

UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

PORTAL PARA SUBMISSÃO DE TAREFAS UTILIZANDO
GRADE COMPUTACIONAL OPORTUNISTA

DIOGO EDEGAR MAFRA

BLUMENAU
2010

2010/2-11

DIOGO EDEGAR MAFRA

**PORTAL PARA SUBMISSÃO DE TAREFAS UTILIZANDO
GRADE COMPUTACIONAL OPORTUNISTA**

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Paulo Fernando da Silva - Orientador

**PORTAL PARA SUBMISSÃO DE TAREFAS UTILIZANDO
GRADE COMPUTACIONAL OPORTUNISTA**

Por

DIOGO EDEGAR MAFRA

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente: _____
Prof. Paulo Fernando da Silva, Mestre – Orientador, FURB

Membro: _____
Prof. Roosevelt dos Santos Junior, Especialista – FURB

Membro: _____
Prof. Mauro Marcelo Mattos, Doutor – FURB

Blumenau, 09 de dezembro de 2010

A imaginação é mais importante que o conhecimento. O conhecimento é limitado. A imaginação envolve o mundo.

Albert Einstein

RESUMO

Este trabalho apresenta o desenvolvimento de um software para a submissão e processamento de tarefas utilizando uma grade computacional oportunista. É desenvolvido o portal para a submissão de tarefas, bem como os componentes necessários para a distribuição e processamento das tarefas na grade. As tarefas submetidas à grade são escritas utilizando o .NET Framework, fazendo uso de uma API própria da grade computacional desenvolvida. Também é apresentada a implementação dos mecanismos de segurança utilizados na grade para impedir que tarefas executem códigos maliciosos.

Palavras-chave: Grade. Processamento. Gerenciamento. Distribuição. Ocioso. *Bag-of-tasks*.

ABSTRACT

This work presents the development of a software for submission and processing of the tasks using an opportunist computational grid. It was developed the portal for submission of tasks, and also the components required for distribution and processing of tasks of the grid. The tasks submitted to the grid are written with .NET Framework, using an API of the grid that has been developed. It's also presented the implementation of the security mechanisms used to prevent tasks from running malicious code.

Key-words: Grid. Processing. Management. Distribution. Idle. Bag-of-tasks

LISTA DE ILUSTRAÇÕES

Figura 1 - Desempenho de acordo com granularidade das tarefas	18
Figura 2 - Desperdício de processamento de acordo com granularidade das tarefas	19
Figura 3 - Componentes da grade.....	25
Figura 4 - Diagrama de classes da API.....	26
Quadro 1 - Exemplo de definição do <i>job</i>	28
Figura 5 - Exemplo de pacote de <i>job</i>	29
Quadro 2 - Exemplo de grade com três tarefas pendentes	30
Quadro 3 - Exemplo de grade processando três tarefas.....	30
Quadro 4 - Exemplo de grade processando três tarefas com replicação	31
Quadro 5 - Estado da grade no momento inicial do exemplo	32
Quadro 6 - Estado da grade após o processamento de uma tarefa	32
Quadro 7 - Estado da grade após o processamento de duas tarefas	33
Figura 6 - Diagrama de casos de uso do portal.....	34
Quadro 8 - Detalhamento do caso de uso UC01.01	34
Quadro 9 - Detalhamento do caso de uso UC01.02	35
Quadro 10 - Detalhamento do caso de uso UC01.03	35
Quadro 11 - Detalhamento do caso de uso UC01.04	36
Quadro 12 - Detalhamento do caso de uso UC01.05	37
Quadro 13 - Detalhamento do caso de uso UC01.06	38
Quadro 14 - Detalhamento do caso de uso UC01.07	39
Figura 7 - Diagrama de casos de uso do executor	40
Quadro 15 - Detalhamento do caso de uso UC02.01	40
Quadro 16 - Detalhamento do caso de uso UC02.02	41
Figura 8 - Diagrama de classes do executor	42
Figura 9 - Diagrama de classes do componente de segurança do executor.....	43
Figura 10 - Diagrama de classes do portal	44
Figura 11 - Diagrama de classes do distribuidor	45
Figura 12 - Diagrama de classes de <i>web services</i>	46
Figura 13 - Diagrama de seqüência do executor	47
Figura 14 - Diagrama de estados de uma tarefa	48
Quadro 17 - Testes unitários.....	49

Quadro 18 - Trecho da classe <code>ExecutorService</code>	50
Quadro 19 - Método <code>LoadTask</code> da classe <code>SecureTaskLoader</code>	51
Quadro 20 - Classe <code>Group</code>	53
Quadro 21 - Classe <code>GroupMap</code>	53
Quadro 22 - Método <code>GetPendingTask</code> da classe <code>DistributionService</code>	54
Quadro 23 - Classe <code>TaskInfo</code>	55
Quadro 24 - Classe <code>GroupInfo</code>	55
Quadro 25 - Método <code>Sort</code> da classe <code>DistributionData</code>	56
Quadro 26 - Classe <code>GroupController</code>	57
Quadro 27 - Arquivo de visualização da tela de listagem de grupos	58
Figura 15 - Tela de listagem de grupos	59
Figura 16 - Tela de inclusão de grupo	60
Figura 17 - Tela de listagem de usuários	60
Figura 18 - Tela de inclusão de usuário.....	61
Figura 19 - Tela de <i>download</i> do executor	61
Figura 20 - Tela de configuração do executor	62
Figura 21 - Tela principal do executor	62
Figura 22 - Tela para obtenção da API da grade	63
Figura 23 - Projeto de uma tarefa	64
Quadro 28 - Implementação da tarefa <code>AntiAliasingTask</code>	64
Quadro 29 - XML de definição do <i>job</i>	65
Figura 24 - Estrutura do arquivo do <i>job</i>	66
Figura 25 - Tela de submissão de <i>jobs</i>	66
Figura 26 - Tela de consulta de <i>jobs</i>	67
Figura 27 - Tela de detalhes do <i>job</i>	67
Figura 28 - Tela de detalhes do <i>job</i> com resultados	68
Figura 29 - Tarefa com código malicioso.....	68
Figura 30 - Processamento da tarefa com erro	69

LISTA DE TABELAS

Tabela 1 - Resultado de testes da grade.....	69
--	----

LISTA DE SIGLAS

API – Application Programming Interface

BOINC – Berkeley Open Infrastructure for Network Computing

LAN – Local Area Network

RF – Requisito Funcional

RNF – Requisito Não Funcional

SQL – Structured Query Language

XML - eXtensible Markup Language

SUMÁRIO

1 INTRODUÇÃO.....	12
1.1 OBJETIVOS DO TRABALHO	13
1.2 ESTRUTURA DO TRABALHO	13
2 FUNDAMENTAÇÃO TEÓRICA	15
2.1 GRADES COMPUTACIONAIS	15
2.2 APLICAÇÕES DO TIPO <i>BAG-OF-TASKS</i>	16
2.3 ESCALONAMENTO DAS TAREFAS DA GRADE	16
2.3.1 Workqueue	17
2.3.2 Workqueue com Replicação	17
2.3.3 Comparação de desempenho.....	18
2.4 SEGURANÇA NA EXECUÇÃO DE TAREFAS NA GRADE	19
2.5 ASP.NET MVC.....	20
2.6 TRABALHOS CORRELATOS.....	21
2.6.1 SETI@home.....	21
2.6.2 BOINC	21
2.6.3 Globus Toolkit	22
3 DESENVOLVIMENTO DO SOFTWARE	24
3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO.....	24
3.2 ESPECIFICAÇÃO	25
3.2.1 Componentes da grade	25
3.2.2 Grupos e usuários do sistema.....	26
3.2.3 API utilizada na criação de tarefas.....	26
3.2.4 Formato do <i>job</i>	27
3.2.4.1 Arquivo XML com a definição do <i>job</i>	28
3.2.4.2 Pacote do <i>job</i>	29
3.2.5 Distribuição das tarefas	30
3.2.5.1 Workqueue com Replicação	30
3.2.5.2 Contabilização dos pontos de processamento.....	31
3.2.6 Casos de uso.....	33
3.2.7 Diagramas de classes.....	41
3.2.7.1 Executor	42

3.2.7.2 Portal.....	43
3.2.7.3 Distribuidor.....	45
3.2.7.4 Web services.....	45
3.2.8 Diagrama de sequência.....	47
3.2.9 Diagrama de estados.....	47
3.3 IMPLEMENTAÇÃO.....	48
3.3.1 Bibliotecas e ferramentas utilizadas.....	48
3.3.2 Implementação dos testes unitários.....	49
3.3.3 Implementação do executor.....	49
3.3.4 Sistema Web.....	52
3.3.4.1 Acesso a dados.....	52
3.3.4.2 Distribuidor.....	53
3.3.4.3 Implementação do portal.....	56
3.4 OPERACIONALIDADE DA IMPLEMENTAÇÃO.....	58
3.4.1 Ambiente.....	58
3.4.1.1 Criar grupo e usuário.....	59
3.4.1.2 Iniciar executor nas máquinas da grade.....	61
3.4.2 Execução do <i>job</i>	63
3.4.2.1 Obter API da grade.....	63
3.4.2.2 Criar tarefa.....	63
3.4.2.3 Criar <i>job</i>	65
3.4.2.4 Submeter <i>job</i>	66
3.4.2.5 Execução do <i>job</i> na grade.....	67
3.4.2.6 Obter resultados.....	67
3.4.2.7 Submeter tarefa com código malicioso.....	68
3.5 RESULTADOS E DISCUSSÃO.....	69
4 CONCLUSÕES.....	71
4.1 EXTENSÕES.....	72
REFERÊNCIAS BIBLIOGRÁFICAS.....	74

1 INTRODUÇÃO

Nas últimas décadas os computadores tem evoluído muito na sua capacidade de processamento. Um computador pessoal em 2001 é tão rápido quanto um supercomputador em 1990. Apesar disso, a quantidade de dados a serem processados também cresceu neste período, o que faz com que muitas tarefas demandem muito tempo de processamento para um simples computador (FOSTER, 2002).

Para resolver este problema, pode-se optar por utilizar um supercomputador composto por vários processadores. Essa é uma solução que traz alto desempenho, porém com um custo muito alto. Supercomputadores utilizam hardwares especiais e normalmente de alto custo. Uma solução alternativa aos supercomputadores é fazer uso de um *cluster* de alto desempenho ou de uma grade computacional.

Um *cluster* é composto de vários computadores conectados localmente, onde cada computador é dedicado exclusivamente ao processamento. São máquinas totalmente dedicadas, normalmente utilizam sistemas operacionais desenvolvidos especialmente a este fim e estão interligadas por uma *Local Area Network* (LAN) de alta velocidade.

Uma grade computacional, por sua vez, é uma combinação heterogênea de computadores. Cada computador da grade pode ter uma arquitetura diferente e estar disposto geograficamente distante um do outro. A grade permite o uso de computadores diferentes e dispostos em vários locais (BUYAYA, 2002, p. 9). Uma das subcategorias de grade computacional são as grades computacionais oportunistas, que fazem uso do tempo ocioso dos computadores para a execução das tarefas (GOLDCHLEGER, 2004, p. 3).

Diante do exposto, este trabalho apresenta um conjunto de softwares que gerenciam uma grade computacional que utilize os recursos ociosos de diversos computadores para executar o processamento das tarefas submetidas a ela. Nesta grade, as tarefas são desenvolvidas utilizando o Microsoft .NET Framework fazendo acesso a uma *Application Programming Interface* (API) própria da grade, e submetidas por meio de um portal acessível via internet. O portal permite ao usuário submeter as tarefas, acompanhar o estado da execução das mesmas e obter o resultado ao final do processamento. As tarefas são do tipo *bag-of-tasks*, as quais são mais bem definidas na fundamentação teórica. Cada computador conectado à grade utiliza seu tempo ocioso para processar tarefas submetidas por meio do portal. O número de computadores conectados à grade pode ser dinâmico, podendo-se adicionar ou remover computadores a qualquer momento, aumentando ou diminuindo a

capacidade de processamento disponível sem qualquer problema no processamento das tarefas.

Neste trabalho são implementados os softwares necessários para a submissão, distribuição e processamento das tarefas da grade.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver um conjunto de softwares que permitam a criação de uma grade computacional que utilize o tempo ocioso de computadores para o processamento de tarefas.

Os objetivos específicos do trabalho são:

- a) disponibilizar um portal web para submissão de tarefas a serem processadas;
- b) distribuir o processamento das tarefas entre os computadores conectados à grade;
- c) permitir o uso do tempo ocioso de computadores para executar o processamento das tarefas;
- d) permitir o acompanhamento do estado de execução das tarefas;
- e) permitir a obtenção dos resultados da execução da tarefa submetida.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está estruturado em quatro capítulos. No segundo capítulo é apresentada a fundamentação teórica utilizada para o desenvolvimento do trabalho. Nele é apresentada a definição de grade computacional e aplicações do tipo *bag-of-tasks*. São discutidos diferentes algoritmos de escalonamento utilizados em grades computacionais, comparando o desempenho entre eles. Também é apresentada a biblioteca ASP.NET MVC, utilizada na implementação do portal para submissão de tarefas e o modo de executar tarefas de forma segura em .NET. O capítulo traz ainda uma descrição de alguns trabalhos correlatos.

No terceiro capítulo é apresentado o desenvolvimento do software. Inicialmente são expostos os requisitos a serem atendidos, a seguir a especificação das diversas áreas do software, sua implementação e a sua operacionalidade.

Por fim, no quarto capítulo, são apresentadas as conclusões e sugestões de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é apresentada a fundamentação teórica necessária para o desenvolvimento da grade computacional, bem como do portal de submissão de tarefas. É feita a definição de uma grade computacional e das tarefas do tipo *bag-of-tasks* que serão processadas pela grade. São expostos os tipos de algoritmos utilizados no escalonamento da grade comparando o desempenho deles de acordo com o tipo de tarefa e a heterogeneidade da grade. Aborda-se a forma de garantir a segurança na execução das tarefas da grade. É dada uma visão geral da biblioteca ASP.NET MVC utilizada no desenvolvimento do portal de submissão de tarefas. E, por fim, são apresentados os trabalhos correlatos.

2.1 GRADES COMPUTACIONAIS

Grades computacionais são sistemas que suportam execução paralela de aplicações em recursos distribuídos e heterogêneos, oferecendo acesso consistente e barato a estes recursos independente do local físico (FOSTER; KESSELMAN; TUECKE, 2001).

Sistemas de computação em grade (*grid computing*) tornaram-se realidade a partir da década de 90 através do desenvolvimento de Ian Foster e sua equipe (BRAGA, 2005, p. 3).

A palavra Grade remete ao termo electrical power grid que designa a rede elétrica. Segundo o próprio Ian Foster, a semelhança entre as duas arquiteturas se deve ao fato que na rede elétrica não nos importamos com a fonte geradora desta, assim como em um ambiente de grade computacional, não devemos nos preocupar com a origem dos ciclos de processamento e sim, que eles estão disponíveis para o uso pelas aplicações. (CONTI, 2009, p. 3).

Uma grade computacional é caracterizada por quatro aspectos (BUYAYA, 2002, p. 13):

- a) múltiplos domínios administrativos e autonomia: recursos da grade estão distribuídos geograficamente através de múltiplos domínios administrativos e sob propriedade de diferentes organizações. A autonomia dos donos dos recursos precisa ser honrada, assim como seu gerenciamento local e políticas de uso;
- b) heterogeneidade: uma grade envolve uma variedade de recursos que são heterogêneos por natureza e irão abranger várias tecnologias;
- c) escalabilidade: uma grade pode crescer de alguns recursos para milhões. Isso aumenta o problema da possível degradação de desempenho;

- d) dinamicidade e adaptabilidade: em uma grade, a falha de um recurso é a regra e não a exceção. De fato, com tantos recursos em uma grade, a probabilidade de alguma falha é alta. Gerenciadores de recursos e aplicações devem adequar dinamicamente seu comportamento e usar os recursos disponíveis e serviços de forma eficiente.

2.2 APLICAÇÕES DO TIPO *BAG-OF-TASKS*

Aplicações do tipo *bag-of-tasks* são aquelas nas quais as tarefas são independentes umas das outras. Apesar da simplicidade, aplicações *bag-of-tasks* são usadas em uma variedade de cenários, incluindo mineração de dados, buscas massivas, simulações, cálculos fractais, biologia computacional e processamento de imagens (CIRNE et al., 2003).

Devido a sua natureza desconexa, este tipo de aplicação adequa-se perfeitamente ao processamento em grade. Cada tarefa pode ser distribuída entre partes distintas da grade, independente de estarem diretamente conectadas ou geograficamente distantes, sem causar qualquer problema ao processamento.

2.3 ESCALONAMENTO DAS TAREFAS DA GRADE

De acordo com Brasileiro, Cirne e Silva (2003, p. 2), o escalonamento de tarefas em uma grade computacional é uma tarefa bastante complexa. Mesmo o escalonamento de aplicações do tipo *bag-of-tasks* não é uma tarefa trivial. A heterogeneidade é uma característica intrínseca da grade, deste modo ela deve ser levada em conta durante o escalonamento. Diversas características devem ser levadas em conta, tais como heterogeneidade, a natureza dinâmica das máquinas e da rede, largura de banda, latência e topologias de rede diversas. Em uma grade computacional normalmente é difícil estimar o tempo de execução de uma tarefa, o tempo pode variar bastante entre tarefas distintas ou entre máquinas diferentes.

Com a dificuldade de estimar o tempo de execução de uma tarefa ou obter mais informações dela, surge a necessidade de se utilizar um algoritmo que faça o escalonamento

de forma eficiente levando em consideração a heterogeneidade da grade.

A seguir são detalhados dois algoritmos que podem ser utilizados no escalonamento da grade.

