UNIVERSIDADE FEDERAL FLUMINENSE

ALEXANDRE SILVA VENITO

MyEasyGridPortal
Uma abordagem para facilitar o acesso a múltiplas grades computacionais usando a tecnologia Portlet

NITERÓI

2006

UNIVERSIDADE FEDERAL FLUMINENSE

ALEXANDRE SILVA VENITO

MyEasyGridPortal
Uma abordagem para facilitar o acesso a múltiplas grades computacionais usando a tecnologia Portlet

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre. Área de concentração: Processamento Distribuído e Paralelo.

Orientador:
Vinod Rebello

NITERÓI

2006

MyEasyGridPortal
Uma abordagem para facilitar o acesso a múltiplas grades computacionais usando a tecnologia Portlet

Alexandre Silva Venito

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre.

Aprovada por:

Prof. Ph.D. Eugene Francis Vinod Rebello / IC-UFF
(Presidente)

Profa. Ph.D. Maria Cristina Silva Boeres / IC-UFF

Prof. D.Sc. Bruno Richard Schulze / LNCC

Niterói, 20 de dezembro de 2006.

Muito obrigado a todos que estiveram ao meu lado durante todo ou parte do tempo que este trabalho levou para ser elaborado.

Agradecimentos

Meus sinceros agradecimentos ao meu orientador que se mostrou sempre paciente, persistente e compreensivo às dificuldades encontradas, ao meu amigo pessoal Rodrigo Lorca que foi meu guru para programação Java e problemas relacionados, ao meu amigo pessoal Arthur Pomeroy que me mostrou os "caminhos das pedras" do sistema operacional utilizado, aos meus colegas da UFF sempre prontos a ajudar, especialmente ao Jacques Silva, incansável e sempre solícito, e aos meus familiares e amigos que compreenderam minha falta de tempo em função da conclusão deste trabalho.

Resumo

A redução dos custos e aumento do poder de processamento e armazenamento de dados dos computadores pessoais, assim como o aumento das taxas de transferência das redes locais tem motivado a idéia da utilização de vários computadores de "uso comum" para a execução de aplicações paralelas e distribuídas. Além disso, a Internet tem crescido significativamente na disponibilidade de banda e redução da latência no tráfego de dados, principalmente devido a utilização de fibra óptica que facilita a comunicação entre pontos fisicamente distantes, uma vez que a distância física é um obstáculo a utilização do poder computacional de um significante número de computadores geograficamente distribuídos. O campo da Computação em Grades, que se caracteriza pela disponibilidade e utilização dos recursos oferecidos por esses "computadores de uso comum", se diferencia da computação distribuída por seu foco no compartilhamento de recursos em larga escala. Através desse compartilhamento, cientistas obtêm acesso, por exemplo, a poder computacional, banco de dados e instrumentos não disponíveis localmente e realizam suas pesquisas como antes não eram possíveis.

Sendo compartilhada e composta de recursos heterogêneos, grades computacionais são ambientes complexos de serem utilizados eficientemente. Numeros *softwares*, conhecidos como *service middleware* estão sendo desenvolvidos para tornar o uso de grades mais transparente para o usuário.

Infelizmente, a variedade de *middlewares* disponíveis e sua aplicação restrita, exige que cientistas e especialistas em computação tenham um profundo conhecimento de seu funcionamento e operação para usufruir de todas as vantagens e funcionalidades oferecidas por eles.

Uma solução que visa facilitar o uso de grades computacionais é a utilização de uma interface padronizada para seu acesso e operação. Portais geralmente são baseados na *Web*, necessitando que o usuário tenha somente conhecimentos básicos de acesso a páginas WWW para sua utilização. Além disso, estas páginas podem ser apresentadas como interface gráfica para facilitar a interatividade com o usuário e ocultar detalhes específicos da operação das grades. As facilidades oferecidas pelos portais fazem dele uma poderosa ferramenta capaz de promover o uso de grades pelas comunidades científicas.

Este trabalho contribui para facilitar a utilização de grades computacionais através da implementação de um portal baseado na padronização de *portlets* proposta "recentemente". A forma como foi desenvolvido permite interoperabilidade entre diferentes *middlewares* e múltiplas grades computacionais simultaneamente, além de permitir que outras funcionalidades sejam incorporadas independetemente da linguagem de programação que tenham sido escritas. Adicionalmente, o portal suporta personalização para que o usuário escolha o grau de especialização a que gostaria de ser exposto.

Palavras-chave: grid portal, gridsphere, portlets, JSR168, easygrid

Abstract

The falling costs and steady increases in processing capacity and data storage of personal computer, as well as throughput improvements on local networks have stimulated the idea of using such non-specialized computers to execute high performance parallel and distributed applications. Futhermore, the Internet has seen dramatic increases in available bandwidth and reductions in latency thanks to fibre optics which makes distance no longer an obstacle to harnessing the power of significant numbers of geographically dispersed computers. The field of Grid Computing distinguishes itself from distributed computing by its focus on large scales collaboration. Through this collaboration, scientist can obtain access to, for example, computing power, databases and instruments not available locally and carry out research which was not possible previously.

Being both shared and composed of heterogeneous resources, computational grids are complex environments to use effectively. Numerous softwares, known as *service middleware*, have and are being developed to make these difficulties transparent to the user. Unfortunately, both the variety of middleware available and their often restricted application (specialization) means that scientists and even computing specialists require a indepth understanding of the workings of, and how to operate, each chosen middleware in order to fully take advantage of the functionality they offer.

One solution that might significantly facilitate the use of computational grids is the adoption of a common standardized interface for access and operation. Portals are generally Web-based and thus only require that the user has a basic knowledge of how to access WWW pages. Furthermore, these pages can be presented in the form of a graphical interface to facilitate interactivity with the user and hide the specific details of operating computational grids. Features offered by portals make them powerful tools to further the use of grids in scientific communities.

This work contributes to facilitate the use of computional grids through the implementation of a portal, based on the recently proposed *portlet* standard. The design approach adopted permits the interoperability between different middlewares and multiple computational grids simultaneous and was developed to allow functionalities to be incorporated independently of the programming language in which they were written. Additionally, the portal supports personalization to allow users to chose the degree of specialization to which they would like to be exposed to.

Keywords: grid portal, gridsphere, portlets, JSR168, easygrid

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O avanço da tecnologia de fabricação dos semicondutores tem proporcionado uma melhoria significativa no poder de processamento dos microprocessadores atuais, assim como tem provocado a redução dos custos destes dispositivos para valores que tornam o computador pessoal um equipamento acessível a uma quantidade cada vez maior da população mundial. As redes computacionais também tornam-se mais eficientes, mais rápidas na transferência de dados e com menor latência, tornando-se mais baratas a cada dia. A associação destes dois fatores, avanço tecnológico e redução de preços, abre novos horizontes e motiva a utilização de grande nmero de máquinas em paralelo para a resolução de problemas antes só atacados pelos supercomputadores.

Um exemplo de aplicação de processamento distribuído pela Internet é o *SETI@home*[1], que é um experimento científico que procura por vida extraterrestre inteligente analisando os sinais captados pelo rádio-telescópio do Observatório de Arecibo em Porto Rico. Para processar todos os sinais captados, seria necessário um super-computador. Como as verbas destinadas ao projeto pelo governo americano não permitiam a compra de um equipamento com essas características, surgiu a idéia de se utilizar a capacidade individual de pequenos computadores trabalhando ao mesmo tempo e analisando pequenos pacotes de dados[2], o que foi conseguido utilizando-se o tempo ocioso dos computadores conectados à Internet. Para isso, foi desenvolvido um programa chamado *BOINC* que é o programa cliente sobre o qual roda o processamento do *SETI@home*, analisando os dados recebidos enquanto apresenta os sinais graficamente na tela e os envia de volta ao site. O usuário de Internet que quiser participar do experimento, instala o *BOINC* em sua máquina. Atualmente, segundo a página de estatísticas de uso do *BOINC*[3], o experimento conta com a

[1] http://setiathome.berkeley.edu/index.php
[2] http://www.setiathome.com.br/
[3] http://www.boincstats.com/index.phpé or=15

colaboração de 536.928 (quinhentos e trinta e seis mil, novecentos e vinte e oito) usuários espalhados pelo mundo.

Atualmente, algumas instituições de ensino e de pesquisa, como a Universidade da Califórnia (*UCLA*)[4] e algumas outras iniciativas como a *Globus Alliance*[5] têm trabalhado no desenvolvimento de sistemas capazes de disponibilizar a um usuário que esteja conectado em rede, normalmente pela internet, recursos computacionais oferecidos pela associação de vários computadores interligados em *clusters*, que são grupos de computadores interligados de forma que seus recursos sejam utilizados como se fossem oferecidos por um único computador. A idéia é que um usuário que necessite de processamento, interligue seu computador a este sistema, utilizando os recursos oferecidos conforme sua necessidade e demanda de processamento.

Dentre as vantagens oferecidas por uma grade computacional, podemos citar a facilidade na agregação de recursos sem a preocupação de sua localização física, acesso distribuído a diversos tipos de recursos, melhor aproveitamento de recursos ociosos, maior disponibilidade do sistema em geral e maior capacidade de processamento se comparado a máquinas isoladas, ou seja, esse tipo de compartilhamento traz benefícios significativos justificando o tempo e dinheiro atualmente investidos nas pesquisas relacionadas [22].

Devido a filosofia de funcionamento muito similar a do sistema de fornecimento de energia elétrica, conhecido em inglês como *Power Grid*, essa associação de computadores foi chamada de *Compute Grid* [28], ou "Grade Computacional" em português, muitas vezes também referida simplesmente como "grades".

Para entender melhor a analogia entre as grades e o sistema de fornecimento de energia elétrica, algumas comparações entre os dois sistemas podem ser feitas, como por exemplo: quando interligamos um equipamento elétrico à rede elétrica, não importa de onde aquela energia consumida está sendo gerada e nem que processo é utilizado em sua geração, hidroelétrico, ou termoelétrico, assim deve ser a grade, quando um processo, seja uma simples tarefa ou um complexo conjunto delas, é enviado para a grade para ser executado, não importa ao cliente, que neste caso é quem está enviando o processo, a topologia utilizada para a sua execução, o processo deve ser simplesmente executado de acordo com o objetivo dos usuários e como se os recursos utilizados para isso fossem locais. O poder computacional oferecido pela grade deve ser, assim como a energia elétrica, um produto simples de ser consumido, bastando que o cliente se conecte à rede de fornecimento

[4] http://www.ucla.edu/
[5] http://www.globus.org/

obedecendo alguns padrões básicos, como o protocolo utilizado.

Na prática, atualmente não conseguimos um funcionamento tão transparente como o do sistema de fornecimento de energia elétrica por várias razões, dentre elas [10]: o processo, ou aplicação que será executada pela grade, deve estar preparado para ser executado em outras máquinas, não deve haver dependências de diretórios com caminhos absolutos no código, o processo não deve necessitar de outros recursos instalados na máquina em que seria executado originalmente, além disso, havendo mais de um processo, estes devem ser independentes e as quantidades de dados migrados devem ser pequenas uma vez que os recursos computacionais utilizados podem estar geograficamente distribuídos e se grandes quantidades de dados forem trocadas, o tempo de transferência pode se tornar considerável em relação ao tempo total de execução. Ainda não existem ferramentas de conversão automática para uso de aplicativos em grades e nem todas as aplicações podem ser processadas de forma paralela. No entanto, novas aplicações podem ser modeladas e desenvolvidas com relativa facilidade para a utilização da infra-estrutura oferecida pelas grades.

Devido a estrutura das grades, os processos ou aplicativos que melhor se adaptam são os que são executados de forma independente, também definidos como *Bag-of-Tasks (BoT)* [6], que apesar da sua simplicidade, são utilizadas em vários cenários como por exemplo a mineração de dados, simulações, cálculo de fractais, computação voltada para biologia e processamento de imagens.

A computação em grades se diferencia da computação distribuída tradicional por seu foco no compartilhamento de recursos em larga escala [11]. Este novo campo da computação cria o conceito de "Organizações Virtuais (*VO*)" [11] que pode ser definido como um grupo de indivíduos ou instituições com regras definidas de compartilhamento de seus recursos, que não precisam estar fisicamente instalados no mesmo local. O conceito de *VOs* possibilita novas formas de negócio, como por exemplo a comercialização o de processamento. Imagine que uma empresa qualquer necessitasse executar um aplicativo com alta demanda de processamento computacional, esta poderia utilizar grades que disponibilizem seus recursos, pagando proporcionalmente ao seu uso. A utilização efetiva do conceito de *VOs* requer interoperabilidade para que seja possível estabelecer relações de compartilhamento de recursos entre todo e qualquer participante, obedecendo obviamente, as políticas de segurança e utilização previamente estabelecidas por cada *site*.

Em relação à sua aplicação, podemos classificar as grades em três tipos básicos [10], sendo eles: grades computacionais, grades centradas em dados e grades interativas. As

grades computacionais são as que têm seu foco no compartilhamento de recursos de processamento, ou seja, de uma forma geral elas oferecem tempo de processamento de suas *CPU's*[6], normalmente utilizadas para computação de alto desempenho. Podem servir a aplicações em geral, mas geralmente são construídas para atender um objetivo específico, como por exemplo as grades dos departamentos da Universidade da Califórnia, abordado na subseção o 2.2.4. Já as grades centradas em dados têm como objetivo oferecer recursos de armazenamento e acesso a dados, funcionando como um sistema de armazenamento massivo porém distribuído. Este tipo de grade por ser composta de um grande nmero de unidades menores em relação à sua capacidade total e oferece não só bastante facilidade de expansão, como também alta confiabilidade, uma vez que os dados podem ser gravados redundantemente em mais de uma das unidades que a compõe. No caso de grades interativas, o principal objetivo é prover serviços de comunicação de alto desempenho e tolerantes a falhas, normalmente utilizadas em aplicações como por exemplo teleconferências, onde porções de dados são dividas entre estas máquinas que compõe a grade garantindo redundância do processamento e caminhos alternativos em seu roteamento.

Evidentemente uma série de desafios surgem junto com esse novo campo da computação, como por exemplo questêms relacionadas a segurança de acesso aos recursos e dados, complexidade na gerência desses recursos em função o da sua distribuição, diferentes interfaces de operação dos sistemas operacionais utilizados devido a sua heterogeneidade, gerência das políticas de segurança e utilização dos diferentes *sites*, e a complexidade agregada a programas que utilizem a estrutura de grades [27].

A utilização das grades computacionais requer uma série de detalhes como o uso de certificado digital, uma espécie de CPF[7] digital para autenticação o e acesso a recursos locais e remotos. As soluções para a utilização dos recursos das grades são implementadas como *middleware*, que é uma interface que permite a interação o de diferentes aplicações geralmente sobre diferentes plataformas de hardware e infra-estrutura para troca de dados, oferecendo serviços de autenticação o e acesso a dados de forma segura, transferência de arquivos, submissão e monitoramento de tarefas, além do monitoramento de recursos, o que exige que o usuário conheça os comandos implementados por esse *middleware* para sua utilização.

Uma ferramenta que auxilie a utilização de grades, automatizando e tornando coman-

[6] *Central Processing Unit*, ou Unidade Central de Processamento em português.

[7] Cadastro de Pessoas Físicas ou CPF é o cadastro da Receita Federal brasileira no qual devem estar todos os contribuintes (pessoas físicas brasileiras ou estrangeiras com negócios no Brasil). O CPF único para cada contribuinte e armazena informações fornecidas pelo próprio e por outros sistemas da Receita Federal.

dos acessíveis por interface gráfica é muito relevante, economizando o tempo do usuário, além de tornar seu acesso mais fácil mesmo para quem não possua conhecimentos específicos. Uma solução o que visa facilitar o uso de grades computacionais é a utilização de portais para seu acesso e operação.

Atualmente existem algumas iniciativas na implementação o de portais para acesso a grades, sendo abordadas mais detalhadamente na subseção 2.2, mas basicamente, além de funcionar como uma interface gráfica, os portais também isolam o usuário de detalhes específicos para a utilização de determinados recursos.

Portais são tipicamente baseados na tecnologia *Web* e é necessário que o usuário tenha somente conhecimentos básicos de acesso a páginas pela internet, servindo como uma interface gráfica, não necessitando nenhum tipo de programação ou de código. Outras facilidades oferecidas tais como personalização o de ambientes, segurança e navegação o [27], fazem do portal uma poderosa ferramenta capaz de permitir a utilização das grades por comunidades científicas sem que seja necessário conhecer todo funcionamento e comandos inerentes à sua operação. Uma outra facilidade que pode ser oferecida pelo portal é o acesso ao estado das aplicações que estão sendo executadas através de dispositivos móveis, tornando seu monitoramento muito mais flexível, além de um sistema automático de informação o capaz de gerar um comunicado, seja através de email ou *paging* por exemplo, referente as tarefas que terminaram e seu respectivo estado, permitindo uma ação o rápida do usuário caso ocorra algum erro. Este tipo de informação o de erros é particularmente útil pois existem processos que levam horas, ou até dias para serem concluídos os e caso o usuário seja logo informado de algum erro, muito tempo pode ser ganho para uma intervenção o corretiva.

A maioria dos portais facilitam as operações básicas como submissão de *jobs*, sendo *job* definido como um processo, ou tarefa, que deve ser executado pela grade, transferência de arquivos entre máquinas remotas, entendendo máquina como um computador que faz parte da grade, verificação o do estado dos *jobs* submetidos e monitoramento das máquinas componentes da grade ou grades as quais o portal tem acesso. No entanto, normalmente são desenvolvidos para uma grade específica ficando dependentes de um *middleware* particular.

Esta dissertação o é o resultado da pesquisa das tecnologias empregadas na implementação o de portais *Web* para acesso a grades computacionais, além da descrição o da experiência da implementação o baseada em *portlets*, abordados na subseção 2.1.5, no desenvolvimento de um portal com funções básicas de autenticação o, monitoramento de

recursos, transferência de arquivos remotos, submissão e monitoramento de *jobs*, e tem como objetivo a contribuição o no esforço de se desenvolver interfaces que facilitem a utilização de grades computacionais. A arquitetura do *MyEasyGridPortal* foi elaborada para permitir o mínimo de interoperabilidade entre diferentes *middlewares* e múltiplas grades computacionais simultaneamente, além de permitir que outras funcionalidades sejam incorporadas independetemente da linguagem de programação o que tenham sido escritas, objetivando ser uma ferramenta bastante flexível para atender usuários com as mais variadas necessidades. Um outro foco abordado pela implementação o deste portal é facilitar a utilização do ambiente *MPI (Message Passing Interface)*, automatizando e monitorando a maioria dos processos básicos e repetitivos, facilitando os trabalhos de pesquisas que utilizam esta têmnica. Este trabalho está assim organizado:

Capítulo 2 descreve as tecnologias utilizadas no desenvolvimento de sistemas *Web* e cita alguns trabalhos relacionados a portais para acesso a grades;

Capítulo 3 explica os conceitos e o funcionamento dos componentes de uma grade computacional e traz considerações quanto ao seu uso, realçando as facilidades e dificuldades, focado na gerência de seus recursos;

Capítulo 4 descreve detalhadamente o contexto em que foi implementado o portal, sua estrutura, define as ações e funções dos elementos que o compõem, apresenta os diagramas de classes, interação o, e sequência dos componentes construídos, assim como as tabelas do banco de dados e as relações entre elas e os elementos utilizados;

Capítulo 5 apresenta conclusãos e trabalhos futuros sugeridos, que podem ser desenvolvidos a partir deste.

Capítulo 2

Tecnologias e trabalhos relacionados

Este capítulo apresenta as tecnologias utilizadas para o desenvolvimento e implementação de portais, servindo como base para o restante desta dissertação. Também serão apresentados alguns portais pesquisados para acesso a grades computacionais.

2.1 Tecnologias *Web*

A *World Wide Web*, também conhecida simplesmente como *Web*, ou ainda *WWW*, que, literalmente em português significa "teia de abrangência mundial", é um sistema dentro da internet formado por servidores *Web* que disponibilizam o acesso a documentos escritos com uma linguagem chamada *HyperText Markup Language* ou *HTML*. A linguagem *HTML* utiliza formato de texto, sendo relativamente de fácil entendimento pelas pessoas e foi criada com o objetivo de permitir integrar informações dos mais variados formatos como sons, imagens e animação em uma mesma página, inclusive acessando recursos externos, ou seja, o conteúdo descrito no documento não precisa estar localizado fisicamente ou logicamente no mesmo servidor. Quando um aplicativo conhecido como navegador, como por exemplo o *Internet Explorer* da *Microsoft* e o *Firefox* da *Mozilla Corporation*, exibe uma página disponível na *Web*, na verdade, a interação com o servidor que forneceu seu conteúdo basea-se em requisições e respostas (*request/response*), ou seja, conforme a ilustração da Figura 2.1, o navegador, ou cliente, após se digitar um endereço *WWW*, requisita ao servidor através do protocolo conhecido como *HTTP* o arquivo padrão para exibição, ou qualquer outro arquivo que faça parte do endereço digitado, o servidor consulta a base de dados e responde com o arquivo solicitado, que é interpretado e exibido pelo navegador.

Uma desvantagem neste processo, é que o arquivo passado para o cliente é estático, ou

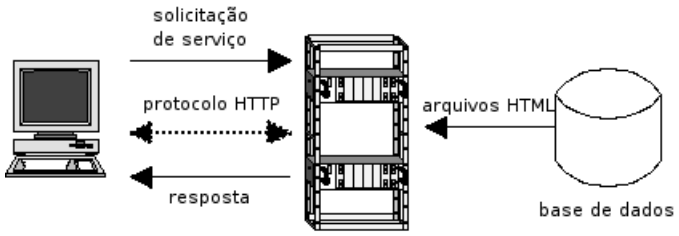


Figura 2.1: Arquitetura *HTTP*

seja, o conteúdo do arquivo é integralmente enviado sem nenhuma alteração. Este tipo de "página", também conhecida como página estática, não funcionaria para aplicações mais dinâmicas que, por exemplo, precisam mostrar o conteúdo de uma tabela de um banco de dados que é atualizado várias vezes. Uma solução para tornar as páginas enviadas pelos servidores mais dinâmicas é a utilização da *Common Gateway Interface*.

2.1.1 *Common Gateway Interface (CGI)*

A *Common Gateway Interface (CGI)* é a especificação de um padrão que permite que uma aplicação executada pelo sistema operacional, também seja utilizada por servidores de informação como os servidores *Web* [26].

Um programa *CGI* é chamado pelo servidor e é executado no momento da solicitação do cliente, passando os dados necessários para a correta exibição do conteúdo desejado. Dessa forma, a integração com programas que executam tarefas específicas, como conexão a banco de dados por exemplo, podem ser utilizados na interação entre usuário e sistema. A Figura 2.2, mostra o cliente solicitando dados ao servidor que executa "comandos *CGI*", que por sua vez consulta uma base de dados, gerando e devolvendo o resultado no formato *HTML* para o servidor que responde ao cliente.

