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

Rafael de Souza Stabile

Módulo do iSPD para a Exportação de Modelos para o GridSim

São José do Rio Preto
2013

Rafael de Souza Stabile

Módulo do iSPD para a Exportação de Modelos para o GridSim

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

Rafael de Souza Stabile

Módulo do iSPD para a Exportação de Modelos para o GridSim

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. Rafael de Souza Stabile

Banca Avaliadora:

Prof.ª Dr.ª Luciana Pavani de Paula Bueno

Prof. Dr. Norian Marranghello

São José do Rio Preto
2013

Aos meus queridos pais Sandra e Luiz Antonio

Ao meu irmão Gabriel e à minha avó Aparecida

A todos os meus familiares, tios, tias e primos

”A felicidade não se resume na ausência de problemas, mas sim na sua capacidade de lidar com eles.”

Albert Einstein

Agradecimentos

Agradeço primeiramente a Deus e à minha família, que é a minha base e que sempre me apoiou em todas as minhas decisões, nunca medindo esforços para me ajudar mesmo nem sempre podendo. Mas principalmente à meus pais, Sandra e Luiz Antônio, que são minha inspiração, se um dia for a metade do que eles hoje são, estarei realizado. À minha mãe, minha amiga, que sempre se preocupa comigo, me insentivando, agradeço pelo amor e carinho dado a mim e pelas horas de trabalho incansavéis até tarde da noite para me proporcionar um conforto nos meus estudos. E à meu pai, meu exemplo de vida a ser seguido, sempre me ajudando em tudo, dedicando todo seu tempo a mim, agradeço por todos seus ensinamentos, amor e companherismo, devo muito a ele.

Agradeço à meu irmão Gabriel, pelas grandes horas que passamos justos, gosto muito desse gordo. À minha avó Aparecida, desde pequeno cuidando de mim, espero poder retribuir um dia o que ela fez por mim. Agradeço a todos os meus tios e tias, mas principalmente à Marilda, Eva, Tereza, Lurdinha, Claudio, Douglas, Rosane, Carlos, Franciane e a todos os outros, muito obrigado por todo o seu amor e carinho, nunca vou me esquecer. E a meus primos, tenho certeza quando digo que ninguem tem primos melhores que os meus, somos amigos e companheiros: Samanta, Pâmela, Tiago, Natália, Helinha, Andreza, Jaqueline, Lais, Carol, Victor, Claudinho, Gustavo, Pati, Bia, Tais, Arthur e todos os outros. Além disso, agradeço a familiares meus importantes na minha vida, Rita, Angela, Mariely, Tia Neuza, Toco, Ana, Erica e outros, obrigado pela enorme ajuda e acolhimento sem nem mesmo me conhecerem, eles poderão sempre contar comigo.

Agradeço a meus orientadores Profª. Renata Spolon Lobato e ao Prof. Aleardo Manacero Jr. pelos ensinamentos e apoio dado a mim. Além de professores, guardarei a recordação de grandes amigos para a minha vida inteira. Assim, agradeço à todos os professores do BCC e à UNESP pela ajuda financeira através da bolsa PIBIC/Reitoria. Além de todo o GSPD, e amigos do grupo.

Por último, agradeço a meus amigos feitos durante o curso: Danilo(Ex gordo), Gabriel C., Willian, Heitor, Matheus, Gustavo, Leo, Saraiva, Leandro, Diogo (Gordo), Japa, Mari, Cássio, Daniel, Mano, Jonh, Jão Marcos, Jão Matheus, Marina, Barbara, Denison, Turquinho, Mario, Maycon, Lais, Veronica e outros, nunca me esquecerei de nunhum deles, eles não fazem ideia o quão são importantes para mim. E aos meus amigos da pensão Afonso, Matheus, Igor, Zão, Brunão, Vinicius e Alan.

Resumo

Hoje há uma crescente necessidade por sistemas computacionais de Alto Desempenho. Dentro desse meio, pode-se destacar o uso de Grades Computacionais, e as formas de avaliação de desempenho das mesmas. Uma forma de se avaliar a performance deste tipo de sistema é usar a simulação. Porém, as ferramentas existentes que simulam grades não proveem ao usuário uma facilidade de manuseio, já que muitas usam a programação para realizar a modelagem de seu sistema. Deste modo, o iSPD foi construído para sanar esta deficiência e, além disso, poder analisar modelos de outros simuladores, bem como exportar seu próprio modelo. Assim, o objetivo deste trabalho é apresentar mais um módulo do iSPD, o módulo de exportação de modelos para o GridSim. O que envolveu analisar o modelo vindo do iSPD e então converter para o GridSim, através de uma gramática de conversão de modelos. Então, foi possível comparar o resultado de simulações dessas duas ferramentas para validar esta conversão. Tal comparação foi feita através de gráficos e dados númericos, com os resultados das simulações de diferentes modelos, com variações de carga, que foram modelados no iSPD e convertidos para o GridSim através do módulo de exportação apresentado nesta monografia. Nos resultados das simulações apareceram diferenças baixas em relação ao resultado das simulações dos diferentes modelos, validando assim o processo de exportação de modelos.

Palavras-chave: Simulação. Grades Computacionais. Exportação

Abstract

Today there is an increasing need for High Performance computer systems. Then, we can highlight the use of Computational Grids, and ways of evaluating their performance. One way to evaluate the performance of this type of system is using the simulation. However, existing tools that simulate grids do not provide an easy way to do it, since many use the programming to perform the modeling of a system. Thus, the iSPD was built to remedy this deficiency and also be able to interpret models from other simulators as well as exporting its own model. The objective of this work is to present another iSPD module, the module for exporting models to GridSim. What involves interpreting the model coming from iSPD and converts it to GridSim, through a model conversion grammar. Then it will be possible to compare the results of simulations of these two tools to validate this conversion.

Keywords: Simulation. Grids. Exportation.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

dtd: Document Type Definition

GSPD: Grupo de Sistemas Paralelos e Sistribuídos

iSPD: iconic Simulator of Parallel and Distributed systems

JVM: Java Virtual Machine

MIPS: milhões de instruções por segundo

XML: Extensible Markup Language

Capítulo 1

Introdução

A computação de alto desempenho vem crescendo nos últimos anos. Isso se deve, principalmente, às aplicações que demandam um alto nível de processamento, presentes tanto no meio acadêmico (bioinformática, física, química), quanto no comércio (animação, comércio eletrônico). Dessa forma, desde o início da computação tem-se investido em formas novas e mais rápidas de se obter resultados (CRICHLOW, 1988), principalmente pelo desenvolvimento de microprocessadores, e de redes de computadores de alta velocidade (TANENBAUM; STEEN, 2007).

Assim, pode-se pensar no uso de supercomputadores para a solução de problemas que demandam alto grau de processamento, porém o custo envolvido na obtenção e no uso de tais máquinas é bastante elevado. Desta forma, uma alternativa para isto é juntar em um só sistema vários computadores conectados por uma rede de alta velocidade, chamando-os então de Sistemas Distribuídos (TANENBAUM; STEEN, 2007), que são justamente várias máquinas conectadas que se comunicam e que coordenam suas ações através de passagem de mensagem (G. DOLLIMORE J., 2001).

Compartilhamento de recursos, segurança, replicação de dados e escalabilidade são alguns dos muitos desafios que envolvem a construção de um sistema distribuído (G. DOLLIMORE J., 2001), além da avaliação de desempenho. Um exemplo deste tipo de sistema são as Grades Computacionais, na qual a principal característica é a heterogeneidade, tanto no *hardware*, quanto no sistema operacional das máquinas, ou mesmo nas redes (TANENBAUM; STEEN, 2007).

Com o aumento da utilização das Grades Computacionais, houve um crescente número de usuários que muitas vezes não entendem de computação, o que gera uma demanda por ferramentas que primeiramente avaliam tais sistemas.

Há três maneiras de se realizar a avaliação de desempenho de Grades Computaci-

onais: *Benchmarks*, Modelagem Analítica e Simulação (JAIN, 1991). Usando *Benchmarks* tem-se com exatidão o resultado buscado, porém este tipo de métrica depende do acesso ao sistema no qual se deseja testar, do contrário, é impossível utilizar-se destas ferramentas. A Modelagem Analítica é o tipo de avaliação que é feita fazendo uso de fórmulas matemáticas, porém tem uma baixa precisão e geralmente é difícil de se realizar. Já a simulação, embora não tenha a exatidão dos *Benchmarks*, é bastante vantajosa, já que é possível fazer uma boa avaliação sem ter acesso ao sistema simulado.

1.1 Motivação

A maior dificuldade encontrada na utilização das ferramentas que utilizam a simulação para obter resultados da avaliação de Grades Computacionais é que, em sua grande maioria são limitadas na geração de modelos ou mesmo não são fáceis de se usar. Mais ainda, os simuladores mais utilizados tem limitação na quantidade de características simuladas e, muitos deles só reconhecem modelos próprios e nem mesmo geram modelos para outras ferramentas do mesmo tipo.

Como exemplo de simuladores de Grade mais utilizados temos o GridSim (ASHRAF; ERLEBACH, 2011) (KUMAR; KUMAR; KUMAR, 2011) (GRIDSIM, 2013), SimGrid (CASANOVA, 2001) (SIMGRID, 2013), Bricks (TAKEFUSA; MATSUOKA, 2000) (BRICKS, 2013), GangSim (GANGSIM, 2012), OptorSim (OPTORSIM, 2012) (CAMERON et al., 2003). Como já mencionado, tais ferramentas são limitadas no que diz respeito à facilidade do usuário, já que para utilizá-las é necessário saber programar, porém a maioria dos usuários são leigos em computação.

Neste sentido, com o objetivo de sanar os problemas já relatados, o Grupo de Sistemas Paralelos e Sistribuídos (GPSD) (GSPD, 2013), vem desenvolvendo a ferramenta de simulação iSPD (*iconic Simulator of Parallel and Distributed systems*) (MANACERO et al., 2012). O iSPD é um simulador de grades computacionais em que o usuário é capaz de inserir métricas e receber resultados facilmente. Esta ferramenta ainda é capaz de interpretar modelos de outros simuladores, ou mesmo exportar seu próprio modelo para a execução em outra ferramenta.

1.2 Objetivo

O objetivo deste projeto foi desenvolver mais um módulo do iSPD. Com ele, o modelo construído no simulador pode ser convertido em um que possa ser simulado em outra ferramenta, no caso deste trabalho, o GridSim.

Desta maneira, um usuário que já tenha especificado seu modelo no iSPD não precisa especificar novamente no GridSim. Ou mesmo usar o módulo de exportação para somente montar uma Grade específica no GridSim, sem que precise programar.

1.3 Organização do texto

Após esta introdução do trabalho, os capítulos apresentam a seguinte organização:

- **Capítulo 2:** Este capítulo apresenta toda a fundamentação teórica no qual este trabalho está inserido, mostrando os simuladores iSPD, GridSim, apresentando outros simuladores de grades computacionais e fazendo uma breve revisão sobre compiladores e linguagens formais.
- **Capítulo 3:** São apresentadas as atividades realizadas para que o módulo de exportação de modelos funcionasse.
- **Capítulo 4:** São mostrados os testes de exportação de diferentes modelos de grades computacionais, para a validação do módulo de exportação.
- **Capítulo 5:** É feita uma conclusão sobre todo o projeto, bem como o relato de suas contribuições.

Capítulo 2

Revisão Bibliográfica

Este capítulo apresenta os principais conceitos sobre os simuladores de Grades Computacionais iSPD e GridSim, bem como uma visão geral sobre outros principais simuladores utilizados atualmente, além de uma breve revisão sobre os conceitos de Compiladores e de Linguagens Formais.

2.1 O iSPD

O simulador de Grades Computacionais iSPD (*iconic Simulator of Parallel and Distributed Systems*) é uma ferramenta desenvolvida pelo GSPD (Grupo de Sistemas Paralelos e Distribuídos) (GSPD, 2013) da UNESP. Ele tem como principal característica a possibilidade da modelagem do sistema em uma interface gráfica, o que o torna diferente das demais ferramentas de simulação, onde a descrição de modelos é feita através de linguagens de programação (MANACERO et al., 2012).

O iSPD é implementado na linguagem Java (JAVA, 2012), e permite a execução de tarefas do tipo *Bag of Tasks*, implicando que os modelos de grade sejam do tipo Mestre-Escravo. Além disso tudo, o iSPD tenta corrigir um problema que atinge todos outros simuladores de grades computacionais, que é a capacidade de interpretação de seu próprio modelo, ou seja, as demais ferramentas de simulação não compreendem, ou mesmo não geram modelos para outros simuladores.

Na Figura 2.1 é apresentada a visão conceitual do iSPD. Nela é possível observar toda a sua estrutura, e as conexões entre seus módulos. Também é mostrado como é realizada a conversão entre linguagens internas, a esta ferramenta, e com aquelas utilizadas pelos demais simuladores. Como aparece na Figura 2.1, o usuário pode gerar um modelo da grade computacional através de uma interface icônica. Esse modelo icônico

é então interpretado para um modelo simulável, que é processado para a obtenção de métricas de desempenho que sejam relevantes ao usuário. O simulador pode importar dados reais de carga de trabalho (SILVA, 2012), e também exportar ou importar o modelo icônico de outros simuladores existentes.

O detalhamento de cada um dos módulos que compõem o iSPD é visto a seguir.

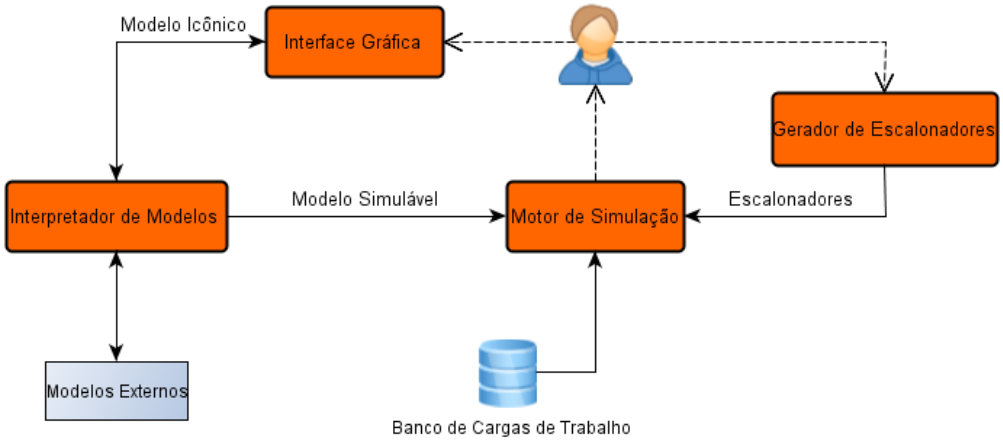


Figura 2.1: Diagrama conceitual do iSPD

2.1.1 Interface Gráfica

Através do módulo de Interface Gráfica é feita a modelagem do sistema e da carga. O procedimento para montar o modelo é feito por meio de ícones. É pela Interface Gráfica que o usuário modela e obtém os resultados de sua simulação (AOQUI et al., 2010). Desta maneira, é através dela que o usuário mantém o acesso aos demais módulos do simulador e realiza qualquer outra operação possível dentro do iSPD.