2.3.1 Workqueue

É um algoritmo de escalonamento que não depende de informações da tarefa a ser processada. Nesse algoritmo as tarefas são atribuídas aos processadores de forma aleatória. Quando o processamento é finalizado, os resultados são enviados de volta e o escalonador atribui uma nova tarefa ao processador (BRASILEIRO; CIRNE; SILVA, 2003, p. 5).

A idéia desse algoritmo é que mais tarefas serão atribuídas a máquinas mais rápidas enquanto que máquinas mais lentas processarão menos. O problema dele é que, quando uma tarefa grande é atribuída para uma máquina lenta a finalização do processamento do *bag-of-tasks* precisará ficar aguardando essa tarefa ser completada (BRASILEIRO; CIRNE; SILVA, 2003, p. 5).

2.3.2 Workqueue com Replicação

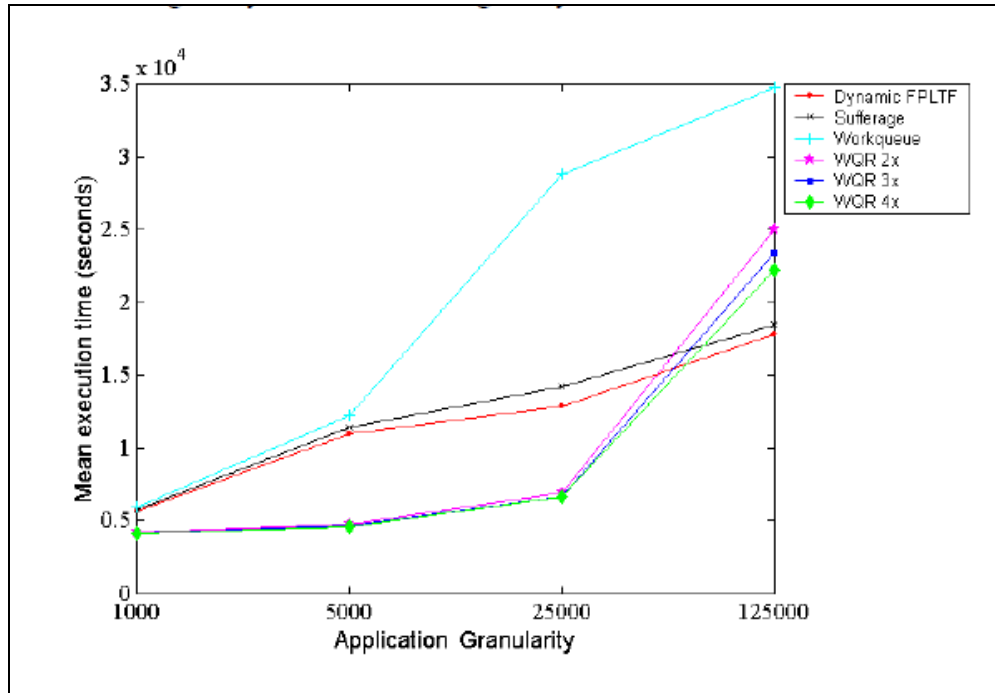
O algoritmo Workqueue com Replicação é uma evolução do Workqueue. Ele visa resolver o problema de atribuir uma tarefa grande a uma máquina lenta e tenta se adequar melhor a heterogeneidade da grade (BRASILEIRO; CIRNE; SILVA, 2003, p. 5).

Esse algoritmo basicamente adiciona a replicação de tarefas ao algoritmo original. Quando uma máquina encontra-se em espera ela obtém uma tarefa que já esteja em execução e passa a executá-la também. Novas cópias são criadas até um número limite pré-definido de réplicas. Com essas réplicas as chances de se atribuir a tarefa a uma máquina mais rápida aumentam e conseqüentemente o tempo de execução tende a diminuir. Assim que a tarefa é finalizada, todas as réplicas pendentes são canceladas.

O ponto negativo desse algoritmo é o desperdício de ciclos de processamento das réplicas canceladas. Como o processamento das réplicas canceladas não foi utilizado, ocorrerá o desperdício. A quantidade desperdiçada de processamento vai depender da quantidade de réplicas e da heterogeneidade das tarefas e máquinas.

2.3.3 Comparação de desempenho

Na figura 1 são apresentados os dados do processamento de tarefas com diferentes tamanhos (1000, 5000, 25000 e 125000 segundos). No comparativo são exibidos os resultados de algoritmos que necessitam de informações das tarefas (Dynamic FPLTF e Sufferage) e algoritmos que não necessitam de tais informações.

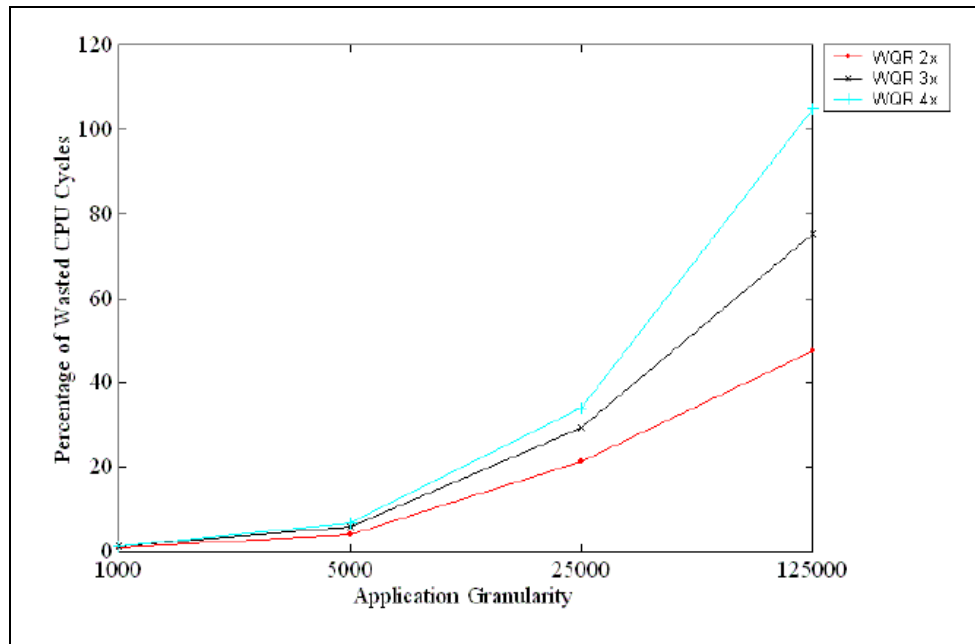


Fonte: Brasileiro, Cirne e Silva (2003, p. 8).

Figura 1 - Desempenho de acordo com granularidade das tarefas

Conforme se pode observar nos resultados, o algoritmo Workqueue é o mais lento entre eles, enquanto o algoritmo Workqueue com Replicação apresentou, em determinadas situações, resultados melhores que os algoritmos que necessitam de informações das tarefas.

Na figura 2 são exibidos os dados comparando a execução do algoritmo Workqueue com Replicação de acordo com granularidade das tarefas e a quantidade de processamento desperdiçado. Pode-se perceber que, à medida que o tamanho das tarefas aumenta, maior o desperdício de processamento.



Fonte: Brasileiro, Cirne e Silva (2003, p. 11).

Figura 2 - Desperdício de processamento de acordo com granularidade das tarefas

Com os dados apresentados, chega-se a conclusão que o algoritmo Workqueue com Replicação apresenta um desempenho muito superior ao algoritmo Workqueue. Também fica claro que com duas réplicas obtém-se um bom desempenho com uma baixa percentagem de processamento desperdiçado. Desse modo, o algoritmo Workqueue com Replicação de duas tarefas se apresentou o mais adequado para uso na grade.

2.4 SEGURANÇA NA EXECUÇÃO DE TAREFAS NA GRADE

Uma grade computacional que permite a submissão de tarefas cria um possível risco aos computadores que executarão estas tarefas. É possível que um código malicioso seja adicionado à tarefa e esse comprometa os computadores da grade. A fim de evitar este problema de segurança, uma possível abordagem é o emprego de *sandboxes*¹ (LOPES, 2006, p. 111).

Para a criação do *sandbox*, é utilizado um recurso do .NET denominado *Application Domain*. Um *Application Domain* é uma separação dentro de um único processo que permite

¹ *Sandbox* é um mecanismo de segurança que permite a execução de uma aplicação de forma isolada do resto do sistema.

que áreas distintas de uma aplicação trabalhem de forma separada, sem acesso direto de uma a outra. Para cada *Application Domain* pode-se definir as restrições de segurança de acesso do código (*Code Access Security*). O recurso de segurança de acesso do código permite restringir as operações que a aplicação poderá executar, evitando assim, a execução de códigos maliciosos (MICROSOFT CORPORATION, 2010a).

O uso do *Application Domain* traz uma desvantagem, ele acarreta em perda de desempenho. Como um *Application Domain* não pode acessar a área de memória de outro, os objetos passados entre eles precisam ser serializados. Com a serialização, cada objeto que trafega entre *Application Domains* precisa ser gravado em um formato intermediário, transmitido para o outro *Application Domain* e reconstruído à partir do formato intermediário salvo anteriormente. Esse processo de serialização faz com que a comunicação entre *Application Domains* seja menos eficiente que a comunicação direta de objetos (ASPALLIANCE, 2006).

Utilizando as rotinas de segurança de acesso do código do .NET Framework, as tarefas da grade são executadas criando um *sandbox*, dando o mínimo possível de permissões a esta tarefa, restringindo assim, o acesso aos discos locais, a internet e a qualquer outro recurso não necessário ao processamento da tarefa.

2.5 ASP.NET MVC

O ASP.NET é a tecnologia incluída no .NET Framework para a criação de aplicações web. Entre as principais características do ASP.NET pode-se destacar (MICROSOFT CORPORATION, 2010b):

- a) separação entre código utilizado na interface e lógica da aplicação;
- b) interface baseada em componentes processados no servidor;
- c) programação baseada em eventos, bastante similar ao modelo utilizado em aplicações *desktop*.

Mais tarde, a Microsoft desenvolveu a biblioteca ASP.NET MVC, que tem como objetivo possibilitar o uso do padrão Modelo-Visão-Controle (MVC) em aplicações ASP.NET.

No padrão MVC o modelo é composto pelo objeto representando o domínio manipulado, não tendo qualquer conhecimento da tela. A representação da tela é feita pela

visão e controle, onde a visão tem o papel de exibição e captura das ações do usuário, e o controle é responsável pelo tratamento destas ações (FOWLER, 2006).

2.6 TRABALHOS CORRELATOS

Alguns sistemas desempenham papel semelhante ao proposto no presente trabalho, cada qual com objetivos e restrições diferentes. Dentre eles foram selecionados o SETI@home (SETI@HOME, 2010), o *Berkeley Open Infrastructure for Network Computing* (BOINC) (BOINC, 2010) e o Globus Toolkit (GLOBUS ALLIANCE, 2010).

2.6.1 SETI@home

O projeto SETI@home tem como principal objetivo detectar vida inteligente fora da terra. Ao contrário de projetos anteriores que utilizavam supercomputadores, o projeto SETI@home proposto por David Gedye objetiva utilizar um grande número de computadores interconectados pela internet (SETI@HOME, 2010).

Para detectar vida inteligente fora da terra, os cientistas analisam ondas de rádio buscando transmissões não originadas na terra. Para esta análise, as recepções de rádio são gravadas, separadas em pequenos blocos e distribuídas entre os computadores da grade. Cada pedaço é processado de forma independente dos demais, permitindo o crescimento da grade de forma global (ANDERSON et al., 2002).

O projeto SETI@home comprovou a viabilidade da computação distribuída utilizando recursos públicos (ANDERSON et al., 2002). Devido ao seu sucesso, o SETI@home serviu como base para a criação do projeto BOINC. O projeto inicial do SETI@home foi finalizado em 22 de dezembro de 2005, quando passou a utilizar a infraestrutura do BOINC (SETI@HOME, 2009).

2.6.2 BOINC

O projeto BOINC é uma plataforma para computação distribuída utilizando recursos

públicos. O projeto é desenvolvido no Laboratório de Ciências Espaciais de Berkeley pelo grupo que desenvolveu e continua a operar o SETI@home (ANDERSON, 2004a, p. 2). Este projeto foi desenvolvido para resolver limitações do SETI@home e criar uma infraestrutura genérica para execução deste tipo de tarefa (ANDERSON, 2004b, p. 4).

BOINC prove a infraestrutura necessária para criação de sistemas distribuídos. Ao contrário do SETI@home, que é voltado especificamente à descoberta de vida inteligente extra-terrestre, ele não é voltado a um fim específico. Ele possui o conceito de projeto, onde cada projeto utiliza a infraestrutura básica do BOINC adaptando-a a suas necessidades. O usuário tem a possibilidade de participar de vários projetos simultaneamente (GOLDCHLEGER, 2004, p. 33-35).

As aplicações a serem executadas pelo BOINC são do tipo *bag-of-tasks*, ou seja, aplicações altamente paralelizáveis sem comunicação entre os nós.

Alguns projetos que utilizam o BOINC são:

- a) SETI@home;
- b) Climateprediction.net;
- c) Einstein@Home;
- d) LHC@home;
- e) Malaria Control Project.

2.6.3 Globus Toolkit

Segundo Lopes (2006, p. 88), “O Globus é o projeto de computação em grade com maior relevância na atualidade. É desenvolvido pela Aliança Globus, um grupo formado por inúmeras instituições ao redor de todo o mundo [...]”.

Esta aliança desenvolve o projeto denominado Globus Toolkit, que provê componentes necessários para o desenvolvimento de uma grade computacional. O *toolkit* inclui softwares para (GLOBUS ALLIANCE, 2010):

- a) segurança;
- b) infraestrutura da informação;
- c) gerenciamento de recursos;
- d) gerenciamento de dados;
- e) comunicação;
- f) tolerância a falhas;

g) portabilidade.

De acordo com Cirne (2002, p. 30), “É interessante notar que a decisão de estruturar Globus como um conjunto de serviços independentes deixa claro que Globus não é uma solução pronta e completa (*plug-and-play*) para construção de *Grids*”. Deste modo, este *toolkit* serve como base para o desenvolvimento de uma grade computacional, e não uma grade pronta para uso.

Outra característica importante do Globus Toolkit é que, conforme dito por Lopes (2006, p. 93), “A infraestrutura de uma grade Globus é composta por máquinas dedicadas, ou seja, máquinas que fornecem à grade todo o seu poder computacional.”

3 DESENVOLVIMENTO DO SOFTWARE

Este capítulo detalha as etapas do desenvolvimento do software. São apresentados os requisitos, a especificação e a implementação do mesmo, mencionando as técnicas e ferramentas utilizadas. Também é apresentada a operacionalidade do software e os resultados obtidos.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Abaixo são apresentados os Requisitos Funcionais (RF) e Requisitos Não-Funcionais (RNF) a serem atendidos pelo software desenvolvido:

- a) permitir a submissão de tarefas por meio de um portal web (RF);
- b) permitir o acompanhamento do estado da execução das tarefas (RF);
- c) permitir a obtenção dos resultados da execução da tarefa submetida (RF);
- d) distribuir, entre os computadores participantes da grade, as tarefas submetidas pelo portal (RF);
- e) tornar disponível uma API a ser utilizada pelas tarefas submetidas à grade (RF);
- f) utilizar o tempo ocioso dos computadores conectados à grade para processar as tarefas (RF);
- g) restringir as permissões de execução da tarefa, impedindo a execução de códigos maliciosos nos computadores participantes da grade (RNF);
- h) ser implementado utilizando o ambiente de desenvolvimento Microsoft Visual Studio 2010 (RNF);
- i) ser implementado utilizando a linguagem C# (RNF);
- j) utilizar a biblioteca ASP.NET MVC 2 para a construção do portal web (RNF);
- k) funcionar no sistema operacional Windows XP ou superior (RNF);
- l) utilizar banco de dados Microsoft SQL Server 2005 para armazenamento de dados utilizados pelo portal web e tarefas submetidas a grade (RNF);
- m) as tarefas submetidas ao portal devem ser desenvolvidas utilizando o Microsoft .NET Framework (RNF).

3.2 ESPECIFICAÇÃO

Esta seção apresenta a especificação da ferramenta, contendo os casos de uso, os diagramas de classes e de seqüência que foram especificados utilizando a ferramenta Enterprise Architect.

3.2.1 Componentes da grade

A grade computacional desenvolvida é composta por três partes: o portal para submissão de tarefas, o sistema de distribuição das mesmas e o programa que faz a execução das tarefas. Na figura 3 é exibida a estrutura e relacionamento dessas três partes.

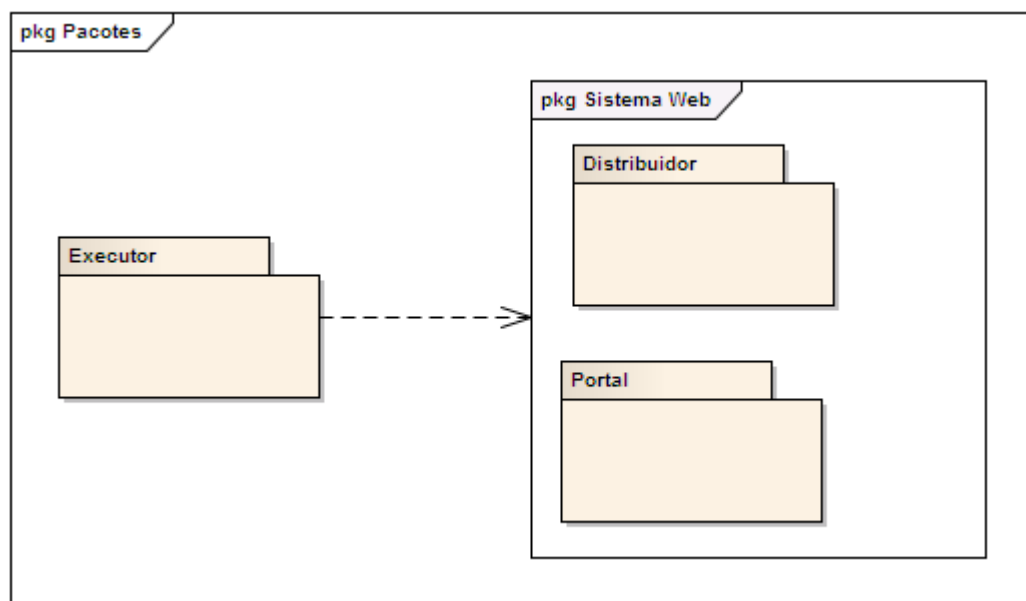


Figura 3 - Componentes da grade

O executor é o programa que deve ser instalado em cada máquina da grade para efetuar o processamento das tarefas. Este programa se conecta ao sistema web para obter as tarefas e enviar os resultados ao final do processamento.