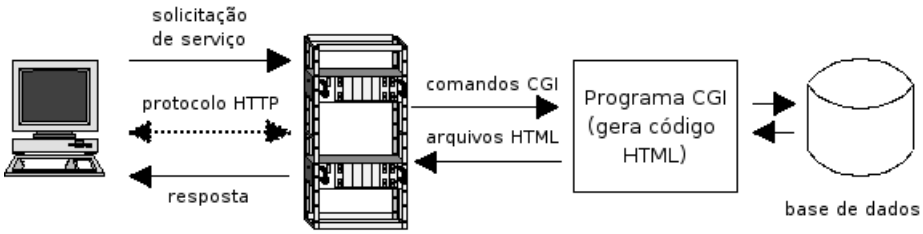


Figura 2.2: Arquitetura *CGI*

No entanto existem desvantagens como o fato do código *HTML* de retorno para o cliente ser escrito junto com o código executável, tornando *CGI's* menos legível e de manutenção mais difícil, como pode ser visto na Figura 2.3 que mostra um pedaço de

código de um programa *CGI*, escrito em C. Repare que os códigos *HTML* são inseridos junto com os comandos "printf", além disso, qualquer modificação necessita uma nova compilação.

```
1. main()
2. {
3.   struct student_struct *s;
4.   VARCHAR username[64];
5.   VARCHAR password[64];
6.   VARCHAR dynamicStmt[128];
7.   VARCHAR localName[64];
8.   float localGrade;
9.   int i;
10.
11.   ...
12.
13.   /* CGI HEADER */
14.   printf("content-type: text/html\r\n\r\n");
15.   /* HTML HEADER */
16.   printf("<HTML> \n<HEAD>\n");
17.   printf("<TITLE>Sample Database Interface</TITLE>\n");
18.   printf("</HEAD>\n");
19.
20.   if (!Initialize()) exit(0);
21.   if (!GetUserName((char*)username.arr)) exit(0);
22.   username.len = strlen(username.arr);
23.   if (!GetPassWord((char*)password.arr)) exit(0);
24.   password.len = strlen(password.arr);
25.
26.   EXEC SQL WHENEVER SQLERROR DO sql_error("ORACLE error--");
27.   EXEC SQL CONNECT :username IDENTIFIED BY :password;
28.
29.   s = (Student *) malloc (sizeof(Student));
30.   if (!s) exit(1);
31.
32.   /* HTML BODY */
33.   printf("<BODY TEXT=\"#FFFFFF\" BGCOLOR=\"#303030\">\n");
34.   printf("<CENTER><H1>Sample Database Interface for %s</H1>\n",
username.arr);
35.   printf("<H2>CS145 Students</H2></CENTER><HR>\n");
36.
37.   ...
38. }
```

Figura 2.3: Fragmento de código *CGI*

Uma outra desvantagem é que cada instância do programa *CGI* executado é visto como um processo diferente e totalmente independente para o sistema operacional, consumindo mais memória no caso de múltiplos acessos simultâneos.

Em uma estrutura de diretórios de um servidor *Web*, os programas *CGI* normalmente ficam no diretório */cgi-bin*. Praticamente qualquer linguagem de programação pode ser utilizada para escrever um programa *CGI* como C/C++, *Perl* e *TCL*, bastando para isso que o sistema permita sua execução.

2.1.2 *Applets*

Os *applets* são programas Java que podem ser embutidos em documentos *HTML*, como páginas *Web*. Enquanto os programas *CGI* são executados no servidor que os hospeda, quando um navegador solicita uma página *Web* que contém um *applet*, o *applet* é baixado

para o navegador e começa a ser executado na máquina cliente que está fazendo o acesso [7]. Uma das vantagens de um aplicativo Java é que ele é independente da plataforma que está sendo executado, uma vez compilado, o programa pode ser executado em qualquer plataforma em que uma *Java Virtual Machine (JVM)*[1] esteja rodando.

Applets são utilizados basicamente quando há necessidade de execução de código do lado do cliente, mas algumas desvantagens que podem ser apontadas são: todo o código tem que ser transferido para a máquina cliente, *applets* muito grandes tornam o acesso lento, além disso, só começam a ser executados após todos os arquivos terminarem de ser transferidos, a execução é relativamente lenta pois precisa ser interpretada localmente pela *JVM*.

2.1.3 *Servlets*

Um *Servlet* é um programa que estende a funcionalidade de um servidor *Web*, gerando conteúdo dinâmico e interagindo com os clientes sendo eles navegadores ou outras aplicações. Os *servlets* não são restritos ao modelo *HTTP* de requisição e resposta, ou seja, eles também podem ser escritos de forma que sejam utilizados por outros aplicativos que precisem de seus serviços, assim, por exemplo, somente um *servlet* pode ficar responsável pelo acesso ao banco de dados e todos os outros aplicativos que necessitem gravar ou recuperar dados desse banco, solicitam este serviço a ele. Esta organização oferece maior flexibilidade e modularidade às implementações.

Os *servlets* correspondem no lado do servidor aos *applets* no lado do cliente e são executados como parte de um servidor *Web*. Os *servlets* tornaram-se tão populares que passaram a ser suportados pelos servidores *Web* mais comumente utilizados, como o *Apache* e o *Microsoft Internet Information Service (IIS)*. Em 1996 a *Sun Microsystems* introduziu *servlets* como aplicações baseadas em Java para adicionar funcionalidade dinâmica a servidores *Web*. Os *servlets* Java têm um modelo de programação similar aos programas *CGI*, na medida em que eles recebem uma solicitação *HTTP* de um navegador *Web* como entrada e espera-se que localizem e/ou construam o conteúdo apropriado para respostas do servidor, no entanto os *servlets* associados com um servidor *Web* rodam dentro de um único processo. Este processo, também chamado de "*container*" roda em uma *JVM*, e ao invés de criar uma nova instância para cada solicitação, o *container* cria um encadeamento de memória para as solicitações dos *servlets*, ou seja, gerencia o contexto dos

[1] *JVM* ou máquina virtual Java, é um programa que carrega e executa os aplicativos Java, convertendo os bytecodes em código executável de máquina.

servlets dentro do seu próprio espaço de memória, sendo responsável por todo o ciclo de vida do *servlet*, desde de sua inicialização, passando a ocupar espaço em memória, durante sua execução, até sua finalização e retirada da memória, garantindo um uso mais eficiente desta se comparado a criar processos diferentes pelo sistema operacional. Isto torna a execução do *servlet* consideravelmente mais eficiente do que o processamento de *CGI's* [8].

Um dos mais populares *containers* utilizados hoje é o *Apache Tomcat*, projeto da *Apache Software Foundation*[2]. *Tomcat*, como é mais conhecido, é o *servlet container* que é usado como implementação oficial de referência da *Sun*. Ele deve ser instalado na máquina que funcionará como servidora *Web* e funciona como o próprio servidor. Por ser um *servlet container*, ele responderá as requisições dos clientes *Web* executando os *servlets* associados a requisição, mas também é capaz de responder a solicitações que retornem documentos puramente *HTML*.

O correto funcionamento de uma aplicação que utilize *servlets* e *Tomcat* deve seguir alguns padrões, como a estrutura de diretórios sob a qual uma aplicação deve ser instalada. A aplicação a ser executada pelo *container* deve ser instalada sob o diretório *webapp*. Maiores detalhes sobre a configuração e desenvolvimento de uma aplicação que utilize *servlets* podem ser encontrados no *site* do projeto *Tomcat*[3].

2.1.4 *Java Server Pages (JSP's)*

Java Server Pages, ou *JSP's* é também uma tecnologia, criada pela *Sun*, orientada a criar páginas *Web* dinâmicas com Java, e que permite que instruções *HTML* estáticas sejam misturadas com instruções *HTML* geradas dinamicamente por *servlets* [23]. A *JSP* não oferece nada que não possa ser alcançado utilizando *servlets* puramente. Na verdade, documentos criados com *JSP's* são traduzidos em *servlets* durante sua execução. A principal, mas sutil diferença entre elas, é a maior conveniência em utilizar instruções *HTML* diretamente no documento do que escrever uma série de comandos "*println*" no código de um *servlet*, como pode ser visto pela Figura 2.4. O *JSP* permite misturar código Java com instruções *HTML*, diferenciando um do outro pelos símbolos "<%" e "%>" que envolvem os comandos Java como pode ser visto na linha 1, já na linha 12 temos uma instrução *HTML* estática.

Uma outra diferença é que o fato de poder "embutir" os conteúdos dinâmicos em

[2]http://www.apache.org
[3]http://tomcat.apache.org/

```
1. <%@ page import="java.util.Date" %>
2.
3.
4. <%!
5. private int accessCount = 0;
6. private Date accessDate = new Date();
7. private String accessHost = "<I>Sem acesso anterior</I>";
8. %>
9. <P>
10. <HR>
11.
12. <A HREF="http//www.my-company.com/">my-company.com</A>.
13. Esta página foi acessada <%= ++accessCount %>
14. vezes desde o último reboot do server. Foi acessada a última vez
de
15. <%= accessHost %> at <%= accessDate %>.
16. <% accessHost = request.getRemoteHost(); %>
17. <% accessDate = new Date(); %>
```

Figura 2.4: Fragmento de código *JSP*

páginas *HTML* comuns, permite que *Web Designers* criem as páginas com editores *HTML* e depois passem para os programadores inserir o conteúdo dinâmico. As *JSP's* também necessitam de um *container* para serem executadas, o que é feito pelo mesmo *container* dos *servlets*, o *Tomcat*.

2.1.5 *Portlets*

Portlets são componentes *Web*, como *servlets*, especificamente projetados para serem agregados em um contexto de uma página *Web* composta. Usualmente vários *portlets* são invocados em uma simples chamada a uma página de um portal. Cada *portlet* produz um fragmento de página que é combinado com outros fragmentos de outros *portlets*, todos juntos para formar a página do portal [15]. A especificação JSR168, criada pela *Java Community Process*[4], define como desenvolver um *portlet* para interagir com um portal *Web*. Em sua versão 1.0 [13], a JSR168 apresenta os conceitos básicos de programação de *portlets* que foram observados neste trabalho.

Para explicar *portlets* é necessário explicar o conceito de portais *Web*.

2.1.5.1 Portais *Web*

Uma página típica de um portal *Web* baseado em *portlets* pode ser vista na Figura 2.5.

A página mostrada pela Figura 2.5 é composta de por diferentes "janelas", cada uma delas possuindo sua própria barra de título e botões para maximizar e minimizar. Cada janela exibida é uma aplicação diferente e totalmente independente da outra, permitindo

[4] *Java Community Process* é um processo formalizado que permite que as partes interessadas se envolvam nas definições de versões futuras e funcionalidades da plataforma Java.

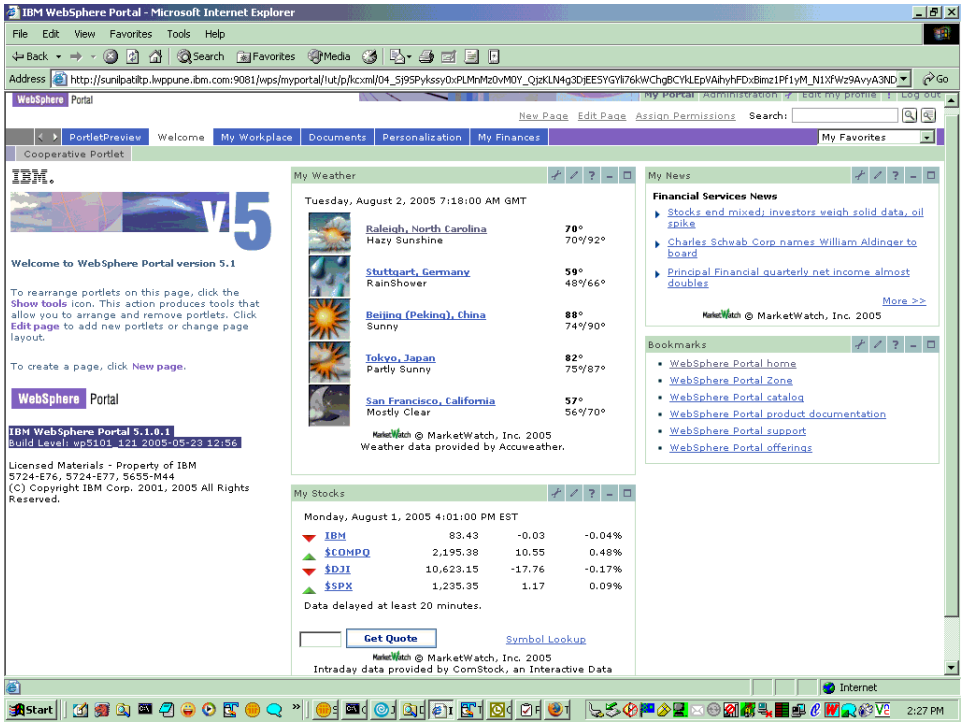


Figura 2.5: *Web portal* baseado no IBM *WebSphere*

que diferentes aplicativos desenvolvidos para interface *Web* sejam executados e monitorados visualmente ao mesmo tempo.

O desenvolvedor do *portlet* deve criar sua aplicação e "empacotá-la" em um arquivo conhecido como *Web Archive*, ou *.war*, que é um formato padrão de arquivos compactados que obedecem a estrutura de diretórios de uma aplicação *Web*, e enviar este arquivo para o administrador do portal que, por sua vez, o instala e disponibiliza para os demais usuários. Em [15], temos a definição de um portal como uma aplicação *Web* que permite personalização, autenticação única também conhecida como *"single sign on"*, agregação de conteúdos de diversas fontes e que hospeda a camada de apresentação do sistema de informação. As funcionalidades de um portal podem ser divididas em 3 partes principais:

Portlet container que é muito similar ao *servlet container*, onde cada *portlet* é instalado e que controla todo o seu ciclo de vida, ou seja, desde de sua inicialização quando é carregado na memória, sua execução e sua finalização com retirada da memória;

Content aggregator ou agregador de conteúdo, que, segundo a especificação de *portlets* é a principal função de um portal, agregar informações de várias fontes ao mesmo tempo. Estas informações, ou conteúdos, podem ser os mais variados dados a serem exibidos através de uma página *WWW*;

Common services ou "serviços comuns", que é o conjunto de serviços que um portal

oferece aos *portlets*, e é uma das principais vantagens associadas, como exemplo podemos citar a capacidade de personalização individual para cada usuário, desta forma, detalhes como a cor das bordas das páginas exibidas não são preocupações de quem escreve o *portlet*, e podem ser configuradas individualmente pelo próprio usuário. Uma outro "serviço comum" oferecido pelo portal é a possibilidade do usuário poder configurar quais *portlets*, dentro dos que estão disponíveis para ele, serão utilizados.

Similar aos *servlets*, *portlets* são componentes *Web* que são instalados dentro de seu *container* e que geram conteúdo dinâmico. Pelo lado técnico, *portlet* é a classe que implementa a interface *javax.portlet.Portlet*. Dentre as semelhanças entre *portlets* e *servlets*, podemos citar o fato dos dois serem executados por *containers* específicos, de gerarem conteúdo dinâmico, de terem seu ciclo de vida gerenciado por seu *container* e por interagirem pelo modelo *request/response*.

Como principais diferenças entre os dois, temos que os *portlets* geram somente fragmentos e não o documento inteiro, não são endereçáveis diretamente por *URL*, e não podem gerar fragmentos de conteúdo aleatórios, ou seja, o conjunto de *portlets* que é apresentado pela página do portal deve gerar conteúdos do mesmo tipo, *text/html* ou *WML* por exemplo [20].

Portlets oferecem algumas funcionalidades adicionais como:

Armazenamento persistente de preferências onde as preferências ajustadas de cada usuário são automaticamente gravadas e recuperadas quando há um novo acesso ao portal pelo mesmo usuário;

Processo de requisição muito mais refinado se comparado a um *servlet*. Um *portlet* pode receber uma requisição devido a uma ação específica do usuário sobre ele, por exemplo, um botão que tenha sido inserido com código *HTML* <*ui:actionsubmit action="doActionLink" key="DOIT"/*>, ao ser acionado, gera a excução direta da classe *doActionLink*, sem que seja necessário nenhum outro código para direcionar a execução para essa classe correspondente;

Modos de operação pelo qual é possível saber em que estado o *portlet* se encontra, se em modo de visualização (*VIEW*), edição (*EDIT*) ou ajuda (*HELP*). Assim, é possível escrever códigos específicos para serem executados em determinadas condições de forma mais simples com as classes *doView()*, *doEdit()* e *doHelp()* respectivamente, que são chamadas nas mudanças de estado dos *portlets*.

2.2 Trabalhos relacionados

Os trabalhos aqui relacionados são os que considerei mais relevantes no que se refere a objetividade e descrição de implementação, ou utilização de grades computacionais através de portais *Web*.

2.2.1 EGEE e GENIUS

O *Enabling Grids for E-sciencE* é um projeto fundado pela Comunidade Européia com o objetivo de ajudar no desenvolvimento e pesquisa de tecnologias relacionadas a grades.

Atualmente, segundo o site oficial do projeto (http://www.eu-egee.org), participam mais de 90 entidades em 32 países, organizados em 13 federações, possui uma infra-estrutura de mais de 20.000 *CPUs* e cerca de 5 Petabytes de armazenamento, executando uma média de 20.000 *jobs* por dia. Mais de 20 aplicações em 10 campos diferentes da ciência utilizam a infra-estrutura da grade EGEE. Algumas entidades que a utilizam são: CERN (*Centre Europeen de Recherche Nucleaire*) na Suíça e o Fermilab (*Fermi National Accelerator Laboratory*) nos Estados Unidos.

O *middleware* utilizado pelo EGEE é baseado no projeto denominado *LHC Computing Grid (LGC)* do CERN, que foi desenvolvido com a missão de suportar o armazenamento e processamento demandados pelo acelerador de partículas conhecido como LHC, localizado nesta mesma instituição na Suíça.

Inicialmente, o projeto foi implementado para duas aplicações de áreas bem conhecidas, a física de altas energias e biomedicina, pelo fato dessas duas áreas já utilizarem computação paralela e distribuída, sendo mais fácil de adaptá-las para o processamento em grades por este fato, servindo como piloto para o desenvolvimento do projeto. Atualmente uma série de outras áreas são atendidas como a química computacional e astrofísica.

Um exemplo de portal que utiliza a estrutura oferecida pelo EGEE é o GENIUS *Grid Portal* (*Grid Enabled web eNvironment for site Independent User job Submission*). O GENIUS, cuja página inicial pode ser vista pela Figura 2.6, permite que cientistas acessem, executem e monitorem suas próprias aplicações que utilizem os recursos oferecidos pela grade EGEE, a partir de um navegador [5].

Pela Figura 2.7 podemos observar a arquitetura do GENIUS *Grid Portal* é baseada no modelo *MVC*[5], ou "*3-tier model*". Nela podemos observar o cliente representado pelo *WEB*

[5] *Model-View-Controller* (*MVC*) é uma arquitetura de software que separa uma aplicação em três

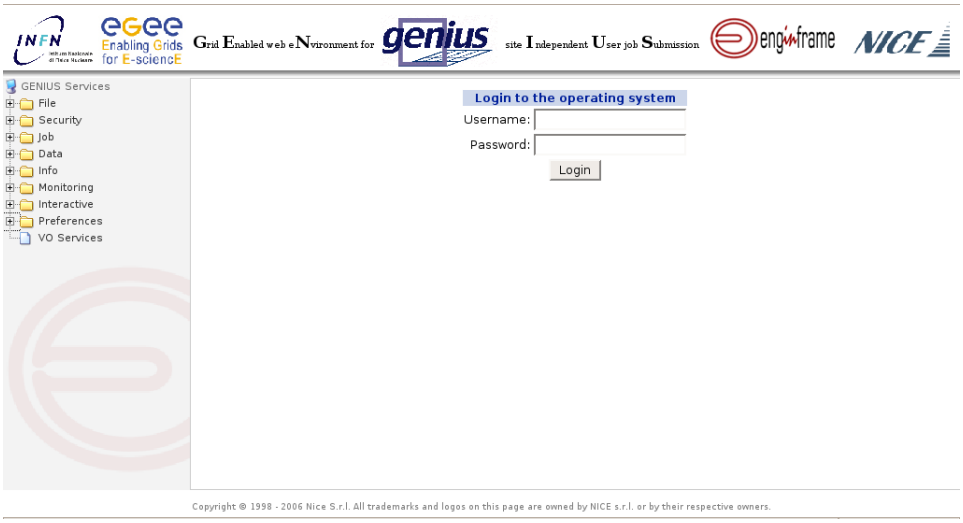


Figura 2.6: Página inicial do GENIUS *Grid Portal*

Browser, ou navegador, na parte superior direita da figura, temos o servidor representado no lado direito, e os recursos remotos, representados na parte inferior direita. Para o cliente utlizar os recursos remotos, deve fazê-lo através do servidor.

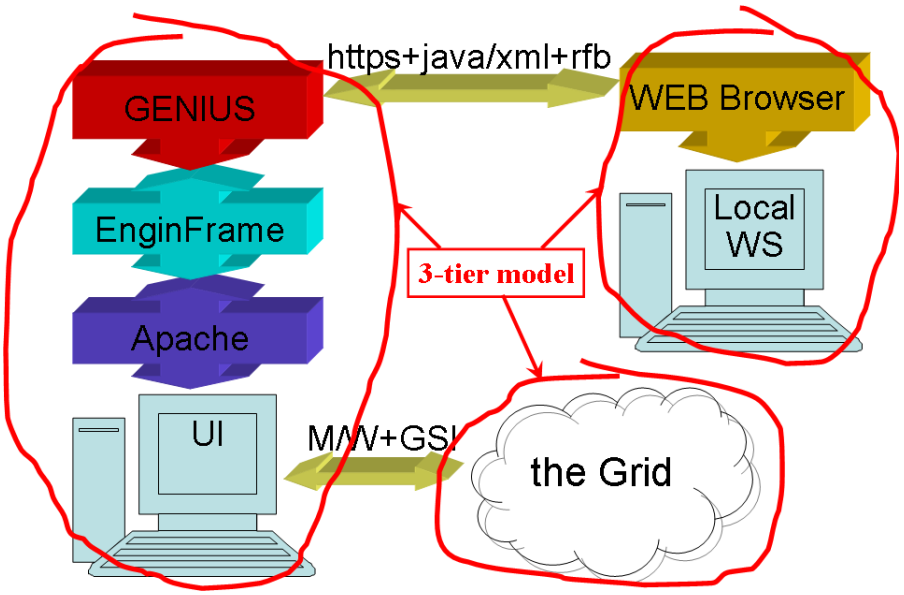


Figura 2.7: Arquitetura do GENIUS *Grid Portal*

O elemento nomeado de *EginFrame*, referente ao servidor representado pela Figura 2.7, é quem faz a interface entre o portal e o *middleware* utilizado para acesso aos recursos. Os serviços básicos oferecidos pelo GENIUS *Grid Portal* são:

Serviços de segurança, que são a garantia de acesso seguro, como conexão *HTTPS*,

componentes distintos, sendo eles: modelo de dados da aplicação, interface com o usuário e controle lógico. Desta forma, uma modificação no aplicativo pode ser feita com impactos mínimos no modelo de dados.

conta no portal para acesso restrito e autenticação por exemplo;

Serviços de submissão de *jobs*, que informa os recursos disponíveis, cria, submete e monitora *jobs*;

Serviços de gerenciamento de dados, que gerencia a criação, duplicação e remoção de dados;

Serviços para dispositivos móveis, que costumiza as informações passadas a esses tipos de dispositivos devido a suas limitações;

2.2.2 GridSphere

O *GridSphere* é um *framework open-source* para construção de portais *Web*, ou seja, ele é uma estrutura desenvolvida para facilitar e suportar a construção de portais, fornecendo bibliotecas, scripts e outros componentes de *software*. Do ponto de vista de aplicativo, ele é um *servlet*, implementado como uma aplicação *Web* e funciona como *container* para os *portlets*, como pode ser visto pela Figura 2.8, onde vemos o navegador, representado pelo primeiro retângulo, acessando o servidor *Web*. A página exibida pelo navegador é composta de vários *portlets*, que são exibidos de acordo com a configuração utilizada pelo usuário. A acesso é redirecionado para o *GridSphere Servlet*, que interage com o *portlet*. O *portlet*, por sua vez, interage com os componentes básicos como o *Portal Layout Engine*, responsável pela exibição do *layout* da página que é enviada para o cliente, o *GridSphere Portlet API*[6], que é a *API* utilizada pelos *portlets*, e os serviços principais oferecidos pelo *GridSphere*, através do componente denominado *Core Services*. Os recursos da grade são acessados pelo próprio *portlet*, que faz uso da *API* para isso.