O objetivo da Interface Gráfica é tornar o processo de modelagem mais intuitivo o possível. Assim, como pode-se observar na Figura 2.2 , basta adicionar os ícones que representam os recursos da grade modelada e conectá-los utilizando das conexões de rede (*links*).

2.1.2 Interpretador de Modelos

O Interpretador de Modelos tem como objetivo a conversão entre diferentes linguagens da representação do modelo definido. Este módulo é dividido em dois outros submódulos, são eles: Interpretador de Modelos Internos e o Interpretador de Modelos

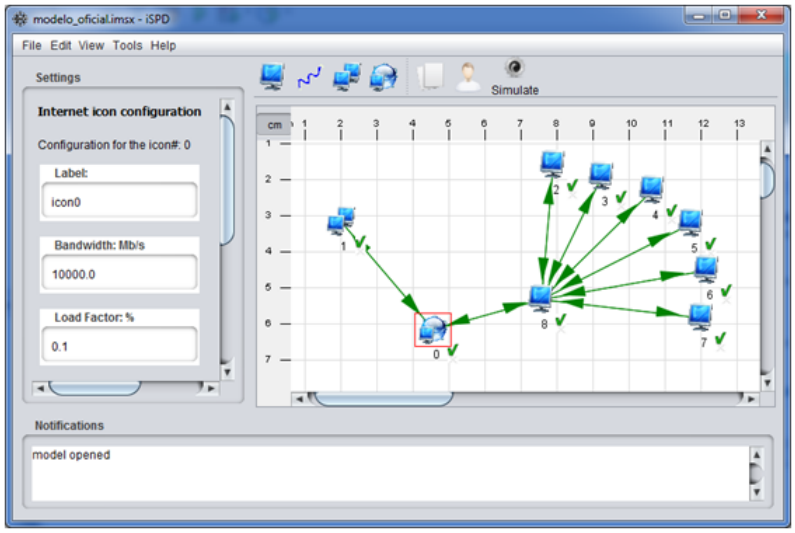


Figura 2.2: Interface Principal ((MENEZES et al., 2012))

Externos.

- **Interpretador de Modelos Internos:** No iSPD, o modelo icônico é construído a partir daquele projetado na Interface Gráfica. E então, chega ao Módulo de Interpretação de Modelos, no qual é convertido para outra linguagem interna ao simulador, construíndo o que é chamado de modelo simulável. É uma linguagem de filas pronta para ser realmente simulada.

- **Interpretador de Modelos Externos:** Esta parte do iSPD é responsável pela comunicação com os demais simuladores de grades computacionais. Ou seja, com este módulo é possivel compreender modelos de outros simuladores, convertendo-os para o modelo icônico. Ou mesmo conseguir fazer com que outros *softwares* simulem um modelo projetado no iSPD.

2.1.3 Motor de Simulação e Gerador de Escalonadores

O Motor de Simulação é o componente do simulador que realmente realiza a simulação (MENEZES et al., 2012). É ele também é o responsável pelo resultado e pelas métricas de simulação devolvidas ao usuário (SILVA, 2012). Durante a realização da simulação, a infraestrutura da grade computacional é mapeada para uma rede de filas.

Já o Gerador de Escalonadores tem como objetivo gerar e gerenciar políticas de escalonamento produzidas pelo próprio usuário para o iSPD (SILVA, 2012).

2.1.4 Modelos Externos

Os modelos externos são aqueles que podem ser convertidos para o iSPD, ou mesmo gerados pelo simulador para que sejam simulados em uma destas ferramentas externas.

Atualmente, o iSPD consegue interpretar modelos do SimGrid e GridSim. Além disso, consegue também gerar modelos para o SimGrid e a partir deste trabalho, também consegue gerar modelos para o GridSim.

2.1.5 Banco de Cargas de Trabalho

Este banco contém cargas de trabalho reais para serem simuladas no iSPD. Tal banco contém informações de *traces*, que consiste na caracterização de uma carga de trabalho pela observação de uma execução real da mesma.

2.2 GridSim

O GridSim é um conjunto de ferramentas de simulação de grades computacionais baseado em Java, mais especificamente da classe SimJava (BUYYA; MURSHED, 2002). Todos os componentes do GridSim se comunicam através de operações de passagem de mensagem definidos no SimJava (SULISTIO et al., 2008). A arquitetura do Gridsim foi construída em forma de camadas, como pode ser observado na Figura 2.3.

De baixo para cima, a primeira camada preocupa-se com a *Java Virtual Machine* (JVM). Já, a segunda camada, refere-se a infraestrutura da simulação baseada em eventos (OLIVEIRA, 2007). A terceira camada direciona-se a modelagem e simulação de entidades da grade, como seus recursos (BUYYA; MURSHED, 2002). Na quarta camada concentram-se os escalonadores dos recursos grade modelada. A quinta e última camada foca na aplicação e na modelagem de recursos, bem como na avaliação de escanoladores e de políticas de gerenciamento de tais recursos.

Especificando um pouco melhor as características do GridSim, temos que esta ferramenta fornece uma maneira de se simular diferentes classes de recursos, usuários e aplicações heterogêneas (BUYYA; MURSHED, 2002). No GridSim, cada usuário pode ter seu próprio escalonador de recursos privado. Assim, é possível fazer com que se otimize os requisitos e objetivos de seu usuário.

Neste simulador, os recursos podem ser modelados para que sejam monoprocessados ou multiprocessados, explorando os conceitos de compartilhamento de espaço e tempo, e sua capacidade é definida em termos de MIPS (milhões de instruções por segundo). Além disso, pode-se destacar:

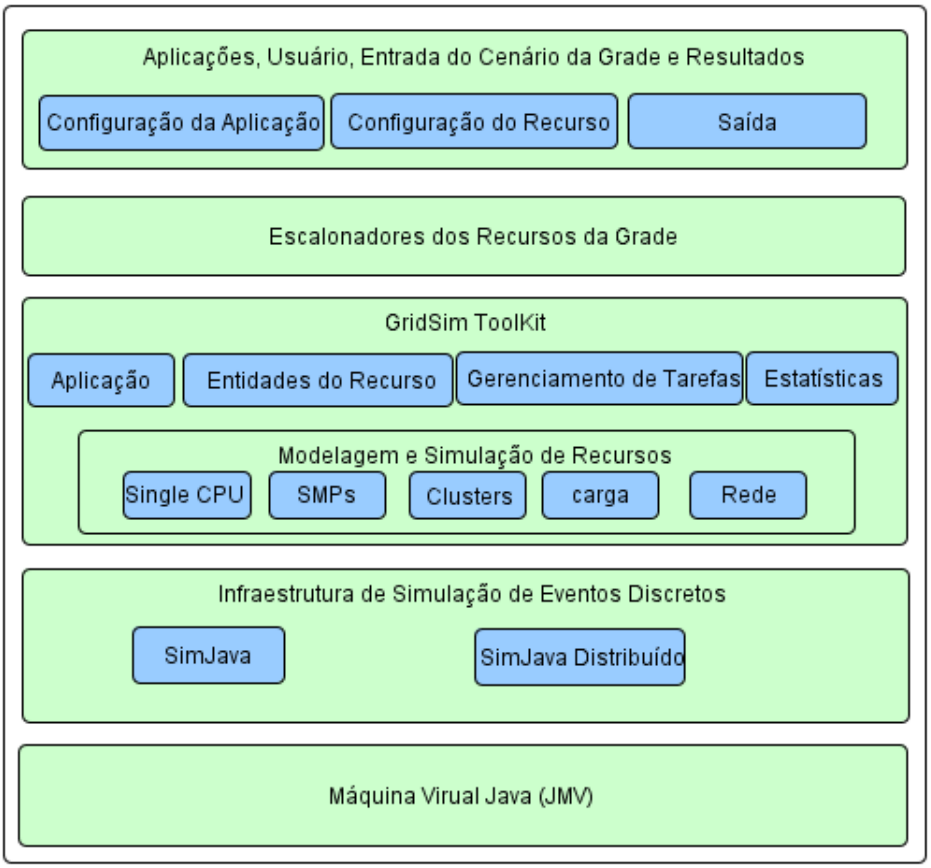


Figura 2.3: Arquitetura GridSim

- Finais de semana e feriados podem ser mapeados quando são criados, ou seja é possível modelar variações de carga incluíndo tais datas.
- Modela a possibilidade de reserva de recursos.
- Grades de dados podem ser modeladas e simuladas.
- Tarefas podem ser heterogêneas, ou seja, elas podem necessitar de muito processamento ou mesmo muita entrada e saída.
- Estatísticas das operações realizadas podem ser gravadas e analisadas.

A modelagem no GridSim é feita utilizando-se da linguagem de programação Java. Ou seja, para se construir um modelo no GridSim é necessário o conhecimento de

conceitos de programação, bem como de orientação a objeto. Através disto, constróem-se os recursos, a rede, as cargas de trabalho e os escalonadores. Um exemplo de modelo completo no GridSim pode ser visto no Anexo A.

O GridSim continua em desenvolvimento até hoje. Em (SULISTIO et al., 2008) é mostrada uma extensão do GridSim, agora para o suporte de Grades de Dados. Em (CLOUDSIM, 2013) o trabalho apresentado mostra uma atualização do simulador, porém agora ele pode simular computação em nuvem, e seu nome é mudado para CloudSim.

Este simulador possui vasta documentação, sendo facilmente encontrada, o que ajuda na sua compreensão para qualquer usuário, facilitando a realização deste projeto.

2.3 Outros Simuladores

Esta sessão dedica-se a apresentar rapidamente outros simuladores de grades computacionais utilizados pela comunidade científica, a fim de mostrar mais uma justificativa da escolha do GridSim para realizar a exportação de modelos do iSPD. Já que a maioria dos projetos apresentados nesta sessão estão parados a um certo tempo.

2.3.1 SimGrid

A ferramenta de simulação SimGrid (SIMGRID, 2013) foi desenvolvida inicialmente com o objetivo principal de se estudar algoritmos de escanolamento (GUERRA; AOKI, 2010). Nesta primeira versão, o simulador era orientado a eventos, porém era bem limitado, como por exemplo, não era possível a modelagem de algoritmos de escalonamento descentralizados.

Na sua versão mais atual, o simulador tem vários ambientes de programação construídos em cima de um único núcleo de simulação (OLIVEIRA, 2007), e várias limitações encontradas anteriormente foram corrigidas.

O SimGrid é implementado na linguagem C, porém ele apresenta um interessante recurso: a possibilidade de especificar a plataforma computacional, os recursos da Grade, a conexão e as tarefas em arquivos de *Extensible Markup Language* (XML) (CASANOVA, 2001).

2.3.2 Bricks

O simulador Bricks (BRICKS, 2013) permite diferentes estratégias de escalonamento de tarefas em Grades Computacionais. Assim, é possível realizar análises e avaliações

de desempenho de tais estratégias (OLIVEIRA, 2007).

Obviamente, o objetivo principal deste *software* é simular o sistema modelado com diferentes configurações, sejam elas mudando a política de escalonamento, topologias de rede ou mesmo alterando as taxas de processamento de servidores, por exemplo.

O Bricks também é implementado em Java. O simulador contém algums componentes para facilitar a especificação de escalonamentos, porém utilizando-se de programação (TAKEFUSA; MATSUOKA, 2000).

2.3.3 OptorSim

O OptorSim (OPTORSIM, 2012) foi desenvolvido com o objetivo de estudar ambientes de grades computacionais típicos, e avaliar algoritmos de réplica da dados (BELL et al., 2003), ou seja, esta ferramenta foi projetada com o propósito de simular um tipo de grade específica, as grades de dados em que a transferência de dados é um fator limitante no desempenho do sistema (OLIVEIRA, 2007).

Para o simulador, os escalonamentos levam em consideração que a replicação significa que são criadas cópias dos dados em recursos que estão distribuídos geograficamente (OLIVEIRA, 2007).

Tanto em (CAMERON et al., 2003), quanto em (BELL et al., 2002), assim como na maioria dos trabalhos encontrados a respeito do simulador, as principais aplicações usadas para o OptorSim constituem na avaliação de estratégias para a replicação.

2.4 Conceitos de Compiladores e Linguagens Formais

Esta sessão destina-se a apresentar alguns dos principais conceitos sobre compiladores e sobre linguagens formais, já que, para a realização deste trabalho, são necessários alguns destes tópicos.

2.4.1 Linguagens Formais

As linguagens formais podem ser representadas por mecanismos reconhecedores, os quais determinam se uma sentença pertence ou não a uma linguagem, como os autômatos, ou por geradores, que são mecanismos capazes de gerar as sentenças da linguagem, as gramáticas.

Segundo a hierarquia de Chomsky, para cada tipo de linguagem há um tipo diferente de reconhecedor e gerador. Por exemplo, para a Linguagem Regular, o mecanismo reconhecedor é o Autômato Finito e o mecanismo gerador é a Gramática Regular. Já

para a linguagem Livre de Contexto, o mecanismo reconhecedor é o Autômato com Pilha e o gerador é a Gramática Livre de Contexto.

2.4.2 Estrutura Geral do Compilador

Um compilador nada mais é do que um *software* cuja finalidade é realizar a tradução entre diferentes tipos de linguagem. Mais ainda, em tipos de linguagens de programação específicas, sendo uma delas aquela em que o usuário tem mais facilidade, ou seja, aquelas ditas de alto nível, e a outra, aquela chamada de baixo nível, sendo essa a que o computador compreende.

A Figura 2.4 mostra a estrutura geral de um compilador. Nela é possível perceber que o compilador é dividido em 6 etapas, agrupados em duas fases: a análise e a síntese. Na análise é feita uma representação intermediária do programa fonte, e na Síntese é feita a construção do programa alvo (SETHI RAVI; JEFFREY; LAM, 2008). Na Figura 2.4 a análise é diferenciada na cor verde, e a síntese na cor laranja.

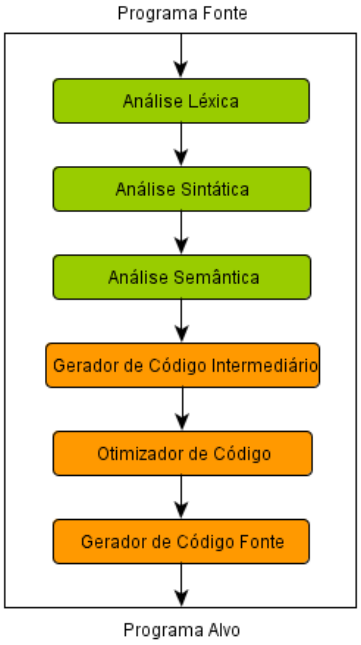


Figura 2.4: Estrutura de um Compilador

2.4.3 Análise de Código

A fase de Análise Léxica constitui em ler caracteres do programa fonte e agrupá-los em conjuntos que tenham um significado coeso, chamados de *Tokens* (SETHI RAVI; JEFFREY; LAM, 2008). Com isso são identificadas também palavras-chave, pontuação e operadores, por exemplo. A implementação da Análise Léxica tem como base o reconhecedor da Linguagem Regular das Linguagens Formais, ou seja, o Autômato Finito.