O acesso a grade, tanto pelo usuário quanto pelo executor, é feito via internet. Com isso, o distribuidor e o portal estão agrupados no mesmo sistema web. Nesse sistema web estão disponíveis as páginas utilizadas pelo usuário para gerenciamento da grade e os *web services* utilizados pelo executor para obtenção das informações das tarefas.

3.2.2 Grupos e usuários do sistema

Cada usuário da grade faz parte de um grupo. Os grupos servem para maior segurança dos dados, para contabilização dos processamentos efetuados e distribuição uniforme dos recursos da grade. O usuário vê apenas dados das tarefas de seu grupo.

Os recursos da grade são distribuídos entre cada grupo de forma uniforme, ou seja, cada grupo tem o direito de usar a quantidade de processamento proporcional a quantidade de processamento cedida à grade. Isso evita que alguém faça uso excessivo da grade sem ceder recursos a ela.

3.2.3 API utilizada na criação de tarefas

As tarefas submetidas à grade são desenvolvidas no .NET Framework utilizando uma API específica da grade. Na figura 4 são apresentadas as interfaces utilizadas na criação das tarefas.

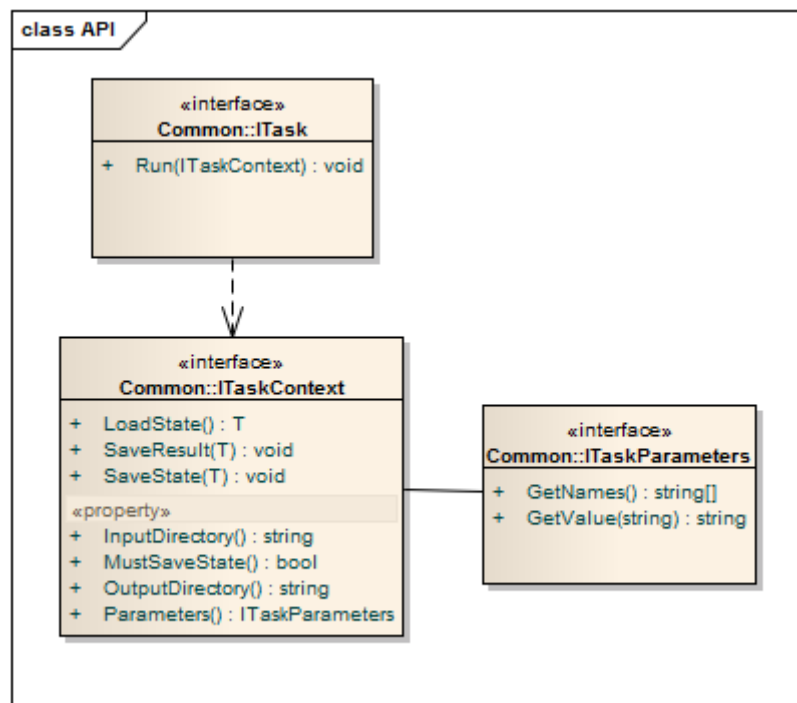


Figura 4 - Diagrama de classes da API

Uma tarefa deve implementar a interface `ITask`, que contém o método `Run`. O método `Run` é chamado pelo executor passando o contexto atual da execução. É nesse método que deve-se implementar a rotina a ser processada pela tarefa.

A interface `ITaskContext` pode ser acessada pela tarefa para obter informações durante a execução. Através dela a tarefa pode carregar e salvar seu estado, ler os parâmetros que foram configurados e obter o nome dos diretórios de entrada e saída de arquivos. Essa interface define os seguintes métodos e propriedades:

- a) `InputDirectory`: define o diretório que contém os arquivos recebidos pela tarefa;
- b) `OutputDirectory`: define o diretório onde os arquivos de saída deverão ser salvos. Ao final da execução, todos os arquivos que estiverem nesse diretório serão enviados como resultado do processamento;
- c) `SaveState`: salva o estado de execução atual da tarefa. Isso permite que uma tarefa seja paralisada e ao iniciar novamente continue o processamento do ponto onde se encontrava anteriormente. O estado de execução é gerenciado pela própria tarefa, é responsabilidade dela definir quais informações serão necessárias para continuar posteriormente a execução no caso dela ser interrompida;
- d) `LoadState`: carrega o estado salvo anteriormente pelo método `SaveState`;
- e) `MustSaveState`: salvar o estado constantemente pode acarretar em um desperdício de processamento durante a execução. Para evitar isso, o executor em um intervalo de tempo define o valor `MustSaveState` indicando que a tarefa deve ser salva. A tarefa pode verificar essa propriedade e salvar o estado apenas quando ela for verdadeira. Isso permite que o estado seja salvo apenas uma vez por minuto ao invés de salvar constantemente;
- f) `SaveResult`: salva o resultado da execução. O resultado da execução é composto pelos arquivos presentes no diretório de saída e pelo valor informado no método `SaveResult`;
- g) `Parameters`: permite obter os parâmetros definidos para a tarefa.

3.2.4 Formato do *job*

Na grade são processados *bag-of-tasks*, ou seja, um conjunto de tarefas similares que são independentes entre si. Cada *bag-of-task* é denominado *job*. O *job* contém as informações do executor (classe implementando a interface `ITask`) e uma lista de tarefas a serem processadas. Todas as tarefas utilizam o mesmo executor, variando entre elas apenas os dados de entrada e os parâmetros.

Para submeter um *job* à grade é criado um arquivo Zip contendo todos os arquivos necessários para o processamento e um arquivo eXtensible Markup Language (XML) definindo as tarefas do *job*. O formato do arquivo Zip e do XML devem respeitar um determinado formato, conforme apresentado a seguir.

3.2.4.1 Arquivo XML com a definição do *job*

Ao submeter um *job* deve-se definir um arquivo XML com as informações das tarefas e do executor. No quadro 1 é exibido um exemplo desse arquivo.

```
<?xml version="1.0" encoding="utf-8" ?>
<job name="AlgumJob">
  <executor>
    <library name="SimpleTask.dll"/>
    <entrypoint name="SimpleTask.SimpleTask" />
    <params>
      <param name="Outro" value="true" />
      <param name="MaisUm" value="1" />
    </params>
  </executor>
  <tasks>
    <task name="Tarefa 1">
      <data>
        <file>1.png</file>
      </data>
      <params>
        <param name="Abc" value="true" />
      </params>
    </task>
    <task name="Tarefa 2">
      <data>
        <file>2.png</file>
      </data>
      <params>
        <param name="Abc" value="false" />
        <param name="Algo" value="1" />
      </params>
    </task>
  </tasks>
</job>
```

Quadro 1 - Exemplo de definição do *job*

O arquivo contém as informações necessárias para a execução das tarefas. Entre essas informações estão:

- a) atributo `name`, define o nome do *job*. Utilizado apenas para facilitar a identificação no portal;
- b) elemento `executor`, especifica a classe utilizada no processamento. Esse elemento contém as seguintes informações:
 - elemento `library`, define o nome do *assembly*² que contém a classe de processamento,
 - elemento `entrypoint`, define o nome da classe de processamento,
 - elemento `params`, define os parâmetros que são passados ao executor em todas as tarefas;
- c) elemento `tasks`, define a lista de tarefas a serem processadas;
 - elemento `data`, define os arquivos de entrada da tarefa,
 - elemento `params`, define os parâmetros a serem passados ao executor ao processar essa tarefa.

3.2.4.2 Pacote do *job*

O *job* submetido à grade deve estar compactado no formato Zip e ter uma estrutura pré-definida. O arquivo Zip deve conter:

- a) arquivo `info.xml`: é o XML contendo as informações do *job*, conforme apresentado anteriormente;
- b) diretório `data`: diretório contendo todos os arquivos de entrada utilizados no *job*;
- c) diretório `executor`: diretório contendo os arquivos do executor.

Na figura 5 é exibido um exemplo de pacote a ser submetido.

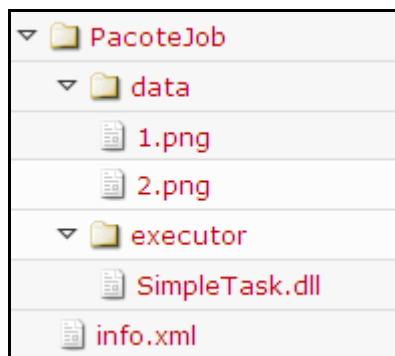


Figura 5 - Exemplo de pacote de *job*

² No .NET Framework, *assembly* é uma biblioteca de código compilado.

3.2.5 Distribuição das tarefas

Como a grade não tem acesso a cada executor, não é ela que atribui as tarefas a cada um. Ao invés da grade enviar as tarefa a cada executor, são os executores que solicitam uma tarefa e a grade determina qual tarefa eles irão processar. A escolha da tarefa que será retornada ao executor é feita por meio de um algoritmo que otimiza a utilização e desempenho da grade.

O algoritmo utilizado no escalonamento das tarefas é o Workqueue com Replicação. Juntamente desse algoritmo foi utilizado um algoritmo para contabilização dos processamentos efetuados por cada grupo que garante o uso proporcional dos recursos cedidos por cada grupo.

3.2.5.1 Workqueue com Replicação

Conforme descrito na fundamentação teórica, este algoritmo cria uma fila de tarefas a serem processadas e, assim que todas estejam atribuídas, cria réplicas para tentar melhorar o desempenho da grade.

No quadro 2 são apresentados os dados de três tarefas que não estão sendo processadas por nenhum cliente.

Nome	Máquinas processando
Tarefa 1	0
Tarefa 2	0
Tarefa 3	0

Quadro 2 - Exemplo de grade com três tarefas pendentes

Após três máquinas requisitarem tarefas, a grade terá o cenário apresentado no quadro 3, onde cada tarefa está sendo processada por uma máquina.

Nome	Máquinas processando
Tarefa 1	1
Tarefa 2	1
Tarefa 3	1

Quadro 3 - Exemplo de grade processando três tarefas

Se uma nova máquina requisitar uma tarefa a ser processada, a grade criará uma

réplica de uma das tarefas, neste caso uma das tarefas estará sendo processada por duas máquinas distintas, conforme exibido no quadro 4.

Nome	Máquinas processando
Tarefa 1	2
Tarefa 2	1
Tarefa 3	1

Quadro 4 - Exemplo de grade processando três tarefas com replicação

Quando uma tarefa é replicada o processamento não é dividido entre as duas máquinas. Cada máquina executará a tarefa completa, de forma independente. Essa réplica visa apenas otimizar o tempo de espera pelo término do processamento. Caso a primeira máquina seja lenta, ou não esteja ligada por um determinado período de tempo, a segunda máquina poderá finalizar o processamento antes. Assim que o processamento da tarefa é finalizado em uma das máquinas todas as réplicas são abortadas.

3.2.5.2 Contabilização dos pontos de processamento

A distribuição das tarefas deve ser proporcional aos recursos cedidos por cada grupo. Ou seja, se um grupo cedeu cem horas de processamento, ele deve ter o direito de utilizar cem horas de processamento da grade.

Para garantir essa proporcionalidade, a grade mantém a quantidade de recursos cedidos e utilizados por cada grupo. Com base nessas informações, a fila de tarefas é ordenada e a próxima requisição de tarefa retornará a tarefa com maior prioridade.

A prioridade de cada tarefa é determinada usando a seguinte fórmula $P = PC/PU$, onde P é a prioridade, PC é o processamento cedido pelo grupo que submeteu a tarefa e PU é o processamento utilizado pelo grupo. Para melhor entendimento, a seguir é apresentada uma simulação de processamento.

No momento inicial, exibido no quadro 5, todos os grupos estão com proporção igual e há quatro tarefas pendentes.

Tarefa	Grupo	Proporção do grupo
Tarefa 0	Grupo 1	1
Tarefa 1	Grupo 1	1
Tarefa 2	Grupo 2	1
Tarefa 3	Grupo 3	1

Grupo	Processamento cedido	Processamento utilizado	Proporção
Grupo 1	100	100	1
Grupo 2	100	100	1
Grupo 3	100	100	1

Quadro 5 - Estado da grade no momento inicial do exemplo

Nesse momento, o Grupo 3 processa a Tarefa 0 e contabiliza 100 pontos de processamento. Após esse processamento a grade estará na situação exibida no quadro 6. Pode-se verificar que a ordem das tarefas foi alterada de acordo com a nova proporção de cada grupo.

Tarefa	Grupo	Proporção do grupo
Tarefa 3	Grupo 3	2
Tarefa 2	Grupo 2	1
Tarefa 1	Grupo 1	0,5

Grupo	Processamento cedido	Processamento utilizado	Proporção
Grupo 1	100	200	0,5
Grupo 2	100	100	1
Grupo 3	200	100	2

Quadro 6 - Estado da grade após o processamento de uma tarefa

A seguir, se o Grupo 1 processar a Tarefa 3, contabilizando 150 pontos, a grade estará

na situação exibida no quadro 7. Novamente a ordem das tarefas é alterada de acordo com a proporção de cada grupo.

Tarefa	Grupo	Proporção do grupo
Tarefa 1	Grupo 1	1,25
Tarefa 2	Grupo 2	1

Grupo	Processamento cedido	Processamento utilizado	Proporção
Grupo 1	250	200	1,25
Grupo 2	100	100	1
Grupo 3	200	250	0,8

Quadro 7 - Estado da grade após o processamento de duas tarefas

Conforme exibido na simulação, a proporção de cada grupo sempre se manterá próxima a um. Isso garante que um grupo não utilizará muitos recursos e compartilhará poucos.

3.2.6 Casos de uso

Nos diagramas de caso de uso foram especificadas as interações do usuário com o sistema de submissão de tarefas. O sistema possui duas partes distintas que o usuário irá interagir: o portal para submissão de tarefas e o programa de execução. Cada uma dessas partes é apresentada em um diagrama de casos de uso distinto.

Pode-se observar nos diagramas que há três tipos de usuários:

- a) usuário normal: não tem nenhum privilégio especial;
- b) administrador do grupo: pode criar usuários pertencentes ao seu grupo;
- c) super administrador: pode executar todas as operações do sistema.

Na figura 6 são apresentados os casos de uso do portal para submissão de tarefas. Neste diagrama são apresentados os casos de uso relacionados à submissão das tarefas, obtenção dos resultados e gerenciamento dos usuários e grupos do portal. Do quadro 8 ao quadro 14 são apresentados os detalhamentos de cada caso de uso.

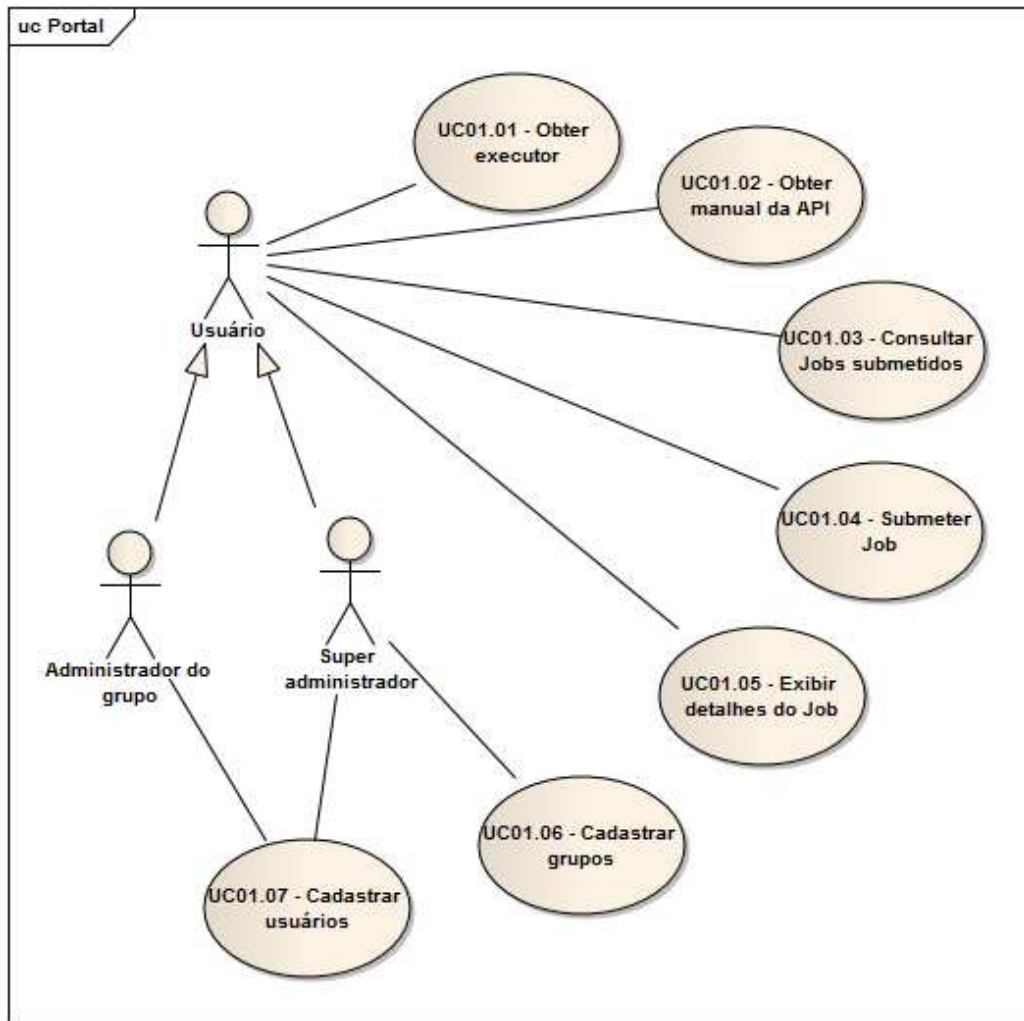


Figura 6 - Diagrama de casos de uso do portal

UC01.01 – Obter executor	
Pré-condições	O usuário deve estar autenticado no sistema.
Cenário principal	<ol style="list-style-type: none"> 1) O usuário acessa a página do executor. 2) O sistema exibe um <i>link</i> para download do programa de execução. 3) O usuário acessa o <i>link</i> e faz o download do executor.
Pós-condições	Download completo do executor.