Na Figura 2.9 podemos observar o cliente, ou navegador, ainda representado pelo primeiro retângulo da figura, acessando o *GridSphere* portal, representado pelo segundo, que por sua vez é implementado pelo *GridSphere Servlet* que funciona como o *container* dos *portlets*, sendo responsável por todo seu ciclo de vida, ou seja, desde de a inicialização, quando o *portlet* é carregado e passa a ocupar a memória, até finalização, passando pela sua renderização na tela do navegador e processamento das ações direcionadas a ele. O retângulo nomeado de *Portlets*, que possui os processos "*inicialize*", "*render*", "*processAction*" e "*destroy*", responsáveis pela inicialização, "*renderização*", processamento

[6] *Application Programming Interface* ou simplesmente *API* é um conjunto de rotinas e padrões estabelecidos por um software para utilização de suas funcionalidades. De modo geral, a *API* é composta por uma série de funções acessíveis somente por programação, e que permitem utilizar características do software menos evidentes ao usuário tradicional.

das ações direcionadas ao *portlet* e destruição do mesmo, é exatamente o mesmo retângulo "*Portlets*" da Figura 2.8.

Algumas características oferecidas pelo *GridSphere* são:

- Compatível com a especificação JSR168;
- Desenvolvimento de *portlet* usando o padrão *Java Server Faces (JSF)*[7];
- Fácil integração de novos *portlets* através de *scripts* já definidos.
- *API* compatível com *IBM's WebSphere® 4.2*

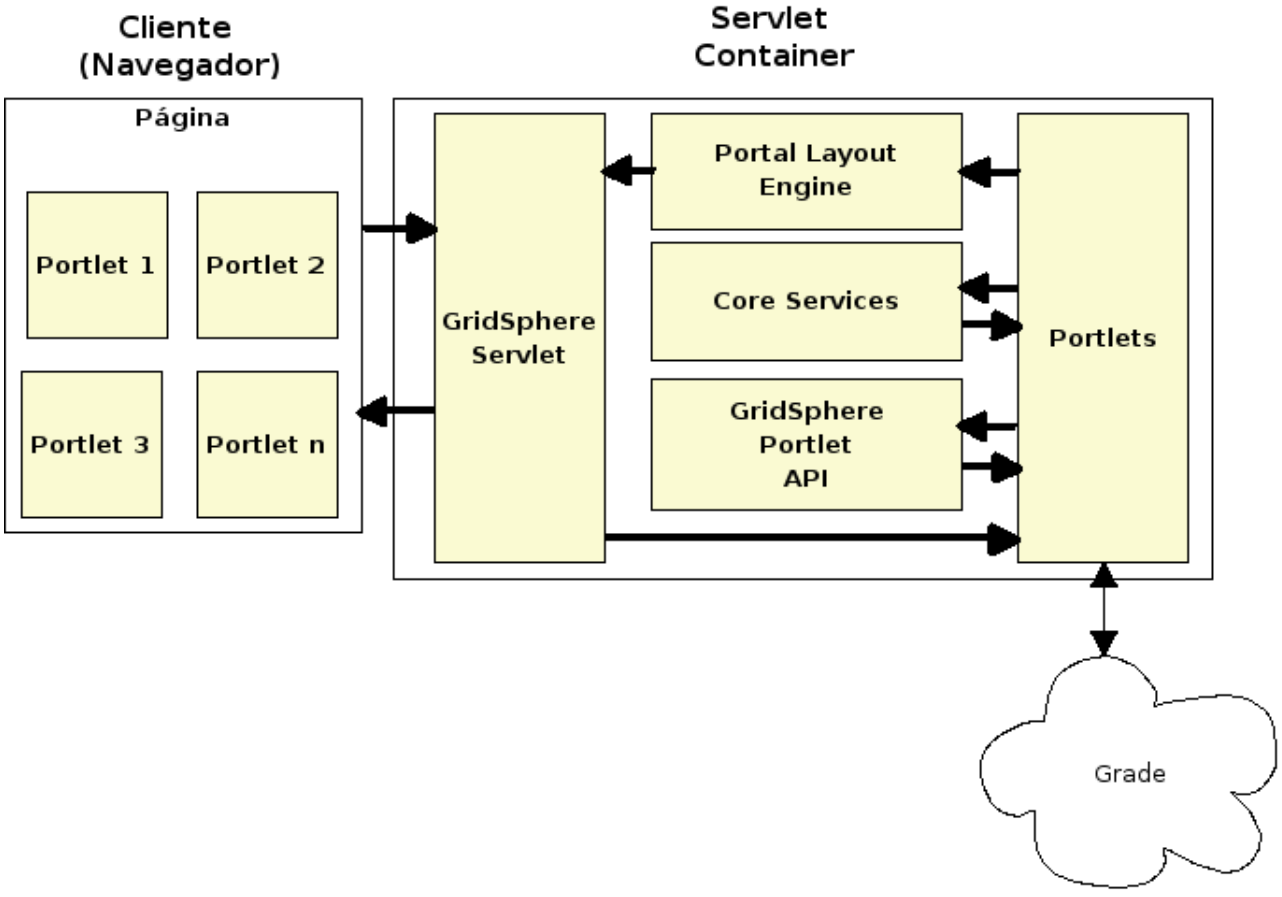


Figura 2.8: *GridSphere* como um aplicativo *Web*

2.2.3 OGCE

O OGCE é um consórcio que desenvolve produtos *open source* para a construção de portais que acessam grades, além de serviços *Web* e *portlets* compatíveis com a especificação

[7] *JavaServer Faces* ou *JSF* é uma especificação que facilita o desenvolvimento da interface com o usuário em aplicações Java usando *Java Server Pages*. Ver 2.1.4.

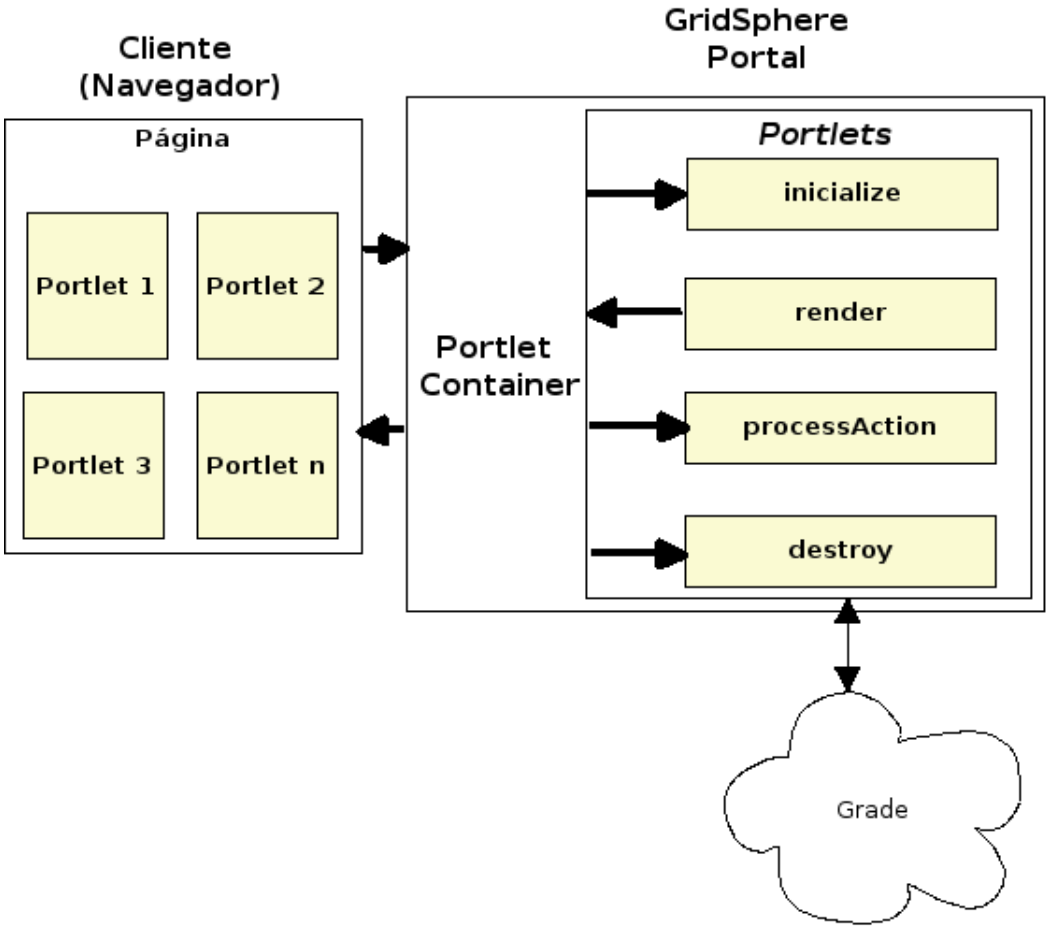


Figura 2.9: Ciclo de vida de um *portlet*

JSR168. O portal OGCE, que é baseado no *GridSphere*, combina *portlets* desenvolvidos pelo *GridPort Team* da universidade do Texas (TACC) e pelo *Extrem Lab* da *Indiana University* dentre outros[8].

O relatório anual apresentado pelo consórcio em 1 de junho de 2006 [1], apresenta alguns projetos e entidades que utilizam e participam de seu desenvolvimento, estando entre eles os seguintes projetos:

- Common Instrument Middleware Architecture, desenvolvido pela Indiana University, permite que experimentos em cristalografia sejam acompanhados e monitorados remotamente;
- TeraGrid User Portal, desenvolvido pelo TACC (*Texas Advanced Computing Center*) da *University of Texas*, que habilita o acesso ao TeraGrid oferecendo serviços de gerenciamento de credencias e monitoramento de servidores;

[8]A lista completa dos colaboradores pode ser encontrada em http://www.collab-ogce.org/ogce2/.

- *Renaissance Computing Institute* (RENCI) BIOPORTAL, desenvolvido pelo próprio instituto, que utiliza o TetraGrid em aplicações na área biomédica e bioinformática;
- LEAD Portal, desenvolvido pela *Indiana University*, utilizado para aplicações de previsão atmosférica.

2.2.4 *UCLA*

O *UCLA Grid Portal* é um desenvolvimento da *UCLA Academic Technology Services* (http://www.ats.ucla.edu/) e se propõe a prover uma interface *Web* única aos *clusters* que formam a grade *UCLA*. O portal também utiliza o *framework GridSphere* e acessa diretamente alguns *clusters* fora de sua grade, como o Hoffman Cluster (http://www.ats.ucla.edu /rct/beowulf/), o Dawson Cluster (http://exodus.physics.ucla.edu/) e Miles Cluster (http://airto.bmap.ucla.edu/) entre outros [9]. A grade possui 9 *clusters* [10], sendo eles os dos departamentos de física, astronomia, química, biologia, ciências atmosféricas, ciências sociais, psicologia, neuro imagens, nano ciências, engenharia elétrica, química e mecânica e ciências dos materiais. O poder computacional das 575 máquinas que formam os 9 clusters é de 7.7 TFlops[11]. O diagrama funcional pode ser visto na Figura 2.10, que mostra um usuário acessando o servidor *Web* que roda o portal através de uma conexão *HTTPS*. O portal faz a autenticação do usuário, redirecionando-o para a grade correspondente através da máquina que mapeia sua credencial para uma conta local, a partir daí, o usuário tem acesso ao conjunto de máquinas que formam o cluster da sua grade.

2.2.5 GPDK

O *Grid Portal Development Kit* é um desenvolvimento do *DOE Science Grid research and development*, fundado pelo Departamento de Energia dos Estados Unidos e fornece um ambiente para o desenvolvimento de novos portais, ou seja, fornece os códigos básicos para esses novos desenvolvimentos, assim como um conjunto de códigos que oferecem serviços básicos de utilização de grades como submissão de tarefas e transferência de arquivos. O *GPDK* tem sua arquitetura baseada no modelo padrão *MVC* onde um navegador cliente se comunica com o servidor *Web* de forma segura através de uma conexão *HTTPS*[12],

[9] A lista completa pode ser encontrada em https://grid.ucla.edu:9443/gridsphere.
[10] Números anunciados na *2006 Worldwide Education and Research Conference*.
[11] 1 TFlop = um trilhão de operações com ponto flutuante por segundo.
[12] *HiperText Transfer Protocol Secure* é um protocolo que faz a transferência de dados segura com criptografia entre o software cliente, e o servidor *Web*.

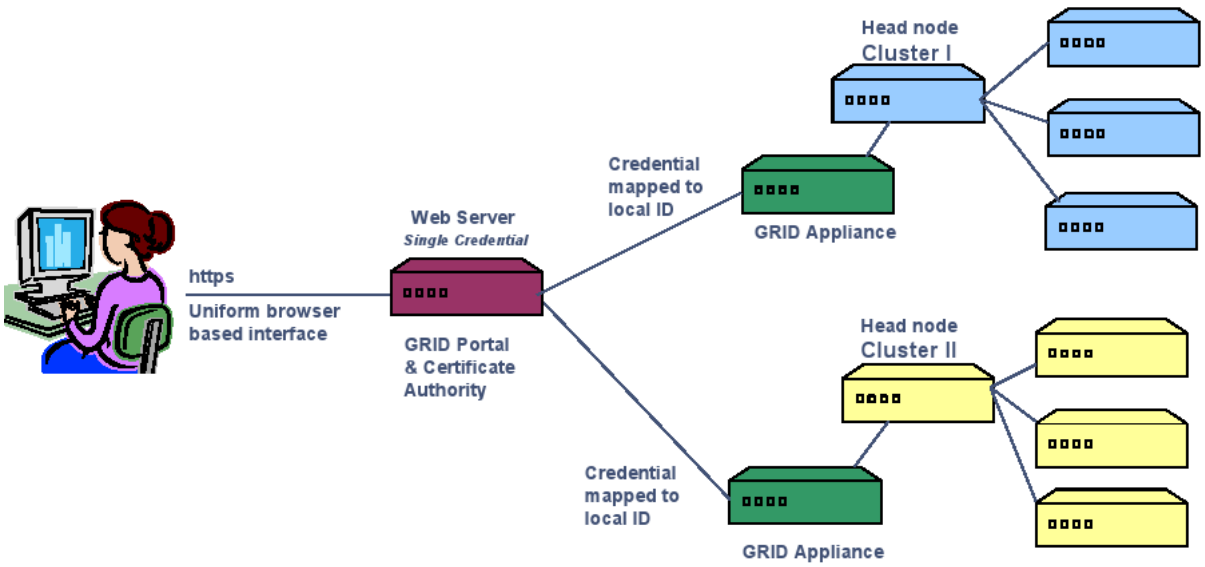


Figura 2.10: Diagrama funcional *UCLA*

o servidor *Web* por sua vez, acessa serviços de grades computacionais através da infra estrutura do *Globus Toolkit* [9]. A Figura 2.11 mostra a arquitetura.

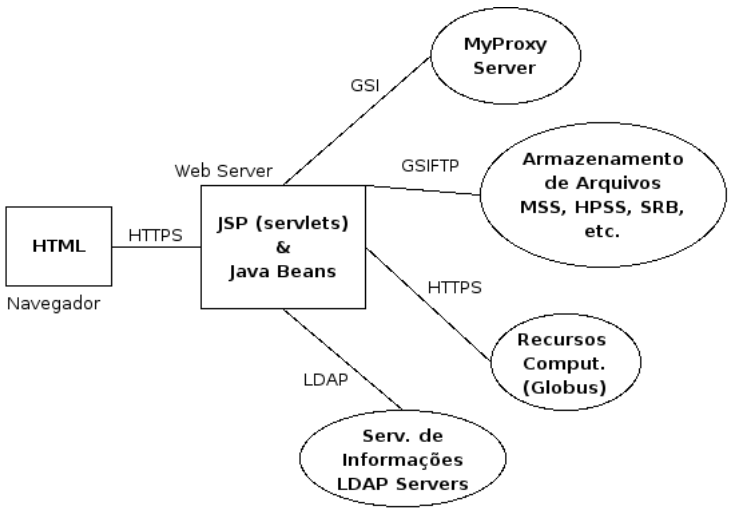


Figura 2.11: Arquitetura *GPDK*

Um *MyProxy Server*, abordado na subseção 3.2, também é utilizado para que o usuário possa acessar a grade sem a necessidade de portar fisicamente sua chave privada, evitando que seja necessário utilizá-la no computador que está sendo usado para o acesso ao portal. O GPDK utiliza *Java Server Pages*[13] e *Java Beans* com *Tomcat*, e atualmente, na página do projeto, existe uma mensagem de que não é mais suportado por seus desenvolvedores.

[13] abordado na subseção 2.1.4

2.2.6 Resumo do capítulo

Várias iniciativas para facilitar o uso das grades computacionais têm sido desenvolvidas por algumas entidades. Essas iniciativas se baseam, na maioria das vezes, em portais *Web* e tem como objetivo permitir o acesso por pessoas que necessitem das vantagens e poder de processamento oferecido pelas grades computacionais, mas que não possuem nenhum conhecimento mais especializado sobre computação. Em geral, as soluções propostas limitam-se a uma grade especifíca, ou a um grupo delas. A padronização *JSR168* é um grande passo na tentativa de criar um padrão que seja compatível com aplicações de portais *Web* mas as tecnologias de *middleware* para computação em grades ainda não convergiram para um padrão.

Capítulo 3

Tecnologias de grades computacionais

A utilização de uma grade computacional, requer, na prática, que uma série de requisitos sejam atendidos para que seus recursos possam ser utilizados de forma adequada. Este capítulo tem por objetivo relatar um exemplo dos procedimentos necessários a essa operação, baseado no *Globus Toolkit* pelo fato deste ser utilizado na maioria das grades computacionais estudadas, além de tecer considerações sobre as facilidades e dificuldades inerentes, sugerindo formas de facilitar sua operação através do uso de um portal.

Em linhas gerais, os passos a serem seguidos para a utilização de uma grade iniciam com a obtenção de um *proxy*, que, por sua vez, é conseguido a partir de um certificado digital que pode ser obtido através de uma Autoridade Certificadora (CA), conforme abordado na subsessão 3.1, ou, através da instalação de um recurso chamado "*simple CA*" disponibilizado pelo *Globus*, que funciona como uma CA, assinando os certificados para fins exclusivamente de testes. Depois, o processo, ou tarefa a ser executada, junto com seus parâmetros e dados de entrada, devem ser transferidos para as máquinas que irão executá-las. O próximo passo então é iniciar a execução da tarefa utilizando o comando apropriado. Uma vez em execução, o progresso deve ser monitorado. Detalhes da utilização e gerência das grades serão abordados nas seções seguintes.

3.1 *X.509 Public Key Infrastructure (PKI)*

Devido a necessidade da segurança das informações digitais, e de restringir acesso aos múltiplos recursos oferecidos pelas grades, deve existir uma forma uniforme de se garantir que o usuário é autorizado a acessar os recursos e que os dados gerados por estes recursos são confiáveis. O método utilizado atualmente é análogo ao que é feito para se garantir a autenticidade de uma assinatura pessoal, onde para ser reconhecida, deve existir uma

confirmação por uma terceira parte de que a assinatura é realmente de quem deveria ser, o que normalmente é feito por um cartório oficialmente reconhecido pelo governo local e acreditado pelas partes que estão envolvidas pelo documento assinado, a chamada *X.509 Public Key Infrastructure*, ou "infra-estrutura de chave pública", nome esse resultado da utilização dos conceitos de criptografia de chave pública elaborados por Diffie e Hellman em 1976 e por Rivest, Shamir e Adleman (RSA) em 1977 [18], foi criada para atender este objetivo baseado em quatro níveis principais de segurança, sendo eles: *confidencialidade* ou *sigilo* que é a garantia de que somente as pessoas ou organizações envolvidas na comunicação possam ler e utilizar as informações transmitidas de forma eletrônica pela rede, *integridade*, que é a garantia de que o conteúdo de uma mensagem ou resultado de uma consulta não será alterado durante seu tráfego, *autenticação*, que é a garantia de identificação das pessoas ou organizações envolvidas na comunicação, e "*não repúdio*", que é a garantia de que o emissor de uma mensagem ou a pessoa que executou determinada transação de forma eletrônica não poderá, posteriormente negar sua autoria.

A infra-estrutura de chave pública baseia-se em criptografia. Três técnicas básicas de criptografia são descritas abaixo, sendo que as duas últimas são utilizadas pela infra-estrutura de chave pública:

- **Criptografia simétrica**

  Nesta técnica, o mesmo código, ou chave como será chamado daqui pra frente, é utilizada para criptografar e descriptografar uma mensagem que, portanto, deve ser de conhecimento tanto do emissor como do receptor da mesma. O algoritmo simétrico bastante utilizado é o *DEA (Data Encryption Algorithm)*, mais conhecido como *DES (Data Encryption Standard)*, cuja chave possui tamanho de 56 bits. Entretanto algoritmos com chaves de até 128 bits já também estão disponíveis, resultando em maior segurança. Uma desvantagem deste tipo de criptografia é o fato da chave ter de ser compartilhada entre transmissor e receptor da mensagem, o que diminui a segurança.