A fase de Análise Sintática constitui em agrupar *tokens* hierarquicamente, usualmente em árvores (SETHI RAVI; JEFFREY; LAM, 2008). Desta forma toda estrutura de uma frase do programa fonte é analisada. A implementação da Análise Sintática é feita com base no Autômato com Pilha da Linguagem Livre de Contexto, de Linguagens Formais.

Na Análise Semântica são feitas algumas verificações para ter certeza que todos os componentes do programa combinam (SETHI RAVI; JEFFREY; LAM, 2008).

2.4.4 Síntese

Na fase de Geração de Código Intermediário é feita uma representação intermediária do programa fonte, próxima à linguagem de máquina (SETHI RAVI; JEFFREY; LAM, 2008), a fim de que se torne mais fácil a conversão posteriormente.

Já a Otimização de código, próxima fase do compilador, tem como objetivo melhorar o desempenho do programa, ou seja, fazer com o que programa execute mais rápido.

A última fase consiste em gerar o código alvo propriamente dito. São selecionadas as posições de memória para cada variável do programa.

2.5 Considerações finais

Neste capítulo apresentou-se a ferramenta sobre o qual a módulo de exportação de modelos foi desenvolvido, o iSPD, mostrando seu funcionamento e seus principais conceitos. Além dele, foi mostrado o GridSim, o simulador alvo da conversão do modelo gerado pelo iSPD.

E mais, este capítulo trouxe outros simuladores de Grades, ressaltando a importância de se escolher o GridSim, já que aqueles estão mais desatualizados. Também, foram revistos os principais tópicos sobre compiladores, já que para a realização deste projeto, foi necessária a utilização de alguns desses conceitos, como a análise de códigos.

Capítulo 3

Metodologia para a Conversão de Modelos

Este capítulo tem como objetivo mostrar tudo o que foi desenvolvido para a realização do projeto. Desta maneira, é apresentado como o iSPD realiza a análise do modelo icônico e como ele guarda os dados desta análise, para posteriormente serem utilizadas na construção do modelo externo. Além disso, é mostrada a gramática para a conversão de modelos.

E também o mais importante: é visto como é estruturado o Exportador e sua relação com o restante do sistema, ou seja, como a análise do modelo icônico é realizada e como seu resultado é utilizado pelo módulo de exportação.

3.1 O modelo icônico do iSPD e sua análise

O modelo icônico é projetado a partir da modelagem realizada pelo usuário do iSPD na interface gráfica do simulador, sendo composto de todos os seus elementos tais como: máquinas, *clusters*, rede, escalonadores e cargas de trabalho. Logo em seguinda, todos os componentes deste modelo, assim como seus parâmetros de configuração, são salvos em um arquivo de *eXtensible Markup Language* (XML).

Um exemplo de um modelo icônico em XML simples pode ser observado na Figura 3.1. Este modelo consiste em duas máquinas, sendo uma delas mestre e a outra escrava, conectadas diretamente por um *link*. E há apenas uma única tarefa a ser processada.

A análise do modelo icônico é feita apenas uma vez pelo simulador da seguinte forma: como já mencionado, quando o usuário do simulador projeta sua grade, todas as suas especificações são transformadas, do modelo icônico, em um arquivo de XML.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE system SYSTEM "iSPD.dtd">
<system version="1">
    <owner id="user1"/>
    <machine id="icon0" load="0.0" owner="user1" power="500.0">
        <master scheduler="RoundRobin">
            <slave id="1"/>
        </master>
        <position x="74" y="74"/>
        <icon_id global="0" local="0"/>
    </machine>
    <machine id="icon1" load="0.0" owner="user1" power="500.0">
        <position x="185" y="74"/>
        <icon_id global="1" local="1"/>
    </machine>
    <link bandwidth="100.0" id="icon2" latency="0.01" load="0.0">
        <connect destination="1" origination="0"/>
        <position x="185" y="74"/>
        <position x="74" y="74"/>
        <icon_id global="2" local="0"/>
    </link>
    <load>
        <random tasks="1" time_arrival="0">
            <size average="3" maximum="3" minimum="3" probability="2.0" type="computing"/>
            <size average="3" maximum="3" minimum="3" probability="2.0" type="communication"/>
        </random>
    </load>
</system>
```

Figura 3.1: Exemplo de modelo icônico

Quando é necessário utilizá-lo, a análise é feita por funções próprias da linguagem Java, que é feita através de regras de um arquivo de *Document Type Definition* (dtd) .

Desta forma, o arquivo em XML é transformado em uma árvore, com todas as informações do modelo projetado pelo usuário. Assim, todos os demais módulos que necessitam do modelo icônico, incluindo o exportador, recebem tal árvore.

Esta estrutura de dados contém todos os ícones do modelo como Usuários, Máquinas, Clusters, Ícone de Internet, Links e Carga de trabalho, bem como dos algoritmos de escalonamento. Desta maneira, para ter acesso a estes dados analisados basta ter acesso a árvore e, durante a exportação, construir as estruturas de dados próprias para cada um dos elementos do modelo projetado.

Internamente ao simulador, como já dito anteriormente na seção 2.1, se o usuário optar por simular no iSPD, o modelo icônico é convertido para o modelo simulável, que é uma linguagem de filas, pronta para ser processada pelo motor de simulação do iSPD.

3.2 Especificação das Gramáticas para a Conversão do Modelo Icônico para o GridSim

Esta sessão apresenta as gramáticas de conversão do modelo icônico do iSPD para o GridSim. Como já dito na seção 3.1, tal modelo é um arquivo XML, o qual segue as regras contidas em um documento dtd.

As regras de construção do modelo icônico definem primeiramente todos os elementos contidos no modelo, sendo eles Máquinas, *Clusters*, *Link*, Internet ou mesmo Usuário. Assim, para cada elemento todos os parâmetros necessários para sua correta configuração são ajustados.

Deste modo, por exemplo, ícones que são do tipo mestre, tem relacionado a eles todos os seus escravos, bem como seu algoritmo de escalonamento. Além disso tudo, também são definidas a carga de trabalho e as posições de cada ícone.

Os ícones de *link* e de internet têm como parâmetros: banda, latência e taxa de ocupação. E os icônes *cluster* e máquina têm como parâmetro poder computacional, taxa de ocupação e usuário.

Mais ainda, cada ícone presente no modelo, bem como possíveis usuários do sistema a ser simulado, são identificados através do parâmentro *id*.

O arquivo dtd completo pode ser visualisado no Anexo B.

Para a transformação do modelo icônico do iSPD para o GridSim foram construídas diversas regras, ou seja, uma gramática de conversão entre os dois tipos de modelo. Serão mostradas as principais regras, aquelas que convertem os elementos do modelo como as máquinas, os *clusters*, os *links*, os ícones de internet e a carga de trabalho.

No caso de máquinas ou *clustes*, que são tratados como recursos pelo GridSim, é escrito um método que cria recursos. A parte da gramática que identifica recursos no modelo icônico é mostrada na figura 3.2.

```
<Maquina>:= <menor>machine id <igual> <nome> load <igual> <ocupado> owner <igual>
<nome> power <igual> <capacidade><maior>
      <menor> position x <igual><pos> y <igual> <pos> <maior>
      <menor>icon_id global <igual> <id> local <igual> <local><maior>

<Cluster>:=<menor>cluster bandwidth <igual> <banda> id <igual> <nome> latency<igual>
<latencia> master<igual><tag> nodes<igual> <nós> owner<igual><nome> power<igual>
<capacidade> scheduler<igual><escal>
      <menor> position x<igual><pos> y<igual><pos><maior>
      <menor> icon_id global<igual><igual> local<igual><local><maior>
```

Figura 3.2: Regra de identificação de recursos

Quando os recursos da grade são identificados, a conversão para o GridSim segue

as regras descritas na Figura 3.3.

```
GridResource NOME_RECURSO = createResource("NOME_RECURSO",  COMUNICAÇÃO,
NÚMERO_DE_MÁQUINAS, PODER_COMPUTACIONAL);
```

Figura 3.3: Regra de criação de recursos

Para as conexões entre elementos da grade, ou seja, os *links*, são escritos os métodos correspondentes no GridSim que criam e configuram *links*. Já para a conversão dos ícones de internet são criados os roteadores no modelo do GridSim. A regras de identificação dos *links* e dos ícones de internet são mostradas na Figura 3.4.

```
<link>:=<menor>link bandwidth<igual><banda> id<igual><nome> latency<igual><latencia>
load<igual><ocupado><maior>
      <menor>connect destination<iguai><id_dest> origination<iguai><id_ori><maior>
      <menor>position x<igual><pos> y=<pos><maior>
      <menor>position x<igual><pos> y=<pos><maior>
      <menor>icon_id global<igual><id> local=<num> <maior>

<internet>:=  <menor>internet bandwidth<igual><banda> id<igual><nome> latency<igual>
<latencia> load=<igual><ocupado><maior>
      <menor>position x<igual><pos> y<igual><pos><maior>
      <menor>icon_id global<igual><id> local<igual><num><maior>
```

Figura 3.4: Regras de identificação de links e internet

Assim que são identificados, a criação de *links* e internet seguem as regras descritas na Figura 3.5.

```
Link NOME_LINK = new SimpleLink("NOME_LINK", BANDA, LATÊNCIA, MTU );

Router NOME_ROTEADOR = new RIPRouter( "NOME_ROTEADOR", FLAG);
```

Figura 3.5: Regras de criação de links e internet

Para a carga de trabalho, são criados os chamados *Gridlets* no GridSim, todos eles contidos em uma lista. Para a criação da carga de trabalho, primeiramente são identificadas as tarefas no modelo icônico a partir de regras mostradas na Figura 3.6.

Uma vez identificadas, as tarefas sofrem a conversão para o modelo GridSim. Tal método segue a regra identificada na Figura 3.7.

Como uma forma de se resumir as regras anteriormente apresentadas, na Figura 3.8 é apresentado como cada elemento que compõe o modelo é convertido para o GridSim. Para o recurso é criado o método *createResource*, para *link* é criado o método *SimpleLink*, para o ícone de internet é criado um elemento roteador do GridSim, para

```
<Carga>:=<menor>load<maior> <menor>random tasks<igual><n_tar> time_arrival<igual><time><menor>
        <menor>size average<igual><media_comp> maximum<igual><max_comp> minimum<igual>
<min_comp> probability<igual><prob_comp> type <igual> computing <maior>
        <menor>size average<igual><media_comm> maximum<igual><max_comm> minimum<igual>
<min_comm> probability<igual><prob_comm> type<igual>communication<maior>
```

Figura 3.6: Regra de identificação de tarefas