Quadro 8 - Detalhamento do caso de uso UC01.01

UC01.02 – Obter manual da API	
Pré-condições	O usuário deve estar autenticado no sistema.
Cenário principal	<ol style="list-style-type: none"> 1) O usuário acessa a página do executor. 2) O sistema exibe um <i>link</i> para o manual da API. 3) O usuário acessa o <i>link</i>. 4) O sistema apresenta o manual da API.
Pós-condições	Manual da API exibido ao usuário.

Quadro 9 - Detalhamento do caso de uso UC01.02

UC01.03 – Consultar jobs submetidos	
Pré-condições	O usuário deve estar autenticado no sistema.
Cenário principal	<ol style="list-style-type: none"> 1) O usuário acessa a tela de consulta de <i>jobs</i>. 2) O sistema apresenta os <i>jobs</i> submetidos pelo grupo do usuário, mostrando detalhes básicos (nome do <i>job</i>, data de submissão e usuário que submeteu). 3) O usuário opta por fazer uma nova submissão, exibir os detalhes de um <i>job</i>, pesquisar os <i>jobs</i> enviados ou ainda finalizar o caso de uso. 4) O usuário seleciona o estado dos <i>jobs</i> a serem filtrados e opta por efetuar a pesquisa. 5) O sistema apresenta os <i>jobs</i> que encontram-se no estado selecionado pelo Usuário. 6) Retorna ao passo 3.

Quadro 10 - Detalhamento do caso de uso UC01.03

UC01.04 – Submeter job	
Pré-condições	O usuário deve estar autenticado no sistema.
Cenário principal	<p>No passo 3 do caso de uso UC01.03 o usuário pode optar por submeter um novo <i>job</i>:</p> <ol style="list-style-type: none"> 1) O sistema apresenta a tela de submissão de <i>jobs</i>; 2) O usuário informa um arquivo contendo os dados do <i>job</i> e opta por enviar o <i>job</i>; 3) O sistema valida os dados do arquivo submetido; 4) O sistema cria um novo <i>job</i> com os dados submetidos.
Exceção	<p>No passo 3, caso seja encontrado algum erro no arquivo submetido:</p> <ol style="list-style-type: none"> 3.1) O sistema apresenta uma mensagem de erro indicando o problema e volta ao passo 1.
Pós-condições	O <i>job</i> submetido deve ter sido criado.

Quadro 11 - Detalhamento do caso de uso UC01.04

UC01.05 – Exibir detalhes do job	
Pré-condições	O usuário deve estar autenticado no sistema.
Cenário principal	<p>No passo 3 do caso de uso UC01.03 o usuário pode optar por exibir os detalhes de um <i>job</i>:</p> <ol style="list-style-type: none"> 1) O sistema apresenta a tela com os detalhes do <i>job</i> e a lista de tarefas pertencentes a ele; 2) O usuário pode fazer o download de todos os resultados do <i>job</i>, ou verificar os resultados de cada tarefa individualmente;
Fluxo alternativo 1	<p>No passo 2 do cenário principal o usuário pode optar por fazer o download dos resultados do <i>job</i>:</p> <ol style="list-style-type: none"> 2.1) O usuário seleciona a opção <code>Download</code> do <code>Job</code>; 2.2) O sistema gera um arquivo contendo o resultado de todas as tarefas do <i>job</i>; 2.3) O usuário pode salvar ou abrir o arquivo gerado.
Fluxo alternativo 2	<p>No passo 2 do cenário principal o usuário pode optar por fazer o download dos resultados de uma tarefa:</p> <ol style="list-style-type: none"> 2.1) O usuário apresenta em cada tarefa os arquivos gerados pelo processamento; 2.2) O usuário seleciona um arquivo gerado pela tarefa; 2.3) O sistema envia o arquivo ao usuário permitindo que ele seja salvo ou aberto diretamente.
Fluxo alternativo 3	<p>No passo 2 do cenário principal o usuário pode optar por fazer verificar o resultado de uma tarefa:</p> <ol style="list-style-type: none"> 2.1) O usuário apresenta em cada tarefa o resultado do processamento; 2.2) O usuário seleciona o resultado da tarefa; 2.3) O sistema exibe o resultado.

Quadro 12 - Detalhamento do caso de uso UC01.05

UC01.06 – Cadastrar grupos	
Pré-condições	O usuário autenticado deve ser um Super Administrador.
Cenário principal	<ol style="list-style-type: none"> 1) O Super Administrador acessa o cadastro de grupos; 2) O sistema apresenta todos os grupos do sistema; 3) O Administrador opta por inserir ou alterar um grupo ou finalizar o caso de uso.
Fluxo alternativo 1	<p>No passo 3 o Super Administrador pode optar por inserir um novo grupo clicando no botão <i>Novo Grupo</i>:</p> <ol style="list-style-type: none"> 3.1) O sistema apresenta a tela para cadastro de um novo grupo; 3.2) O Super Administrador informa os dados do grupo e opta por salvar; 3.3) O sistema valida as informações; 3.4) O sistema grava as informações.
Fluxo alternativo 2	<p>No passo 3 o Super Administrador pode optar por alterar um grupo clicando no botão <i>Exibir detalhes</i>:</p> <ol style="list-style-type: none"> 3.1) O sistema apresenta os dados para alteração; 3.2) O Super Administrador edita os dados do grupo e opta por salvar; 3.3) O sistema valida as informações; 3.4) O sistema grava as informações.
Exceção	No passo 3.4 (cenários <i>Inserir Grupo</i> e <i>Alterar Grupo</i>), caso não sejam informados todos os dados, o sistema apresenta uma mensagem informando que os campos são obrigatórios. Retorna ao passo 3.1.

Quadro 13 - Detalhamento do caso de uso UC01.06

UC01.07 – Cadastrar usuários	
Pré-condições	O usuário autenticado deve ser um Administrador.
Cenário principal	<ol style="list-style-type: none"> 1) O Administrador acessa o cadastro de usuários; 2) O sistema apresenta os usuários do sistema. Quando o usuário autenticado for um Super Administrador exibe todos os usuários. Caso seja um Administrador do Grupo exibe apenas os usuários do seu grupo; 3) O Administrador opta por inserir ou alterar um usuário ou finalizar o caso de uso.
Fluxo alternativo 1	<p>No passo 3 o Administrador pode optar por inserir um novo usuário clicando no botão <i>Novo Usuário</i>:</p> <ol style="list-style-type: none"> 3.1) O sistema apresenta a tela para cadastro de um novo usuário; 3.2) O Administrador informa os dados do usuário e opta por salvar; 3.3) O sistema valida as informações; 3.4) O sistema grava as informações.
Fluxo alternativo 2	<p>No passo 3 o Administrador pode optar por alterar um usuário clicando no botão <i>Exibir detalhes</i>:</p> <ol style="list-style-type: none"> 3.1) O sistema apresenta os dados para alteração; 3.2) O Administrador edita os dados do usuário e opta por salvar; 3.3) O sistema valida as informações; 3.4) O sistema grava as informações.
Exceção	No passo 3.4 (cenários <i>Inserir Usuário</i> e <i>Alterar Usuário</i>), caso não sejam informados todos os dados, o sistema apresenta uma mensagem informando que os campos são obrigatórios. Retorna ao passo 3.1.

Quadro 14 - Detalhamento do caso de uso UC01.07

Na figura 7 são apresentados os casos de uso do executor de tarefas. Neste diagrama são apresentados os casos de uso relacionados ao programa instalado em cada máquina da grade para efetuar o processamento das tarefas. No quadro 15 e quadro 16 são apresentados os detalhes de cada caso de uso.

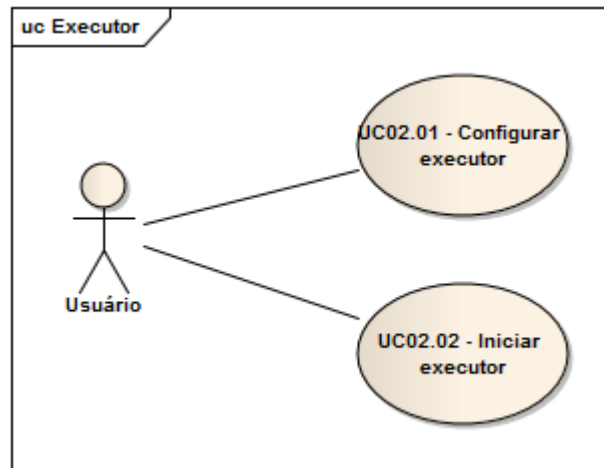


Figura 7 - Diagrama de casos de uso do executor

UC02.01 – Configurar executor	
Cenário principal	<ol style="list-style-type: none"> 1) O sistema apresenta a tela de configuração do executor; 2) O Usuário informa o seu nome, sua senha e o nome da máquina; 3) O usuário opta por salvar a configuração; 4) O sistema valida os dados informados; 5) O sistema verifica o desempenho do computador; 6) O sistema salva os dados da configuração; 7) O sistema fecha a tela de configuração e inicia o processamento das tarefas.
Fluxo alternativo 1	<p>No passo 4, caso o usuário ou a senha estejam inválidos:</p> <ol style="list-style-type: none"> 4.1) O sistema apresenta uma mensagem informando que o usuário ou a senha estão inválidos.
Fluxo alternativo 2	<p>No passo 4, caso o nome da máquina já esteja cadastrada:</p> <ol style="list-style-type: none"> 4.1) O sistema informa que já existe uma máquina cadastrada com esse nome e pergunta ao usuário se ele realmente deseja usar esse nome; 4.2) O usuário responde se deseja usar esse nome; 4.3) Caso o usuário tenha confirmado, continua o caso de uso. Caso contrário, volta ao passo 2.

Quadro 15 - Detalhamento do caso de uso UC02.01

UC02.02 – Iniciar executor	
Cenário principal	1) O usuário inicia o executor; 2) O sistema carrega os dados da máquina; 3) O sistema obtém uma nova tarefa a ser executada; 4) O sistema executa a tarefa; 5) O sistema envia o resultado da tarefa; 6) Volta ao passo 3.
Fluxo alternativo 1	No passo 2, caso os dados da máquina não estejam definidos: 2.1) Executa o caso de uso UC02.01; 2.2) Ao final da execução do caso de uso UC02.01 continua a execução deste caso de uso.
Fluxo alternativo 2	No passo 3, caso não haja nenhuma tarefa a ser executada: 3.1) O sistema aguarda por algum tempo; 3.2) Volta ao passo 3.
Fluxo alternativo 3	No passo 3, caso o executor já contenha uma tarefa sendo executada: 3.1) Carrega tarefa que estava sendo executada; 3.2) Continua a execução do caso de uso.
Exceção	No passo 4, caso ocorra um erro: 4.1) O sistema envia o resultado de erro; 4.2) Continua a execução do caso de uso no passo 6.

Quadro 16 - Detalhamento do caso de uso UC02.02

3.2.7 Diagramas de classes

No projeto há duas aplicações, a primeira delas é o programa de execução de tarefas e a segunda é o sistema web. No sistema web encontram-se o portal utilizado para submeter as tarefas e o distribuidor, que distribui as tarefas aos executores. A seguir, são apresentadas as principais classes do sistema.

3.2.7.1 Executor

Na figura 8 são apresentadas as principais classes e interfaces responsáveis pela execução das tarefas.

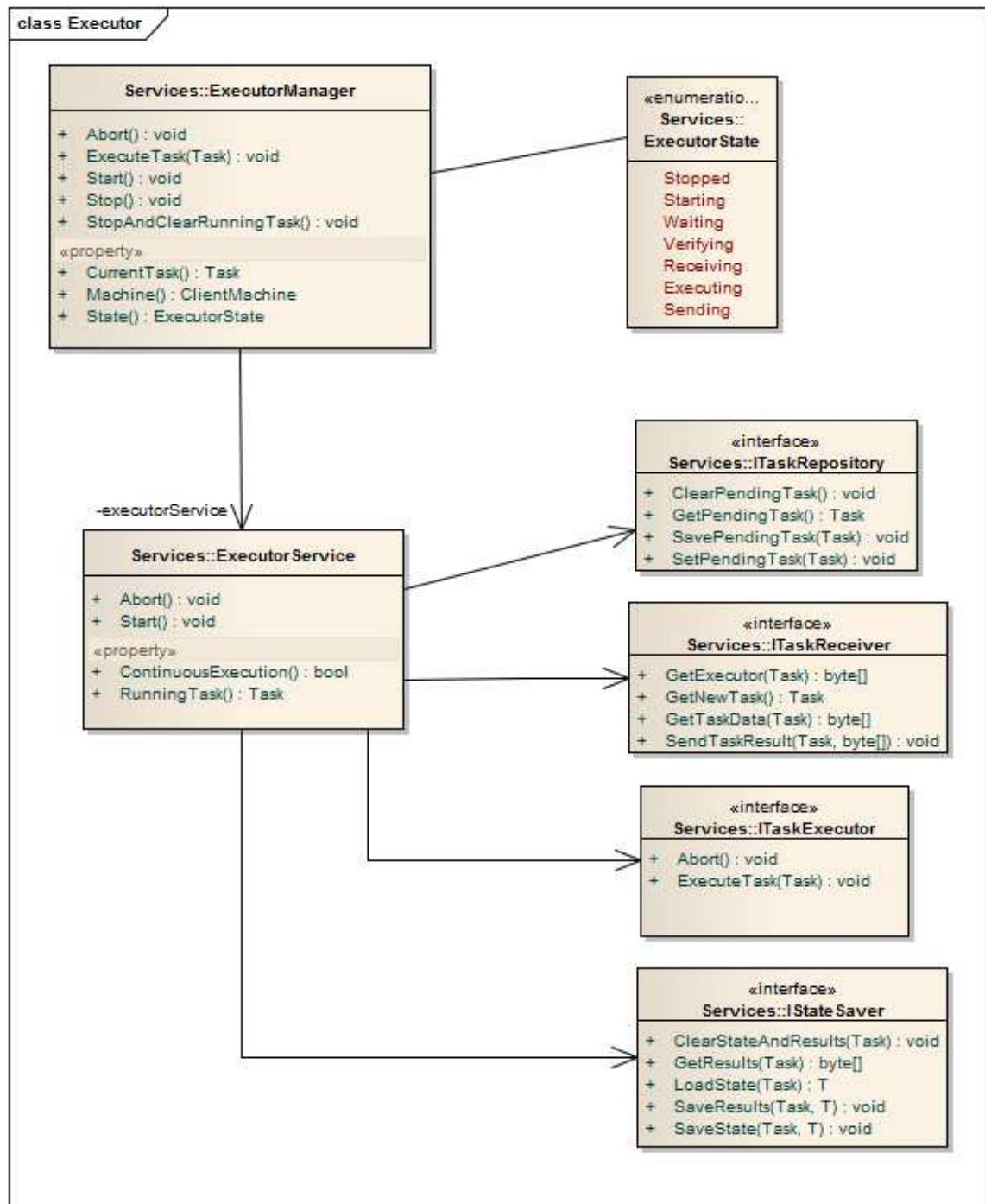


Figura 8 - Diagrama de classes do executor

A classe `ExecutorManager` é responsável por gerenciar a execução das tarefas. Ela inicia a execução da tarefa quando o executor é iniciado, para quando ele é fechado e disponibiliza as informações da tarefa atualmente em execução. A execução pode estar em

diversos estados, sendo representados pelo enumerador `ExecutorState`.

A classe `ExecutorService` é responsável pela coordenação da execução em si. Ela obtém a tarefa a ser executada, efetua o processamento, envia o resultado e repete o ciclo indefinidamente. Na verdade essa classe é apenas a coordenadora, cada ação é feita por uma classe específica. No diagrama pode-se verificar as interfaces:

- `ITaskRepository`: armazena os dados da tarefa atual;
- `ITaskReceiver`: obtém a tarefa e envia os dados processados;
- `ITaskExecutor`: efetua o processamento da tarefa;
- `ITaskReceiver`: salva os dados da tarefa durante o processamento.

Na figura 9 são apresentadas as classes responsáveis por carregar a tarefa e executá-la. A classe `TaskExecutor` implementa a interface `ITaskExecutor` que é utilizada pelo `ExecutorService`, conforme visto anteriormente. A tarefa é carregada pela classe `SecureTaskLoader`. Nessa classe a tarefa é carregada de forma segura para evitar que execute comandos potencialmente perigosos. Na implementação são apresentados mais detalhes de como é carregada a tarefa.