- **Criptografia assimétrica**

  Nesta técnica cada usuário possui um par de chaves, mantendo uma em segredo (chave privada) e tornando a outra pública (chave pública). O tamanho destas chaves varia de 512 a 2048 bits e um dos algoritmos mais utilizado é o RSA . Duas propriedades destas chaves valem ser destacadas, o fato de uma mensagem criptografada por uma das chaves somente poder ser descriptografada pela outra chave

correspondente do par e o fato do conhecimento da chave que é pública, não permitir a descoberta da chave privada. Esta técnica de criptografia garante privacidade e autenticidade a uma mensagem, por exemplo, caso um usuário denominado genericamente de "A" deseje que somente um outro usuário "B" leia sua mensagem, o primeiro deve criptografar a mensagem a ser transmitida com a chave pública de B, assim, somente B pode descriptografá-la pois somente sua chave privada é capaz disso, desta forma A garante privacidade. Caso o usuário A deseje garantir a autenticidade de sua mensagem, este deve criptografá-la com sua chave privada, assim, quem recebê-la só poderá descriptografá-la utilizando a chave pública da A, o que garante que a mensagem partiu realmente dele. A mistura das duas técnicas garante privacidade e autenticidade, ou seja, caso A deseje que somente B seja capaz de descriptografar sua mensagem e que tenha certeza de sua origem, o usuário A deve criptografar a mensagem original com sua chave privada, dando autenticidade, uma vez que só sua chave pública será capaz de descriptografá-la e depois criptografar o resultado com a chave pública de B, garantindo que somente B será capaz de descriptografá-la com sua chave privada, assim, quando B recebê-la, inicialmente deve utilizar sua própria chave privada para fazer a primeira descriptografia, e depois utilizar a chave pública de A, conforme pode ser visto pela Figura 3.1.

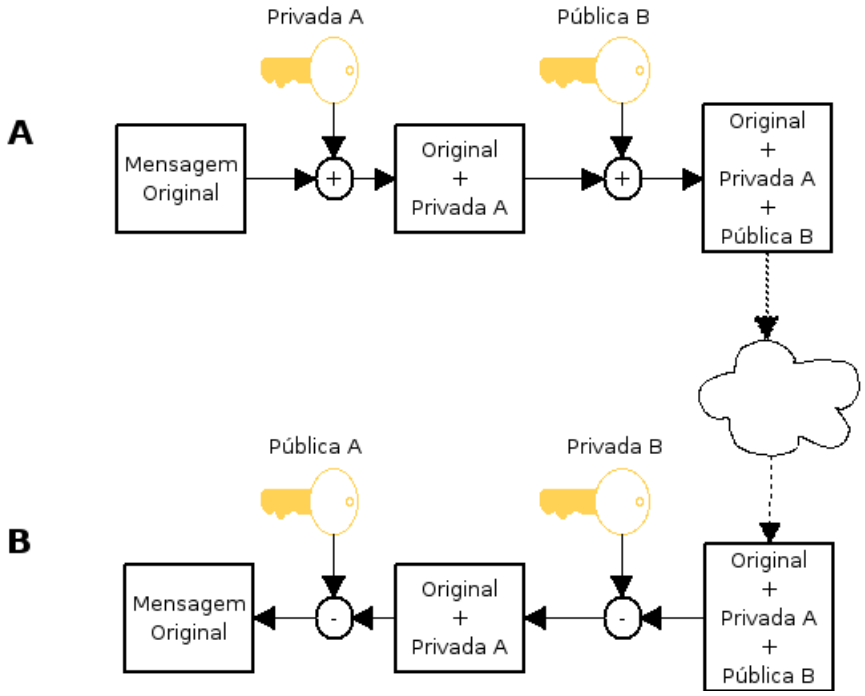


Figura 3.1: Autenticidade e privacidade

- **Função *HASH* ou Resumo de Mensagem**

Esta técnica permite que, ao ser aplicada à uma mensagem de qualquer tamanho, seja gerado um resumo criptografado de tamanho fixo e bastante pequeno, como por exemplo 128 bits. Este resumo também é conhecido como *message digest*. Algumas das propriedades desta função são: não é possível fazer a operação reversa, ou seja, dado um resumo é impossível obter a mensagem original e duas mensagens diferentes, quaisquer que sejam, não podem produzir um mesmo resumo. É utilizada, junto com a criptografia assimétrica, para gerar a Assinatura Digital, que é o resumo da mensagem conseguido através da função *hash*, criptografado com a chave privada do emissor. A vantagem em se utilizar a Assinatura Digital é que a autenticidade e privacidade, conforme descrito no item anterior, pode ser conseguida sem a necessidade de criptografar toda a mensagem com a chave privada do emissor, o que torna o processo mais rápido. Neste caso, conforme a Figura 3.2, o emissor aplica a função *hash* na mensagem original conseguindo o resumo desta, criptografa o resumo com sua chave privada, criptografa a mensagem original com a chave pública do receptor e envia as duas juntas, o receptor descriptografa a mensagem com sua chave privada, aplica o mesmo algoritmo de função *hash* obtendo o resumo, descriptografa a Assinatura Digital com a chave pública do emissor, e compara o resultado com o resumo obtido.

Com as técnicas de criptografia citadas é possível garantir autenticidade e privacidade entre emissor e receptor como foi visto, o que se aplica também à troca de dados entre os navegadores e os servidores *Web*. No caso de acessar um "*site* seguro", o navegador reconhece que a comunicação deve ser criptografada e gera uma chave de criptografia simétrica que será utilizada somente durante aquela sessão, criptografa esta chave com a chave pública do *site* e a envia para o servidor *Web* que está sendo acessado, que a recupera descriptografando-a com sua chave privada, utilizando-a então enquanto durar a sessão. A criptografia simétrica é utilizada durante a sessão devido a sua simplicidade e custo de processamento em relação a assimétrica.

Embora a privacidade da sessão seja garantida, ainda existe a possibilidade do *site*, ou recurso, ser acessado por alguém que não é realmente quem ele diz que é. Qualquer um poderia divulgar uma chave pública em nome de outra pessoa ou entidade, ou seja, assim como no modo tradicional existe uma terceira entidade que garante a autenticidade das assinaturas nos documentos, existe também uma entidade considerada confiável, denominada Autoridade Certificadora, que é responsável por dar "validade" a uma chave pública. A Autoridade Certificadora assina digitalmente a chave pública do usuário emitindo então o Certificado Digital.

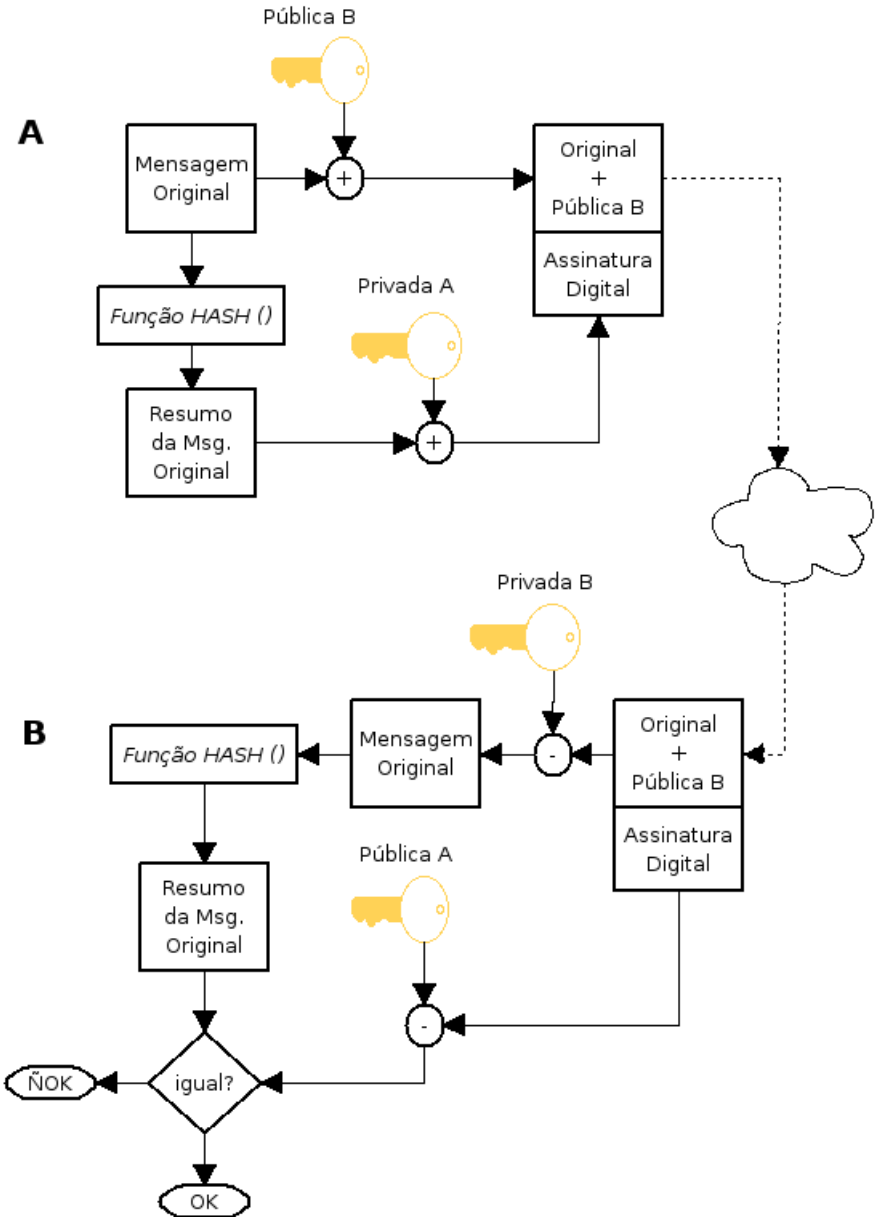


Figura 3.2: Autenticidade e privacidade com a função *hash*

Basicamente um Certificado Digital possui os seguintes campos [16]:

- **Número de versão ou *version***

  É o campo que possui a versão, utilizado para informar o formato do certificado em função da evolução dos padrões. Atualmente está na versão 3.

- **Número de série ou *serial number***

  É o número de série do certificado. Este número deve ser único para cada certificado emitido pela mesma Autoridade Certificadora. É pelo número de série que os certificados revogados são controlados.

- **Assinatura ou *signature***

  É a assinatura digital do certificado, ou seja, é o valor da função *hash* do próprio certificado, criptografado com a chave privada da Autoridade Certificadora que o emitiu. Um dos algoritmos mais utilizados atualmente para a função *hash* é o MD5. É por esse campo que o receptor verifica a autenticidade e integridade do certificado.

- **Nome ou *subject***

  Contém o nome do proprietário do certificado no formato *DN (Distinguished Name)*, que é uma forma de apresentar o nome com alguns campos conforme a Tabela 3.1. O nome *DN* é único para cada usuário.

  O nome "/O=Grid /OU=GlobusTest /OU=simpleCA-mxsec.ic.uff.br /OU=ic.uff.br /CN=Usuario 01" é um nome no formato *DN*.

Tabela 3.1: Tabela de campos *DN*

Campo	Abreviação	Descrição
Nome Comum	CN	Nome sendo certificado
email	E	email
Organização	O	Nome da organização
Unidade Organizacional	OU	Unidade da organização
Localidade	L	Cidade
Estado ou território	ST	Estado ou território
País	C	Sigla do país (padrão ISO)

- **Nome do emissor ou *issue name***

  É a identidade do emissor do certificado, também no formato de nome *DN*. É por este campo que um navegador sabe qual chave pública utilizar para a validação da conexão.

- **Período de Validade ou *validity period***

  Contém o período pelo qual o certificado é válido. Normalmente possui uma data de início e fim, e cada Autoridade Certificadora pode estabelecer um período de validade para seus certificados.

- **Chave pública**

  É a chave pública propriamente dita do usuário que está sendo certificado.

3.2 Certificado *Proxy* e o serviço *MyProxy*

Com o objetivo de garantir um acesso seguro a seus recursos, as grades utilizam a infra estrutura de chave pública [12].

A chave privada de um usuário deve ser guardada de forma mais segura possível, uma vez que é sua identidade, e se for utilizada indevidamente por terceiros, pode comprometer seu proprietário. Em função desta preocupação em se guardar a chave privada, uma solução foi pensada para que processos ou outros usuários possam ser autenticados como o usuário original do certificado "delegando-se" um novo certificado de validade bem mais restrita. Este novo certificado, também chamado de *proxy*, é emitido pelo proprietário do certificado original que também o assina, funcionando como uma procuração.

A Figura 3.3 representa o processo de delegação de certificados digitais. Nela podemos observar que a Autoridade Certificadora, ou simplesmente chamada de *CA*, emite o certificado do usuário com uma validade tipicamente de 1 ano, assinando-o. O usuário por sua vez, pode emitir um *proxy* com uma validade bem menor, tipicamente de 12 horas, e o assina. A emissão de certificados pode continuar, a princípio indefinidamente mas nenhum certificado delegado poderá ter uma validade maior do que o certificado que o originou, assim, ainda que possa existir uma falha de segurança dos *proxies*, sua validade é bem mais restrita.

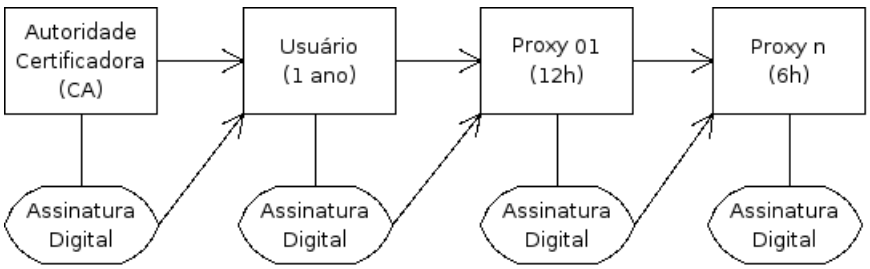


Figura 3.3: Delegação de Certificados Digitais

Quando há um estabelecimento de conexão segura, ambos os lados devem confiar nas CAs que assinaram os certificados emitidos pela outra parte. No caso do *proxy*, como o seu emissor foi o próprio usuário, ou ainda um outro *proxy*, a verificação deve ser feita "hierarquicamente" até chegar a uma *CA* na qual a parte que está fazendo a verificação confie. Tomando a Figura 3.3 como referência, o lado que esteja fazendo a verificação deverá chegar até a entidade que assinou o primeiro certificado da "cadeia de *proxies*".

Para que o usuário não precise portar fisicamente sua chave privada a fim de gerar *proxies* quando se fizer necessário, foi criado um serviço de armazenamendo de *proxies* que

pode ser acessado remotamente, serviço esse chamado *MyProxy* [25]. Com o *MyProxy*, também conhecido como repositório de credenciais, o usuário submete o primeiro *proxy* assinando-o com sua chave privada, dando uma validade típica de alguns dias ou horas a este, que a partir de então, gera outros *proxies* quando é solicitado.

O nome "/O=Grid /OU=GlobusTest /OU=simpleCA-mxsec.ic.uff.br /OU=ic.uff.br /CN=Usuario 01 do GT4 /CN=500098902 /CN=1452282786 /CN=1672141601", mostra o primeiro *CN* que designa o nome do dono do certificado, Usuario 01 do GT4, o segundo *CN* é o nome do primeiro *proxy* criado pelo certificado original, e assim sucessivamente como os próximos *CN's*. Este *proxy* especificamente, foi gerado pelo *MyProxy* e enviado para o portal durante a requisição pelo *Proxy Portlet*.

3.3 *Globus Toolkit*

O *Globus Toolkit* é um *software* de código aberto, desenvolvido pela *Globus Alliance*, e utilizado para "construir" sistemas de grades computacionais. Devido a forma como atua, ou seja, permitindo a interação das máquinas para formar a grade, é também chamado de *middleware*. Sua estratégia de desenvolvimento é similar a do sistema opercional Linux, onde há colaboração de várias partes [12].

A idéia de ser um *toolkit*, ou "caixa de ferramentas", é oferecer serviços e bibliotecas para o monitoramento, descoberta e gerenciamento de recursos, segurança, comunicação, detecção de falhas e portabilidade. É composto por um conjunto de componentes que podem ser usados juntos ou de forma independente para o desenvolvimentos de outras aplicações, foi desenvolvido para ser utilizado por desenvolvedores de aplicativos para grades computacionais, reduzindo o volume de trabalho necessário para se atingir os objetivos básicos de interoperabilidade entre as grades e os aplicativos. O público alvo não é o usuário final da grade, como um cientista ou administrador, e sim os desenvolvedores de aplicativos.

Todos os serviços disponibilizados pelo *Globus* foram desenvolvidos em "C" com seu código fonte aberto, possibilitando que colaboradores reparem erros ou façam qualquer tipo de modificação [21]. Para o usuário, estes serviços acabam se tornando transparentes, pois uma vez instalados e configurados, essas funções são chamadas automaticamente com o uso de comandos específicos do *Globus*, que pode ser instalado em diferentes plataformas, como Linux, Solaris e HPUX.

Os principais serviços do *Globus*, em qualquer versão, são os seguintes:

- ***GRAM - Globus Resource Allocation Manager***

  Este serviço permite executar *jobs* remotamente, assim como controlar e monitorar cada um dos *jobs*. Alguns comandos que implementam este serviço são: ***globus-job-run***, que permite execução de um *job* remotamente de forma interativa; ***globus-job-submit***, que coloca o *job* na fila de execução do servidor remoto retornando uma identificação de processo que é utilizado posteriormente pelos comandos ***globus-job-status***, ***globus-job-cancel***, ***globus-job-get-output*** e ***globus-job-clean***, para conhecer o estado do *job*, cancelar, retornar o resultado e limpá-lo da fila, respectivamente.

- ***GSI - Grid Security Infrastructure***

  É a infra estrutura de segurança da grade. Utiliza a infra estrutura de chave pública conforme abordado na subsessão 3.1. Permite que o usuário faça uma única autenticação, também conhecida como *"single sign on"*.

  Na configuração local de cada máquina participante da grade, existe um arquivo chamado "*grid-mapfile*", normalmente localizado em "/etc/grid-security/" que mapeia uma credencial para uma conta de usuário local, ou seja, quando um usuário da grade acessa aquela máquina com sua respectiva credencial, seu acesso é mapeado para uma conta local que está relacionada neste arquivo. A linha seguinte foi retirada do arquivo citado.

  "/O=Grid/OU=GlobusTest/OU=simpleCA-mxsec.ic.uff.br
  /OU=ic.uff.br/CN=Usuario 02 do GT4" usergrid01

  Onde, neste caso, o usuário "Usuario 02 do GT4" acessa esta máquina como se fosse "usergrid01". Este tipo de mapeamento também é importante para contabilizar a utilização dos recursos disponibilizados. A princípio vários nomes, ou várias credenciais podem apontar para a mesma conta, desta forma, por exemplo, usuários pertencentes a um projeto específico podem ser mapeados todos para a mesma conta que pode ser uma conta de projeto.

- ***MDS - Monitoring and Discovery Service***

  O *MDS* é responsável por armazenar informações sobre vários aspectos da grade como sistemas operacionais, memória disponível e espaço em disco, e é constituído por dois serviços internos, o *GIIS* e o *GRIS*, sendo este último o responsável por

obter as informações da máquina onde está sendo executado, ou seja, as informações locais. Já o *GIIS* (*Grid Index Information Service*) reúne em um único servidor as informações de vários *GRISs*. O *GIIS* solicita aos *GRISs* as informações coletadas em um intervalo de tempo definido pelo administrador, não tendo o usuário nenhum controle sobre isso. Esse tempo pode variar bastante, mas tipicamente fica entre 1 minuto a 20 minutos.

- ***GridFTP - Grid File Transfer Protocol***

  O *GridFTP* é um protocolo para transferência de arquivos entre máquinas que compõe as grades e é baseado no *FTP - File Transfer Protocol*. O canal de controle e de transferência são separados [4], permitindo que uma terceira parte transfira arquivos entre dois outros servidores. Um servidor *GridFTP* deve estar ativo na máquina que irá aceitar a conexão para a transferência do arquivo. Normalmente a porta "2811/tcp" é utilizada para este serviço. O *GlobusToolkit* disponibiliza tanto o servidor *GridFTP*, implementado por *globus-gridftp-server*, como o cliente, *globus-url-copy*.

- ***RFT - Reliable File Transfer***

  A partir da versão 3.0, o *GlobusToolkit* oferece este serviço de transferência de arquivos. O *RFT* é um serviço baseado na *Web*, do tipo *Web Services Resource Framework (WSRF*[1]). Normalmente os "serviços *Web*" possuem uma característica chamada "*stateless*", ou seja, não guardam o estado entre as invocações. A vantagem em se utilizar serviços *stateless*, é que a conexão com o servidor que fará a transferência não precisa ser mantida, sendo desfeita tão logo o comando seja processado, permanecendo somente a conexão entre os dois pontos de trasferência. No entanto, o *RFT* foi implementado de forma que o estado das transferências seja guardado em banco de dados, permitindo uma maior gerência sobre elas.

  Este serviço também permite a funcionalidade "*job scheduler*" que é o termo utilizado para descrever o mecanismo de "agendamento de tarefas" que o *GRAM* utiliza. Basta fornecer uma lista com todos os *URL's* de origem e destino dos arquivos a serem transferidos, que o *RFT* gerencia esta transferência de acordo com os recursos disponíveis.

[1] *WSRF* é uma família de especificações da *OASIS (Organization for the Advancement of Structured Information Standards)* para serviços baseados na *Web*

3.4 Gerência do certificado digital

Segurança é uma preocupação constante quando se utiliza uma estrutura de computação em grades [17]. O modelo utilizado para garantir a segurança de acesso aos recursos é o *x.509 PKI system (Public Key Infrastructure System)* [19], abordado na subsessão 3.1. A identificação do usuário, quando este deseja acessar os recursos oferecidos, pode ser feita utilizando-se diretamente o certificado digital assinado pela Autoridade Certificadora, ou pela utilização de um *proxy*, que é uma autorização com tempo de validade limitado a algumas horas, emitido a partir do certificado original, também já abordado subsessão 3.2.

O *Globus* possui um comando chamado "*grid-proxy-init*" que cria um *proxy* a partir do certificado original do usuário, que deve estar sob seu diretório *home*, utilizando-o então, a partir de sua criação e enquanto for válido, para fazer acesso a todos os recursos que forem necessários, sendo esta característica conhecida como "*single sign on*", ou seja, só há a necessidade do usuário se identificar uma única vez. O comando citado irá interagir com o usuário pedindo uma senha para descriptografar seu certificado antes de utilizá-lo, tendo sido armazenado desta forma para garantir um nível a mais de segurança uma vez que ficar sob a conta do usuário na própria máquina deixa-o muito vulnerável a cópias indevidas. O *proxy* criado por este comando é armazenado sob o diretório */tmp* da máquina local que está sendo utilizada, com o nome formado pela designação "x509up_u(UID)", onde (UID) é um inteiro correspondente a identificação do usuário no sistema.