```
Gridlet NOME_GRIDLET = new Gridlet( ID, TAXA_PROCESSAMENTO,
TAMANHO_ENTRADA, TAMANHO_SAÍDA);
```

Figura 3.7: Regra de criação de tarefas no GridSim

as tarefas é criada uma lista de *GridLets* e por fim para os usuários é criada uma lista de *ResourceUser*.

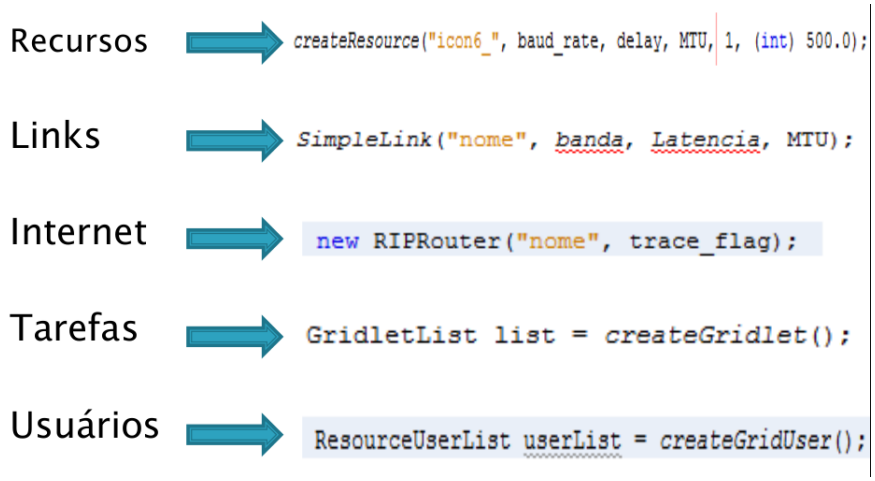


Figura 3.8: Resumo das conversões

Seguindo todas as regras que foram mostradas nesta seção, o exportador pode ser desenvolvido. A seção 3.3 mostra como foi montado o código do exportador de modelos icônicos.

3.3 Desenvolvimento do Exportador

Primeiramente, no código do Exportador, são criadas listas para armazenar todos os componentes do modelo construído no iSPD. São elas para Usuários, Máquinas, Clusters, Ícones de Internet, Links e para a Carga de trabalho, totalizando assim 6 listas. Tais listas são preenchidas com seus respectivos conteúdos a partir da análise do modelo

icônico.

Então, é criado o arquivo Java onde será escrito todo o modelo. As primeiras linhas deste arquivo são as bibliotecas necessárias para a correta execução posterior do modelo.

Primeiramente, são construídas no modelo gerado duas classes. Uma delas, a classe Mestre, trata exclusivamente das máquinas mestres, sua comunicação com suas máquinas escravas e o modo de envio e recebimento das tarefas. A outra classe construída, a Modelo, é a que contém a maior parte das informações da Grade modelada, sendo elas as máquinas escravas e instanciação de objetos que representam os mestres, bem como o modo de conexão entre eles. Tal construção pode ser visualizada na Figura 3.9, que é um diagrama de classes UML de um modelo genérico no GridSim produzido pelo Exportador. Ou seja, toda exportação realizada terá essas duas classes, mudando apenas o conteúdo dos métodos internos de cada classe.

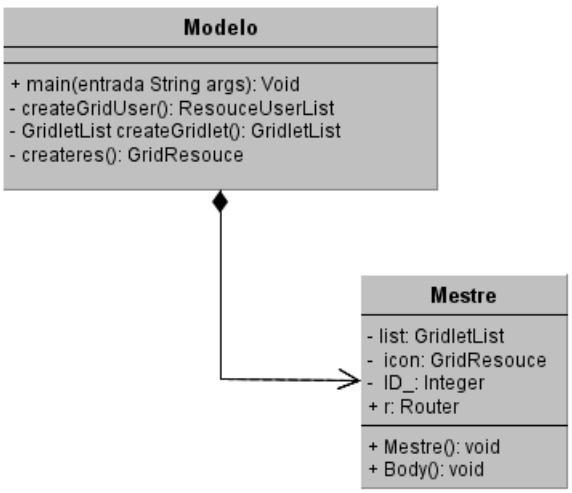


Figura 3.9: Diagrama de Classes UML do Modelo no GridSim

Para a construção completa na classe Modelo, são utilizadas as listas com as informações de cada componente da grade, percorrendo cada uma delas. O método *createres()* refere-se à construção de recursos da grade. Assim, dependendo dos parâmetros passados para este método são retornados os recursos desejados sendo eles máquinas ou mesmo *clusters*. Dentro deste método, para a criação de recursos são instanciados objetos da classe *GridResource*, própria do GridSim.

Na mesma classe ainda, em *createGridlet()*, é construído todo o conjunto de tarefas do modelo. Já no método *createGridUser()* são criados todos os usuários da grade

modelada. Por fim, na *main()* é onde todas as informações são juntadas, inclusive instanciando objetos que representam os mestres, construindo recursos que são passados através do método *createres()*, e conectando todos através dos elementos de rede.

Tais conexões são feitas a partir da lista de *links*, em que cada elemento contém a informação de origem e destino desses elementos de conexão. Para a criação de *links* no GridSim são instanciados objetos do tipo *Simplelink*. Porém, para utilizá-lo é preciso criar um elemento do tipo roteador no modelo do GridSim, que não existe no iSPD. Para isso, tal tarefa é realizada adaptando os parâmetros do modelo para a sua construção, ou seja, são criados os roteadores entre os elementos na grade para então conectar todos.

Independente disto, os ícones de Internet que aparecerem no modelo desenvolvido no iSPD são automaticamente transformados em roteadores no modelo exportado no GridSim.

Já na classe Mestre, o método *Mestre()* é apenas o construtor da classe, e o método *body()* contém as informações do algoritmo de escalonamento, a comunicação com suas máquinas escravas, e como é feito o recebimento das tarefas processadas. Completando assim, o modelo exportado para o GridSim.

3.4 Incorporação do Conversor com o Simulador iSPD

No iSPD, o código que propriamente gera o arquivo em Java contendo o modelo exportado é todo feito em uma única classe, chamada de *Exportador_GridSim*. Tal classe recebe apenas as informações do modelo icônico do iSPD, que já foi analisado. O código desta classe foi descrito na sessão 3.3.

Na Figura 3.10 aparece um diagrama de classes em UML do módulo de exportação de modelos icônicos, no qual aparece a informação de como o código do exportador é incluído no sistema do simulador iSPD. Como é possível observar na figura, para realizar a conversão de um modelo, são necessárias três classes do iSPD, são elas: JPrincipal, sendo essa a principal classe do sistema, que contém, além da própria exportação, também as principais ações que o usuário pode realizar dentro do simulador; a classe AreaDesenho, de onde vem os parâmetros do modelo; e a classe Exportador_GridSim, que contém o código de como deve ser realizada a exportação do modelo.

Detalhando os métodos da classe Exportador_GridSim, temos que *Escreve()* é o método que propriamente constrói o arquivo do modelo exportado em Java, a partir dos parâmetros vindos da análise do modelo icônico do iSPD, todo este processo já foi previamente detalhado. Já *Recebedoc()* é o método onde o arquivo que contém todos

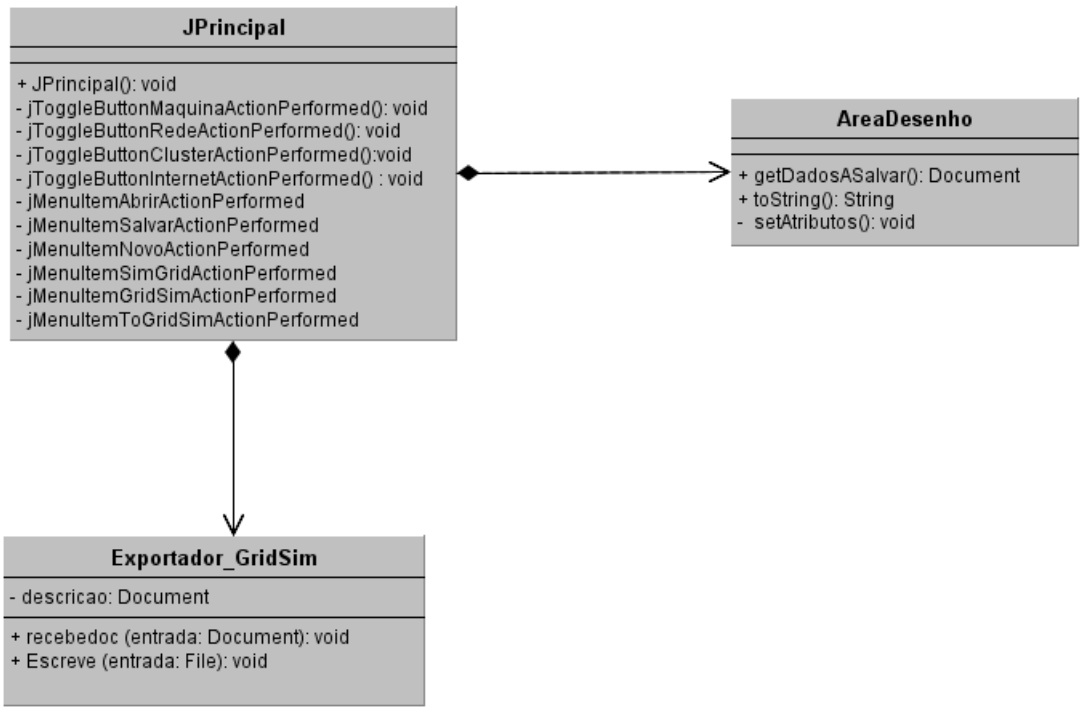


Figura 3.10: Diagrama de Classes UML do Módulo de Exportação

os parâmetros que podem ser obtidos do modelo icônico é recebido pela classe.

A classe AreaDesenho já estava feita, porém é necessário apresentá-la, pois é partir dela que todos os parâmetros do modelo são analisados e são passados para as demais classes do simulador que os necessitam. Mais especificamente, o método que realiza isto chama-se *getDadosASalvar()*. Os demais métodos presentes nesta classe realizam e também salvam o modelo projetado na área de desenho.

Continuando com os detalhamentos, a classe JPrincipal é uma das principais de todo o simulador. Esta classe também já existia, porém foram adicionadas mais funcionalidades a ela. O método *JPrincipal()* é apenas o construtor da classe. Os outros métodos se referem às principais ações do usuário dentro do sistema, tais como adicionar na área de desenho do simulador máquinas, *clusters*, *links* e ícones de internet, além de ações mais básicas como construir um novo modelo, salvar ou abrir um existente. O método *jMenuItemToGridSimActionPerformed* preocupa-se justamente com a exportação do modelo para o GridSim. Para realizá-lo, dentro deste método é instanciado um objeto da classe Exportador_GridSim, utilizando-se de seus métodos para receber parâmetros e escrever o arquivo em um modelo no GridSim.

É importante ressaltar que tanto a classe JPrincipal quanto a AreaDesenho contém mais métodos e funcionalidades do que aquelas mostradas na Figura 3.10, porém eles não são necessários para o entendimento ou desenvolvimento deste projeto.

A classe Exportador_GridSim está contida no pacote de ExportadordeModelos. Já JPrincipal e ÁreaDesenho estão contidas no pacote de InterfaceGráfica do simulador iSPD.

Desta maneira, o usuário que queira exportar um modelo para o GridSim, primeiro deve projetar sua grade, bem como sua carga de trabalho completos no iSPD e, em seguida, ir à barra de menus, encontrar *File -> Export -> To Gridsim Model*. O simulador pedirá o diretório no qual o arquivo em java, com o modelo já exportado para o Gridsim, deverá ser salvo. Todo este processo pode ser visto na Figura 3.11.

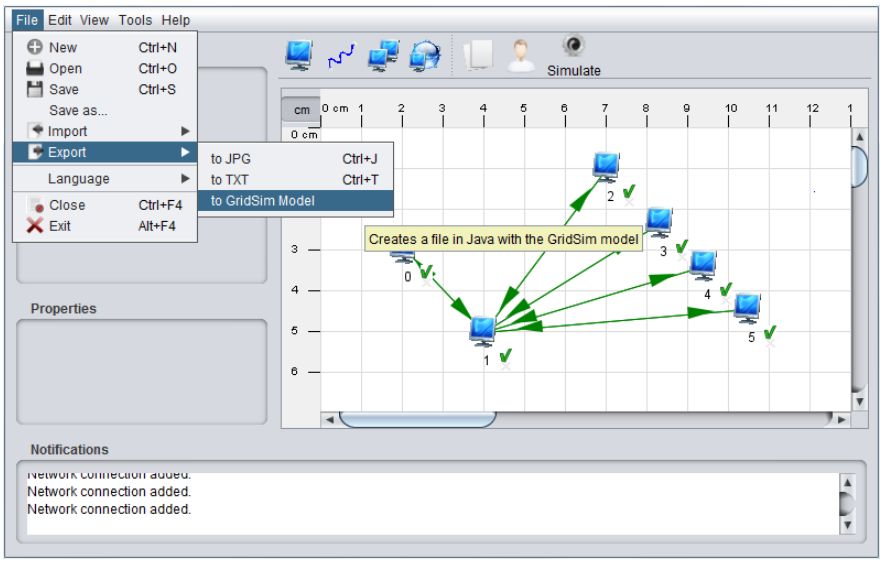


Figura 3.11: Realização da Exportação

É importante destacar que para o uso do módulo de exportação, é necessário que o modelo esteja completo, ou seja, a configuração de todos os parâmetros das máquinas, dos *clusters*, dos algoritimos de escalonamento e das cargas de trabalho devem ser totalmente preenchidos.

Para finalizar, na Figura 3.12 são apresentados os casos de uso de como ocorre a exportação de modelos para o GridSim. Como é possível ver, o usuário deve primeiro montar seu próprio modelo ou abrir um já existente, então poderá exportá-lo. Ou se desejar, simulá-lo no próprio iSPD e receber seus resultados. Na Tabela 3.1 é apresentada a tabela dos casos de uso.

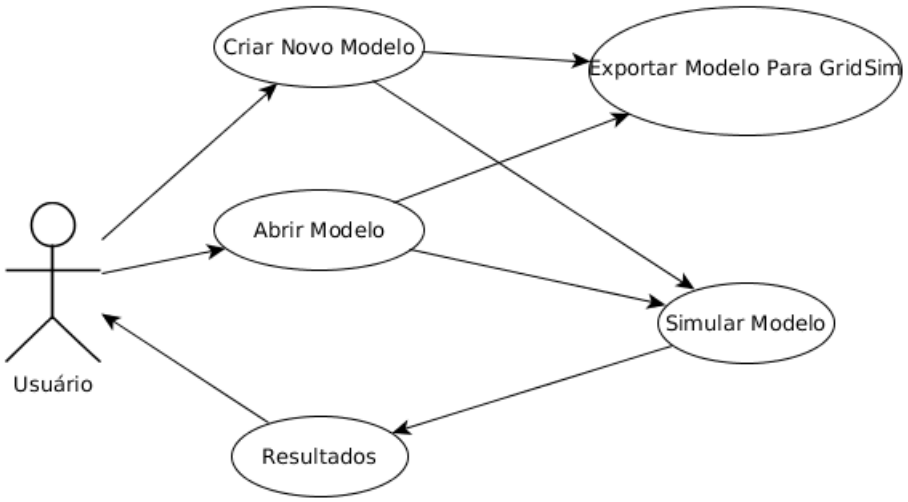


Figura 3.12: Casos de uso

3.5 Considerações Finais

Neste capítulo apresentou-se tudo o que foi desenvolvido para que a exportação do modelo icônico para o GridSim pudesse ocorrer. Foi apresentada a criação de nova classe no código do iSPD, além da criação de métodos nas classes já existentes para integrar com o restante do sistema. E também, foi detalhado como o código no GridSim em Java é construído, seus métodos e classes.

Ainda mais, apresentou-se toda a base para o desenvolvimento deste módulo, como a apresentação das gramáticas e da interpretação do próprio modelo icônico.

O próximo capítulo apresenta a validação da exportação e quais testes foram realizados.

Tabela 3.1: Casos de Uso

Caso de Uso	Quem inicia a ação	Descrição do caso de uso
Criar Novo Modelo	Usuário	O usuário acessa o sistema para criar um novo modelo a partir dos ícones da interface gráfica.
Abrir Modelo	Usuário	O usuário acessa o sistema para abrir um modelo previamente projetado.
Simular Modelo	Usuário	A partir do modelo pronto, o usuário o simula.
Resultados	Usuário	Depois da simulação os resultados são entregues ao usuário.
Exportar Modelo Para GridSim	Usuário	A partir do modelo pronto, o usuário pode exportá-lo para o GridSim

Capítulo 4

Testes e Validação

Neste capítulo são apresentados os testes que validam a exportação de modelos através dos tempos de simulação. Para apresentá-los, primeiramente é mostrada uma grade modelada no iSPD, e então todo o processo de conversão é feito, obtendo assim, um modelo no GridSim. Além disso, a mesma grade modelada, bem como sua carga de trabalho, também é feita diretamente, sem o uso do exportador. Então todos os modelos são simulados e obtém-se o tempo de simulação, ou seja, o tempo que levaria para que o sistema proposto executasse a carga especificada. Para finalizar, todos os resultados são comparados.

São apresenteados três diferentes modelos para os testes. Cada modelo sofre uma variação na carga de trabalho. Assim, cada modelo terá um baixo volume de comunicação e computação, em seguida baixo volume de comunicação e alto índice de computação, e então alto volume de comunicação e baixo índice de computação, e finalmente alto vomule de comunicação e computação.

Os testes realizados são mostrados nas próximas seções. Os resultados apresentados representam a média de cinco simulações dos três modelos, aumentando-se a quantidade de tarefas a serem executadas.

4.1 Teste com um Modelo Simples

Este primeiro teste ocorre com um modelo de grade computacional composta de um único mestre que distribui tarefas às suas sete máquinas escravas, todas conectadas com *links* de 100Mb/s com uma latência de 0,1s. Os recursos deste sistema tem uma capacidade de processamento de 500MFlops. Na Figura 4.1 é mostrado tal modelo. Como é possível observar, o mestre é a máquina identificada com o número 0 (zero),

conectada diretamente as demais máquinas do modelo.

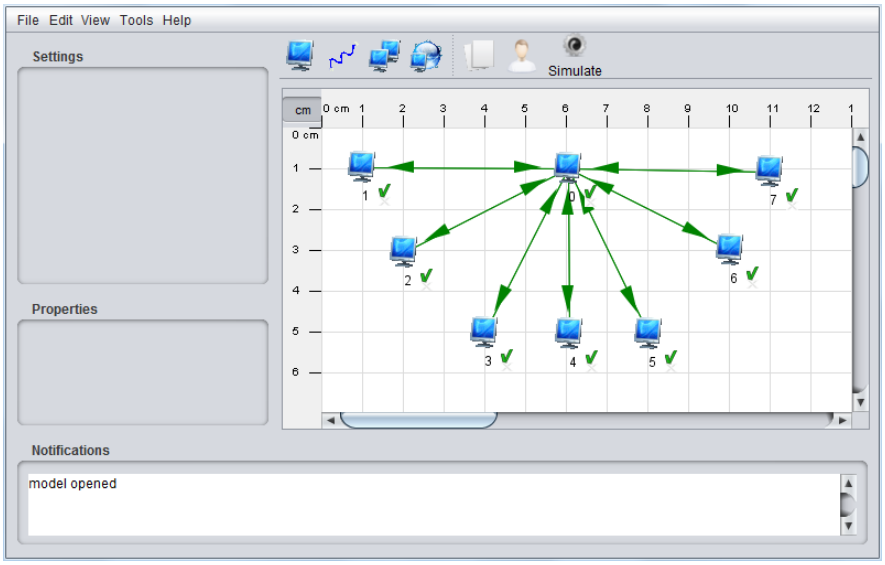


Figura 4.1: Grade do Primeiro Teste

Assim sendo, este modelo foi exportado para um simulável no GridSim. Para exemplificar o que acontece quando tal processo ocorre, na Figura 4.2 é mostrado um pedaço do código exportado. Nele aparece a parte necessária para a criação dos recursos. Como é possível observar, são chamadas sete vezes o método *createResource*, visto na Figura 4.2 nas linhas 1 (um) a 7 (sete), o qual faz a criação propriamente dita, a partir de parâmetros tal como a capacidade de processamento. E em seguida todos estes recursos são adicionados em uma lista de escravos, visto na Figura 4.2 nas linhas 10 (dez) a 16 (dezeseis), que posteriormente será usada na criação do mestre, passando seus escravos como parâmetro para a sua instanciação.

Mais ainda, além destes dois modelos, o convertido e aquele feito no iSPD, foi montado mais um no GridSim, com as mesmas características daquele realizado primeiramente. Então foram realizadas as simulações variando-se a carga de trabalho.

4.1.1 Carga com baixo volume de comunicação e computação

As tarefas deste teste tem 50 Mflops de computação, ou seja, a tarefa necessita de 50 Mflops para ser processada. E tem também 3 Mbits de comunicação. O resultados das médias dos tempos das simulações são apresentados no gráfico da Figura 4.3.

A maior diferença entre o resultado do modelo simulado no iSPD e aquele convertido para o GridSim foi de 5,55% que ocorreu com 250 tarefas. Além disso, a diferença média