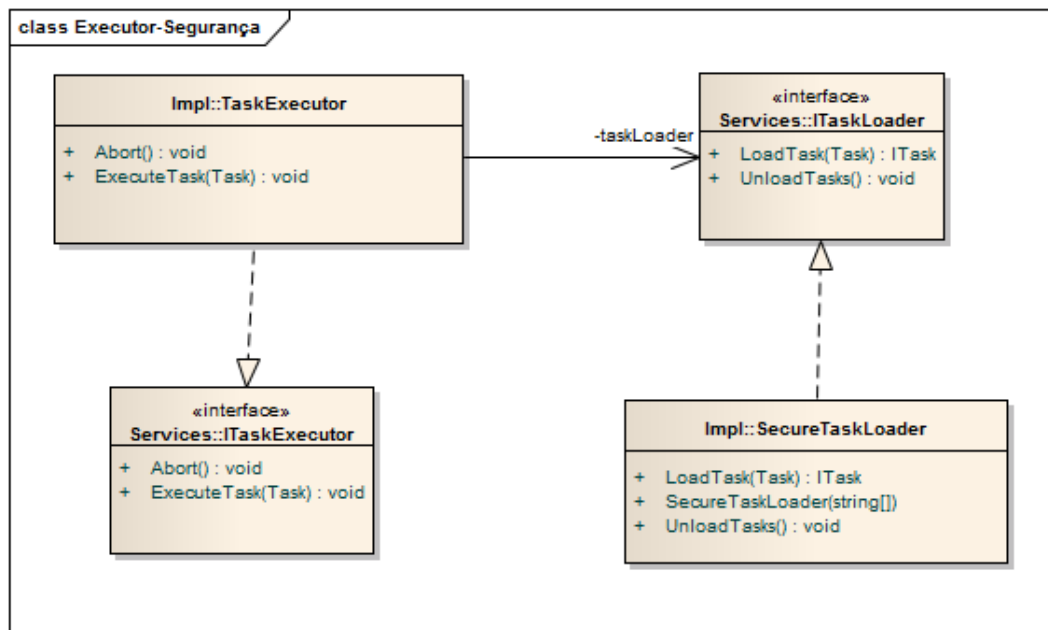


Figura 9 - Diagrama de classes do componente de segurança do executor

3.2.7.2 Portal

Na figura 10 são apresentadas as classes de modelo do portal de submissão de tarefas. Essas classes são utilizadas no acesso a dados, no distribuidor e na exibição das telas.

A classe `Job` representa um *bag-of-tasks* submetido à grade. Cada `Job` contém um executor e uma coleção de `Task`. Todas as tarefas utilizam o mesmo executor do `Job`, variando apenas os dados de entrada e os parâmetros.

A classe `TaskProcessor` representa um processamento daquela tarefa. Quando uma máquina inicia o processamento é criado um `TaskProcessor` com seus dados e, ao final, são registrados os dados do processamento. Uma tarefa pode ter mais de um `TaskProcessor` e este pode estar executando, finalizado, com erro ou abortado.

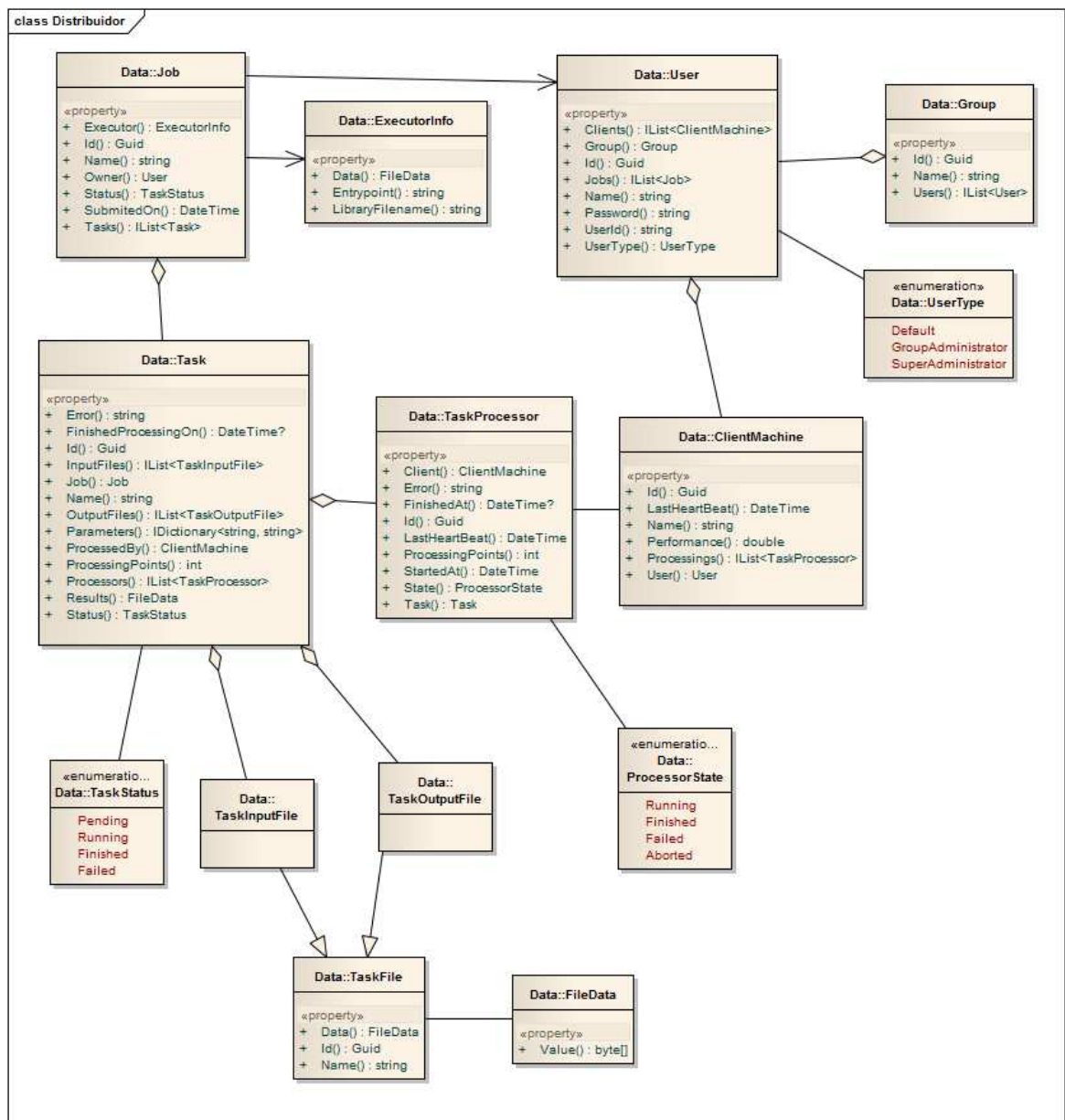


Figura 10 - Diagrama de classes do portal

A classe `Group` contém apenas o nome do grupo e uma lista de usuários que fazem parte do grupo. Cada usuário contém suas informações, como nome e tipo de usuário, senha, e

uma lista de máquinas. Quando o executor é configurado, uma máquina é associada ao usuário com as informações daquele executor.

3.2.7.3 Distribuidor

Na figura 11 é apresentada a interface que define os métodos responsáveis pela distribuição das tarefas e recebimento dos resultados.

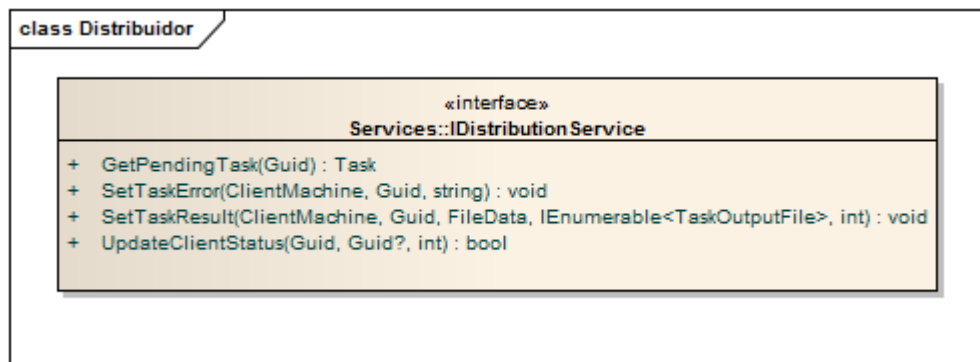


Figura 11 - Diagrama de classes do distribuidor

Na classe de distribuição de tarefas estão presentes os seguintes métodos:

- a) `GetPendingTask`: retorna uma tarefa a ser processada;
- b) `SetTaskError`: se ocorrer um erro durante o processamento esse método é chamado passando-se o erro ocorrido;
- c) `SetTaskResult`: ao final do processamento de uma tarefa esse método é chamado passando-se os resultados da mesma;
- d) `UpdateClientStatus`: esse método é chamado em um intervalo de tempo pré-definido para indicar que o executor está processando a tarefa. Como resultado desse método é retornado um valor booleano indicando se a execução deve ser cancelada (caso ela já tenha sido finalizada por outra máquina da grade).

3.2.7.4 Web services

O executor acessa o distribuidor por meio de *web services*. Na figura 12 são apresentadas as definições dos *web services* disponibilizados ao executor.

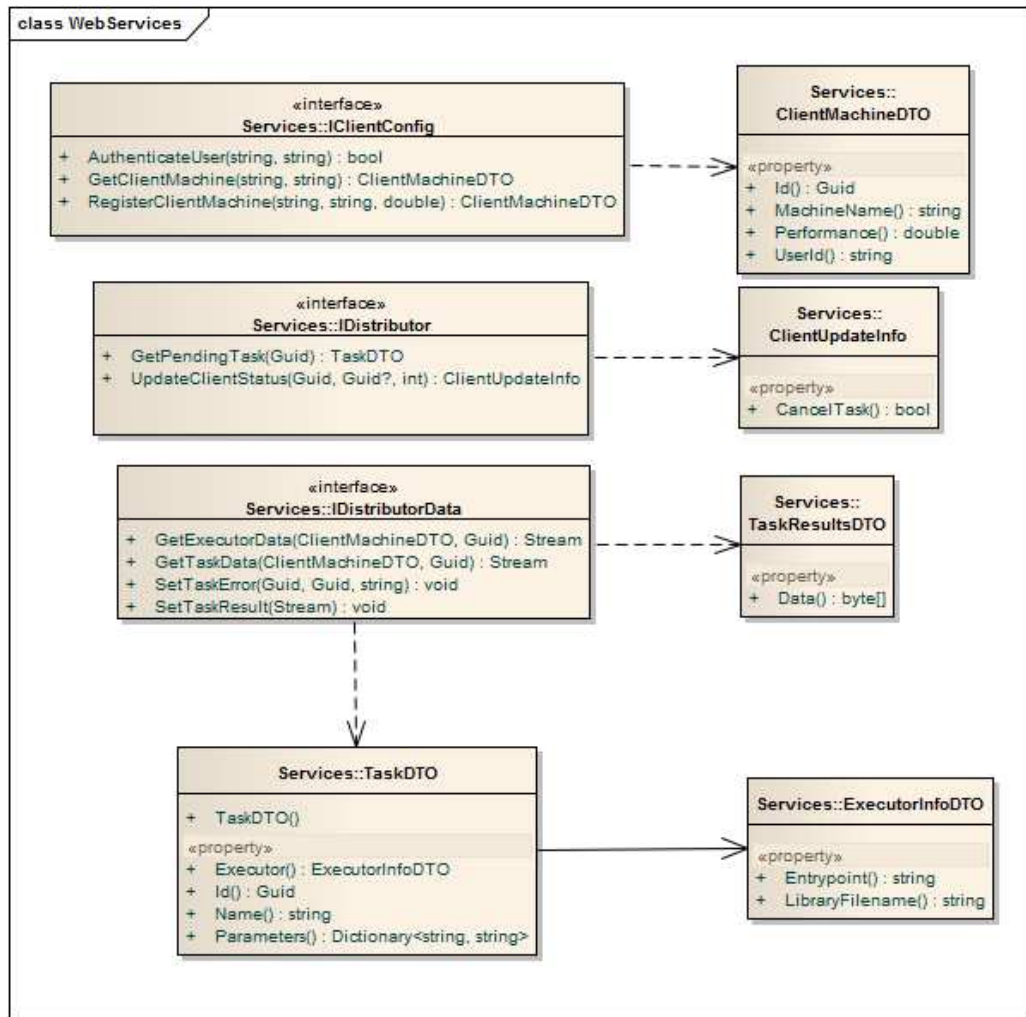


Figura 12 - Diagrama de classes de *web services*

A interface `IClientConfig` define o *web service* acessado durante a configuração do executor. Nele estão os métodos `AuthenticateUser` - utilizado na autenticação do usuário, `GetClientMachine` - utilizado para verificar se o usuário já possui uma máquina registrada com o mesmo nome e `RegisterClientMachine` - responsável por registrar a máquina do usuário.

A interface `IDistributor` contém os métodos `GetPendingTask` e `UpdateClientStatus`, que são usados respectivamente para obter uma nova tarefa e para atualizar o estado da tarefa atual.

A interface `IDistributorData` define os métodos utilizados pelo executor para receber e enviar os dados das tarefas. O método `GetExecutorData` é responsável por obter o *assembly* que será chamado para efetuar a execução da tarefa, enquanto que o método `GetTaskData` obtém os dados utilizados durante essa execução. Ao final da execução, se a tarefa for processada com sucesso, o método `SetTaskResult` é chamado passando-se os dados processados. Em caso de falha o método `SetTaskError` é chamado passando-se a

mensagem de erro.

3.2.8 Diagrama de sequência

Na figura 13 é apresentado o diagrama de sequência do executor. A classe `ExecutorService` recebe a tarefa a ser executada, efetua o processamento e ao final envia os resultados.

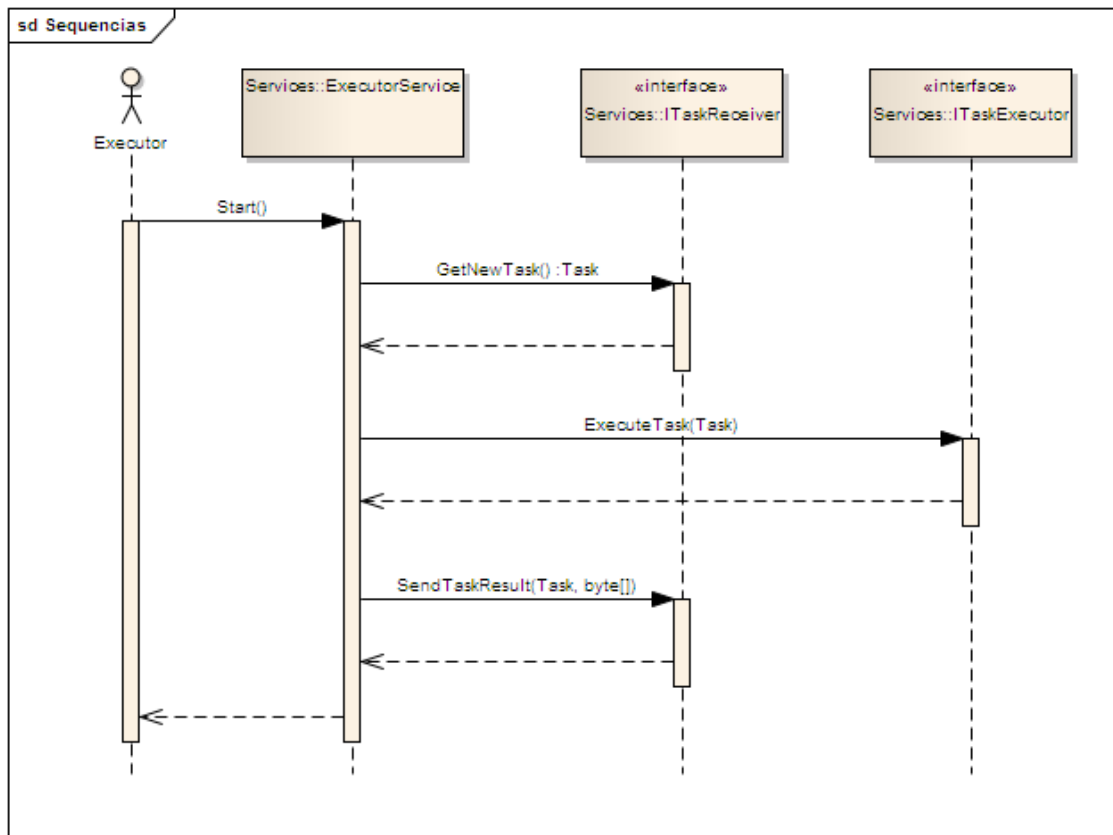


Figura 13 - Diagrama de sequência do executor

A sequência das atividades efetuadas pelo executor é bastante simples. Primeiramente ele obtém uma tarefa a ser executada, a seguir processa essa tarefa e por fim, envia o resultado obtido.

3.2.9 Diagrama de estados

Na figura 14 é apresentado o diagrama de estados de uma tarefa. Quando uma tarefa é submetida ela encontra-se no estado `Pending`, ao iniciar passa para o estado `Running` e ao

final, em caso de sucesso vai para o estado `Finished` e em caso de falha vai para o estado `Failed`.

