | [<inicia_for>] <PONTO_VIRGULA> [<expressao>] <PONTO_VIRGULA> [<atualiza_for>])<FECHA_PAR> <declaraçao>

<inicia_for> ::= [<FINAL>] <tipo> <identificador> <declaraçao_variavel_local> |<declaraçao_lista _expressao>

<declaraçao_lista_expressao> ::= <declaraçao_expressao> (<VIRGULA> <declara-

çao_expressao>)*

<atualiza_for> ::= <declaraçao_lista_expressao>

<declaraçao_break> ::= <BREAK> [<identificador>] <PONTO_VIRGULA>

<declaraçao_continue> ::= <CONTINUE> [<identificador>] <PONTO_VIRGULA>

<declaraçao_return> ::= <RETURN> [<expressao>] <PONTO_VIRGULA>

<declaraçao_throw> ::= <THROW> <expressao> <PONTO_VIRGULA>

<declaraçao_synchronized> ::= <SYNCHRONIZED> <ABRE_PAR> <expressao> <FECHA_PAR> <bloco>

<declaraçao_try> ::= <TRY> <bloco> (<CATCH> <ABRE_PAR> <parametro_formal> <FECHA_PAR> <bloco>)* [<FINALLY> <bloco>]

<anotaçao> ::= <ARROBA> <nome> <ABRE_PAR> (<identificador> <IGUAL> | <FECHA_PAR>) <anotaçao_normal> | <ARROBA> <nome> <ABRE_PAR> <membro_simples_ anotaçao> | <marcador_anotaçao>

<anotaçao_normal> ::= <ARROBA> <nome> <ABRE_PAR> [<valores_membros_pares>] <FECHA _PAR>

<marcador_anotaçao> ::= <ARROBA> <nome>

<membro_simples_anotaçao> ::= <ARROBA> <nome> <ABRE_PAR> <valor_membro> <FECHA _PAR>

<valor_membros_pares> ::= <valor_membro_par> (<VIRGULA> <valor_membro_par>)*

<valor_membro_par> ::= <identificador> <IGUAL> <valor_membro>

<valor_membro> ::= <anotaçao> | <valor_membro_vetor_inicializador> | <expressao_condicional>

<valor_membro_vetor_inicializador> ::= <ABRE_CHAVE> <membro_valor> (<VIRGULA> <membro_valor>)* [<VIRGULA>] <FECHA_CHAVE>

<tipo_declaraçao_anotaçao> ::= <ARROBA> <INTERFACE> <identificador> <anotaçao_tipo_ corpo>

<anotaçao_tipo_corpo> ::= <ABRE_CHAVE> (<declaraçao_anotaçao_tipo_membro>)* <FECHA _CHAVE>

<declaraçao_anotaçao_tipo_membro> ::= <tipo> <identificador> <ABRE_PAR> <FECHA_PAR> [valor_default] <PONTO_VIRGULA> | <classe_interface_declaraçao> | <enum_declaraçao> | <declaraçao_anotaçao_tipo> | <campo_declaraçao>) | (<PONTO_VIRGULA>)

<valor_default> ::= <DEFAULT> <valor_membro>

<RUNSIGNEDSHIFT> ::= <MAIOR> <MAIOR> <MAIOR>

<RSIGNEDSHIFT> ::= <MAIOR><MAIOR>