```
1  GridResource icon1 = createResource("icon1_",  baud_rate,  delay,  MTU, 1, (int)500.0);
2  GridResource icon2 = createResource("icon2_",  baud_rate,  delay,  MTU, 1, (int)500.0);
3  GridResource icon6 = createResource("icon6_",  baud_rate,  delay,  MTU, 1, (int)500.0);
4  GridResource icon7 = createResource("icon7_",  baud_rate,  delay,  MTU, 1, (int)500.0);
5  GridResource icon3 = createResource("icon3_",  baud_rate,  delay,  MTU, 1, (int)500.0);
6  GridResource icon5 = createResource("icon5_",  baud_rate,  delay,  MTU, 1, (int)500.0);
7  GridResource icon4 = createResource("icon4_",  baud_rate,  delay,  MTU, 1, (int)500.0);
8  GridletList list = createGridlet();
9  ArrayList esc0 = new ArrayList();
10 esc0.add(icon1);
11 esc0.add(icon2);
12 esc0.add(icon3);
13 esc0.add(icon4);
14 esc0.add(icon5);
15 esc0.add(icon6);
16 esc0.add(icon7);
```

Figura 4.2: Parte do código exportado

foi de 3,75%.

4.1.2 Carga com baixo volume de comunicação e alto índice de computação

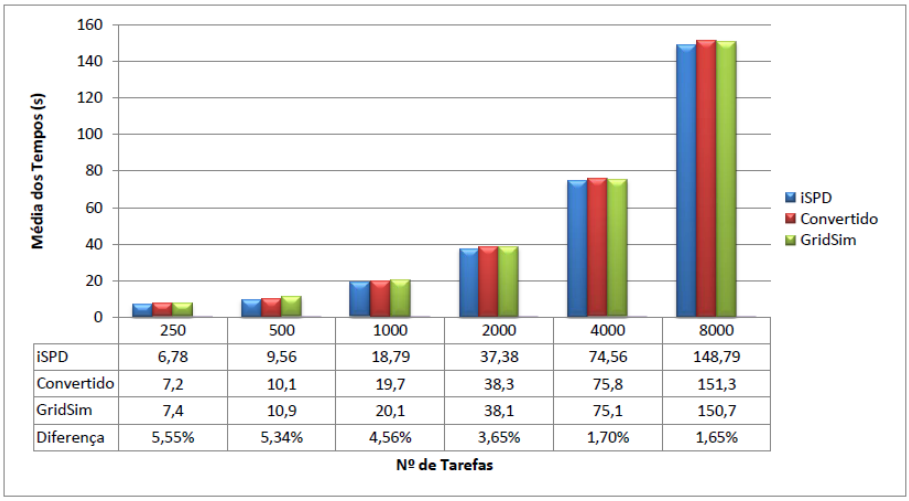
As tarefas deste teste tem 5000 Mflops de computação. E tem também 3 Mbits de comunicação, ou seja, essas tarefas necessitam de 3 Mbits para trafegarem na rede da grade. O resultados das médias dos tempos das simulações são apresentados no gráfico da Figura 4.4.

A diferença máxima entre os resultados do modelo simulado no iSPD e do convertido para o GridSim foi de apenas 5,06% que ocorreu com 2000 tarefas. Além disso, a diferença média destes resultados foi de 2,85%.

4.1.3 Carga com alto volume de comunicação e baixo índice de computação

As tarefas deste teste tem 50 Mflops de computação e 500 Mbits de comunicação. O resultados das médias dos tempos das simulações são apresentados no gráfico da Figura 4.5.

A maior diferença entre o resultado do modelo simulado no iSPD e do convertido para o GridSim ocorreu com 1000 tarefas e foi de 7,4%. Além disso, a média das diferenças foi de 3,08%.



	250	500	1000	2000	4000	8000
iSPD	6,78	9,56	18,79	37,38	74,56	148,79
Convertido	7,2	10,1	19,7	38,3	75,8	151,3
GridSim	7,4	10,9	20,1	38,1	75,1	150,7
Diferença	5,55%	5,34%	4,56%	3,65%	1,70%	1,65%

Figura 4.3: Gráfico com os resultados do teste 1 com baixa comunicação e computação

4.1.4 Carga com alto volume de comunicação e alto índice de computação

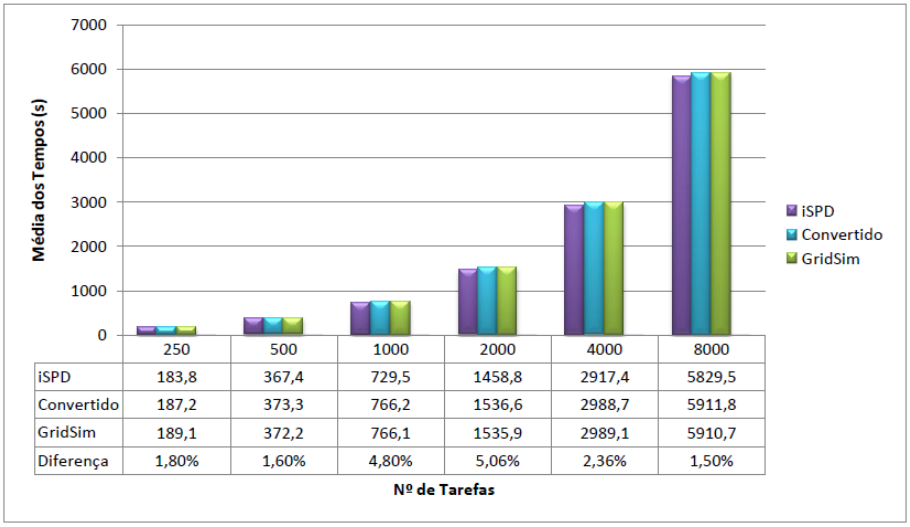
As tarefas deste teste tem 5000 Mflops de computação e 500 Mbits de comunicação. O resultados das médias dos tempos das simulações são apresentados no gráfico da Figura 4.6.

A diferença máxima entre o resultado do modelo simulado no iSPD e do convertido para o GridSim ocorreu com 500 tarefas e foi de 7,35%. Além disso, a diferença média foi de 3,6%.

4.2 Teste com um Modelo com Nós Remotos

A grade modelada para os seguintes testes é apresentada na Figura 4.7. Nela, existe um único mestre, representado pelo ícone de número zero, que envia tarefas a seus recursos escravos, que neste caso são cinco máquinas e um *cluster*. As máquinas estão representadas na Figura 4.7 com os números de 1 (um) a 5 (cinco) e o *cluster* está identificado com o número 6 (seis). Todos os seus recursos estão conectados diretamente ao mestre com excessão da máquina 1 (um), no qual existe um ícone de internet que o conecta.

Todas as máquinas tem uma capacidade de processamento de 500 MFlops. E estão conectadas com uma banda de 100 Mb/s, com uma latência de apenas 0,1s, inclusive a do ícone de internet.



	250	500	1000	2000	4000	8000
iSPD	183,8	367,4	729,5	1458,8	2917,4	5829,5
Convertido	187,2	373,3	766,2	1536,6	2988,7	5911,8
GridSim	189,1	372,2	766,1	1535,9	2989,1	5910,7
Diferença	1,80%	1,60%	4,80%	5,06%	2,36%	1,50%

Figura 4.4: Gráfico com os resultados do teste 1 com baixa comunicação e alta computação

Como sempre, foi feita a conversão do modelo para o GridSim através do Módulo de Exportação de Modelos, obtendo, como já dito anteriormente, um código em Java. Uma das diferenças deste modelo consiste na criação de *cluster* no sistema. Assim sendo, como pode ser observado na Figura 4.8, através da chamada do método *createResource* é construído o *cluster*.

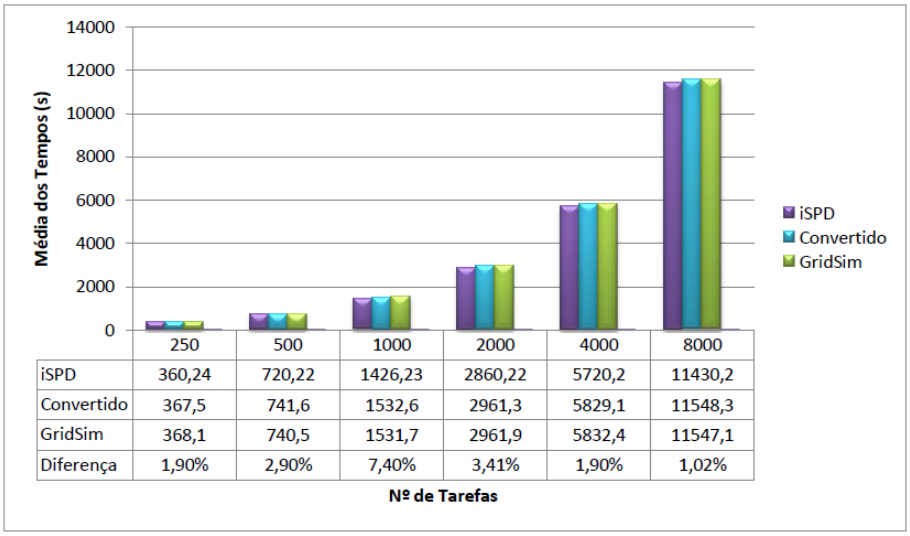
No modelo há ainda o ícone de internet que faz a conexão entre duas máquinas. Quando tal fato ocorre, é criado um elemento roteador no GridSim. Observando a Figura 4.9, pode ser verificada a criação de um roteador que liga os recursos *icon0* e *icon1*, através do método *attachHost*.

E então foi montado mais um modelo diretamente no GridSim. Os três modelos foram simulados variando-se a carga de trabalho.

4.2.1 Carga com baixo volume de comunicação e computação

As tarefas deste teste tem 50 Mflops de computação e também 3 Mbits de comunicação. O resultados das médias dos tempos das simulações são apresentados no gráfico da Figura 4.10.

A maior diferença entre o resultado do modelo simulado no iSPD e aquele convertido para o GridSim foi de 5,76% que ocorreu com 500 tarefas. E também, a diferença média foi de 4,25%.



	250	500	1000	2000	4000	8000
iSPD	360,24	720,22	1426,23	2860,22	5720,2	11430,2
Convertido	367,5	741,6	1532,6	2961,3	5829,1	11548,3
GridSim	368,1	740,5	1531,7	2961,9	5832,4	11547,1
Diferença	1,90%	2,90%	7,40%	3,41%	1,90%	1,02%

Figura 4.5: Gráfico com os resultados do teste 1 com alta comunicação e baixa computação

4.2.2 Carga com baixo volume de comunicação e alto índice de computação

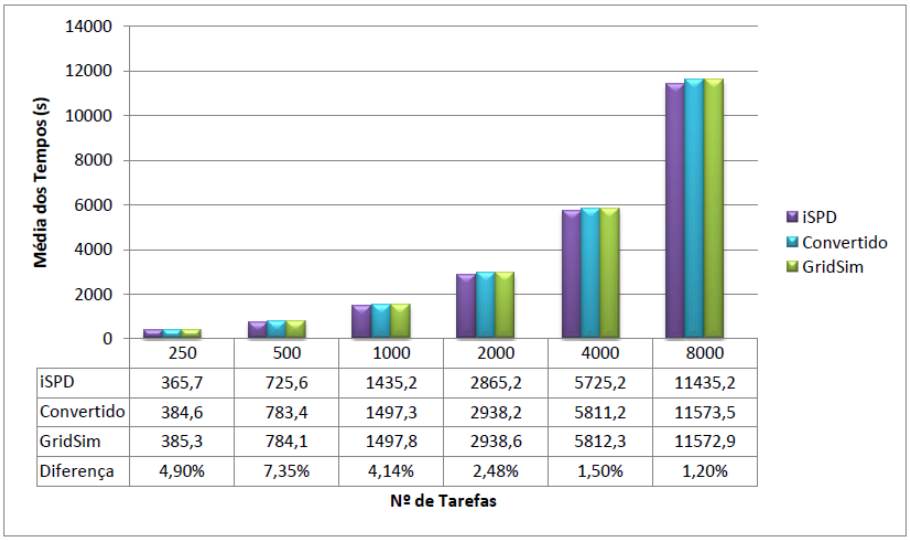
As tarefas deste teste tem 5000 Mflops de computação e 3 Mbits de comunicação. O resultados das médias dos tempos das simulações são apresentados no gráfico da Figura 4.11.

A diferença máxima entre os resultados do modelo simulado no iSPD e do convertido para o GridSim foi de apenas 5,8% que ocorreu com 1000 tarefas. Além disso a diferença média destes resultados foi de 4,15%.

4.2.3 Carga com alto volume de comunicação e baixo índice de computação

As tarefas deste teste tem 50 Mflops de computação e 500 Mbits de comunicação. O resultados das médias dos tempos das simulações são apresentados no gráfico da Figura 4.12.

A maior diferença entre o resultado do modelo simulado no iSPD e do convertido para o GridSim ocorreu com 500 tarefas e foi de 6,7%. E mais, a média das diferenças foi de 3,65%.



	250	500	1000	2000	4000	8000
iSPD	365,7	725,6	1435,2	2865,2	5725,2	11435,2
Convertido	384,6	783,4	1497,3	2938,2	5811,2	11573,5
GridSim	385,3	784,1	1497,8	2938,6	5812,3	11572,9
Diferença	4,90%	7,35%	4,14%	2,48%	1,50%	1,20%

Figura 4.6: Gráfico com os resultados do teste 1 com alta comunicação e computação

4.2.4 Carga com alto volume de comunicação e alto índice de computação

As tarefas deste teste tem 5000 Mflops de computação e 500 Mbits de comunicação. O resultados das médias dos tempos das simulações são apresentados no gráfico da Figura 4.13.

A diferença máxima entre o resultado do modelo simulado no iSPD e do convertido para o GridSim ocorreu com 500 tarefas e foi de 8,2%. Além disso, a diferença média foi de 5,22%.

4.3 Teste com um Modelo com Dois Servidores

O sistema distribuído construído para este terceiro teste conta com dois mestres e a comunicação entre todos os recursos da grade tem apenas 10Mb/s de banda, com exceção dos *clustes* que continuam com 100Mb/s. Como é possível observar na Figura 4.14, os mestres são os ícones zero, com os escravos cinco, sete, oito, nove, além da máquina dois com os escravos um, três, quatro e seis. Cada mestre tem uma comunicação exclusiva com cada um de seus escravos, e apenas as máquinas mestres tem ligação.

Foi realizada exportação deste modelo e montado um modelo diretamente no GridSim. Como há duas máquinas mestres neste exemplo, é mostrado na Figura 4.15, o código no GridSim onde aparece instanciados dois objetos da classe ”Mestre”, com os

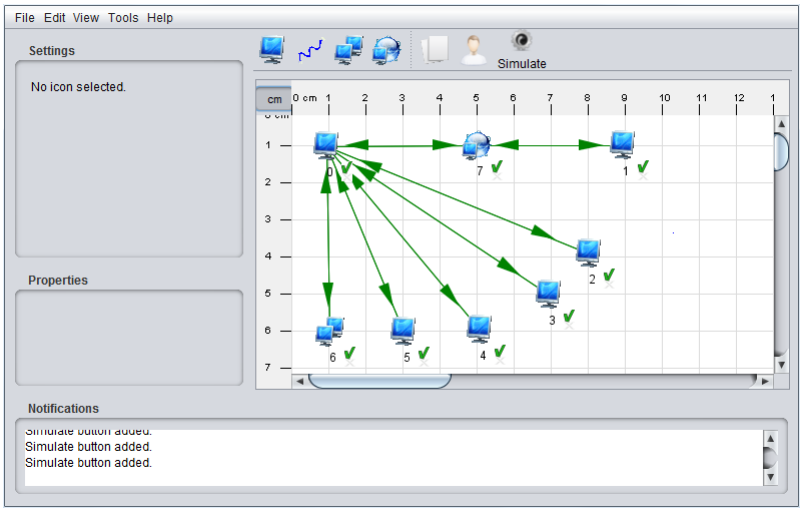


Figura 4.7: Grade do Segundo Teste