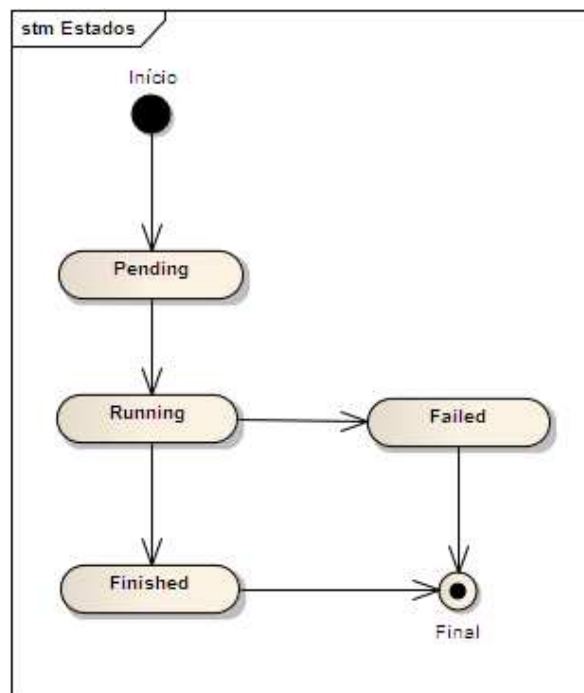


Figura 14 - Diagrama de estados de uma tarefa

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

Para facilitar o entendimento, o processo de implementação do software foi dividido em etapas. Inicialmente são expostas as bibliotecas e técnicas usadas em todo o desenvolvimento. A seguir são apresentadas as implementações de cada parte do sistema: o executor, o distribuidor e o portal para submissão de tarefas. Por fim, é demonstrada a operacionalidade da implementação.

3.3.1 Bibliotecas e ferramentas utilizadas

A implementação do software foi realizada utilizando a linguagem de programação C#, em conjunto com as bibliotecas: ASP.NET MVC – para a construção da interface web;

NHibernate (MAULO, 2009) – para o mapeamento objeto-relacional; Fluent NHibernate (GREGORY, 2010) – para o mapeamento dos objetos utilizados pelo NHibernate; SciMark (TAING, 2010) – para a execução do *benchmark* da máquina da grade; NUnit (NUNIT, 2007) – para os testes unitários.

O ambiente de desenvolvimento escolhido foi o Microsoft Visual Studio 2010. Para a gravação dos dados foi utilizado o Microsoft SQL Server 2008.

3.3.2 Implementação dos testes unitários

O software foi desenvolvido utilizando testes unitários. Ao implementar uma determinada funcionalidade, primeiro era criado um teste unitário que verificava o comportamento esperado e a seguir era feita a implementação. Para a criação dos testes unitários foi utilizada a biblioteca NUnit.

No quadro 17 são exibidos alguns testes unitários criados durante a implementação do software.

```
[Test]
public void AuthenticateUser_WithAnInvalidPassword_MustReturnFalse()
{
    ClientConfig clientConfig = new ClientConfig(session);
    bool authenticated = clientConfig.AuthenticateUser("testel", "senha_errada");

    authenticated.ShouldBeFalse();
}

[Test]
public void GetClientMachine_WithInvalidUser_ReturnNothing()
{
    ClientConfig clientConfig = new ClientConfig(session);

    ClientMachineDTO clientMachineDto = clientConfig.GetClientMachine("usuário
inválido", "machine1");

    clientMachineDto.ShouldBeNull();
}
```

Quadro 17 - Testes unitários

3.3.3 Implementação do executor

O executor é uma aplicação *desktop* utilizada em cada máquina da grade. Sua principal rotina consiste em obter uma tarefa, executá-la e enviar o resultado ao servidor. No quadro 18

é exibido o trecho da classe `ExecutorService` que realiza essa operação.

```
public class ExecutorService
{
    //...

    public void Start()
    {
        do
        {
            LoadTaskToRun();
            ExecuteTask();
            SendTaskResult();
        }
        while (ContinuousExecution);
    }

    //...
}
```

Quadro 18 - Trecho da classe `ExecutorService`

Para garantir a segurança da máquina, a tarefa é carregada em *sandbox*. O *sandbox* é definido criando-se um `AppDomain` separado onde são dadas as permissões mínimas necessárias para a execução da tarefa. Um `AppDomain` em .NET é uma separação lógica dentro de um único processo. Normalmente uma aplicação possui apenas um `AppDomain`, porém, no executor, o segundo `AppDomain` foi criado para restringir o acesso do código potencialmente malicioso.

A classe `SecureTaskLoader` é responsável por carregar a tarefa em um `AppDomain` separado. No quadro 19 é exibido o método `LoadTask` dessa classe.

```

// Carrega a tarefa dentro de um AppDomain seguro
public ITask LoadTask(Task task)
{
    // Obtém o diretório da aplicação
    string dir1 = Path.GetDirectoryName(this.GetType().Assembly.Location);

    // Define as permissões do AppDomain que será criado
    // Permite a execução e o acesso aos diretório de entrada e saída
    PermissionSet permSet = new PermissionSet(PermissionState.None);
    permSet.AddPermission(new SecurityPermission(
        SecurityPermissionFlag.Execution));
    permSet.AddPermission(new FileIOPermission(
        FileIOPermissionAccess.Read, task.Executor.Directory));
    permSet.AddPermission(new FileIOPermission(
        FileIOPermissionAccess.Read, dir1));
    permSet.AddPermission(new FileIOPermission(
        FileIOPermissionAccess.PathDiscovery, task.Executor.Directory));
    permSet.AddPermission(new FileIOPermission(
        FileIOPermissionAccess.PathDiscovery, dir1));
    permSet.AddPermission(new FileIOPermission(
        FileIOPermissionAccess.AllAccess, task.InputDirectory));
    permSet.AddPermission(new FileIOPermission(
        FileIOPermissionAccess.AllAccess, task.OutputDirectory));

    foreach (var secureDirectory in secureDirectories)
    {
        // Permite que a tarefa acesse determinados diretórios
        permSet.AddPermission(new FileIOPermission(
            FileIOPermissionAccess.AllAccess, secureDirectory));
    }

    // Cria um AppDomain com as restrições de segurança
    AppDomainSetup s = new AppDomainSetup();
    s.ApplicationBase = task.Executor.Directory;

    StrongName fullTrustAssembly = typeof(WrappedTask).Assembly
        .Evidence.GetHostEvidence<StrongName>();
    AppDomain appDomain = AppDomain.CreateDomain(
        "TaskDomain", null, s, permSet, fullTrustAssembly);

    // Obtém o nome do assembly a ser executado
    string libraryName = Path.GetFileNameWithoutExtension(
        task.Executor.LibraryFilename);

    try
    {
        // Instancia a classe dentro do AppDomain que irá chamar a tarefa
        Type type = typeof(WrappedTask);
        ObjectHandle handle = Activator.CreateInstanceFrom(appDomain,
            type.Assembly.ManifestModule.FullyQualifiedName,
            type.FullName);
        WrappedTask wrappedTask = (WrappedTask)handle.Unwrap();
        // Obtém a instância da tarefa dentro do AppDomain
        wrappedTask.LoadTask(dir1,
            task.Executor.Directory,
            libraryName,
            task.Executor.EntryPoint);
        currentAppDomain = appDomain;
        return wrappedTask;
    }
    catch (FileNotFoundException)
    {
        string msg = String.Format(Messages.LIBRARY_NOT_FOUND,
            task.Executor.LibraryFilename, task.Name);
        throw new InvalidOperationException(msg);
    }
}

```

Quadro 19 - Método LoadTask da classe SecureTaskLoader

No método `LoadTask`, inicialmente são definidas as permissões que o `AppDomain` terá, como, por exemplo, permissão de execução e acesso ao diretório de entrada e saída de arquivos. A seguir, o `AppDomain` é criado com essas permissões e a tarefa é carregada nele. O método retorna essa tarefa carregada.

O executor também é responsável por coletar os pontos de processamento de uma determinada tarefa. Os pontos de processamento são calculados utilizando duas informações: o tempo de processamento da tarefa e o desempenho da máquina.

Durante a configuração do executor é executado um *benchmark* na máquina utilizando a biblioteca SciMark e é registrado o seu desempenho. Quando uma tarefa é executada, o valor medido no *benchmark* é multiplicado pelo tempo de processamento e o valor resultante representa os pontos de processamento. O *benchmark* garante que máquinas mais rápidas ganharão mais pontos que uma máquina lenta.

3.3.4 Sistema Web

No sistema web estão o distribuidor e o portal, ambos são acessados via web. O distribuidor é acessado pelo executor via *web services* e o portal é acessado pelo usuário diretamente pelo navegador.

O sistema web foi desenvolvido em ASP.NET utilizando a biblioteca ASP.NET MVC. A seguir são dados maiores detalhes da implementação do distribuidor e do portal.

3.3.4.1 Acesso a dados

Para o acesso a dados do sistema web foi utilizada a biblioteca NHibernate. Essa biblioteca faz o mapeamento objeto-relacional, simplificando as rotinas que acessam o banco de dados. Com ela pode-se trabalhar diretamente com objetos, sem a necessidade de escrever comandos usando a *Structured Query Language* (SQL).

No quadro 20 pode-se verificar a implementação de uma classe utilizada no acesso a dados. Essa classe, conforme pode ser constatado, não tem nenhuma dependência do framework de acesso a dados.

```

// Classe representando um Grupo cadastrado no sistema
public class Group
{
    // Identificador do grupo (usado internamente pelo sistema)
    public virtual Guid Id { get; set; }
    // Nome do grupo
    public virtual string Name { get; set; }
    // Lista de usuários pertencentes ao grupo
    public virtual IList<User> Users { get; protected set; }

    // Construtor da classe
    public Group()
    {
        // Ao instanciar um grupo, inicializa a lista
        // de usuários com 0 registros.
        Users = new List<User>();
    }
}

```

Quadro 20 - Classe Group

A maneira tradicional de se mapear os objetos utilizados pelo NHibernate é via arquivos XML. No projeto optou-se por utilizar a biblioteca Fluent NHibernate, que permite que o mapeamento seja feito dentro do projeto, em C#.

No quadro 21 pode-se observar o mapeamento da classe `Group`. O mapeamento é feito definindo-se uma classe que estende `ClassMap<T>` na qual cada propriedade é mapeada. No mapeamento são usadas convenções, como por exemplo, o nome do campo da tabela por padrão é o mesmo nome da propriedade.

```

// Classe utilizada para mapear a tabela do Grupo
public sealed class GroupMap : ClassMap<Group>
{
    public GroupMap()
    {
        // Define o campo Id como chave primária da tabela
        Id(x => x.Id);
        // Mapeia o campo nome (não nulo e único)
        Map(x => x.Name).Not.Nullable().Unique();
        // Mapeia a lista de usuários do grupo
        HasMany(x => x.Users).Cascade.AllDeleteOrphan();
    }
}

```

Quadro 21 - Classe GroupMap

3.3.4.2 Distribuidor

A principal rotina do distribuidor é a atribuição de uma tarefa a um executor. Essa rotina é implementada no método `GetPendingTask` da classe `DistributionService`, conforme exibido no quadro 22.

```

// Método que obtém uma tarefa pendente a ser processada
public Task GetPendingTask(Guid clientId)
{
    // Realiza a operação dentro de uma transação
    using (var transaction = session.BeginTransaction())
    {
        // Obtém a lista de tarefas
        var data = distributorCache.GetData();
        // Obtém a primeira tarefa a ser processada
        TaskInfo t = data.Tasks.FirstOrDefault(x => x.RunningBy < 2);
        if (t == null)
        {
            // Se não há tarefas pendentes, não retorna nada.
            return null;
        }
        else
        {
            // Se há uma tarefa pendente, indica qual
            // máquina está processado-a
            ClientMachine clientMachine =
                session.Load<ClientMachine>(clientId);
            Task pendingTask = session.Load<Task>(t.Id);
            pendingTask.Status = TaskStatus.Running;
            TaskProcessor processor = new TaskProcessor();
            processor.Client = clientMachine;
            processor.StartedAt = DateTime.Now;
            processor.State = ProcessorState.Running;
            processor.Task = pendingTask;
            processor.LastHeartBeat = processor.StartedAt;
            pendingTask.Processors.Add(processor);

            // Finaliza a transação
            transaction.Commit();

            // Atualiza a quantidade de máquinas executando a tarefa
            // e reordena a lista de tarefas pendentes
            t.RunningBy++;
            data.Sort();

            // Retorna a tarefa a ser executada
            return pendingTask;
        }
    }
}

```

Quadro 22 - Método GetPendingTask da classe DistributionService

Nesse método, primeiramente é pego um *cache* contendo a lista de tarefas pendentes (chamada do método `distributorCache.GetData()`). Nesse *cache* é pega a primeira tarefa da lista que não esteja sendo executada por duas máquinas. Essa lista é ordenada de acordo com a pontuação de cada grupo, conforme é exibido mais adiante. Se uma tarefa foi encontrada, é criada a informação definida na classe `TaskProcessor`, que define quais máquinas estão processando a tarefa, bem como data de início, fim e outras informações relevantes.

O valor retornado na chamada `distributorCache.GetData()` contém uma lista de tarefas representadas pela classe `TaskInfo`, que é exibida no quadro 23.

```

// Classe contendo os dados da tarefa
// (utilizada na fila de tarefas pendentes)
public class TaskInfo
{
    public Guid Id { get; set; }
    public GroupInfo Group { get; set; }
    public int RunningBy { get; set; }
    public DateTime SubmittedOn { get; set; }

    // Obtém a quantidade de pontos da tarefa
    // Esse valor determina a sua posição na fila
    public double Points
    {
        get { return Group.Points - RunningBy*1000; }
    }
}

```

Quadro 23 - Classe TaskInfo

A classe `TaskInfo` depende da classe `GroupInfo`, exibida no quadro 24, que contém as informações de pontuação de cada grupo.

```

// Classe representando um Grupo
// (utilizada na fila de tarefas pendentes)
public class GroupInfo
{
    public Guid Id { get; set; }
    private long givenPoints;
    private long usedPoints;
    private double points;

    // Indica a quantidade de pontos cedidos a grade
    public long GivenPoints
    {
        get { return givenPoints; }
        set { givenPoints = value; CalculatePoints(); }
    }

    // Indica a quantidade de pontos usados
    public long UsedPoints
    {
        get { return usedPoints; }
        set { usedPoints = value; CalculatePoints(); }
    }

    // Relação entre pontos cedidos e usados pelo grupo
    public double Points
    {
        get { return points; }
    }

    // Recalcula a relação entre pontos cedidos e usados
    private void CalculatePoints()
    {
        double u = usedPoints == 0 ? 1 : usedPoints;
        points = givenPoints/u;
    }
}

```

Quadro 24 - Classe GroupInfo

No método `CalculatePoints` da classe `GroupInfo` é aplicada a fórmula para cálculo dos pontos do grupo. A propriedade `Points` desta classe é utilizada pela propriedade `Points` da classe `TaskInfo` para calcular os pontos de cada tarefa.

A ordenação das tarefas é feita no método `Sort` da classe `DistributionData`,

conforme exibido no quadro 25.

```
// Faz a ordenação das tarefas
public void Sort()
{
    // Ordena a lista de tarefas (usando lamda expression)
    Tasks.Sort((x,y)=>
    {
        int idx = y.Points.CompareTo(x.Points);
        if (idx == 0)
        {
            // Se duas tarefas tem os mesmos pontos
            // a mais antiga tem maior prioridade
            idx = x.SubmittedOn.CompareTo(y.SubmittedOn);
        }

        return idx;
    });
}
```

Quadro 25 - Método Sort da classe DistributionData

As tarefas são ordenadas de acordo com seus pontos (propriedade `Points`), que por sua vez dependem da pontuação do seu grupo. Caso duas tarefas tenham pontuação igual, a tarefa mais antiga terá maior prioridade.

3.3.4.3 Implementação do portal

O portal para submissão de tarefas foi desenvolvido utilizando a biblioteca ASP.NET MVC. Com o padrão MVC a lógica da tela, a classe de modelo e a visualização da tela ficam bem definidas, cada uma sendo implementada separadamente.

As classe de modelo utilizadas no portal estão expostas na figura 10. Essas classes são utilizadas tanto no portal, como no distribuidor e também para a persistência dos dados.

As classes de controle, por conterem apenas a lógica da tela, são bastante simples. No quadro 26 é apresentado um trecho da classe `GroupController` que representa a classe de controle da tela de cadastro de grupos .

```

// Controller da tela de Grupos
public class GroupController : Controller
{
// ...
// Método chamado ao exibir a lista de grupos
public ActionResult Index()
{
// Obtém os grupos do sistema ordenando pelo nome
// Na ordenação é usada uma lambda expression
var groups = session
    .Query<Group>()
    .OrderBy(x => x.Name)
    .ToList();

// Renderiza a View passando a lista de grupos
return View(groups);
}

// Método chamado ao exibir a edição de um grupo
public ActionResult Edit(Guid id)
{
// Obtém o grupo pelo Id.
Group group = session.Load<Group>(id);
GroupModel model = new GroupModel();
model.Name = group.Name;
model.Id = group.Id;

// Renderiza a View passando o grupo a ser exibido.
return View(model);
}
// ...
}

```

Quadro 26 - Classe GroupController

Na classe `GroupController`, o método `Index` é chamado ao exibir a listagem de grupos, enquanto que o método `Edit` é chamado ao exibir a tela de edição. Na listagem, é feita uma consulta no banco de dados utilizando o NHibernate, retornando todos os grupos ordenados pelo nome. Na edição é carregado o grupo com o `Id` especificado e este é exibido na tela.

A visualização das telas é definida em um arquivo ASPX que contém um *template* da página. Esse arquivo ASPX recebe o modelo processado pela classe de controle, e a partir dele processa os dados a serem exibidos. No quadro 27 é exibido um trecho do arquivo de visualização da tela de listagem de grupos.

```

<table>
  <tr>
    <th>&nbsp;</th>
    <th style="min-width: 150px">Nome</th>
    <th>Usuários</th>
  </tr>
  <% foreach (var group in ViewData.Model) { %>
    <tr>
      <td>
        <a title="Exibir detalhes" class="icon"
          href='<%: Url.Action("Edit", new {Id=group.Id}) %>'>
        <img border="0" width="16" height="16"
          src='<%: Url.Content("~/Content/img/detail.png") %>' />
        </a>
      </td>
      <td><%: group.Name %></td>
      <td align="center"><%: group.Users.Count %></td>
    </tr>
  <% } %>
</table>

```

Quadro 27 - Arquivo de visualização da tela de listagem de grupos

3.4 OPERACIONALIDADE DA IMPLEMENTAÇÃO

Esta seção tem por objetivo apresentar a operacionalidade do software desenvolvido. Para tanto, primeiramente é apresentada a montagem do ambiente da grade computacional, o que envolve a criação de grupos e usuários, bem como a configuração das máquinas que farão a execução das tarefas.