Considerando que o acesso ao portal pode ser feito de qualquer máquina com conexão à internet, e que armazenar a chave privada no servidor em que o portal está sendo executado não é seguro, uma facilidade bastante útil é poder, a partir da máquina através da qual o portal está sendo acessado, gerar um *proxy* localmente utilizando a chave privada armazenada em um dispositivo externo de memória, como por exemplo uma memória *usb*, e transmitir esse *proxy* de forma segura para o servidor que executa o portal. Para isso, deve ser desenvolvido um *applet* a ser executado pelo *browser*, capaz de ler um dispositivo local, gerar e enviar o *proxy* criado. A grande desvantagem deste sistema é a necessidade de se portar fisicamente a chave privada. Uma outra possibilidade, é a utilização de um repositório de *proxies*, conhecido como *MyProxy* [25], conforme abordado na subsessão 3.2. O *MyProxy* acompanha a versão 4.0.1 do *Globus Toolkit* e o usuário pode armazenar um *proxy* no servidor *MyProxy*, utilizando os comandos apropriados encontrados no manual do mesmo e recuperá-lo para a devida utilização. Qualquer outra ação, como transferência de arquivos, submissão de *jobs*, ou consulta ao estado da máquina, depende da obtenção

e validade deste *proxy*, sendo desta forma, a primeira coisa a ser feita para utilização do portal.

3.5 Monitoramento dos servidores

Os sistemas de computação em grades podem ser compostos de muitos recursos ou servidores com as mais variadas configurações de *hardware* e *software*. Conhecer a configuração dos servidores, assim como a ocupação do tempo de processamento das *CPUs*, a fila de tarefas a serem executadas, e sua disponibilidade, podem ajudar na melhor opção para submissão de uma tarefa, podendo esta opção ser escolhida pelo próprio usuário, ou até mesmo por algum algoritmo escrito para isso.

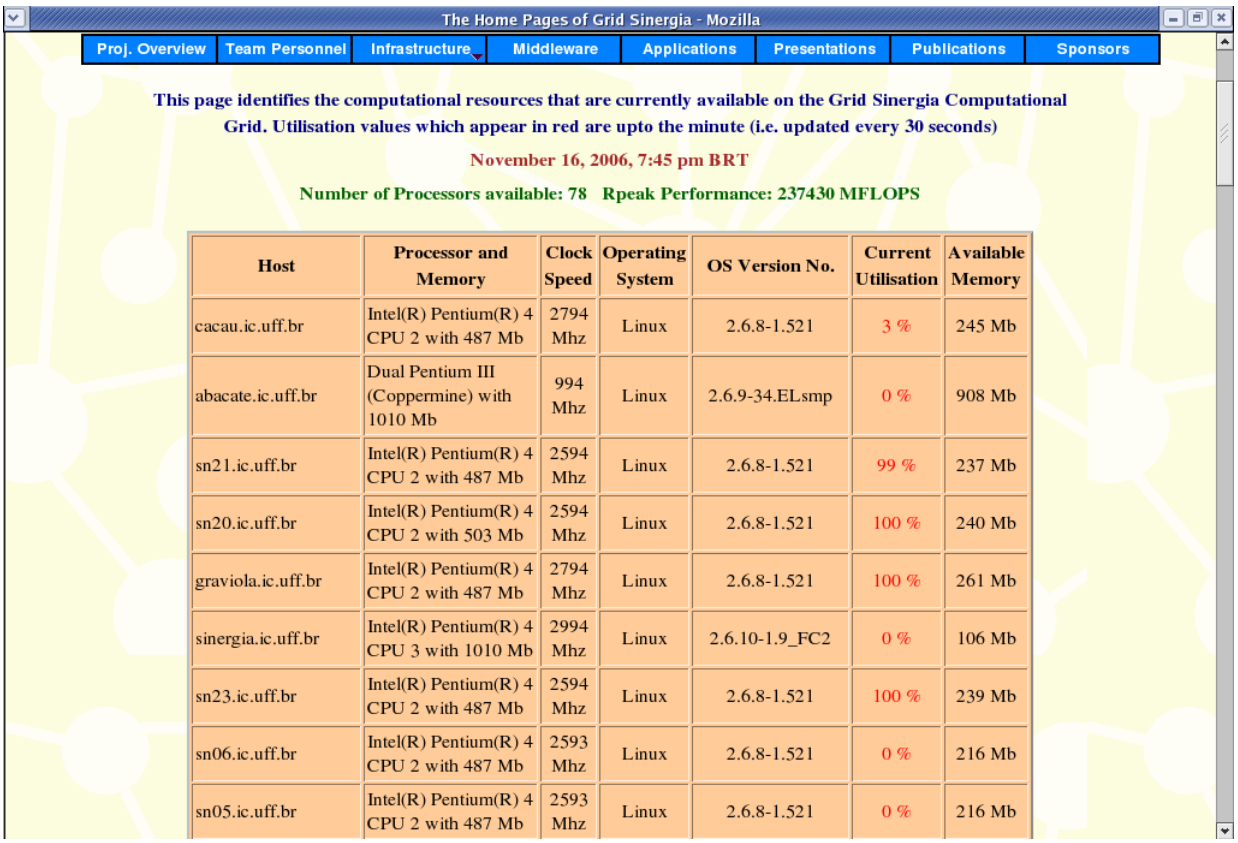
O *Globus* oferece o serviço chamado *MDS*, abordado na subsessão 3.3, que pode ser utilizado para a obtenção dos estados das máquinas monitoradas, no entanto, a versão 4.0.1 não é compatível com o *MDS* utilizado nas versões anteriores e não existe uma interface gráfica para visualização rápida da situação em particular dos servidores que o usuário tem acesso.

Com informações relevantes sobre o estados dos servidores, o usuário pode selecionar quais utilizar. A Figura 3.4 mostra a página do *Grid* Sinergia, em *HTML*, gerada a partir dos dados coletados pelo serviço *MDS*.

3.6 Gerência da transferência de arquivos

A troca de arquivos entre os servidores que compõe a grade é algo comum e essencial a sua operação. O portal deve possuir uma interface gráfica similar a um gerenciador de arquivos para que a transferência seja realizada de forma fácil. Dependendo do tipo de *job* a ser executado, muitas vezes é necessária a transferência de arquivos somente entre dois servidores, mas outras vezes há a necessidade de se transferir o mesmo arquivo para vários servidores ao mesmo tempo.

Muitas grades acessam recursos de *clusters*, onde normalmente existe uma máquina com uma função de *front end*, ou seja, uma máquina que recebe todas as solicitações e as encaminha para os nós que o compõe. Neste caso, normalmente o *cluster* utiliza *NFS (Network File System)* que é um sistema de compartilhamento de arquivos em rede que permite a exportação e montagem do sistema de arquivos de uma máquina através da rede para uma máquina remota, permitindo que os usuários acessem estes arquivos exatamente



Host	Processor and Memory	Clock Speed	Operating System	OS Version No.	Current Utilisation	Available Memory
cacau.ic.uff.br	Intel(R) Pentium(R) 4 CPU 2 with 487 Mb	2794 Mhz	Linux	2.6.8-1.521	3 %	245 Mb
abacate.ic.uff.br	Dual Pentium III (Coppermine) with 1010 Mb	994 Mhz	Linux	2.6.9-34.ELsmp	0 %	908 Mb
sn21.ic.uff.br	Intel(R) Pentium(R) 4 CPU 2 with 487 Mb	2594 Mhz	Linux	2.6.8-1.521	99 %	237 Mb
sn20.ic.uff.br	Intel(R) Pentium(R) 4 CPU 2 with 503 Mb	2594 Mhz	Linux	2.6.8-1.521	100 %	240 Mb
graviola.ic.uff.br	Intel(R) Pentium(R) 4 CPU 2 with 487 Mb	2794 Mhz	Linux	2.6.8-1.521	100 %	261 Mb
sinergia.ic.uff.br	Intel(R) Pentium(R) 4 CPU 3 with 1010 Mb	2994 Mhz	Linux	2.6.10-1.9_FC2	0 %	106 Mb
sn23.ic.uff.br	Intel(R) Pentium(R) 4 CPU 2 with 487 Mb	2594 Mhz	Linux	2.6.8-1.521	100 %	239 Mb
sn06.ic.uff.br	Intel(R) Pentium(R) 4 CPU 2 with 487 Mb	2593 Mhz	Linux	2.6.8-1.521	0 %	216 Mb
sn05.ic.uff.br	Intel(R) Pentium(R) 4 CPU 2 with 487 Mb	2593 Mhz	Linux	2.6.8-1.521	0 %	216 Mb

Figura 3.4: Página *MDS* do *Grid* Sinergia

como se fossem locais à máquina remota, de forma transparente quanto a sua localização física. No caso em que dados tenham de ser transferidos para mais de uma máquina, e que o *NFS* é utilizado, basta que somente uma transferência seja feita, preferencialmente para o servidor *NFS*.

Uma outra forma de se gerenciar a transferência de arquivos é através da utilização de *workflows*, onde as ações são encadeadas em uma representação gráfica do que deve ser executado. O resultado de uma ação, ou transferência, serve de entrada para outra.

3.7 Gerência da submissão de tarefas

A escolha de qual máquina da grade utilizar para a execução de um *job*, normalmente basea-se nas informações coletadas pelo sistema de gerenciamento disponibilizado, no caso do *Globus*, o serviço de *MDS*. No entanto, alguns dos parâmetros das máquinas da grade que são monitoradas, também são extremamente dinâmicos, como a ocupação de *CPU* e disponibilidade de memória, que são recursos essenciais para uma escolha mais apropriada de qual máquina utilizar, uma vez que alguns tipos de *jobs* requerem mais

tempo de processamento e outros maior quantidade de memória disponível por exemplo, porém, a interpretação destes valores requer conhecimento específico do comportamento de recursos e aplicações. Os parâmetros oferecidos pelo serviço *MDS* normalmente não são muito claros para usuários sem conhecimentos em computação como pesquisadores na área biológica por exemplo, tornando-se contraproducentes em algumas situações. O monitoramento dos servidores, conforme abordado na subsessão 3.5, permite que sejam implementados meios para uma decisão automática, baseado nos parâmetros monitorados, de qual máquina utilizar, uma vez que estes parâmetros são armazenados em banco de dados, acessível a qualquer outro aplicativo. Essa característica pode ser bastante útil na implementação de algorítmos utilizados para escalonamento dinâmico de *jobs* em uma grade [24].

A submissão de um *job* não garante que este será executado até sua conclusão, muito menos que será concluído sem nenhum erro, portanto, os *jobs* submetidos devem poder ser monitorados. Os aplicativos executados pelas grades são, em geral, aplicativos que demandam muito tempo para sua conclusão, tipicamente horas ou até dias, e sendo a disponibilidade dos recursos oferecidos pela grade variáveis com o tempo, é difícil de se prever qual será o tempo total de execução de um aplicativo até seu término, o que torna o recurso de monitoramento bastante útil, permitindo que as informações sobre o estado das tarefas sejam conhecidas praticamente em tempo real, uma vez que qualquer evento como término ou erro em um *job*, pode ser reportado através de email ou qualquer outro meio de comunicação.

3.7.1 Conclusão do capítulo

As tecnologias utilizadas nas grades computacionais são bem conhecidas como a infra-estrutura de chave pública e transferência de arquivos baseada no protocolo *FTP*, no entanto, sua utilização ainda requer que o usuário atue através de linha de comando, digitando os comandos a serem executados pela grade e interpretando seus resultados. Além disso, as transferências de arquivos, assim como a submissão de tarefas, podem se tornar tarefas de difícil gerência uma vez que os recursos são distribuídos e que as políticas de segurança e utilização destes podem variar de *site* para *site*.

Capítulo 4

Implementação

Este capítulo visa relatar a experiência e implementação do *MyEasyGridPortal*, um portal para a transferência de arquivos, execução de *jobs* e aplicações *MPI*, em particular aquele baseado no *middleware EasyGrid*. O objetivo é descrever detalhadamente os componentes utilizados e desenvolvidos, justificando as escolhas para o claro entendimento das decisões tomadas durante o desenvolvimento, de forma que seja possível a continuidade deste trabalho.

4.1 A implementação

O *MyEasyGridPortal* foi desenvolvido baseado no *framework GridSphere*, previamente abordado na subseção 2.2.2. A opção por utilizá-lo se deu pelas tecnologias empregadas em seu desenvolvimento já serem bastante difundidas e suportadas no desenvolvimento de aplicações *Web*, como Java, *Java Server Page* e *Servlets*. O fato deste *framework* suportar *portlets* escritos segundo a especificação JSR168 também pesou na decisão por utilizá-lo, uma vez que, a princípio, qualquer portal que também atenda a esta especificação poderá suportar os *portlets* desenvolvidos. Outro ponto relevante é que algumas entidades de ensino e pesquisa já o utilizam, conforme abordado na subseção 2.2.3. No *site* do projeto [2], na parte de documentação, foi disponibilizado, a partir de 01 de agosto de 2006, portanto depois do início desta dissertação de mestrado, um guia de desenvolvimento de *portlets* segundo as especificações da JSR168, sendo que o *status* deste documento ainda é "em desenvolvimento", mas já possui informações sobre a instalação do *framework*, assim como um passo a passo de como escrever um *portlet*.

4.1.1 Arquitetura do *MyEasyGridPortal*

O arquitetura do *MyEasyGridPortal* foi pensada de forma que torna-se possível a utilização de seus resultados e benefícios por outros aplicativos, como um escalonador de tarefas por exemplo, independente da linguagem em que estes são escritos, e independente dos próprios *portlets* implementados. A idéia foi permitir que os trabalhos já desenvolvidos sejam utilizados pelo portal, assim como os trabalhos futuros, o que torna a independência da linguagem utilizada uma necessidade, uma vez que a maioria dos aplicativos foi escrito em C. Por este motivo, os serviços comuns oferecidos pela API do *GridSphere*, como a obtenção de credenciais, não foram utilizados, pois caso contrário, para uma interação com outros aplicativos, exigiria que estes fossem escritos em Java e que houvesse também interação direta com o próprio portal.

O *MyEasyGridPortal* utiliza os comandos implementados pelo *Globus* para interagir com a grade e com o sistema operacional, desta forma, não havendo mudança no nome desses comandos de uma versão para outra do *Globus*, também não há necessidade de nenhuma modificação ou recompilação do código do portal, bastando que seja atualizada a versão do *Globus* que está instalada na máquina que hospeda o próprio portal. Como está mostrado pela Figura 4.1, a arquitetura utilizada é a do *GridSphere*, acrescida da interação com o sistema operacional para o acesso ao banco de dados e aos comandos implementados pelo *Globus*. O acesso ao *MyProxy Server* e a outras grades é feito através da Internet.

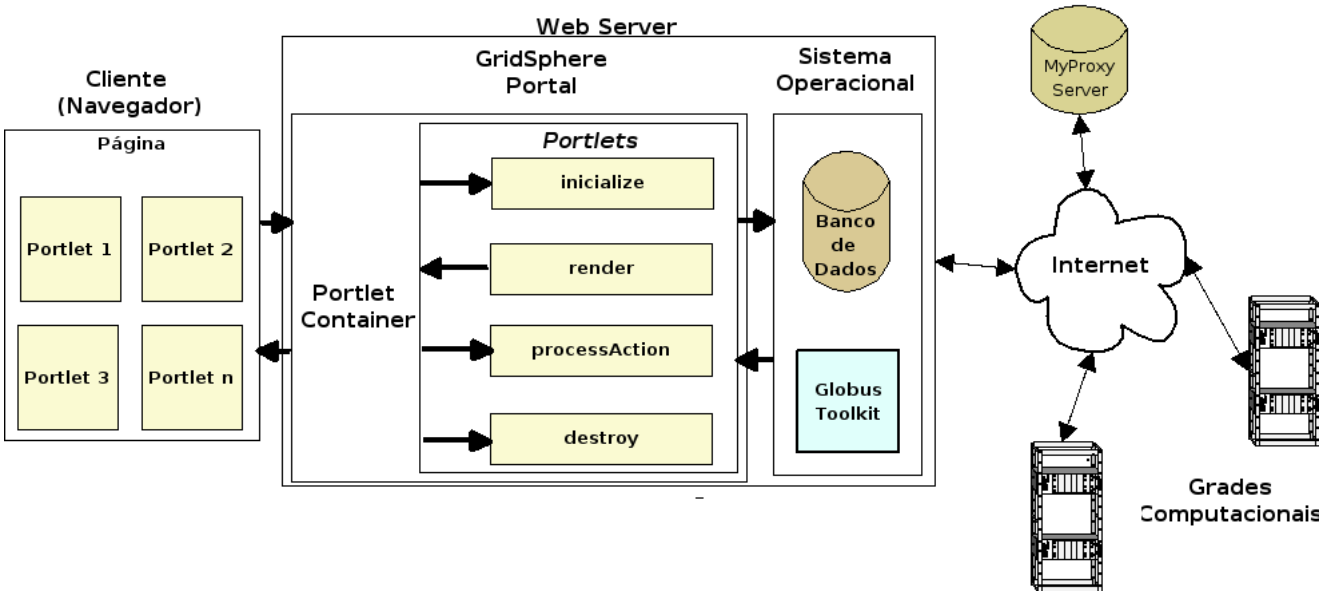


Figura 4.1: Arquitetura do *MyEasyGridPortal*

O portal executa os comandos como se fosse o usuário da grade, que pode ser diferente

do usuário do portal, usando para isso o comando *sudo*[1]. É na tabela do banco de dados, utilizado pelo portal, que guarda os servidores cadastrados nas contas dos usuários, que é mantido um mapeamento relacionando usuário da grade e usuário do portal, desta forma há flexibilidade para que um mesmo usuário do portal possa utilizar certificados digitais diferentes para acessar grades e contas diferentes. O recurso de poder utilizar mais de um certificado por usuário foi idealizado para permitir, por exemplo, que o mesmo usuário do portal possa acessar recursos de grades diferentes que não reconheçam a mesma Autoridade Certificadora. Suponha que uma determinada entidade resolva emitir seus próprios certificados para não depender de uma terceira parte ou para ter maior controle sobre quem utiliza seus recursos, neste caso, somente a grade disponibilizada por essa entidade é quem reconheceria os certificados utilizados por seus usuários, assim, um usuário do *MyEasyGridPortal* poderia fazer acesso a uma grade dessas, além de outras, utilizando a mesma interface, bastando que possua certificados assinados por entidades reconhecidas por cada grade respectivamente.

4.1.2 Configuração do sistema

Toda implementação foi feita e testada utilizando os seguintes recursos de *software*:

- Linux, versão do Kernel acima de 2.6
- MyProxy Server 2.3
- Globus Toolkit 4.0.1
- Java SDK 1.5
- GridSphere 2.1.2
- OpenSSH_3.9p1, OpenSSL 0.9.7a
- Apache Ant 1.6.5
- Apache Tomcat 5.5.16
- LAM 7.0.6/MPI
- MySql 14.12 Distrib 5.0.22
- gcc 3.4.4

[1] *sudo* é um programa utilizado pelo Unix e Linux que permite que um usuário execute um comando como se fosse outro. As limitações para essa execução, pela questão da segurança, podem ser definidas em um arquivo de configuração do *sudo* chamado *sudoers*.

4.1.3 Classes comuns

Os programas escritos em Java consistem de partes que são chamadas de classes [7]. A classes são compostas por atributos e por outras partes chamadas de métodos que, ao serem invocados, realizam operações e retornam resultados. A reutilização de código no Java é feita através de classes disponibilizadas através de bibliotecas chamadas de APIs.

A divisão dos aplicativos em módulos, em geral facilitam sua manutenção, legebilidade e reutilização. No caso de Java, essa modularização é feita através de classes que geralmente oferecem serviços comuns como acesso a banco de dados ou a arquivos, tornando sua reutilização possível por várias outras classes que necessitem destes serviços. Na implementação do *MyEasyGridPortal*, o acesso ao banco de dados, assim como o acesso a arquivos e chamadas ao sistema operacional foram implementados através de classes que são consideradas classes comuns, ou seja, todos os portlets as utilizam, bastando que inicialize uma instância dela. A Figura 4.2 mostra um esquema simplificado da interação entre os *portlets* e as classes comuns.

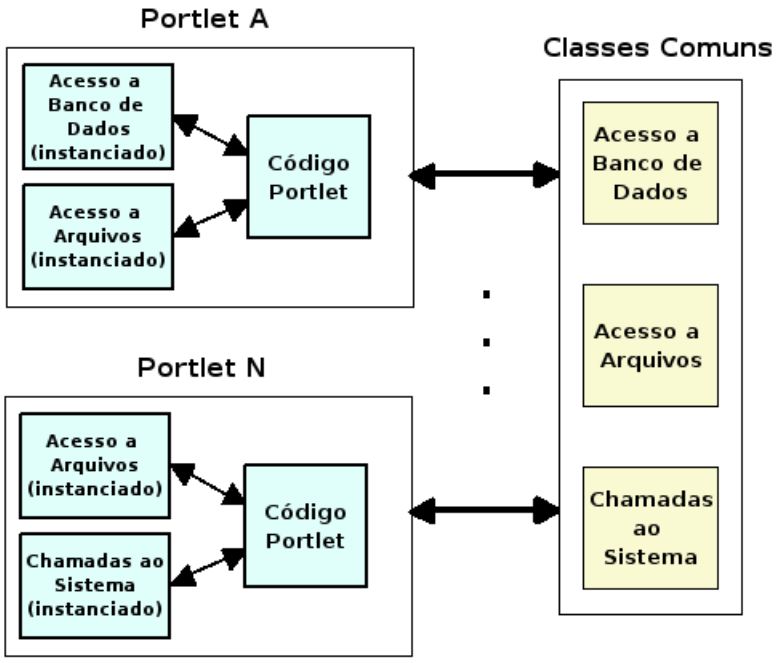


Figura 4.2: Classes comuns do *MyEasyGridPortal*

As classes comuns implementadas foram:

- **DBManipulation** - Esta classe faz acesso ao banco de dados do portal e é reutilizada também pelas outras classes comuns;

- **ExecObj** - Todas as chamadas ao sistema operacional para a execução de comandos são feitas através desta classe, que também disponibiliza o resultado ou erro retornado da chamada;
- **ListServers** - Esta classe basicamente faz a interface entre as chamadas das *JSPs* e o banco de dados, utilizando a classe *DBManipulation*.
- **GeneralDataCollect** - Os dados dinâmicos dos servidores da grade que estão cadastrados na conta do usuário do portal, são atualizados através desta classe que é instanciada como uma *thread*, ou seja, é um processo paralelo, disparado pelo portal com a função de consultar o servidor e atualizar seu estado na tabela correspondente do banco de dados. Dados dinâmicos são os dados de ocupação de *CPU* e quantidade de memória disponível.
- **UpDateServer** - Esta classe é responsável por inicializar as *threads GeneralDataCollect*.

Os atributos e métodos das classes descritas serão detalhados ao longo da descrição da implementação dos *portlets*.

Tabela 4.1: Tabela *jobs*

Field	Type	Null	Key	Default	Extra
id	int(2)	NO	PRI	NULL	auto_increment
user_grid	varchar(20)	YES		NULL	
user_server	varchar(20)	YES		NULL	
server	varchar(20)	YES		NULL	
job_id	varchar(70)	YES		NULL	
status	varchar(10)	YES		NULL	
monitor	tinyint(1)	YES		NULL	

4.1.4 Implementação sob o *GridSphere*

O *GridSphere*, conforme abordado na subseção 2.2.2, oferece um ambiente para o desenvolvimento de portais. A página inicialmente mostrada quando se faz um acesso ao *MyEasyGridPortal* pode ser vista pela Figura 4.3, onde o texto mostrado foi personalizado.