```
GridResource icon1 = createResource("icon1_",  baud_rate,  delay,  MTU, 1, (int)500.0);

GridResource icon6 = createResource("icon6_",  baud_rate,  delay,  MTU, 5, (int)500.0);

GridletList list = createGridlet();
```

Figura 4.8: Parte do Código Exportado do cluster

parâmetros que são seu nome, sua lista de escravos, sua lista de tarefas e escalonamento.

4.3.1 Carga com baixo volume de comunicação e computação

As tarefas deste teste têm 50 Mflops de computação e também 3 Mbits de comunicação. O resultados das médias dos tempos das simulações são apresentados no gráfico da Figura 4.16.

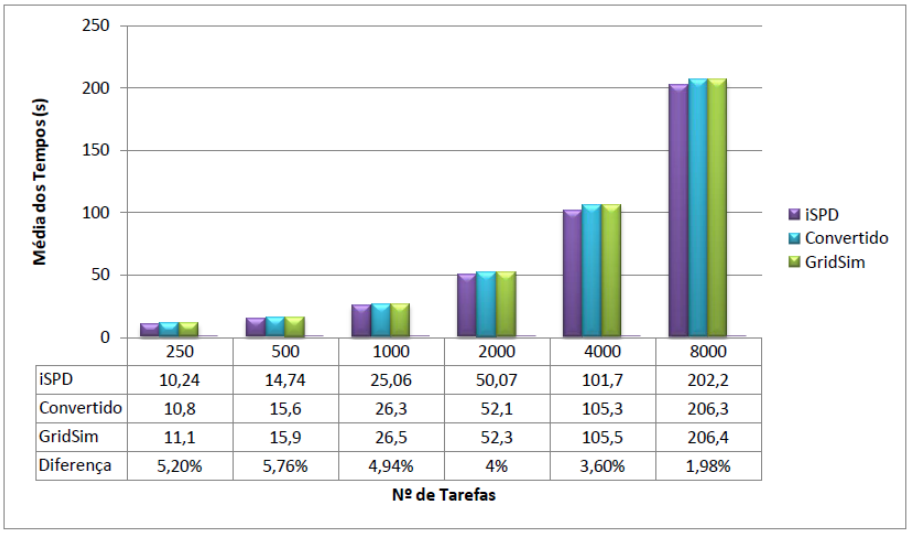
A maior diferença entre o resultado do modelo simulado no iSPD e aquele convertido para o GridSim foi de 1,7% que ocorreu com 4000 tarefas. E a diferença média foi de 1,35%.

4.3.2 Carga com baixo volume de comunicação e alto índice de computação

As tarefas deste teste têm 5000 Mflops de computação e 3 Mbits de comunicação. O resultados das médias dos tempos das simulações são apresentados no gráfico da Figura

```
Router r_icon7 = new RIPRouter("icon7_",trace_flag);
r_icon7.attachHost(icon1, resSched);
r_icon7.attachHost(icon0, resSched);
```

Figura 4.9: Parte do Código Exportado da Internet



	250	500	1000	2000	4000	8000
iSPD	10,24	14,74	25,06	50,07	101,7	202,2
Convertido	10,8	15,6	26,3	52,1	105,3	206,3
GridSim	11,1	15,9	26,5	52,3	105,5	206,4
Diferença	5,20%	5,76%	4,94%	4%	3,60%	1,98%

Figura 4.10: Gráfico com os resultados do teste 2 com baixa comunicação e computação

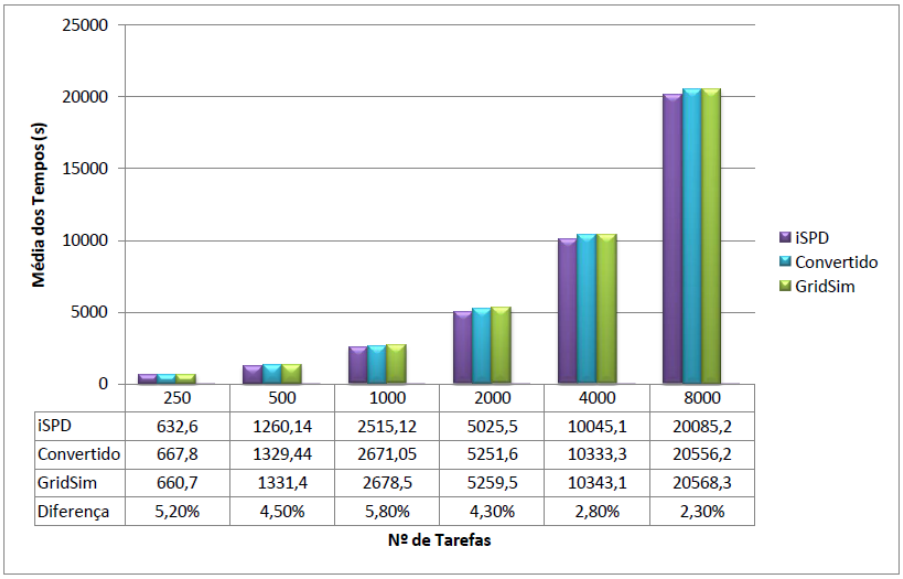
4.17.

A diferença máxima entre os resultados do modelo simulado no iSPD e do convertido para o GridSim foi de apenas 2,3% que ocorreu com 500 tarefas. Além disso, a diferença média destes resultados foi de 1,35%.

4.3.3 Carga com alto volume de comunicação e baixo índice de computação

As tarefas deste teste têm 50 Mflops de computação e 500 Mbits de comunicação. O resultados das médias dos tempos das simulações são apresentados no gráfico da Figura 4.18.

A maior diferença entre o resultado do modelo simulado no iSPD e do convertido para o GridSim ocorreu com 250 tarefas e foi de 2,5%. Além disso, a média das diferenças foi de 1,85%.



	250	500	1000	2000	4000	8000
iSPD	632,6	1260,14	2515,12	5025,5	10045,1	20085,2
Convertido	667,8	1329,44	2671,05	5251,6	10333,3	20556,2
GridSim	660,7	1331,4	2678,5	5259,5	10343,1	20568,3
Diferença	5,20%	4,50%	5,80%	4,30%	2,80%	2,30%

Figura 4.11: Gráfico com os resultados do teste com baixa comunicação e alta computação

4.3.4 Carga com alto volume de comunicação e alto índice de computação

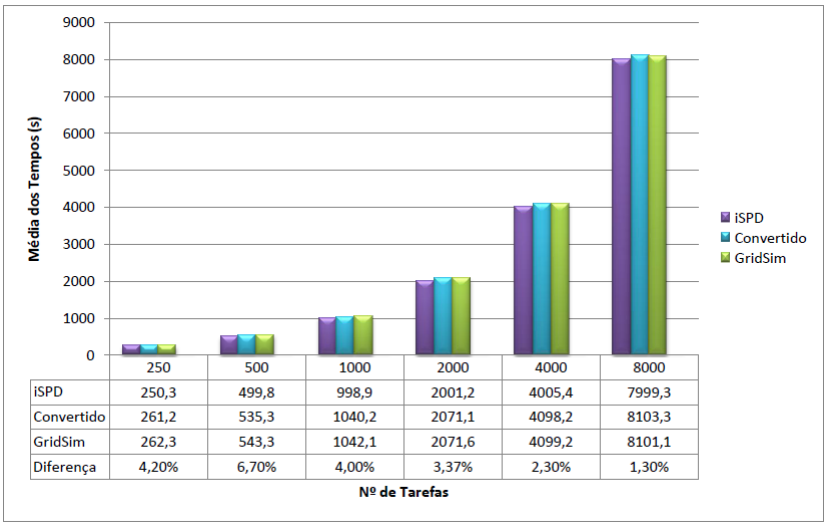
As tarefas deste teste têm 5000 Mflops de computação e 500 Mbits de comunicação. O resultados das médias dos tempos das simulações são apresentados no gráfico da Figura 4.19.

A diferença máxima entre o resultado do modelo simulado no iSPD e do convertido para o GridSim ocorreu com 250 tarefas e foi de 4,08%. E também, a diferença média foi de 2%.

4.4 Considerações finais

Os testes apresentados neste capítulo mostram que a exportação de diferentes modelos de grade computacional feitos no iSPD, bem como a carga de trabalho e sua distribuição por entre as máquinas, pode ser realizada sem problemas.

Tais testes mostram ainda que os tempos entre as simulações são equivalentes, ou seja, os cálculos para a obtenção dos tempos mostram-se parecidos com uma diferença pequena. Porém, as diferenças existentes mostram que os tempos de simulação do GridSim foram sempre maiores que os do iSPD. Isso ocorre porque aquele simulador considera mais atrasos na rede nos cálculos. Todavia nada que comprometa a eficiência

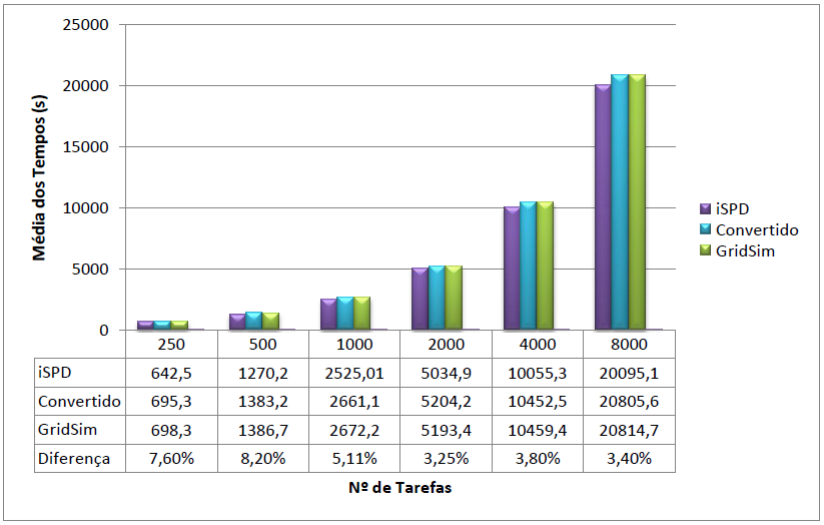


	250	500	1000	2000	4000	8000
iSPD	250,3	499,8	998,9	2001,2	4005,4	7999,3
Convertido	261,2	535,3	1040,2	2071,1	4098,2	8103,3
GridSim	262,3	543,3	1042,1	2071,6	4099,2	8101,1
Diferença	4,20%	6,70%	4,00%	3,37%	2,30%	1,30%

Figura 4.12: Gráfico com os resultados do teste 2 com alta comunicação e baixa computação

da simulação do iSPd ou mesmo do processo de exportação.

No próximo capítulo são apresentadas conclusões detalhadas de todo o trabalho e as direções futuras do simulador iSPD.



	250	500	1000	2000	4000	8000
iSPD	642,5	1270,2	2525,01	5034,9	10055,3	20095,1
Convertido	695,3	1383,2	2661,1	5204,2	10452,5	20805,6
GridSim	698,3	1386,7	2672,2	5193,4	10459,4	20814,7
Diferença	7,60%	8,20%	5,11%	3,25%	3,80%	3,40%

Figura 4.13: Gráfico com os resultados do teste 2 com alta comunicação e computação

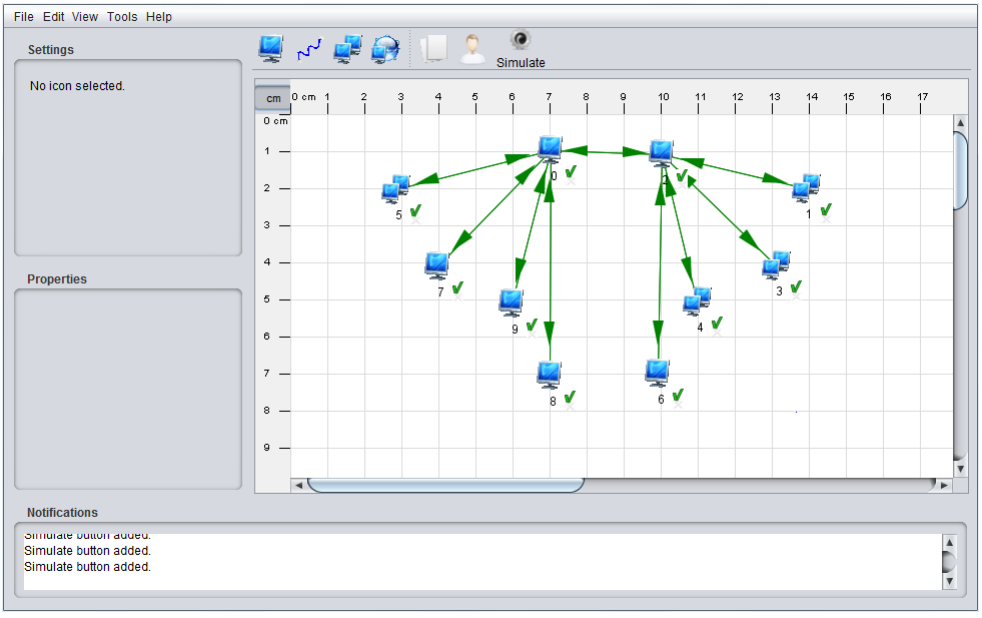
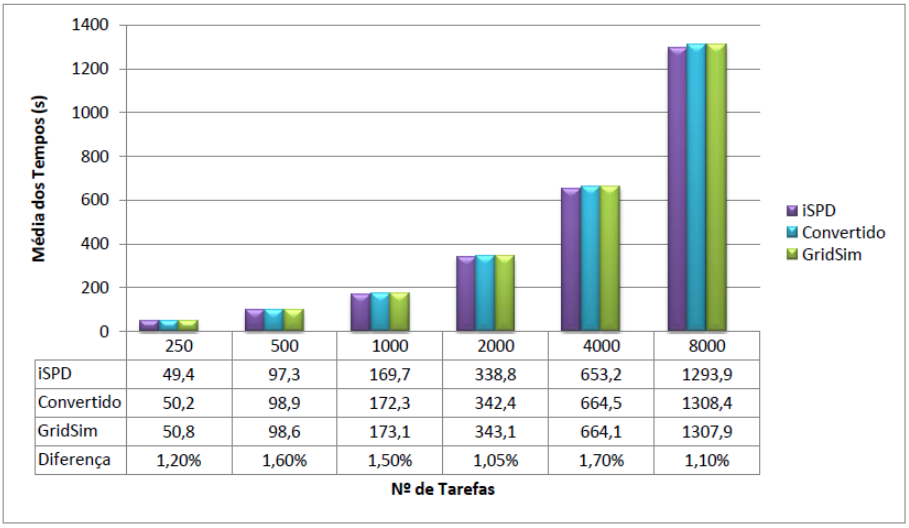


Figura 4.14: Grade do Terceiro Teste

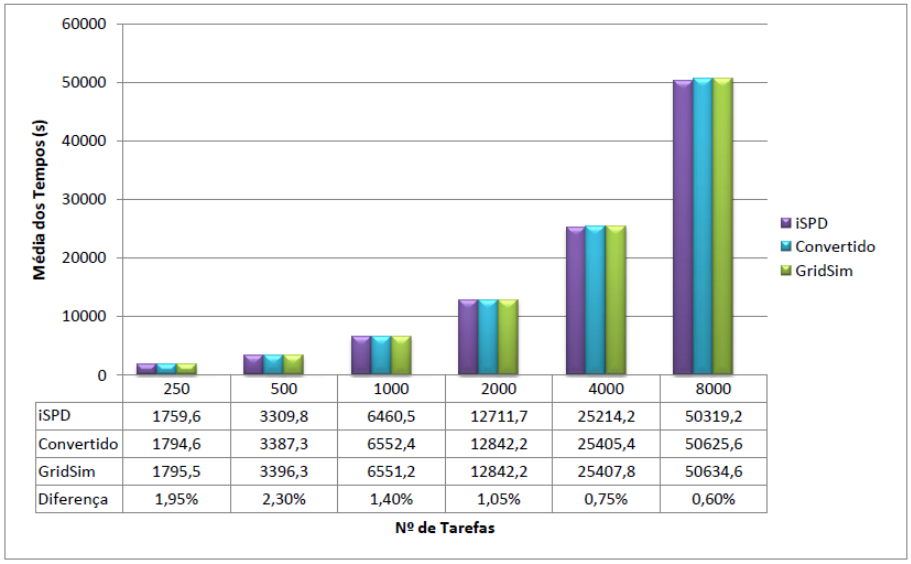
```
ArrayList esc1 = new ArrayList();
esc1.add(icon1);
esc1.add(icon3);
esc1.add(icon4);
esc1.add(icon6);