A seguir é apresentada a execução de tarefas na grade. São apresentados os passos necessários para a criação de uma tarefa, submissão dela na grade e obtenção dos resultados após o processamento.

3.4.1 Ambiente

Antes de submeter qualquer tarefa é necessário configurar o ambiente da grade computacional. Essa configuração consiste na criação de grupos e usuários e na configuração das máquinas que farão a execução das tarefas, conforme detalhado a seguir.

3.4.1.1 Criar grupo e usuário

Acessando o portal com um usuário que tenha o papel de super administrador, pode-se acessar o menu **Grupos** onde são exibidos os grupos existentes e onde é possível criar um novo grupo. Na figura 15 é exibida a tela de listagem de grupos.



Usuário: **diogo** [Sair]

PrestoGrid
Portal de submissão de tarefas

[Inicial](#) [Jobs](#) [Executor](#) [Grupos](#) [Usuários](#)

Grupos

Apenas usuários com papel de "Super Administrador" podem criar grupos.

Novo grupo

Nome	Usuários
Grupo 1	1

Figura 15 - Tela de listagem de grupos

Ao clicar no *link* **Novo grupo** é exibida ao usuário a tela apresentada na figura 16. Nessa tela é necessário apenas definir o nome do grupo a ser criado e a seguir clicar em **Criar**.

The screenshot shows the 'Grupos' (Groups) page in the PrestoGrid application. The user is logged in as 'diogo'. The page has a navigation menu with 'Inicial', 'Jobs', 'Executor', 'Grupos', and 'Usuários'. The main content area is titled 'Grupo' and contains a form to create a new group. The form has a text input field labeled 'Nome' with the value 'Novo grupo' and a 'Criar' button. Below the form is a 'Voltar' link. To the left of the form, there is an icon of a person and a text box explaining that only users with the 'Super Administrador' role can create groups.

Figura 16 - Tela de inclusão de grupo

Depois de criado o grupo, pode-se acessar o menu *Usuários* que exibe a lista de usuários cadastrados no sistema, conforme exibido na figura 17.

The screenshot shows the 'Usuários' (Users) page in the PrestoGrid application. The user is logged in as 'diogo'. The page has a navigation menu with 'Inicial', 'Jobs', 'Executor', 'Grupos', and 'Usuários'. The main content area is titled 'Usuários' and contains a table listing the users. Above the table is a 'Novo usuário' link with a plus icon. To the left of the table, there is an icon of two people and a text box explaining that users with the 'Super Administrador' role can view all users, while users with the 'Administrador do Grupo' role can only view users in their group.

Nome	UserID	Tipo	Grupo
diogo	diogo	Super administrador	Grupo 1

Figura 17 - Tela de listagem de usuários

Ao clicar no *link* Novo usuário é apresentada a tela exibida na figura 18. Nessa tela o usuário deve definir os dados do usuário como por exemplo, nome, identificação e tipo de usuário.

Usuário: diogo [Sair]

PrestoGrid
Portal de submissão de tarefas

Inicial Jobs Executor Grupos **Usuários**

Usuário

Nome
Novo usuário

Identificação
novo_usuario

Senha
••••

Grupo
Novo grupo

Tipo de usuário
Usuário comum

Criar

[Voltar](#)

Como você possui o papel de "Super Administrador" poderá criar usuários de qualquer grupo.

Usuários com papel de "Administrador do Grupo" só podem criar usuários pertencentes ao seu grupo e não podem criar usuários com o papel de "Super Administrador".

Figura 18 - Tela de inclusão de usuário

Cada pessoa que deseja submeter ou executar tarefas deve ter um usuário no sistema. Esse usuário obrigatoriamente fará parte de um grupo.

3.4.1.2 Iniciar executor nas máquinas da grade

Após criar o usuário deve-se fazer o download do programa de execução da grade e instalar esse programa nas máquinas que farão a execução das tarefas. O download é feito no portal, acessando o menu `Executor`, conforme exibido na figura 19.

Usuário: diogo [Sair]

PrestoGrid
Portal de submissão de tarefas

Inicial Jobs **Executor** Grupos Usuários

Executor

Aqui você pode fazer o download do programa de execução da grade e obter mais informações de como criar uma tarefa.

Download

Faça o download do programa no link abaixo para executar tarefas da grade em seu computador.

[Download do executor](#)

Figura 19 - Tela de *download* do executor

O executor não precisa ser instalado, ele é empacotado em um arquivo Zip e pode ser descompactado em qualquer diretório. Ao fazer o download dele basta descompactar o arquivo Zip e executar o programa.

Ao iniciar o executor pela primeira vez é exibida a tela de configuração, conforme figura 20. Nessa tela deve-se informar o usuário e senha da pessoa, bem como o nome da máquina no qual o executor está instalado. O nome da máquina é utilizado apenas para melhor identificação no portal.

Ao confirmar os dados, o sistema executa um teste de desempenho na máquina e configura o executor para acessar a grade. O teste de desempenho pode demorar alguns segundos, variando conforme a velocidade do processador.



Figura 20 - Tela de configuração do executor

Ao final da configuração o executor ficará monitorando as tarefas a serem executadas. Assim que um *job* for submetido à grade o executor iniciará o seu processamento. Na Figura 21 é apresentada a tela exibida ao usuário enquanto o executor está aguardando uma tarefa a ser executada.



Figura 21 - Tela principal do executor

3.4.2 Execução do *job*

A seguir são descritos os passos necessários para a criação de um *job*, submissão deste e obtenção dos resultados.

3.4.2.1 Obter API da grade

Ao criar uma tarefa a ser executada na grade é necessário referenciar as bibliotecas utilizadas pelo executor. Para obter essas bibliotecas e maiores informações a respeito da criação de tarefas, o usuário deve acessar o *link* Tutorial dentro do menu Executor, conforme exibido na figura 22.



Figura 22 - Tela para obtenção da API da grade

3.4.2.2 Criar tarefa

Depois de obter as bibliotecas utilizadas na grade é possível criar a tarefa a ser submetida. Deve-se criar um novo projeto do tipo `Class Library` no Visual Studio e nele criar uma classe implementando a interface `ITask`, conforme exibido na figura 23.

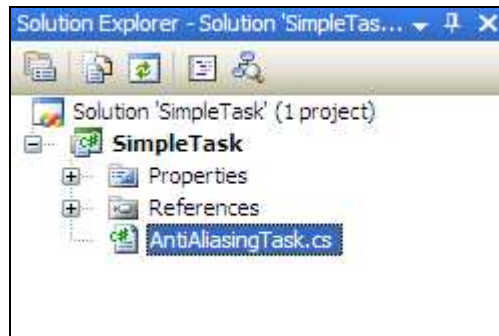


Figura 23 - Projeto de uma tarefa

No quadro 28 pode-se verificar a implementação de uma tarefa.

```

public class AntiAliasingTask : ITask
{
    public void Run(ITaskContext context)
    {
        // Obtém o parâmetro que indica quantas vezes
        // deve aplicar o antialiasing
        int repeatCount;
        if (!int.TryParse(context.Parameters.GetValue("RepeatCount"),
            out repeatCount))
        {
            repeatCount = 1;
        }

        // Busca no diretório de entrada os
        // arquivos a serem processados
        string[] files = Directory.GetFiles(
            context.InputDirectory, "*.jpg");
        foreach (var file in files)
        {
            // Carrega a imagem
            Bitmap bitmap = new Bitmap(file);

            // Aplica o antialiasing
            for (int i = 0; i < repeatCount; i++)
            {
                ProcessaAntialiasing(bitmap);
            }

            // Salva o arquivo processado
            string novoArquivo = Path.Combine(context.OutputDirectory,
                Path.GetFileName(file));
            bitmap.Save(novoArquivo, ImageFormat.Jpeg);
        }
        //...
    }
}

```

Quadro 28 - Implementação da tarefa AntiAliasingTask

Na classe apresentada no quadro 28 o método `ProcessaAntialiasing` foi omitido pois não é o foco do exemplo e para manter o código simples.

A classe `AntiAliasingTask` lê as imagens que estão no diretório de origem, aplica o *antialiasing* e por fim, salva a imagem processada no diretório de saída. O parâmetro `RepeatCount` define quantas vezes será processado o *antialiasing*.

Após compilar o projeto será gerado um *assembly* denominado `SimpleTask` que será utilizado na criação do *job*.

3.4.2.3 Criar *job*

Após implementar a classe que fará o processamento, deve-se definir o *job* a ser submetido à grade. No *job* são definidas as tarefas a serem executadas usando o *assembly* criado anteriormente.

É criado um diretório que contém duas subpastas denominadas *data* e *executor*. No diretório *executor* ficam os arquivos utilizados para efetuar o processamento, nesse caso o *assembly* *SimpleTask.dll*. No diretório *data* ficam os arquivos a serem processados. Neste caso ficarão quatro imagens a serem processadas.

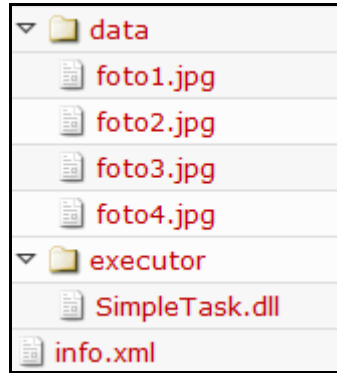
Após copiar os arquivos para as devidas pastas é criado um arquivo XML denominado *info.xml* que contém a definição das tarefas. O quadro 29 apresenta o XML utilizado na configuração deste exemplo.

```
<?xml version="1.0" encoding="utf-8" ?>
<job name="Processa imagens">
  <executor>
    <library name="SimpleTask.dll"/>
    <entrypoint name="SimpleTask.AntiAliasingTask" />
    <params>
      <param name="RepeatCount" value="2" />
    </params>
  </executor>
  <tasks>
    <task name="Imagem 1">
      <data>
        <file>foto1.jpg</file>
      </data>
    </task>
    <task name="Imagem 2">
      <data>
        <file>foto2.jpg</file>
      </data>
    </task>
    <task name="Imagem 3">
      <data>
        <file>foto3.jpg</file>
      </data>
    </task>
    <task name="Imagem 4">
      <data>
        <file>foto4.jpg</file>
      </data>
    </task>
  </tasks>
</job>
```

Quadro 29 - XML de definição do *job*

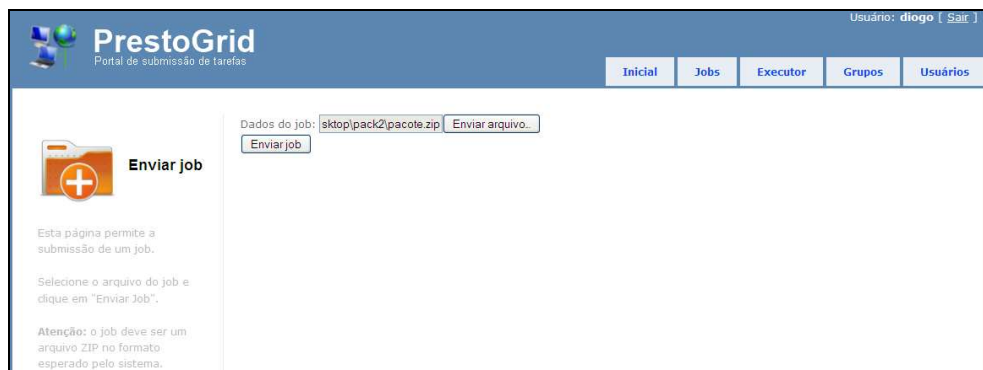
No XML foi definida a classe utilizada no processamento, e quatro tarefas. Cada tarefa processa uma imagem.

Ao final deve-se compactar no formato Zip o diretório do *job*. Esse diretório estará com a estrutura exibida na figura 24.

Figura 24 - Estrutura do arquivo do *job*

3.4.2.4 Submeter *job*

Para submeter um *job* o usuário acessa o menu `Job` e clica no link `Enviar Job`. Ao efetuar o envio é apresentada a tela exibida na figura 25. Nessa tela o usuário seleciona o arquivo Zip e faz *upload* dele.

Figura 25 - Tela de submissão de *jobs*

Após submeter um *job* o usuário é redirecionado para a lista de *jobs*, conforme exibido na figura 26.

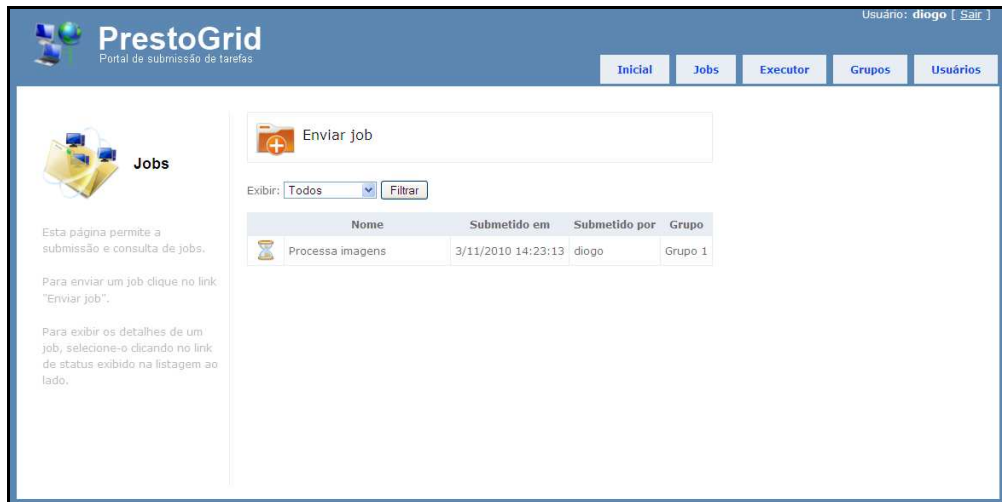


Figura 26 - Tela de consulta de *jobs*

3.4.2.5 Execução do *job* na grade

Nos testes foram criados três usuários, cada um com um executor. Foi então submetida uma tarefa à grade contém quatro tarefas. Conforme exibido na figura 27, três tarefas são processadas simultaneamente, cada uma por um executor. Quando um dos executores finaliza seu processamento a quarta tarefa, que continua pendente, é então processada.

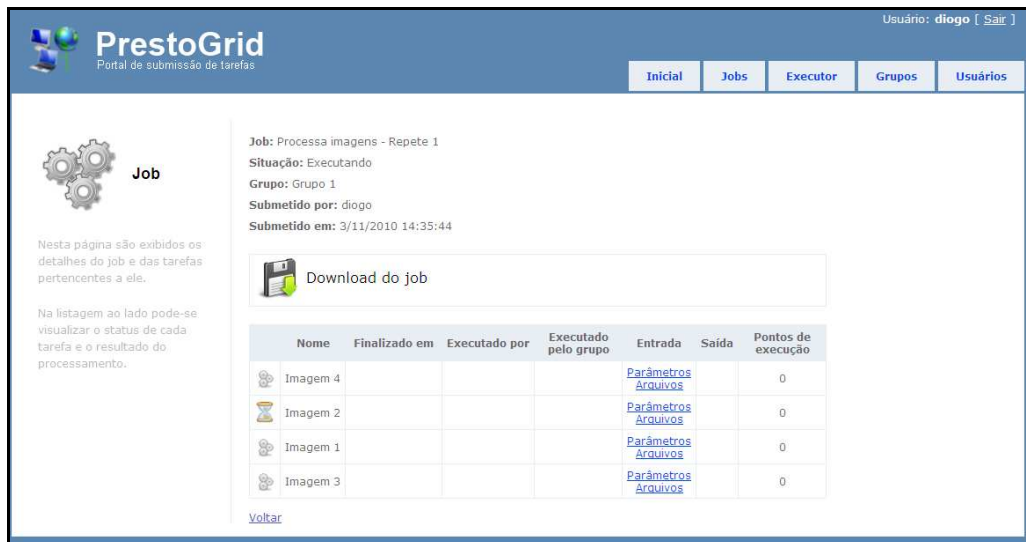


Figura 27 - Tela de detalhes do *job*

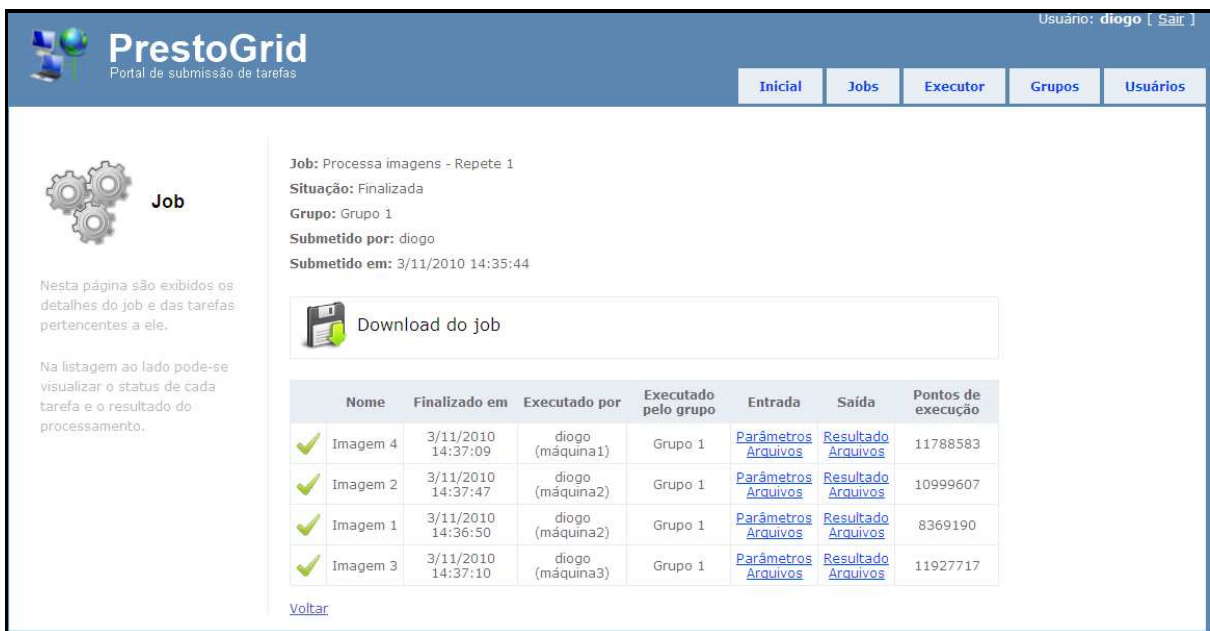
3.4.2.6 Obter resultados

Ao exibir os detalhes de um *job* é possível obter os dados usados como entrada para o

processamento, assim como os resultados da tarefa.