Após ter se identificado com seu *login* e senha, o usuário tem acesso ao *Welcome portlet*, mostrado pela Figura 4.4, onde suas informações básicas como nome, *email*, *ti-*

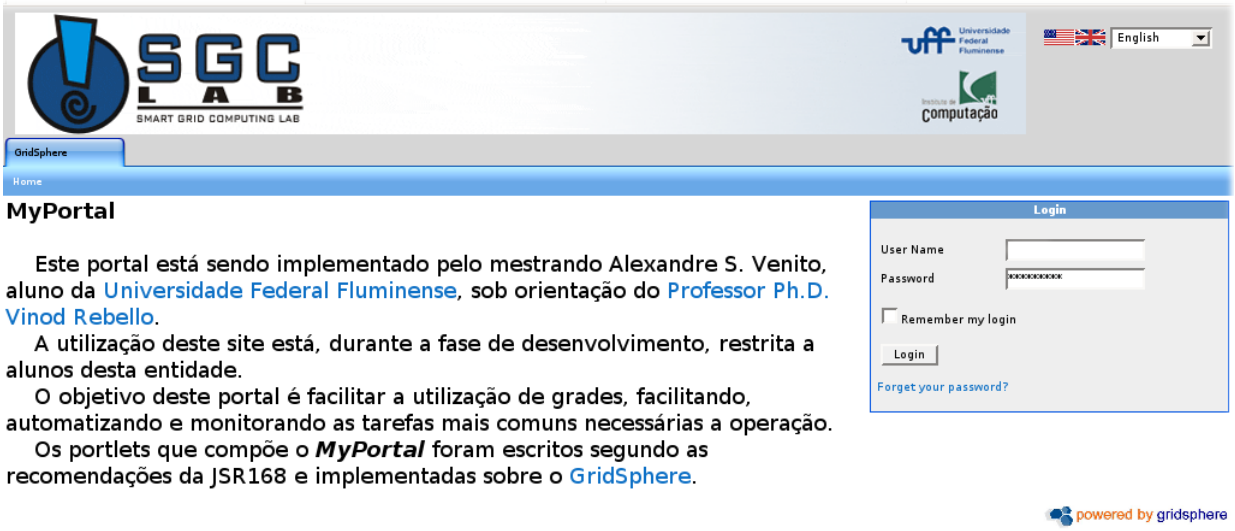


Figura 4.3: Página inicial do *MyEasyGridPortal*

mezone e *portlets* que utiliza podem ser configuradas. Os dados dos usuários, assim como sua senha, são armazenados em banco de dados próprio utilizado pelo *GridSphere*.

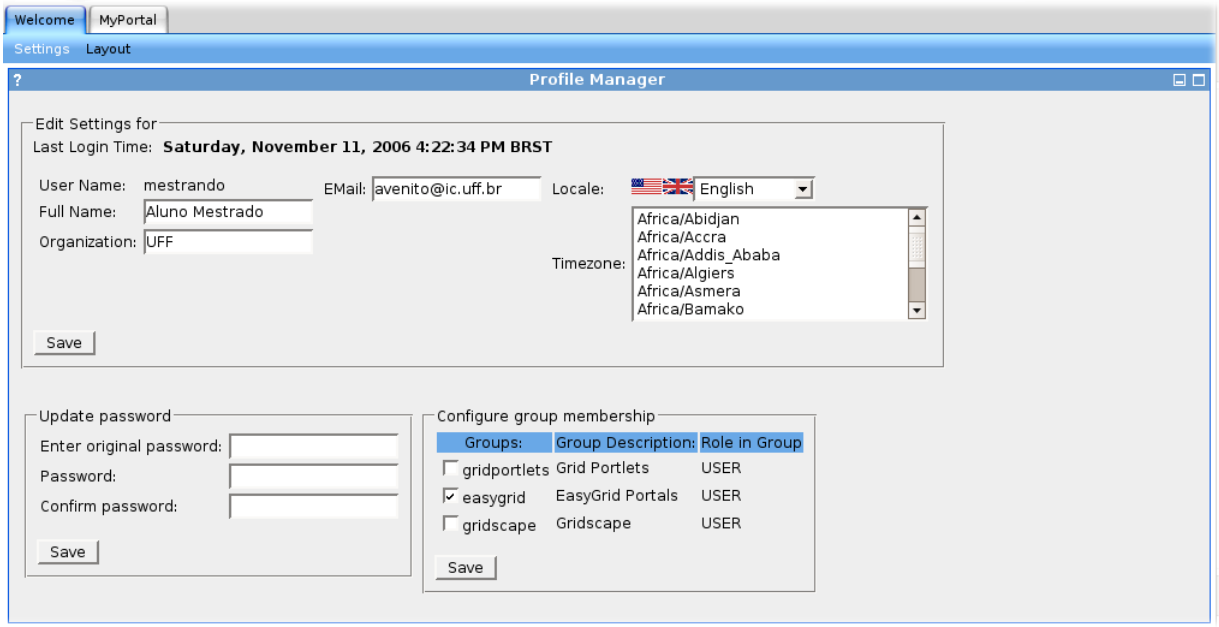


Figura 4.4: *Welcome portlet*

Os *portlets* implementados neste trabalho estão agrupados e disponíveis como *submenus* na aba "*MyEasyGridPortal*", como pode ser visto pela Figura 4.5, os *submenus* são: *Proxies*, *Servers*, *Files*, *Jobs* e *Help*. Se o usuário possuir permissões de administrador, o *Administration portlet* estará disponível para gerência do portal, a página inicial é mostrada pela Figura 4.6. O administrador também disponibiliza os demais *portlets* para os usuários ou grupos de usuários através da página mostrada pela Figura 4.7, onde pode-

mos observar os *portlets GridFTP, Help, Broadcast, Jobs, MPICH, Servers* e *MPILAM* selecionados para usuários com função "*User*".

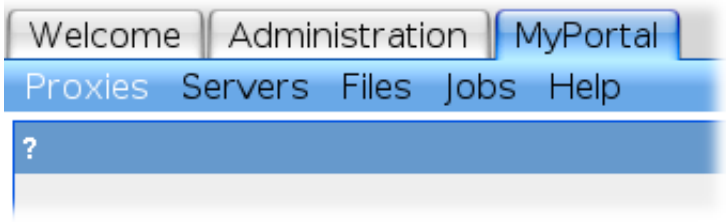


Figura 4.5: Aba *MyEasyGridPortal*

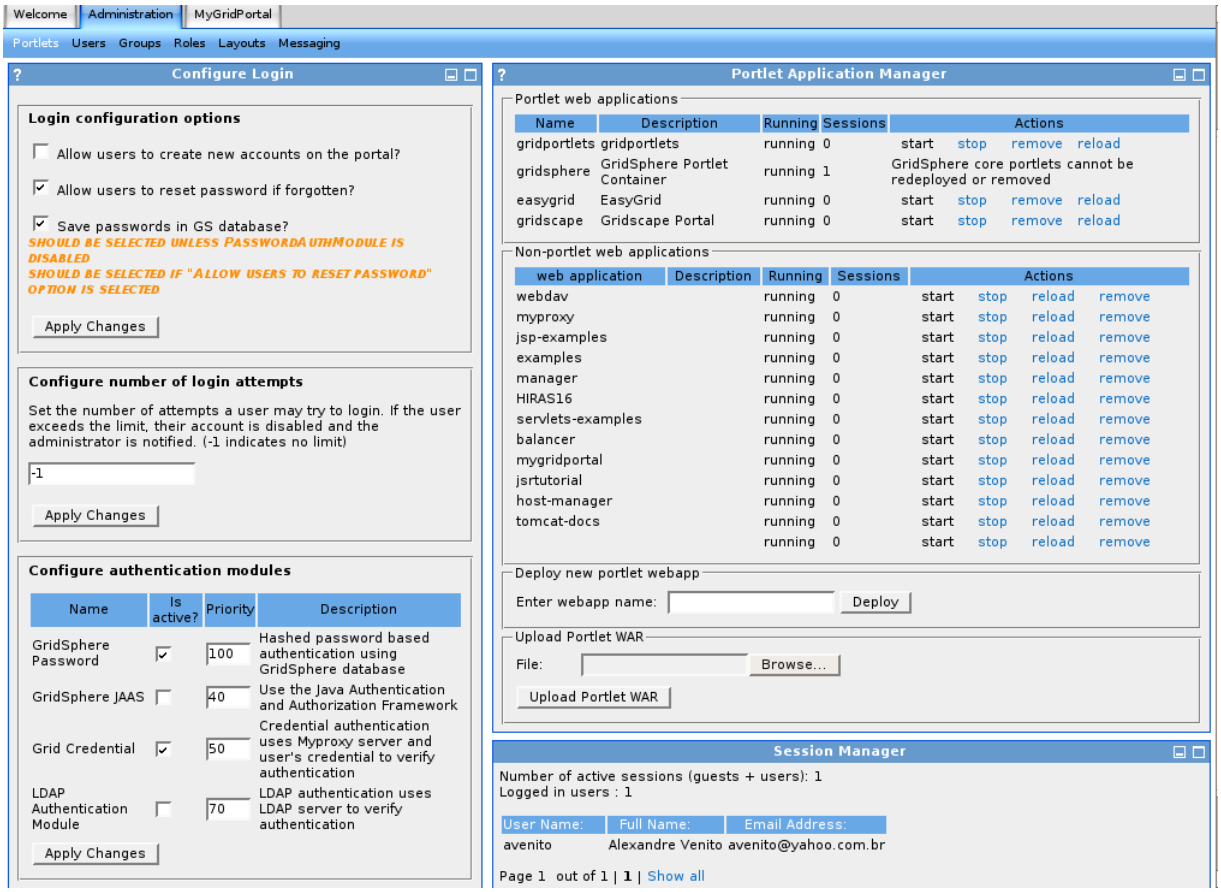


Figura 4.6: *Administration portlet*

4.1.5 Desenvolvimento e ciclo de vida de um *portlet*

Portlets são componentes *Web*, conforme abordado na subseção 2.1.5, portanto, do ponto de vista de instalação e configuração, podem ser tratados exatamente da mesma forma que outros aplicativos *Web*. A estrutura de diretórios na qual o *portlet* deve ser instalado é a mesma que a de um *servlet*. A hierarquia de diretórios mostrada na Figura 4.8 é a estrutura na qual o *MyEasyGridPortal* foi implementado.

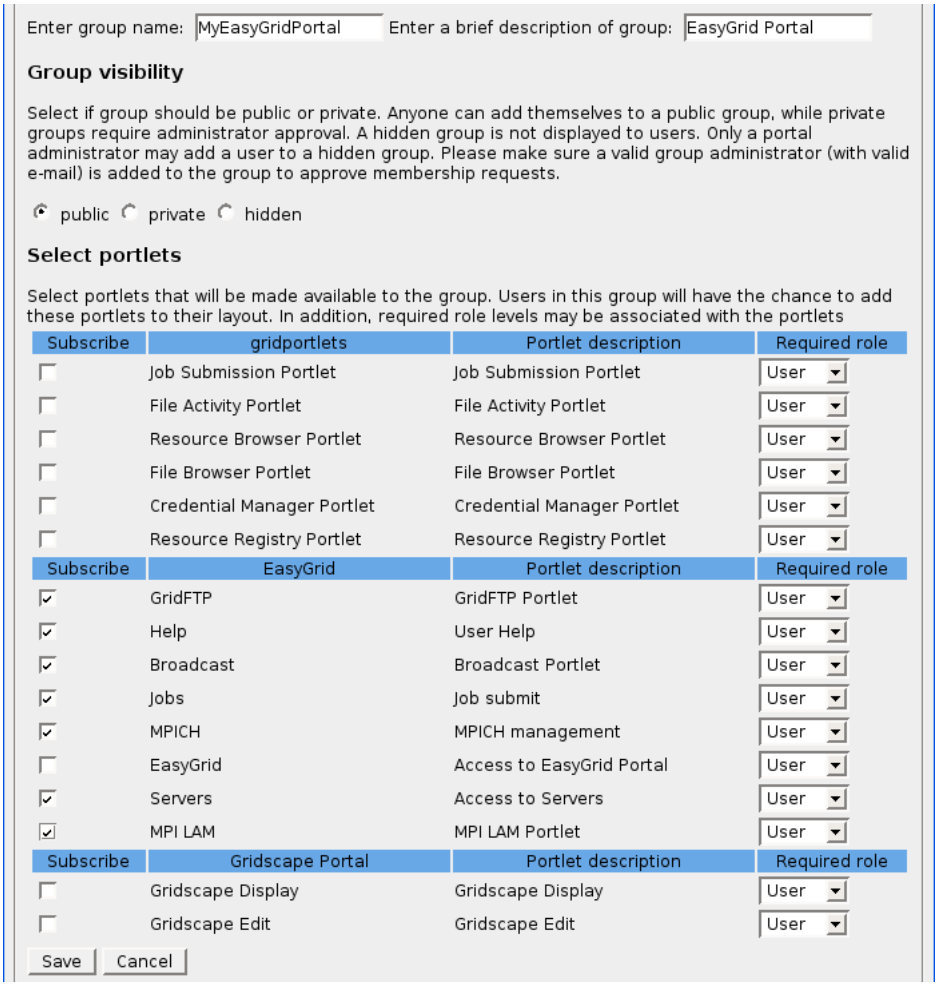


Figura 4.7: Disponibilização de *portlets*

Assim como em *servlet*, existem arquivos de configuração que são escritos em *xml*[2]. Todos os arquivos de configuração descritos ficam armazenados sob o diretório *WEB-INF* da aplicação.

- **web.xml** - Este arquivo define as características *Web* do *portlet*, incluindo o nome das classes que o implementam e os dados de inicialização.
- **portlet.xml** - É neste arquivo que estão definidas as características da aplicação *portlet*, ele possui os parâmetros de configuração como nome da aplicação no portal, títulos da barras de título e outros dados específicos do *portlet* a que se refere.
- **layout.xml** - Este arquivo define onde os portlets serão dispostos na página que será enviada para o cliente, se ocuparão todo o espaço disponível ou qual percentual da página será utilizada.

[2] *Extensible Markup Language* - Uma linguagem universal para permitir a troca de informações de forma estruturada.

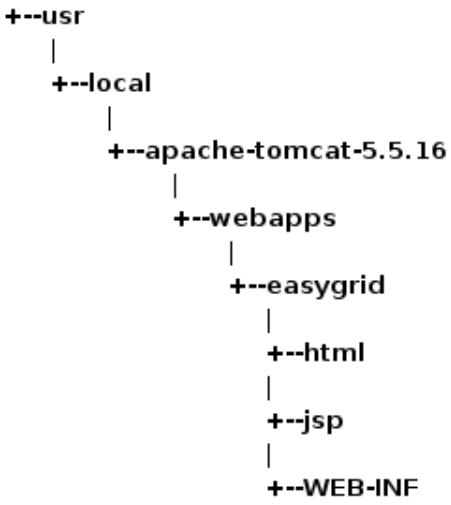


Figura 4.8: Estrutura de diretórios

- **group.xml** - Este arquivo é utilizado para agrupar os *portlets* e definir quais as permissões necessárias para a utilização de cada um deles. Este recurso é útil por exemplo para agrupar *portlets* afins e permitir que um deles seja utilizado somente por usuários com permissões de administrador ou super usuário.

O ciclo de vida de um *portlet* pode ser divido em três partes:

1. **Criação do *portlet***, quando é inicializado e passa a ocupar a memória;
2. **Processamento das requisições**, que é o tratamento das solicitações das ações originadas da interação do usuário com o portal e;
3. **Remoção e retirada da memória pelo *garbage collector***[3], quando o *portlet* é finalizado e retirado da memória deixando de ocupá-la.

A criação ocorre com a invocação do construtor da classe do *portlet*, que é uma função de inicialização que permite atribuir valores iniciais a campos internos da própria classe, e que deve ser minimamente implementada por:

public PortletClass() { }; (onde *PortletClass* é o nome da classe que implementa o *portlet*)

A próxima função, ou método a ser chamado pelo *container* quando este inicializa o

[3] *Garbage collector*, ou coletor de lixo em português, é um processo usado no gerenciamento de memória nos sistemas computacionais. Com este recurso é possível recuperar a zona de memória que um programa não esteja utilizando mais.

portlet é o *init()*, que o torna ativo. Um objeto do tipo *Portletconfig*, que contém as configurações definidas no arquivo web.xml[14], deve ser passado para este método. O *init()* do *Server portlet* está abaixo descrito, nele, o objeto denominado *config* é passado para a super classe e a variável "*DEFAULT_VIEW_PAGE*" é inicializada com o nome do método que deve ser invocado como *default*.

```
public void init(PortletConfig config) throws PortletException {
super.init(config);

DEFAULT_VIEW_PAGE = "showServersList";
}
```

Em todos os eventos disparados pela interação entre o usuário e o portal, como por exemplo o acionamento de um botão da página que está sendo exibida, o método *service()* é invocado e é responsável, por sua vez, por invocar o método correspondente a ação, se ela existir. O código a seguir, retirado do arquivo *viewServerList.jsp*, que é responsável por apresentar a lista de máquinas que estão cadastradas na conta do usuário do portal, renderiza um botão na página enviada para o cliente, que dispara um método chamado "*newServer*" quando é pressionado.

```
<ui:actionsubmit action="newServer"key="SERVERS_NEW_LABEL"/>
```

Quando o botão correspondente é pressionado, o método *newServer()* é invocado e é passado como parâmetro o objeto *ActionFormEvent*, que contém os objetos *ActionRequest* e *ActionResponse*. O fragmento de código a seguir foi retirado do arquivo *Server.java* que implementa o *Server portlet*, nele, o objeto *ActionFormEvent* é passado e o *ActionRequest* é recuperado pelo método *getActionRequest()*.

```
public void newServer(ActionFormEvent event) throws PortletException {
ActionRequest request = event.getActionRequest();
```

Assim foram implementadas todas as ações dos *portlets* desenvolvidos. E finalmente, quando o *container* determinar que um *portlet* não é mais necessário, o método *destroy()* é invocado para a liberação dos recursos alocados.

4.1.6 Implementação dos *portlets*

Os subitens seguintes visam descrever a implementação dos *portlets* que compõe o *MyEasyGridPortal* através de diagramas de casos de uso, classes, sequência e de colaboração, quando aplicáveis.

4.1.6.1 *Proxies*

O acesso aos recursos de uma grade somente são permitidos depois da autenticação do usuário através da utilização do certificado digital, abordados nas subseções 3.1, e 3.2. O *Proxies portlet* e está sob o *submenu Proxies* e foi implementado para acessar um repositório de credencias resgatando-a para ser utilizada pelo portal. Para tanto, o *portlet* faz uma chamada ao sistema operacional para a execução do comando "*myproxy-get-delegate*", que se conecta ao servidor *MyProxy* especificado pelo usuário e que contém um *proxy* seu, previamente armazenado, criando um novo *proxy* sob a conta que é informada na página de obtenção da credencial. O diagrama de caso de uso pode ser visto pela Figura 4.9. O usuário pode criar um *proxy* a partir de um servidor *MyProxy*, ou deletar algum já existente.

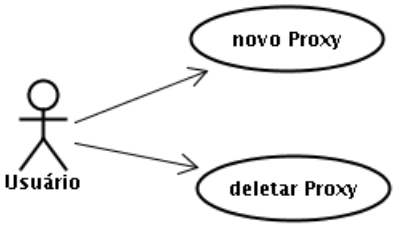


Figura 4.9: Caso de uso do *Proxies portlet*

A Figura 4.10 mostra a página inicial do *Proxies portlet*, nela podemos observar dois botões que permitem a inclusão ou exclusão de um *proxy*, e dois *proxies* com os seguintes campos:

- ***Label*** - É um nome dado pelo usuário quando o *proxy* é criado. Este parâmetro serve para uma identificação mais rápida no caso do usuário possuir mais de um *proxy*. Os dois *proxies* mostrados pela Figura tem os *labels* "User 01" e "User 02" respectivamente.
- ***Status*** - Mostra se o *proxy* está ativo ou expirado.
- ***Subject*** - É o nome DN do *proxy*, conforme abordado na subseção 3.1.

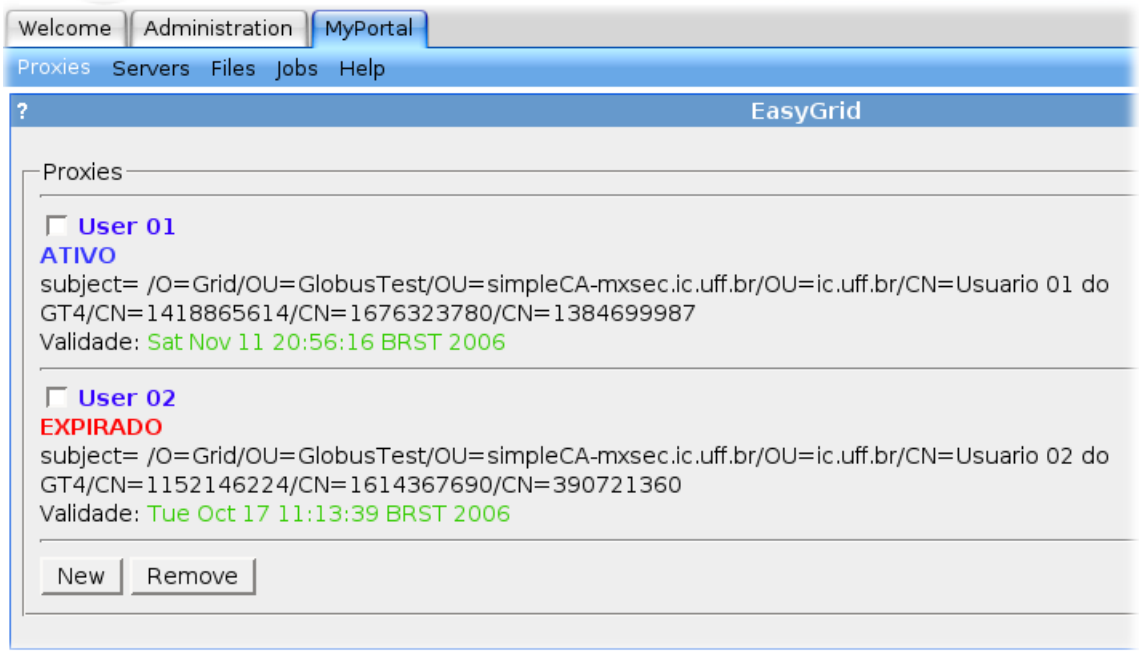


Figura 4.10: Página inicial do *Proxies portlet*

- **Validade** - Mostra até quando este *proxy* é válido.