Mestre icon2 = new Mestre("icon2_", link, list, esc1, 1);
Router r_icon2 = new RIPRouter("router_1", trace_flag);
r_icon2.attachHost(icon2, resSched);

ArrayList esc2 = new ArrayList();
esc2.add(icon5);
esc2.add(icon7);
esc2.add(icon8);
esc2.add(icon9);

Mestre icon0 = new Mestre("icon0_", link, list, esc2, 1);
Router r_icon0 = new RIPRouter("router_2", trace_flag);
r_icon0.attachHost(icon0, resSched);
```

Figura 4.15: Parte do Código Exportado



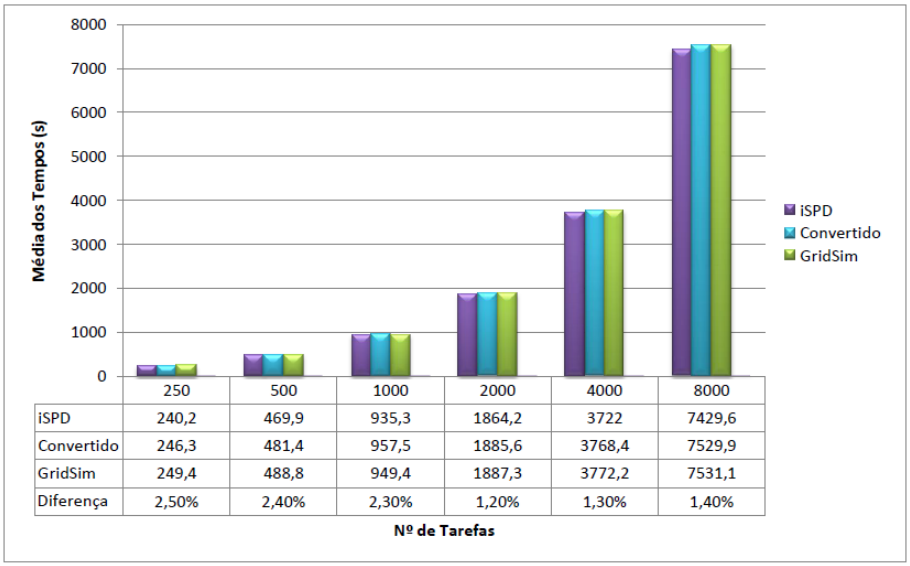
	250	500	1000	2000	4000	8000
iSPD	49,4	97,3	169,7	338,8	653,2	1293,9
Convertido	50,2	98,9	172,3	342,4	664,5	1308,4
GridSim	50,8	98,6	173,1	343,1	664,1	1307,9
Diferença	1,20%	1,60%	1,50%	1,05%	1,70%	1,10%

Figura 4.16: Gráfico com os resultados do teste 3 com baixa comunicação e computação



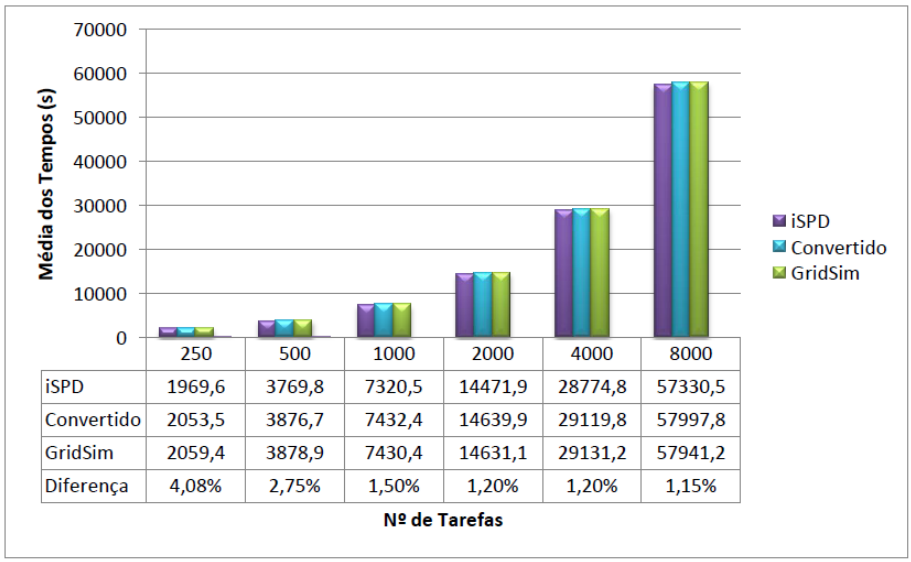
	250	500	1000	2000	4000	8000
iSPD	1759,6	3309,8	6460,5	12711,7	25214,2	50319,2
Convertido	1794,6	3387,3	6552,4	12842,2	25405,4	50625,6
GridSim	1795,5	3396,3	6551,2	12842,2	25407,8	50634,6
Diferença	1,95%	2,30%	1,40%	1,05%	0,75%	0,60%

Figura 4.17: Gráfico com os resultados do teste 3 com baixa comunicação e alta computação



	250	500	1000	2000	4000	8000
iSPD	240,2	469,9	935,3	1864,2	3722	7429,6
Convertido	246,3	481,4	957,5	1885,6	3768,4	7529,9
GridSim	249,4	488,8	949,4	1887,3	3772,2	7531,1
Diferença	2,50%	2,40%	2,30%	1,20%	1,30%	1,40%

Figura 4.18: Gráfico com os resultados do teste 3 com alta comunicação e baixa computação



	250	500	1000	2000	4000	8000
iSPD	1969,6	3769,8	7320,5	14471,9	28774,8	57330,5
Convertido	2053,5	3876,7	7432,4	14639,9	29119,8	57997,8
GridSim	2059,4	3878,9	7430,4	14631,1	29131,2	57941,2
Diferença	4,08%	2,75%	1,50%	1,20%	1,20%	1,15%

Figura 4.19: Gráfico com os resultados do teste 3 com alta comunicação e computação

Capítulo 5

Conclusões

Este trabalho apresentou os conceitos de análise e conversão de modelos de diferentes simuladores de grades computacionais, mais especificamente entre o iSPD e o GridSim. Para isso também foi mostrada suas principais características, arquitetura e como se realiza a modelagem em cada uma destas ferramentas. Além disso, foi feita uma revisão de Linguagems Formais e de Compiladores, já que são importantes para o entendimento deste projeto.

Assim, a maior contribuição deste trabalho se deve no aumento da facilidade de uso do iSPD. Desta maneira, um usuário que já tenha especificado seu modelo no iSPD, não precisa construí-lo novamente no GridSim, se assim o desejar, caso queria um segundo resultado de simulação. Ou mesmo caso queira apenas usar a Interface Gráfica do iSPD para uma modelagem mais fácil.

5.1 Considerações

O simulador de Grades Computacionais iSPD tem como principal objetivo facilitar a iteração que o usuário tem com a ferramenta de simulação, seja modelando seu sistema, ou mesmo recebendo os resultados, buscando em todos os casos ser simples e intuitiva em seu uso. Deste modo, este trabalho apresenta uma contribuição de mais uma funcionalidade desta ferramenta.

As conclusões que podem ser tiradas deste trabalho é que ele diminui o trabalho do usuário, que não precisa modelar novamente sua grade computacional no simulador externo ao iSPD. Além disso, possibilita a comparação dos resultados de simulações de ferramentas distintas. E também, a possibilidade do uso da interface gráfica para o usuário que deseja simular sua grade computacional no GridSim sem que se precise

utilizar-se da programação. Mais ainda, este trabalho contribui para aumentar uma funcionalidade que não é presente em outros simuladores, que é a possibilidade de interpretação de modelos externos e exportação de modelos internos.

5.2 Problemas Encontrados

Os principais problemas encontrados durante a realização do projeto são mostrados a seguir:

- Dificuldade no entendimento da correta construção do modelo no GridSim, já que é necessária a utilização de métodos específicos do simulador e modos próprios de utilizá-los. Embora há que se ressaltar que a documentação relativa ao GridSim é bastante ampla.
- Atualização da linguagem icônica do iSPD. Durante a realização do projeto, foi necessária a mudança da linguagem icônica utilizada no iSPD, simplificando, depois desta troca, a análise do modelo.

5.3 Direções Futuras

Dentre as próximas etapas de desenvolvimento deste projeto, destacam-se:

- Possibilidade de conversão de modelos de mais simuladores, tanto importando quanto exportando.
- Possibilidade de caracterização de tarefas, a fim de dar suporte à simulação de Grade de Dados.
- Possibilidade de simulação de Computação em nuvem, implicando em caracterizar tarefas que suportem tal sistema.

5.4 Publicações

Este trabalho vem sendo desenvolvido como Iniciação Científica desde 2011 e desde setembro de 2012 tem sido comtemplado com bolsa PIBIC/Reitoria. E foi publicado em congressos e eventos.

1. *Desenvolvimento de Plataforma de Simulação de Grades Computacionais: Módulo de Interpretação de Modelos Extermos e Módulo de Exportação de Modelos Icônicos*. Apresentado na III Escola Regional de Alto Desempenho de São Paulo (ERAD-SP, 2012). (STABILE et al., 2012)

2. *Desenvolvimento de Plataforma de Simulação de Grades Computacionais: Módulo de Exportação de Modelos Icônicos*. Apresentado no XXIV Congresso de Iniciação Científica da Unesp (XXIV CIC, 2012). (STABILE; LOBATO; MANACERO, 2012)

3. *Desenvolvimento de Plataforma de Simulação de Grades Computacionais: Módulo de Exportação de Modelos Icônicos*. Apresentado no XXV Congresso de Iniciação Científica da Unesp (XXIV CIC, 2013). (STABILE; LOBATO; MANACERO, 2013)

4. Relatórios técnicos de atividades apresentadas à Reitoria, em razão das atividades de iniciação científica sob financiamento da bolsa PIBIC/Reitoria.

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 Paulo, 2010. CD-ROM, p. 1–2.

ASHRAF, J.; ERLEBACH, T. A hybrid scheduling technique for grid workflows in advance reservation environments. In: *High Performance Computing and Simulation (HPCS), 2011 International Conference on*. [S.l.: s.n.], 2011. p. 98 –106.

BELL, W. et al. Simulation of dynamic grid replication strategies in optorsim. In: *Proc. of the ACM/IEEE Workshop on Grid Computing*. [S.l.]: Springer-Verlag, 2002. ISBN 3-540-58021-2.

BELL, W. H. et al. Optorsim - a grid simulator for studying dynamic data replication strategies. *International Journal of High Performance Computing Applications*, 2003.

BRICKS. Bricks: A performance evaluation system for grid computing scheduling algorithms. Disponível em <http://ninf.apgrid.org/bricks/>. 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.

CAMERON, D. G. et al. Evaluating scheduling and replica optimisation strategies in optorsim. *GRID '03 Proceedings of the 4th International Workshop on Grid Computing*, IEEE Computer Society Washington, DC, USA, p. 53, 2003.

CASANOVA, H. Simgrid: a toolkit for the simulation of application scheduling. In: *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001*. [S.l.: s.n.], 2001. p. 430–437.

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

CRICHLOW, J. M. *An Introduction to Distributed and parallel computing*. [S.l.]: Prentice Hall, 1988.

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

GANGSIM. Gangsim: A simulator for grid scheduling studies with support for uslas. Disponível em <http://people.cs.uchicago.edu/ cldumitr/GangSim/>. 2012.

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

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

GUERRA, A.; AOKI, V. Plataforma de simulação de grades computacionais: Interface icônica e interpretador de modelos externos. In: . Monografia de Conclusão de Curso, Universidade Estadual Paulista (UNESP). São José do Rio Preto: [s.n.], 2010.

JAIN, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. 2nd edition. ed. [S.l.]: John Wiley & Sons, 1991.

JAVA. Java home page. Disponível em <http://www.java.com/pt BR/>. 2012.

KUMAR, S.; KUMAR, N.; KUMAR, P. Genetic algorithm for network-aware job scheduling in grid environment. In: *Recent Advances in Intelligent Computational Systems (RAICS), 2011 IEEE*. [S.l.: s.n.], 2011. p. 615 –620.

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. et al. Scheduler simulation using ispd, an iconic-based computer grid simulator. In: *Proc. of Intl Symp on Computers and Communications*. Capadocia, Turkey: [s.n.], 2012. (ISCC'12, CDROM), p. 1–6.

OLIVEIRA, L. J. *Comparação de ferramentas de simulação de grades computacionais*. Tese (Monografia) — Universidade Estadual Paulista, São José do Rio Preto, 2007.

OPTORSIM. Simulating data access optimization algorithms - optorsim. Disponível em <http://optorsim.sourceforge.net/>. 2012.

SETHI RAVI, U.; JEFFREY; LAM. *Compiladores: princípios, técnicas e ferramentas*. [S.l.]: Pearson Addison Wesley, 2008.

SILVA, D. Implementação de um banco de traces de cargas de trabalho para o ispd. In: . Monografia, Universidade Estadual Paulista (UNESP). São José do Rio Preto: [s.n.], 2012.

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

STABILE, R. et al. Desenvolvimento de plataforma de simulação de grades computacionais: Módulo de interpretação de modelos externos e módulo de exportação de modelos icônicos. In: *Anais da III Escola Regional de Alto Desempenho*. [S.l.: s.n.], 2012.

STABILE, R.; LOBATO, R.; MANACERO, A. Desenvolvimento de plataforma de simulação de grades computacionais: Módulo de exportação de modelos icônicos. In: *XXIV Congresso de Iniciação Científica*. [S.l.: s.n.], 2012.

STABILE, R.; LOBATO, R.; MANACERO, A. Desenvolvimento de plataforma de simulação de grades computacionais: Módulo de exportação de modelos icônicos. In: *XXV Congresso de Iniciação Científica*. [S.l.: s.n.], 2013.

SULISTIO, A. et al. A toolkit for modelling and simulating data grids: an extension to gridsim. *Concurr. Comput. : Pract. Exper.*, John Wiley and Sons Ltd., Chichester, UK, v. 20, p. 1591–1609, September 2008. ISSN 1532-0626. Disponível em: <http://-/dl.acm.org/citation.cfm?id=1400265.1400270>.

TAKEFUSA, A.; MATSUOKA, S. Performance issues in client-server global computing. *International Workshop on Global and Cluster Computing (WGCC'2000)*, 2000.

TANENBAUM, A. S.; STEEN, M. V. *Sistemas Distribuídos Princípios e Paradigmas*. 2nd. ed. [S.l.]: Ed. Pearson, 2007.

Apêndice A

Modelo no GridSim

Neste anexo, aparece um modelo no GridSim no qual há um mestre que distribui 2000 tarefas a seus quatro recursos. O código foi dividido em partes para ser melhor explicado.

Na Figura A.1 aparece o código de escalonamento Workqueue feito no modelo, os métodos *GridSim.GridletSubmit()* e *Gridsim.GridletReceive()* são usados, respectivamente para mandar e receber tarefas dos recursos, esta parte do código perterce a classe Mestre do modelo. O escalonamento é feito a partir da lista Escravos_, primeiramente mandando tarefas a cada um deles e posteriormente, recebendo os resultados dos processamentos

Já na Figura A.2, é mostrado todo o processo de criação de recursos, chamando o método *createResource*, passando como parâmetro o nome do recurso, especificações da comunicação, quantidade de máquina e capacidade de processamento. Então todos estes recursos são adicionados em uma lista de escravos. Logo em seguida, é instanciado um objeto da classe mestre, passando como parâmetro o *link*, que é a comunicação com seus escravos, lista de suas tarefas e lista de seus escravos.

Assim, na Figura A.3 é mostrado como são conectados os recursos ao mestre, através do roteador *r_icon0*. Para cada um dos escravos do modelo, é usado o método *attachHost*.

Mais ainda, na Figura A.4 aparece como se cria máquinas no GridSim. Primeiramente é criada a lista *mList* onde será armazenado cada uma delas. Em um laço de repetição são criadas máquinas a partir dos parâmetros id, número de elementos de processamento e capacidade computacional e então são adicionadas em tal lista.

Então na Figura A.5 é mostrado todo o código do método *createResource*, a criação de recursos, através da instanciamento de um objeto da classe *GridResouce*, própria do GridSim.

E finalmente, na Figura A.6 aparece como é criado a carga de trabalho, no caso a criação de Gridlets. A partir de um laço, com a quantidade total de tarefas a serem criadas, são feitos primeiramente *length*, que é quantidade a ser computada e *file_size*, que é a taxa necessária para o transporte desta tarefa até o recurso. Assim, é instanciado um *Gridlet*, para a criação efetiva da carga.