Na figura 28 pode-se verificar que em cada tarefa são exibidos alguns links onde pode-se obter as informações de entrada e saída da tarefa. Isso possibilita obter os dados individualmente de cada tarefa.

Para obter todos os dados resultantes do processamento do *job* pode-se clicar no *link* Download do *job*. Ao clicar nesse *link* é feito o download de um arquivo Zip que contém os arquivos de saída e resultados de todas as tarefas processadas.



PrestoGrid
Portal de submissão de tarefas

Usuário: diogo [Sair]

Inicial Jobs Executor Grupos Usuários

Job

Job: Processa imagens - Repete 1
 Situação: Finalizada
 Grupo: Grupo 1
 Submetido por: diogo
 Submetido em: 3/11/2010 14:35:44

Download do job

	Nome	Finalizado em	Executado por	Executado pelo grupo	Entrada	Saída	Pontos de execução
✓	Imagem 4	3/11/2010 14:37:09	diogo (máquina1)	Grupo 1	Parâmetros Arquivos	Resultado Arquivos	11788583
✓	Imagem 2	3/11/2010 14:37:47	diogo (máquina2)	Grupo 1	Parâmetros Arquivos	Resultado Arquivos	10999607
✓	Imagem 1	3/11/2010 14:36:50	diogo (máquina2)	Grupo 1	Parâmetros Arquivos	Resultado Arquivos	8369190
✓	Imagem 3	3/11/2010 14:37:10	diogo (máquina3)	Grupo 1	Parâmetros Arquivos	Resultado Arquivos	11927717

[Voltar](#)

Figura 28 - Tela de detalhes do *job* com resultados

3.4.2.7 Submeter tarefa com código malicioso

O mecanismo de segurança da grade impede que códigos maliciosos sejam executados. Para demonstrar isso pode-se submeter uma tarefa que acesse algum diretório do sistema, como demonstrado na figura 29.


```
public class AcessaDisco : ITask
{
    public void Run(ITaskContext context)
    {
        // Tenta listar os arquivos do diretório C:\
        string[] arquivos = Directory.GetFiles(@"C:\");
    }
}
```

Figura 29 - Tarefa com código malicioso

Ao executar essa tarefa será gerado um erro de falta de permissão. A tarefa então ficará

com a indicação de erro conforme exibido na figura 30. Na coluna saída da tarefa pode-se exibir os detalhes do erro.

Job: Acessa Disco
Situação: Com erro
Grupo: Grupo 1
Submetido por: diogo
Submetido em: 12/12/2010 16:28:04

 **Download do job**


	Nome	Finalizado em	Executado por	Executado pelo grupo	Entrada	Saída	Pontos de execução
	Tarefa 1	12/12/2010 16:29:27	diogo (diogo-pc)	Grupo 1	Parâmetros Arquivos	Erro	0

Figura 30 - Processamento da tarefa com erro

3.5 RESULTADOS E DISCUSSÃO

Além dos testes realizados durante a implementação, foi realizado um teste comparando o desempenho de um determinado processamento sendo realizado em apenas uma máquina e usando várias máquinas na grade. Neste teste, inicialmente um *job* com quatro tarefas foi processado em apenas uma máquina, coletando-se ao final o tempo de processamento e o uso do processador. A seguir, repetiu-se a operação utilizando de duas a cinco máquinas.

Na tabela 1 são apresentados os resultados obtidos nos testes. Os pontos de processamento correspondem a quantidade de processamento utilizado para processar o *job*. 100% representa o processamento sem desperdício, 120% representa o processamento com 20% de desperdício.

Tabela 1 - Resultado de testes da grade

Quantidade de máquinas	1	2	3	4	5
Tempo de processamento	03m 15s	1m 43s	1m 35s	0m 53s	0m 51s
Pontos de processamento	100%	102%	119%	102%	124%

O desperdício de processamento ocorre devido a replicação de tarefas. Nesse teste, como havia 4 tarefas, ao utilizar um número ímpar de máquinas o desperdício de

processamento ficou maior.

Também realizou-se um teste a fim de verificar a quantidade de processamento utilizada durante a operação normal de um *desktop*. Para esse teste, foram utilizados 9 computadores, todos com o processador Intel Core2Duo E6600 (com a frequência de 2400MHz). Os computadores foram analisados durante um período normal de trabalho, por 9 horas, sendo utilizados no desenvolvimento de software. Obteve-se como resultado um uso de apenas 7,32% do processador de cada máquina. Esse valor pode variar conforme o uso feito do computador, mas fica claro que o uso da grade permite o aproveitamento de grande parte do processamento que normalmente seria desperdiçado.

Acerca do portal de submissão de tarefas, pelo fato de estar acessível via web, possibilita que qualquer pessoa que tenha um usuário cadastrado possa submeter tarefas. O acompanhamento das tarefas e obtenção do resultados também é bastante simples, por serem feitos diretamente no portal.

A distribuição das tarefas foi bastante eficaz. O algoritmo Workqueue com Replicação se mostrou adequado na execução das tarefas do tipo *bag-of-tasks*. A replicação adotada pelo algoritmo, embora gere desperdício de processamento, consegue fazer com que os *jobs* sejam finalizados em menos tempo.

O software desenvolvido atendeu totalmente os requisitos elicitados. Comparando o software com as ferramentas correlatas observa-se que ao contrário do BOINC e do Globus Toolkit, o software desenvolvido fornece a infraestrutura completa para a utilização da grade. Em comparação ao SETI@home, a ferramenta desenvolvida tem a vantagem de ser de uso geral, e não específica a um projeto. De maneira geral, a ferramenta desenvolvida apresenta fácil utilização, tanto para o usuário que deseja executar tarefas na grade como para aquele que deseja submeter tarefas.

4 CONCLUSÕES

Atualmente o poder de processamento de um computador é sub-utilizado na maior parte do tempo. Em média, menos de 10% da capacidade de processamento de um processador é utilizada. Essa capacidade de processamento pode ser usada para outros fins, como já foi explorado pelo projeto SETI@home e pela proposta apresentada neste trabalho.

O presente trabalho atingiu seus objetivos, criando um software para a submissão e gerenciamento de uma grade computacional. O portal para submissão de tarefas se mostrou de simples utilização e a forma utilizada para criação de tarefas também não apresenta grandes dificuldades. Pelo fato de utilizar o .NET Framework, que é conhecido por uma grande quantidade de desenvolvedores, a criação de tarefas a serem submetidas a grade é bastante simples. O uso do .NET Framework também possibilita que as tarefas sejam criadas em uma variedade de linguagens, tais como C#, VB.NET, C++ e outras, basta que a linguagem seja capaz de compilar para o .NET Framework.

Durante o desenvolvimento do trabalho percebeu-se que a grade possui uma complexidade bem maior que a inicialmente prevista. Uma grade computacional é composta por várias partes que se interligam, cada uma dessas partes é complexa o suficiente para ser objeto de estudo por si só. Cada parte da grade é passível de um estudo mais aprofundado, sendo possível a extensão do presente trabalho a fim de aprimorar essas partes. O escalonador, por exemplo, que é responsável pela distribuição das tarefas a serem processadas, possui uma série de algoritmos que podem ser utilizados e se adaptam melhor a determinadas circunstâncias. Também o executor pode ser aperfeiçoado, provendo mais serviços durante a execução das tarefas.

Um ponto observado durante o desenvolvimento é a segurança na execução das tarefas. Como a grade pode ser utilizada por um grande número de pessoas, de entidades e grupos diferentes, é necessário ter completo controle do que cada tarefa pode executar. Uma tarefa com código mal-intencionado não pode comprometer os computadores participantes da grade e ainda comprometer a confiabilidade da grade como um todo. Os mecanismos de segurança oferecidos pelo .NET Framework se mostraram adequados a esse fim e de fácil utilização.

A principal vantagem em relação aos demais projetos é a fácil utilização da grade, tanto pelo usuário que deseja processar tarefas como por aquele que deseja submeter tarefas. O uso do programa de execução é bastante simples, não requer a instalação de nenhum

componente adicional e a única configuração exigida é o usuário e senha do usuário na grade. A criação de tarefas é bastante simplificada, bastando a criação de um *assembly* .NET e posteriormente empacotado em um Zip com o formato aceito pela grade. Uma desvantagem do projeto é que sua utilização está restrita ao ambiente Windows, porém, é possível adaptar o projeto para que ele seja utilizado em outros ambientes.

A principal limitação do software desenvolvido diz respeito ao tipo de processamento efetuado. A grade é capaz de processar apenas tarefas do tipo *bag-of-tasks*, ou seja, tarefas que não tem dependências entre si. O processamento de tarefas fortemente acopladas e interdependentes não é adequado ao software desenvolvido.

4.1 EXTENSÕES

Uma grade computacional é bastante complexa e possibilita estudos em diversas áreas. Como extensões para o presente software sugere-se:

- a) possibilitar o uso de linguagens dinâmicas, tais como Ruby e Python, usando para isso os próprios recursos oferecidos pelo .NET e as implementações IronRuby e IronPython. Isso possibilita que ao submeter uma tarefa seja enviado um script no lugar de um *assembly*;
- b) criar uma ferramenta para automatizar a criação de *jobs*, facilitando a criação do Zip contendo as tarefas. É possível criar uma ferramenta de propósito geral, que monte qualquer pacote de *job* ou uma ferramenta específica, voltada para um único cenário que esteja adaptada para aceitar os parâmetros de entrada específicos de um tipo de tarefa. Por exemplo, se existe uma tarefa que processa imagens poderia ser criada uma ferramenta onde o usuário escolhe as imagens de entrada e os parâmetros de processamento e esta gere o *job* a ser submetido;
- c) desenvolver uma ferramenta para monitorar os *jobs* em execução e obter os resultados. Isso facilitaria o usuário que submete muitos *jobs*, fazendo com que ele não precise entrar constantemente no portal para verificar o estado da execução dos seus *jobs*;
- d) adaptar o software para utilizar a biblioteca Mono (NOVELL, 2010), possibilitando que a grade rode em outros sistemas operacionais tais como Linux, Mac OS X e FreeBSD;

- e) ajustar o portal de submissão de tarefas para exibir mais detalhes das máquinas participantes da grade. Pode-se exibir quantas máquinas participam da grade, quantas estão ativas e inativas no momento, quanto cada uma processou e outras informações relevantes;
- f) exibir um *ranking* de usuários e grupos que mais contribuem para a grade, incentivando uma maior participação dos usuários, conseguindo assim mais recursos computacionais.

REFERÊNCIAS BIBLIOGRÁFICAS

- ANDERSON, David P. **BOINC**: a system for public-resource computing and storage. [S.l.], 2004a. Disponível em: <http://boinc.berkeley.edu/grid_paper_04.pdf>. Acesso em: 26 mar. 2010.
- _____. **Public computing**: reconnecting people to science. [S.l.], 2004b. Disponível em: <<http://boinc.berkeley.edu/boinc2.pdf>>. Acesso em: 26 mar. 2010.
- ANDERSON, David P. et al. **SETI@home**: an experiment in public-resource computing. [S.l.], 2002. Disponível em: <<http://www.seas.gwu.edu/~jstanton/courses/cs235/papers/p56-anderson.pdf>>. Acesso em: 26 mar. 2010.
- ASPALLIANCE. **The perfect service**: part 2. [S.l.], 2006. Disponível em: <http://aspalliance.com/750_The_Perfect_Service__Part_2.2>. Acesso em: 12 dez. 2010.
- BRAGA, Reinaldo B. **Grades computacionais**. 2005. 53 f. Monografia (Tecnólogo em Telemática) – Curso Superior de Tecnologia em Telemática, Centro Federal de Educação Tecnológica do Ceará, Fortaleza.
- BRASILEIRO, Francisco V.; CIRNE, Walfredo; SILVA, Daniel P. da. **Trading cycles for information**: using replication to schedule bag-of-tasks applications on computational grids. Campina Grande, 2003. Disponível em: <<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.59.556&rep=rep1&type=pdf>>. Acesso em: 06 nov. 2010.
- BOINC. **BOINC**: software open-source para a computação voluntária e rede de computação. [S.l.], 2010. Disponível em: <<http://boinc.berkeley.edu/>>. Acesso em: 20 maio 2010.
- BUYA, Rajkumar. **Economic-based distributed resource management and scheduling for grid computing**. 2002. 180 f. Tese (Doutorado em Ciência da Computação) – School of Computer Science and Software Engineering Monash University, Melbourne. Disponível em: <<http://www.buya.com/thesis/thesis.pdf>>. Acesso em: 20 maio 2010.
- CIRNE, Walfredo. **Grids computacionais**: arquiteturas, tecnologias e aplicações. Campina Grande, 2002. Disponível em: <<http://www2.lsd.ufcg.edu.br/~walfredo/papers/Grids%20Computacionais-%20WSCAD.pdf>>. Acesso em: 26 mar. 2010.
- CIRNE, Walfredo et al. **Running bag-of-tasks applications on computational grids**: the MyGrid approach. Campina Grande, 2003. Disponível em: <<http://www.lsd.ufcg.edu.br/~walfredo/papers/Running%20Bag%20of%20Tasks%20Applications%20v14.pdf>>. Acesso em: 26 mar. 2010.

CONTI, Fabieli de. **Grades computacionais para processamento de alto desempenho**. Santa Maria, 2009. Disponível em: <<http://www-usr.inf.ufsm.br/~andrea/elc888/artigos/artigo3.pdf>>. Acesso em: 26 mar. 2010.

FOSTER, Ian. **The grid**: a new infrastructure for 21st century science. [S.l.], 2002. Disponível em: <<http://research.calit2.net/cibio/archived/thegrid.pdf>>. Acesso em: 26 mar. 2010.

FOSTER, Ian; KESSELMAN, Carl; TUECKE, Steven. **The anatomy of the grid**: enabling scalable virtual organizations. [S.l.], 2001. Disponível em: <<http://www.globus.org/alliance/publications/papers/anatomy.pdf>>. Acesso em: 26 mar. 2010.

FOWLER, Martin. **GUI architectures**. [S.l.], 2006. Disponível em: <<http://martinfowler.com/eaDev/uiArchs.html>>. Acesso em: 16 maio 2010.

GLOBUS ALLIANCE. **About Globus**. [S.l.], [2010?]. Disponível em: <<http://www.globus.org/toolkit/about.html>>. Acesso em: 25 mar. 2010.

GOLDCHLEGER, Andrei. **InteGrade**: um sistema de middleware para computação em grade oportunista. 2004. 124 f. Dissertação (Mestrado em Ciências) – Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo.

GREGORY, James. **Fluent NHibernate**: fluent mapping for your model. [S.l.], 2010. Disponível em: <<http://fluentnhibernate.org>>. Acesso em: 06 nov. 2010.

LOPES, Rafael F. **MAG**: uma grade computacional baseada em agentes móveis. 2006. 128 f. Dissertação (Mestrado em Engenharia de Eletricidade) – Centro de Ciências Exatas e Tecnologia, Universidade Federal do Maranhão, São Luís.

MAULO, Fabio. **NHibernate FORGE**: the official new home for the NHibernate for .NET community. [S.l.], 2009. Disponível em: <<http://nhforge.org>>. Acesso em: 06 nov. 2010.

MICROSOFT CORPORATION. **Code access security**. [S.l.], 2010a. Disponível em: <[http://msdn.microsoft.com/en-us/library/930b76w0\(VS.71\).aspx](http://msdn.microsoft.com/en-us/library/930b76w0(VS.71).aspx)>. Acesso em: 25 mar. 2010.

_____. **What is ASP.NET Web Forms**. [S.l.], 2010b. Disponível em: <<http://www.asp.net/web-forms/what-is-web-forms>>. Acesso em: 16 maio 2010.

NOVELL. **Main page**: Mono. [S.l.], [2010?]. Disponível em: <<http://www.mono-project.com/>>. Acesso em: 06 nov. 2010.

NUNIT. **NUnit home**. [S.l.], 2007. Disponível em: <<http://www.nunit.org/>>. Acesso em: 06 nov. 2010.

SETI@HOME. **SETI@home**: the search for extraterrestrial intelligence. [S.l.], 2009. Disponível em: <<http://seticlassic.ssl.berkeley.edu>>. Acesso em: 25 mar. 2010.

_____. **About SETI@home**. [S.l.], 2010. Disponível em: <http://setiathome.berkeley.edu/sah_about.php>. Acesso em: 25 mar. 2010.

TAING, Ricky. **A collection of phoenix-compatible C# benchmarks**. [S.l.], [2010?]. Disponível em: <<http://www.cs.ucsb.edu/~ckrintz/racelab/PhxC#Benchmarks/>>. Acesso em: 06 nov. 2010.