Ao acionar o botão "New", uma outra página é exibida para que o usuário entre com os dados necessários para que um *proxy* seja recuperado a partir de um *MyProxy server*. Os campos, mostrados pela Figura 4.11, são os seguintes:

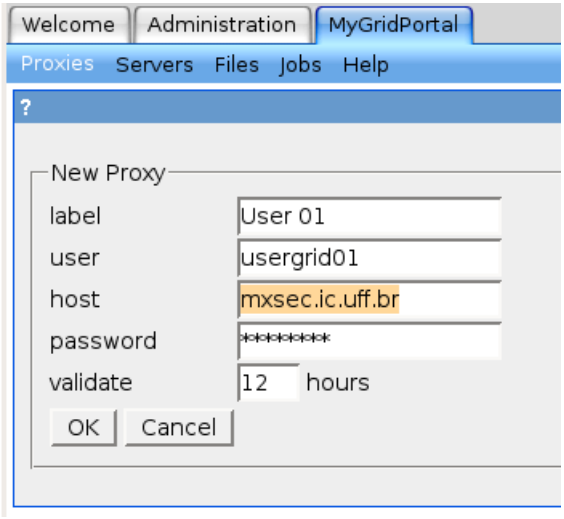


Figura 4.11: Página de novo *proxy*

- ***label*** - Nome pelo qual o *proxy* será referido. É o nome que é utilizado para substituir o *DN*.

- ***user*** - Nome do usuário sob o qual o *proxy* foi armazenado no repositório.
- ***host*** - Nome do *host* no qual o *MyProxy server* está rodando. Isto permite flexibilidade ao usuário, uma vez que este pode utilizar o servidor *MyProxy* que desejar e confiar.
- **password** - Senha sob a qual o *proxy* foi armazenado no repositório.
- **validate** - Tempo de validade do *proxy* a ser criado. É importante observar que este tempo de validade está condicionado ao parâmetro de validade máxima passado ao servidor *MyProxy*, e também ao prazo máximo de validade do próprio *proxy* quando este foi submetido ao servidor. Ou seja, nenhum *proxy* criado pode ter data de validade posterior ao que o criou, e a validade máxima dos *proxies* posteriormente criados a partir deste, também pode ser definida através de um parâmetro na linha de comando, no momento em que o usuário armazena seu *proxy* no servidor.

O diagrama de sequência do *Proxies portlet* é mostrado na Figura 4.12, nela podemos observar que quando um novo *proxy* é solicitado, o *portlet* aponta o arquivo *viewNew.jsp* para que o portal o utilize para montar a página que será enviada para o usuário, mostrada na Figura 4.11. Ao acionar o botão "OK", o método *getCredential()* é disparado, fazendo com que o *portlet* faça uma solicitação ao *ExecObj*, responsável pela interação com o sistema operacional, para que este execute o comando *myproxy-get-delegation*, utilizando os parâmetros fornecidos pelo usuário. Logo em seguida o resultado desta operação é solicitado e, não havendo nenhum erro, o *proxy* criado sob a conta do usuário passado como parâmetro tem uma referência guardada no banco de dados através de uma solicitação à classe *DBManipulation*, abordada em 4.1.3. O fato do portal armazenar o *proxy* sob a conta do usuário passado como parâmetro, significa que esta conta deve existir no sistema operacional que está sendo uitlizado na máquina que hospeda o portal, sendo essa uma responsabilidade do seu administrador. Finalmente o *portlet* aponta o arquivo *viewMain.jsp*, o arquivo padrão, para ser exibido pelo portal, que exibe a página inicial que pode ser vista pela Figura 4.10.

As Tabelas 4.2 e 4.3 mostram a descrição dos campos que a compõe e parte do conteúdo desta, respectivamente.

Os campos da Tabela 4.2 são:

- ***id*** - É a identificação do registro da tabela, usado como chave primária e com auto incremento.

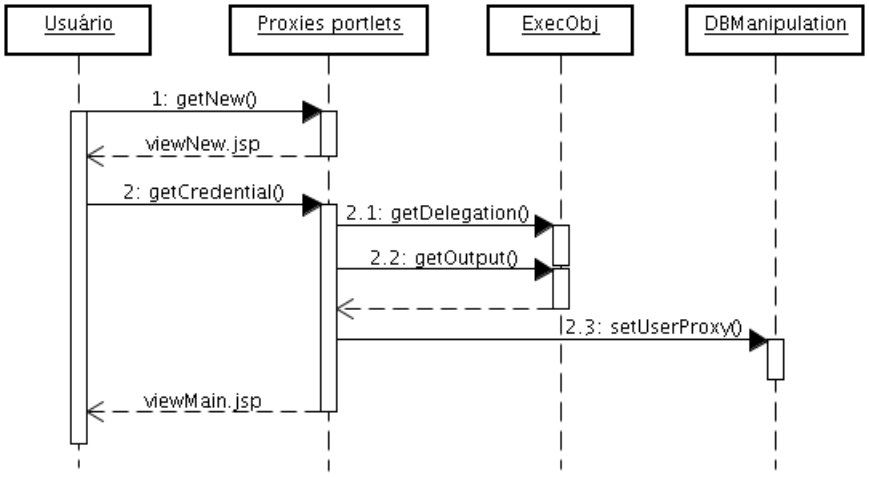


Figura 4.12: Diagrama de sequência do *Proxies portlet*

Tabela 4.2: Tabela *proxy*

Field	Type	Null	Key	Default	Extra
id	int(2)	NO	PRI	NULL	auto_increment
user_grid	varchar(20)	YES		NULL	
user_server	varchar(20)	YES		NULL	
renew	tinyint(1)	YES		NULL	
proxy	varchar(25)	YES		NULL	

- ***user_ grid*** - Identifica o usuário do portal ao qual aquele *proxy* pertence.
- ***user_ server*** - Representa o usuário sob qual o *proxy* foi armazenado no *MyProxy*. Armazena o campo "usuário" mostrado pela Figura 4.11.
- ***renew*** - Este campo tem por objetivo informar se aquele registro de *proxy* deve ser renovado automaticamente ou não.
- ***proxy*** - O nome do arquivo sob o qual o *proxy* é armazenado é aqui mostrado.

Tabela 4.3: Conteúdo da tabela *proxy*

id	user_grid	user_server	renew	proxy	label
34	avenito	usergrid01	NULL	/tmp/x509up_u491	User 01
31	avenito	usergrid02	NULL	/tmp/x509up_u13128	User 02

Podemos observar pela Tabela 4.3 que os *proxies* alí referenciados pertecem ao mesmo usuário do portal, descrito pelo campo *user_ grid*. Observamos também que o *proxy* de *id* igual a 34 e *label* igual a "User 01", tem "usergrid01" como *user_ server*, e o nome do

arquivo que o contém é */tmp/x509up_u491*. O campo *user_server* armazena o nome do usuário que é usado pelo sistema operacional, da máquina que roda o portal, sob o qual será guardado o *proxy*. Sempre que uma chamada ao sistema operacional for feita, o portal utilizará o usuário armazenado neste campo como um dos parâmetros desta chamada. Esta característica permite que o acesso aos recursos oferecidos pela grade seja feito diretamente com o *proxy* do próprio usuário, ao invés de um *proxy* pertencente ao portal, o que permite a contabilidade do uso dos recursos além de um maior controle sobre quem acessa quais recursos.

Os campos *ca* e *renew* não estão preenchidos. Estes campos não foram utilizados na implementação, mas podem ser futuramente para armazenar o nome da autoridade certificadora e indicar se o certificado deve ser renovado automaticamente.

4.1.6.2 *Servers*

Depois da obtenção do *proxy*, o próximo passo é criar uma lista dos servidores que o usuário tem acesso, cujo procedimento será explicado posteriormente a descrição da Figura 4.13 que mostra a página inicial do *Servers portlet*, encontrado sob o *submenu Servers*. Podemos observar os seguintes campos:

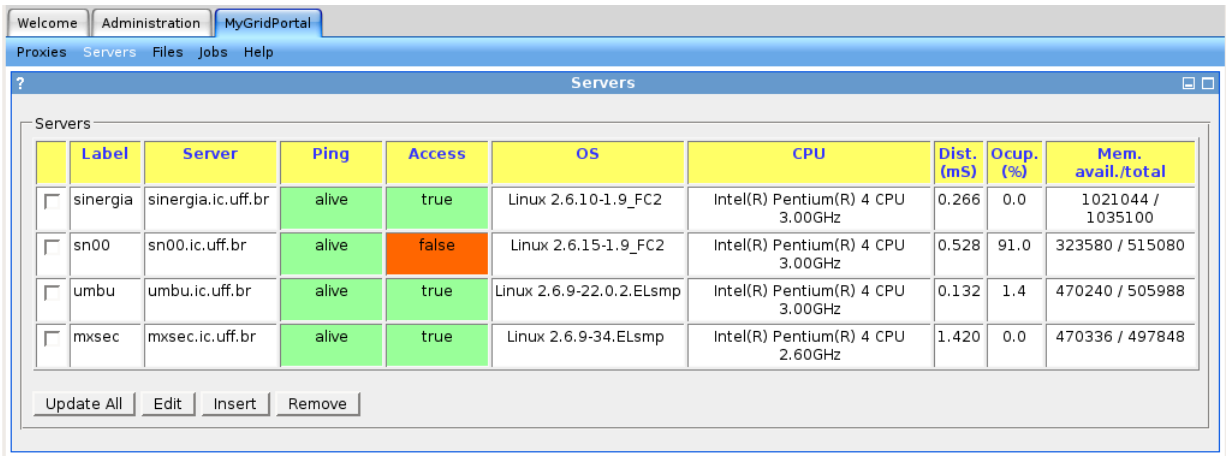


Figura 4.13: Página inicial do *Servers portlet*

- ***Label*** - Nome pelo qual o servidor é referido.
- ***Server*** - Nome do servidor, o mesmo obtido pelo comando *hostname* do Linux.
- ***Ping*** - Este campo pode assumir dois valores, *alive* ou *no response (no resp.)*, sendo que corresponde a resposta do comando *ping*[4]. Por este campo é possível saber se

[4] *Ping* é o nome de uma ferramenta que provê um teste básico se determinado equipamento de rede está funcionando e é alcançável pela rede do equipamento de onde é disparado o teste.

um servidor está ativo ou não, considerando que ele está configurado para responder a uma solicitação do tipo *ping*.

- ***Access*** - Através deste campo é possível saber se o servidor permite uma conexão para a utilização de seus recursos pelo usuário. Pode assumir dois valores, *true* ou *false*. Pela Figura 4.13 é possível verificar que o servidor "sn00.ic.uff.br" não está acessível para conexão, apesar de estar respondendo a solicitação de *ping*. O *proxy* utilizado para esta verificação é o que foi selecionado no momento em que o servidor foi inserido pelo usuário, utilizando a página própria pra isso pelo *Servers portlet*. Este campo é necessário ser atualizado constantemente pois o acesso aos recursos é controlado pelos administradores das próprias máquinas, podendo mudar o acesso sem aviso prévio e sem controle do usuário.

- ***OS*** - Este campo mostra qual o sistema operacional e versão que estão instalados no servidor. Este dado é coletado automaticamente pelo *daemon* descrito na subseção 4.1.6.7.

- ***CPU*** - Descreve o nome e o tipo do processador utilizado pelo servidor

- ***Dist.(mseg)*** - Representa a distância, em milisegundos, entre a máquina que hospeda o portal e o servidor listado.

- **Ocup.(%)** - Através deste campo é possível saber qual a ocupação da *CPU* na última solicitação. A atualização deste campo depende da solicitação do usuário através do botão "Atualiza todos", descrito na subseção 4.1.6.3.

- **Mem. disp./total** - Este campo mostra qual a memória disponível e total, em *KBytes*, do servidor na última solicitação

O diagrama com os casos de uso pode ser visto pela Figura 4.14. Nele podemos observar que "Recarregar"(disparado pelo botão "*Refresh*"), estende "Atualizar todos" (disparado pelo botão "*Update All*") pelo fato de ser um caso de uso específico para quando as *threads* disparadas ainda não terminaram de atualizar o banco de dados. Quando o botão "*Update All*" é acionado, uma *thread* para cada servidor da lista do usuário é disparada afim de obter e atualizar o estado do mesmo. Para que a página seja recarregada rapidamente, o *Servers portlet*, através do *viewJobs.jsp*, remonta a página enviada para o usuário com o botão "*Refresh*" no lugar do "*Update All*". O botão "*Refresh*" deve ser acionado novamente até que o "*Update All*" tome seu lugar, indicando que todas as *threads* já terminaram e que o estado mostrado é o mais atual.

Quanto as ações que o usuário pode tomar, temos: inserir um servidor na lista, editar os dados correspondentes, excluir um ou mais servidores, e obter o estado atualizado de todos os servidores listados.

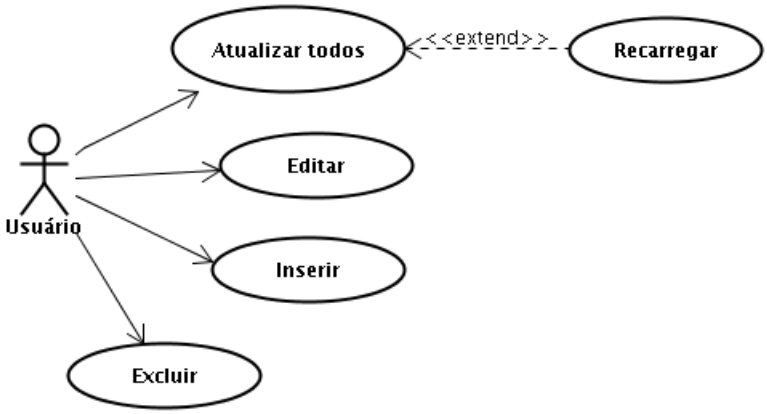


Figura 4.14: Casos de uso do *Servers portlet*

Para a inserção de servidores na lista, após acionar o botão "*Insert*", a página para inserção é exibida, conforme a Figura 4.15 com os seguintes campos que devem ser preenchidos:

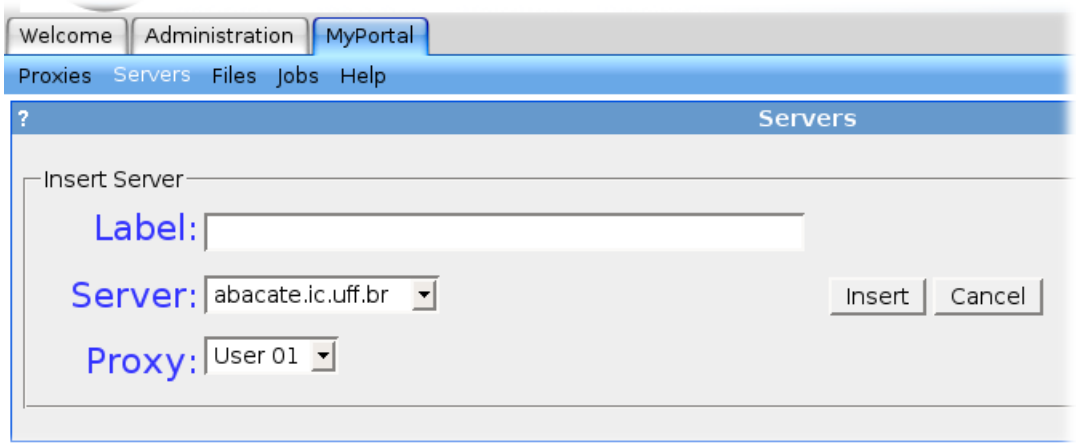


Figura 4.15: Página de inserção de servidores

- ***Label*** - Nome pelo qual o servidor será referido.
- ***Server*** - Este campo na verdade é uma caixa *select* contendo a lista de todos os servidores disponíveis. O usuário deve selecionar um deles. A lista de servidores que é exibida, foi previamente cadastrada pelo administrador do portal.
- ***Proxy*** - Este também é uma caixa *select* que contém os *labels* dos *proxies* que estão cadastrados na conta do usuário do portal, estejam eles ativos ou não.

A inserção de um servidor na lista tem seu diagrama de sequência mostrado pela Figura 4.16. Ao acionar o botão "*Insert*", o método *newServer()* é invocado, redirecionando o arquivo *viewNew.jsp*, responsável por mostrar a página com os campos de dados para a inserção de um novo servidor na lista, representada pela Figura 4.15. Depois de preenchido os campos correspondentes, ao acionar o botão "*Insert*" da página de inserção, o método *addServer()* do *portlet* é chamado, o que dispara uma chamada a outro método de mesmo nome para o *DBManipulation*, que por sua vez, insere o servidor na tabela apropriada, e finalmente o *portlet* redireciona o arquivo "*showServerList.jsp*" para o portal exibir novamente a página inicial. Para a inserção de outros servidores, esse procedimento deve ser repetido. Os servidores são adicionados independentemente para cada usuário.

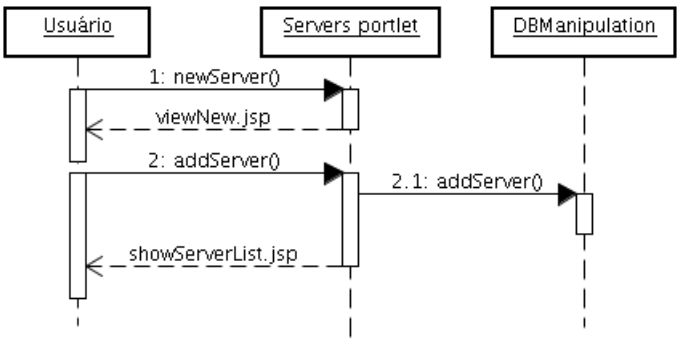


Figura 4.16: Diagrama de sequência de inserção de servidores

Duas tabelas são utilizadas pelo portal para armazenar os dados referentes aos servidores que são acessados. A primeira delas, mostrada pela Tabela 4.4 armazena as informações referentes aos servidores cadastrados pelo administrador do portal, que são os servidores que aparecem no campo *Server*, mostrado pela Figura 4.15. Quando o usuário insere um servidor em sua lista de acesso, um novo registro é inserido na Tabela 4.5.

Descrição dos campos da Tabela 4.4:

- ***id*** - É a identificação do registro da tabela, usado como chave primária e com auto incremento.
- ***server*** - Nome do servidor.
- ***alive*** - Este campo indica se o servidor está respondendo a uma requisição *ping*.
- ***nfs_ server*** - Indica qual o servidor no qual o *nfs* está montado, se houver.
- ***so*** - Indica o nome do sistema operacional utilizado pelo servidor.

Tabela 4.4: Tabela *default_servers*

Field	Type	Null	Key	Default	Extra
id	int(2)	NO	PRI	NULL	auto_increment
server	varchar(20)	NO		NULL	
alive	tinyint(1)	YES		NULL	
nfs_server	tinyint(1)	YES		NULL	
so	varchar(15)	YES		NULL	
version	varchar(15)	YES		NULL	
cpu_model	varchar(30)	YES		NULL	
clock	varchar(5)	YES		NULL	
globus_version	varchar(10)	YES		NULL	
distance	varchar(5)	YES		NULL	
mem_total	int(8)	YES		NULL	
mem_free	int(8)	YES		NULL	
cpu_load	varchar(5)	YES		NULL	
updating	tinyint(1)	YES		NULL	

- ***version*** - Indica a versão do sistema operacional utilizado.
- ***cpu_ model*** - Modelo da *CPU*.
- ***clock*** - Indica a frequência do *clock* da *CPU*.
- ***globus_ version*** - Representa a versão do *Globus Toolkit* utilizada.
- ***distance*** - Indica a distância do servidor em milisegundos, tomando como referência a máquina que hospeda o portal.
- ***mem_ total*** - Inidica a memória total instalada no servidor.
- ***mem_ free*** - Indica a memória total disponível no momento da consulta.
- ***cpu_ load*** - Representa a ocupação média da *CPU*.
- ***updating*** - Este campo sinaliza que o servidor deve ser atualizado. É utilizado pelo *daemon Job Monitor*, abordado na subseção 4.1.6.7.

Descrição dos campos da Tabela 4.5:

- ***id*** - É a identificação do registro da tabela, usado como chave primária e com auto incremento.
- ***user_ portal*** - Representa o nome do usuário do portal que utiliza aquele servidor

Tabela 4.5: Tabela *servers*

Field	Type	Null	Key	Default	Extra
id	int(4)	NO	PRI	NULL	auto_increment
user_portal	varchar(20)	YES		NULL	
user_server	varchar(20)	YES		NULL	
home	char(70)	YES		NULL	
access	tinyint(1)	YES		NULL	
icon	int(1)	YES		NULL	
server	varchar(20)	YES		NULL	
label	varchar(20)	YES		NULL	
distance	varchar(5)	YES		NULL	
updating	tinyint(1)	YES		NULL	

- ***user_ server*** - Representa o nome do usuário pelo qual o servidor é acessado. Na prática este campo é utilizado para identificar qual *proxy* será usado no acesso àquele servidor.
- ***home*** - Representa o diretório *home* que é acessado pelo usuário no servidor.
- ***access*** - Inidica se o acesso foi feito com sucesso. Um valor positivo, ou *true*, indica que o servidor pôde ser acessado com o *proxy* especificado.
- ***icon*** - Este campo foi inserido com a finalidade de ser utilizado para apontar uma imagem que servirá como ícone na apresentação na página enviada para o navegador.
- ***server*** - Refere-se ao campo *id* do servidor da tabela *default_ servers*.
- ***label*** - Refere-se ao *label* pelo qual o servidor será referenciado.
- ***distance*** - Indica a distância, em milisegundos, do servidor em relação à máquina em que roda o portal.
- ***updating*** - Este campo sinaliza que existe uma *thread* de atualização do estado do servidor em andamento. É utilizado pelo *viewJobs.jsp* para exibir o botão "*Update All*" ou "*Refresh*" na página para o usuário. Caso exista algum servidor com este campo verdadeiro, o botão "*Refresh*" é exibido.

Quando o botão "*Update All*" é acionado, uma *thread* responsável pela atualização dos dados do referido servidor é disparada para cada servidor listado, atualizando os dados no banco de dados do portal. Para que a página exibida para o usuário seja recarregada rapidamente, independente do término de todas as *threads*, o *label* do botão *Update All* é

trocado para *Refresh*, indicando que os dados exibidos ainda devem ser atualizados e que a página deve ser recarregada. O *label* só voltará a ser *Update All* se no momento em que a página for recarregada, todas as *threads* disparadas já tiverem terminado, desta forma, o usuário pode ter certeza que os dados exibidos estão atualizados.

4.1.6.3 *GridFTP* e *Broadcast*

Tendo adicionado os servidores em sua lista, o usuário já pode utilizar o *GridFTP portlet* para a transferência ponto a ponto de arquivos, localizado sob o *submenu Files*. A Figura 4.17 mostra a página inicial do *portlet* em questão. Podemos observar que a página está divida em duas partes, com os botões de cópia e transferência de arquivos entre elas. O campo *Server* mostra uma lista dos servidores que estão cadastrados na conta do usuário do portal, e o campo *Directory* mostra o nome do diretório do servidor acessado remotamente pelo usuário. Este é o diretório *home* do usuário que está mapeado pelo arquivo *grid-mapfile*, abordado na subseção 3.3. Selecionados os servidores e o arquivo a ser copiado ou movido entre eles, os botões de cópia e transferência podem ser utilizados. Somente é possível mover ou copiar um arquivo de cada vez, a operação com múltiplos arquivos é indicada como trabalhos futuros.