```
    if (this.Escal == 1) { //O escalonador Ã© Workqueue
      int cont = 0;
      int k;
      Gridlet gl;
      for (k = 0; k < Escravos_.size() && cont < list.size(); k++, cont++) {
        int num = resList.get(k).get_id();
        list.get(cont).setUserID(this.ID_);
        System.out.println("Submetendo tarefa" + list.get(cont).getGridletID() + " Para
REcurso" + num);

        super.gridletSubmit((Gridlet) list.get(cont), num, 0.0, true);

      }
      int res = 0;
      while (cont < list.size() || res < list.size()) {
        gl = super.gridletReceive();
        System.out.println("Recebi de " + gl.getResourceID() + " Status:" +
gl.getGridletStatus()+"No tempo:"+ GridSim.clock());

        res++;
        int num = gl.getResourceID();
        System.out.println("Relogio:"+GridSim.clock());
        if (cont < list.size()) {
          System.out.println("Enviando:"+  list.get(cont).getGridletID() + " PAra: "+ num);
          list.get(cont).setUserID(this.ID_);
          super.gridletSubmit((Gridlet) list.get(cont), num, 0.0, true);
```

Figura A.1: Código do Escalonamento Feito no Mestre

```
double delay = 0.1;
int MTU = 1500;
FIFOScheduler resSched1 = new FIFOScheduler("GridResSched1");
GridResource icon1 = createResource("icon1_", baud_rate, delay, MTU, 1, (int) 400);
GridResource icon2 = createResource("icon2_", baud_rate, delay, MTU, 1, (int) 400);
GridResource icon3 = createResource("icon3_", baud_rate, delay, MTU, 1, (int) 400);
GridResource icon4 = createResource("icon4_", baud_rate, delay, MTU, 1, (int) 400);

Link link = new SimpleLink("link_", 400000, 0.1, 1500);

ArrayList esc1 = new ArrayList();
esc1.add(icon1);
esc1.add(icon2);
esc1.add(icon3);
esc1.add(icon4);

GridletList list = createGridlet();
FIFOScheduler resSched_ = new FIFOScheduler("GridResSched_");
Mestre1 icon0 = new Mestre1("icon0_", link, list, esc1, 1);
Router r_icon0 = new RIPRouter("router_1", trace_flag);
r_icon0.attachHost(icon0, resSched_);
ResourceUserList userList = createGridUser();
```

Figura A.2: Código da instanciação de Recursos e do Mestre do Modelo

```
FIFOScheduler rSched = new FIFOScheduler("r_Sched");
FIFOScheduler rSched_ = new FIFOScheduler("r_Sched_");

r_icon0.attachHost(icon2, rSched);
r_icon0.attachHost(icon1, rSched_);
r_icon0.attachHost(icon3, rSched);
r_icon0.attachHost(icon4, rSched);

GridSim.startGridSimulation();
```

Figura A.3: Código da Conexão dos Recursos ao Roteador

```
MachineList mList = new MachineList();
for (int i = 0; i < n_maq; i++) {

  mList.add(new Machine(i, 1, cap));
```

Figura A.4: Código da Criação de Máquinas

```
String arch = "Sun Ultra";
String os = "Solaris";
double time_zone = 9.0;
double cost = 3.0;

ResourceCharacteristics resConfig = new ResourceCharacteristics(arch, os, mList,
ResourceCharacteristics.TIME_SHARED, time_zone, cost);

long seed = 11L * 13 * 17 * 19 * 23 + 1;
double peakLoad = 0.0;
double offPeakLoad = 0.0;
double holidayLoad = 0.0;

LinkedList Weekends = new LinkedList();
Weekends.add(new Integer(Calendar.SATURDAY));
Weekends.add(new Integer(Calendar.SUNDAY));
LinkedList Holidays = new LinkedList();
GridResource gridRes = null;

try {
  gridRes = new GridResource(name, new SimpleLink(name + "_link", baud_rate, delay,
MTU), seed, resConfig, peakLoad, offPeakLoad, holidayLoad, Weekends, Holidays);

} catch (Exception e) {
  e.printStackTrace();
}
```

Figura A.5: Código da Criação de Recursos

```
  private static GridletList createGridlet() {
    double length;
    long file_size;
    Random random = new Random();
    GridletList list = new GridletList();

    int num_tar = 2000;
    for (int i = 0; i < num_tar; i++) {

      length = GridSimRandom.real(4.0, 1.0, 1.0, random.nextDouble());
      file_size = (long) GridSimRandom.real(4.0, 1.0, 1.0, random.nextDouble());
      Gridlet gridlet = new Gridlet(i, 4, 125, 125);
      list.add(gridlet);
      gridlet.setUserID(0);

    }
    return list;

  }
}
```

Figura A.6: Código da Criação de Tarefas

Apêndice B

Regras do Modelo Icônico

As regras de formação do modelo icônico, que contém todas as especificações de informações dos elementos da grade a ser simulada, bem como a carga de trabalho, são apresentadas a seguir:

<!ELEMENT system (owner*,(machine|link|cluster|internet)*,load?)>
<!ATTLIST system version CDATA #REQUIRED>

<!ELEMENT owner EMPTY>
<!ATTLIST owner id CDATA #REQUIRED>

<!ELEMENT position EMPTY>
<!ATTLIST position x CDATA #REQUIRED>
<!ATTLIST position y CDATA #REQUIRED>

<!ELEMENT icon id EMPTY>
<!ATTLIST icon id global CDATA #REQUIRED>
<!ATTLIST icon id local CDATA #REQUIRED>

<!ELEMENT slave EMPTY>
<!ATTLIST slave id CDATA #REQUIRED>

<!ELEMENT master (slave*)>
<!ATTLIST master scheduler CDATA —>

<!ELEMENT machine (master?,position,icon id)>
<!ATTLIST machine id ID #REQUIRED>
<!ATTLIST machine power CDATA 0.0 >
<!ATTLIST machine load CDATA 0.0 >
<!ATTLIST machine owner CDATA user1 >

```
<!ELEMENT cluster (position,icon id)>
<!ATTLIST cluster id ID #REQUIRED>
<!ATTLIST cluster nodes CDATA 0 >
<!ATTLIST cluster power CDATA 0.0 >
<!ATTLIST cluster bandwidth CDATA 0.0 >
<!ATTLIST cluster latency CDATA 0.0 >
<!ATTLIST cluster scheduler CDATA — >
<!ATTLIST cluster owner CDATA user1>
<!ATTLIST cluster master CDATA true>

<!ELEMENT internet (position,icon id)>
<!ATTLIST internet id ID #REQUIRED>
<!ATTLIST internet bandwidth CDATA 0.0 >
<!ATTLIST internet load CDATA 0.0 >
<!ATTLIST internet latency CDATA 0.0 >

<!ELEMENT connect EMPTY>
<!ATTLIST connect origination CDATA #REQUIRED>
<!ATTLIST connect destination CDATA #REQUIRED>

<!ELEMENT link (connect,position,position,icon id)>
<!ATTLIST link id ID #REQUIRED>
<!ATTLIST link bandwidth CDATA 0.0 >
<!ATTLIST link load CDATA 0.0 >
<!ATTLIST link latency CDATA 0.0 >

<!ELEMENT load (file|random|node+)>

<!ELEMENT trace EMPTY>
<!ATTLIST trace file path CDATA #REQUIRED>

<!ELEMENT random (size,size)>
<!ATTLIST random owner CDATA user1 >
<!ATTLIST random tasks CDATA 0 >
<!ATTLIST random time arrival CDATA 0 >

<!ELEMENT node (size,size)>
<!ATTLIST node application CDATA application0 >
<!ATTLIST node owner CDATA user1 >
<!ATTLIST node id master CDATA ???>
<!ATTLIST node tasks CDATA 0 >

<!ELEMENT size EMPTY>
```

<!ATTLIST size type (computing|communication) #REQUIRED>
<!ATTLIST size maximum CDATA 0 >
<!ATTLIST size average CDATA 0 >
<!ATTLIST size minimum CDATA 0 >
<!ATTLIST size probability CDATA 0.0 >