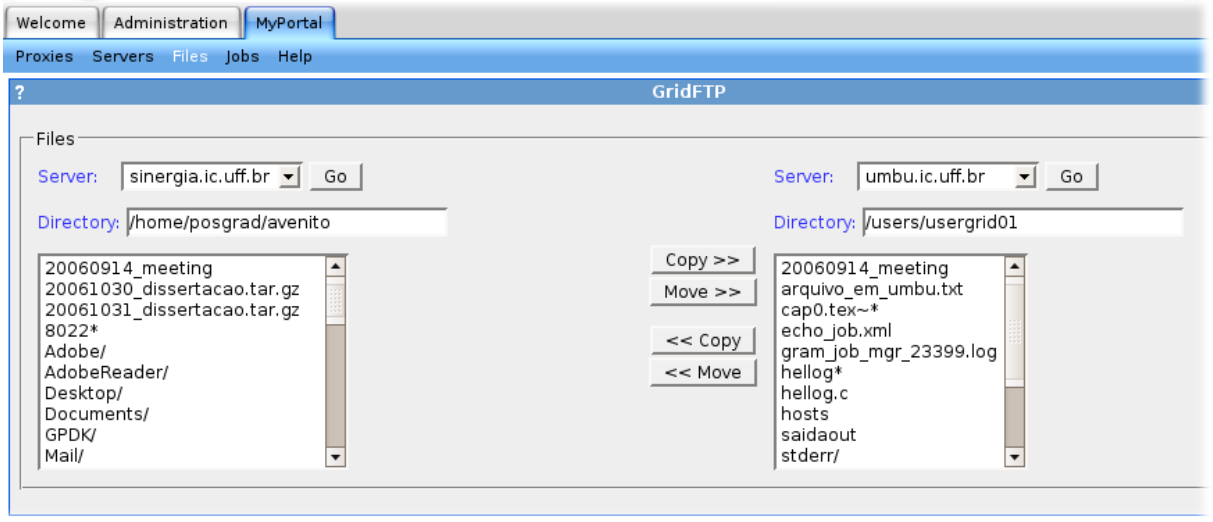


Figura 4.17: Página inicial do *GridFTP portlet*

Para a transferência do arquivo selecionado, o camando *globus-url-copy* é utilizado pelo portlet, fazendo uma chamada ao sistema operacional e colocando esta transferência em segundo plano. A execução desta tarefa de transferência em segundo plano permite que página exibida para o usuário seja recarregada rapidamente. Não há tabelas implementadas para este *portlet*.

O *Broadcast portlet* tem por função fazer a transferência de "um arquivo para vários pontos" de uma só vez. Ele não foi completamente implementado, sendo indicado também como trabalhos futuros, mas foi projetado para trabalhar utilizando o serviço *Reliable File Transfer*, abordado na subseção 3.3. Apesar de estar disponível somente a partir da versão 3.0 do *GlobusToolkit*, o *RFT* utiliza o *GridFTP* para fazer as transferências [3], o que o torna compatível com versões anteriores a 3.0.

A Figura 4.18 mostra a página inicial do *Broadcast portlet*, que é muito similar a do *GridFTP portlet*, com excessão dos *labels Source* e *Target* encontrados acima das *ListBoxes* e de somente um botão de cópia. Também podemos observar, pela mesma figura, que existe uma "barra de título" onde lemos "*GridFTP*", acima do *portlet*. Por estarem sob o mesmo *submenu*, os dois *portlets* são exibidos ao mesmo tempo, neste caso, o anterior foi minimizado, ficando somente sua barra de título como indicação.

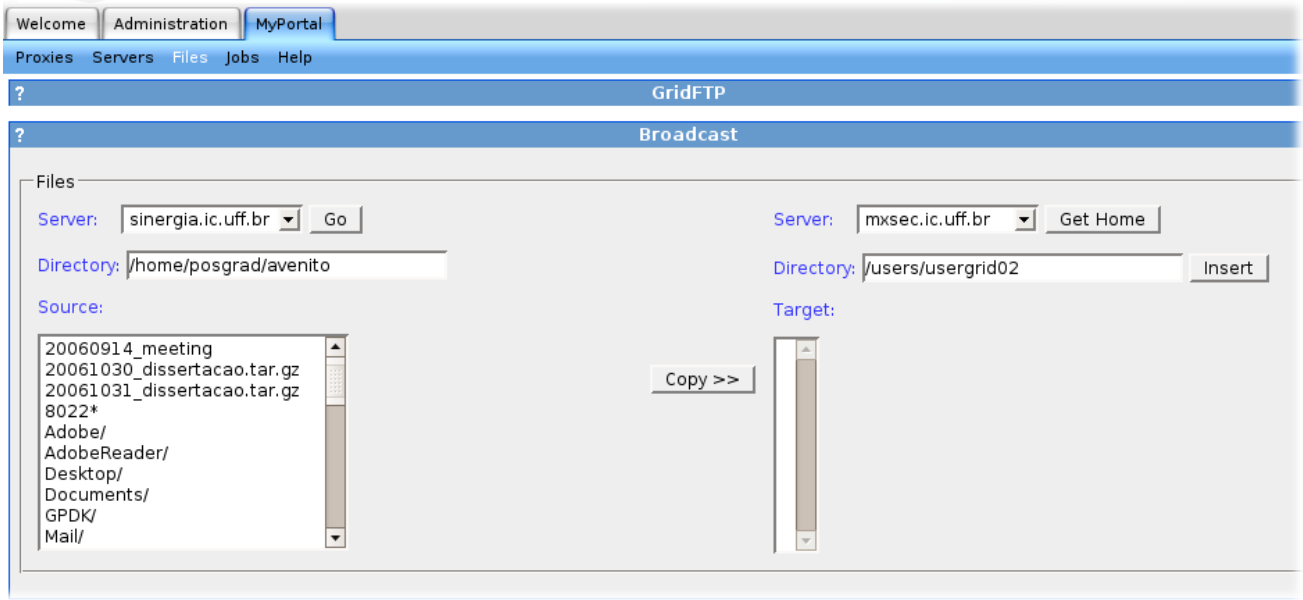


Figura 4.18: Página inicial do *Broadcast portlet*

Após selecionar o arquivo a ser copiado na *ListBox Source*, o usuário deve selecionar os destinos, servidor e caminho para o qual o arquivo será copiado, usando para isso os campos *Server* e *Directory* da direita da página (Figura 4.18). O botão "*Insert*" deve ser pressionado para que o destino seja relacionado na *ListBox Target*. Depois de ter selecionado todos os destinos, o botão "*Copy»*" pode ser pressionado para que a lista de origem e destino seja passada para o serviço *RFT*.

4.1.6.4 *Jobs*

Tendo transferido o *job* a ser executado para a máquina remota da grade pelo *GridFTP portlet*, o *Jobs portlet* deve ser utilizado para sua execução. Este se encontra sob o *submenu Jobs*. Podemos observar que a página inicial, mostrada pela Figura 4.19, está dividida em duas partes horizontais denominadas *Submit* e *Jobs*, cada uma delas com seus respectivos campos.

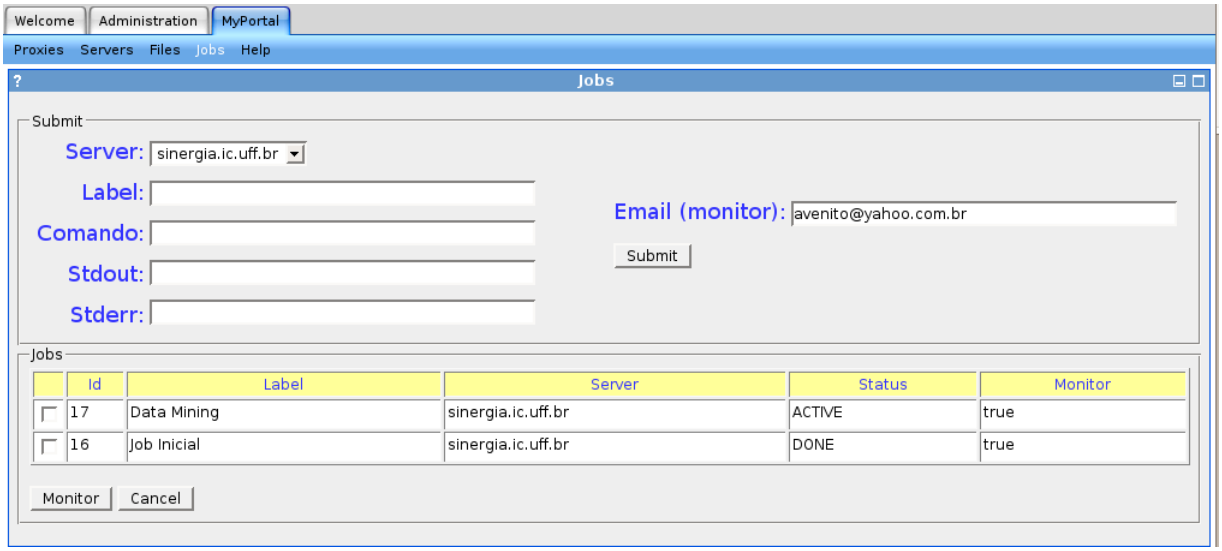


Figura 4.19: Página inicial do *Jobs portlet*

Campos de *Submit*:

- ***Server*** - Este campo apresenta uma lista dos servidores que estão cadastrados na conta do usuário do portal.
- ***Label*** - Nome pelo qual o *job* será referenciado.
- **Comando** - É neste campo que o comando referente ao *job* a ser executado deve ser inserido. Todo o caminho assim como os argumentos necessários devem ser inseridos.
- ***Stdout*** - Este campo aponta a saída do resultado do job caso o usuário deseje redirecioná-la para algum arquivo ou dispositivo.
- ***Stderr*** - Este campo aponta o arquivo de saída de erros.
- ***Email (monitor)*** - Cada *job* pode ser monitorado e ter seu estado informado através do envio de uma mensagem. Este campo, que traz o email do usuário do portal como padrão, exibe para qual endereço será enviada esta informação.

Depois dos campos devidamente preenchidos, o usuário deve submeter o *job* à execução acionando o botão *Submit*, que o submete à máquina selecionada pelo campo *Server*, utilizando o comando "*globus-job-submit*", e o insere em uma tabela que mantém seu estado para ser apresentado na segunda parte da página, denominada *Jobs*. Os campos da lista apresentada nesta segunda parte são:

- ***Id*** - Este campo apresenta a identificação do *job* na tabela *jobs* do banco de dados.
- ***Label*** - Nome do *job* que foi dado quando este foi submetido à execução.
- **Servidor** - É o nome do servidor ao qual o *job* foi submetido.
- ***Status*** - Apresenta o estado do job, sendo atualizado pelo *daemon* descrito na subsessão 4.1.6.7.
- ***Monitor*** - Este campo indica se o *job* está sendo monitorado pelo *daemon*. Caso indique *true*, um email será enviado para o endereço do usuário do portal quando houver mudança do estado do *job*.

Ainda pela Figura 4.19 podemos observar que dois *jobs* foram submetidos ao servidor de nome *sinergia.ic.uff.br*, estando o primeiro deles ainda em execução e o segundo já terminado. O botão "*Monitor*" marca o *job* para que este seja monitorado pelo *daemon*, alterando o campo "monitor" descrito pela Tabela 4.1.3. Para que um *job* seja retirado da lista e finalizado caso ainda não tenha terminado sua execução, o botão "*Cancel*" deve ser acionado.

4.1.6.5 *MPI LAM* e *MPICH*

O *MPI LAM portlet* funciona de maneira similar ao *Jobs portlet*, mas é específico para a submissão de aplicações que utilizam a biblioteca *MPI-LAM*. Antes das aplicações *MPI LAM* poderem ser executadas, é necessário inicializar o *daemon* responsável pela comunicação e execução remota nos servidores que serão utilizados, o que é feito pelo comando "*lamboot*". A Figura 4.20 mostra a página inicial do *portlet* onde podem ser vistos os seguintes campos:

- ***User*** - Este campo refere-se ao usuário do *proxy* que será utilizado para a conexão com os servidores. No caso do usuário possuir mais de um *proxy*, este deve escolher qual utilizará.

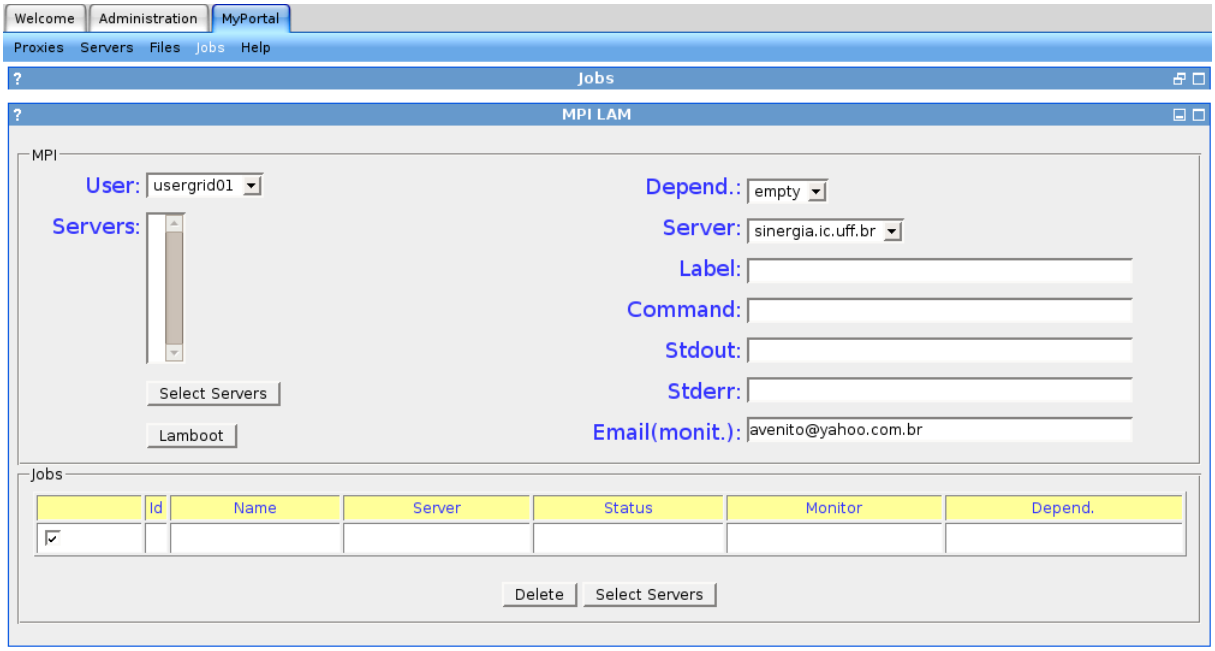


Figura 4.20: Página inicial do *MPI LAM portlet*

- ***Servers*** - Os servidores selecionados na página exibida quando o botão "Select Servers" é acionado, mostrada pela Figura 4.21, são listados neste campo, conforme pode ser visto pela Figura 4.22. Após serem selecionados, o botão "*Lamboot*" deve ser acionado para que o processo de *lamboot* seja inicializado. Esta ação coloca um *job* com o *label Lamboot* na lista da parte de baixo da página. O cancelamento deste *job* é feito pelo comando "*wipe*", responsável por parar os *daemons* já inicializados anteriormente.

- **Depend.** - Este campo é utilizado para condicionar o início da execução do *job* ao término de um outro *job*.

- ***Server***, ***Label***, **Comando**, **Stdout**, **Stderr** e **Email (monit.)** - Estes campos têm exatamente a mesma função que no *Job portlet*.

O diagrama de sequência do *portlet* pode ser visto pela Figura 4.23, onde vemos que a primeira ação do usuário é a seleção dos servidores que serão utilizados. Depois, o processo de *lamboot* é iniciado. O usuário pode submeter o *job MPI* logo em seguida, no entanto, este só será iniciado após o término do processo de *lamboot*. Esse recurso é particularmente interessante quando o "*lamboot*" consta de muitos servidores, levando muito tempo para ser finalizado. O *job* a ser executado é condicionado ao término deste.

O *MPICH portlet* foi projetado para a utilização do *MPICH-G2* sobre o *Globus* e não foi completamente implementado, sendo indicado como trabalhos futuros. A Figura 4.24

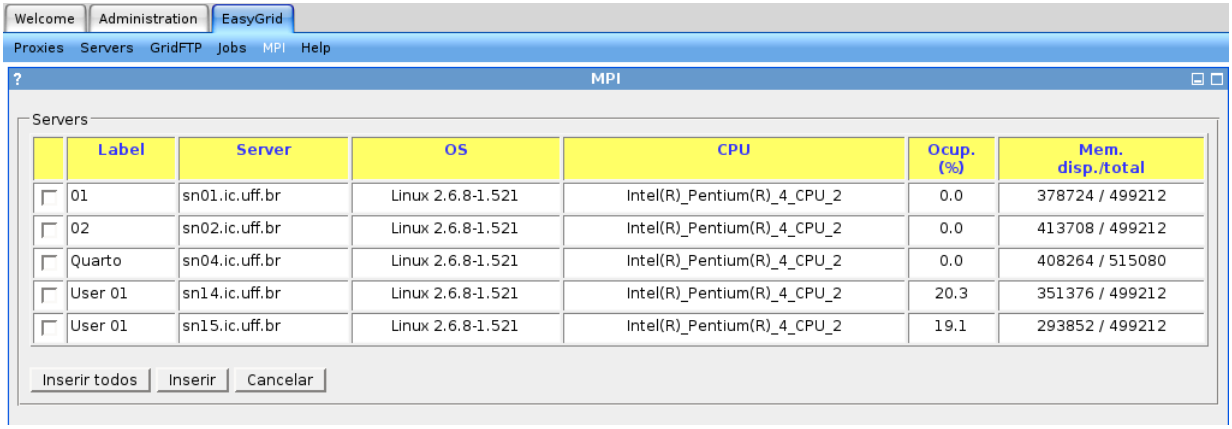


Figura 4.21: Seleção de servidores do *MPI LAM portlet*

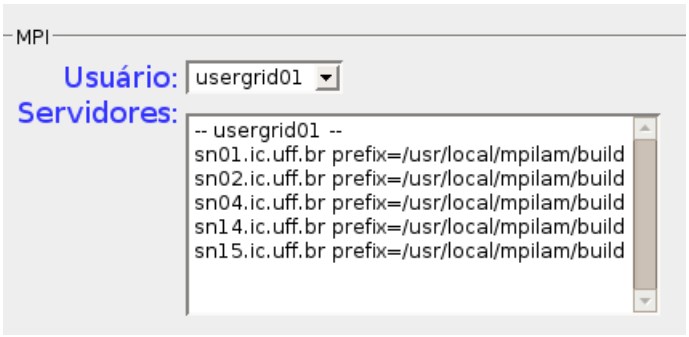


Figura 4.22: Servidores selecionados

mostra a tela inicial deste *portlet*.

4.1.6.6 *Help*

O *Help portlet* é na verdade somente uma descrição básica sobre os conceitos de funcionamento e da estrutura de uma grade computacional. O usuário também encontra uma descrição, passo à passo para que possa utilizar o portal.

4.1.6.7 *Job Monitor* e *Resource Monitor*

A submissão dos *jobs* pelo usuário do portal é controlada e monitorada através de uma tabela do banco de dados descrito nas subseções 4.1.6.4 e 4.1.6.5. Para que estas tabelas sejam atualizadas, o portal executa um *daemon* em segundo plano, em intervalos regulares, que acessa o banco de dados afim de saber quais tarefas estão sendo executadas e que necessitam ter seu estado consultado ao servidor que as executa. Todas as tarefas que estão sinalizadas para serem monitoradas, de todos os usuários, são lidas do banco e verificadas. Os casos de uso do *daemon* são mostrados pela Figura 4.25.

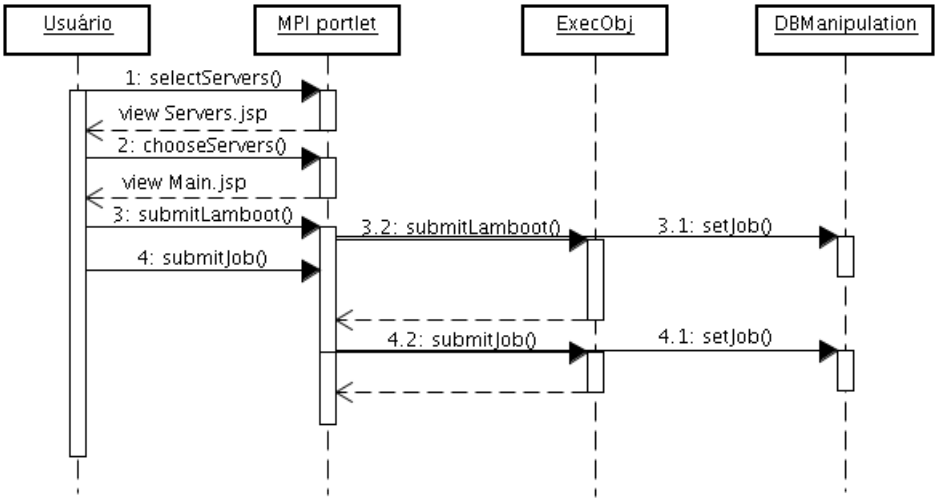


Figura 4.23: Diagrama de sequência do *MPI portlet*

A periodicidade com que o *daemon* executa seu ciclo deve ser ajustada dinamicamente de acordo com o número de *jobs* a serem consultados. Para esta implementação, foi estabelecido um tempo inicial de 30 segundos para um novo ciclo de consultas. O diagrama de sequência pode ser visto pela Figura 4.26, onde podemos observar que a primeira coisa a ser feita é a consulta ao banco de dados, depois o servidor responsável pela tarefa a ter seu estado atualizado é consultado através de uma chamada ao sistema operacional, cujo retorno é então atualizado na tabela correspondente. Dependendo do estado anterior, ou seja, se houve mudança em relação ao estado anterior, um *email* é enviado para o usuário informando sua mudança. Após o tempo de espera ajustado, em que o *daemon* encontrasse em repouso, o ciclo se reinicia.

A interação do *Job monitor* já é um exemplo de integração de um aplicativo externo com os resultados que o portal produz, uma vez que apesar de ter sido escrito em Java, em nenhum momento há a necessidade de comunicação direta com os *portlets* implementados.

Um outro *daemon*, chamado *Resource Monitor*, foi implementado para atualizar as informações menos dinâmicas dos servidores que compõe a grade, como a versão do sistema operacional ou o modelo da *CPU*. O intervalo em que essas verificações são feitas é de 12h. Como há a necessidade de autenticação na conexão com o servidor para que essas informações sejam obtidas, este *daemon* utiliza os *proxies* que estiverem ativos nas contas dos usuários. As informações sobre o nome do servidor e que *proxy* utilizar, são retiradas da tabela *servers*, no entanto, os dados atualizados ficam armazenados na tabela *default_servers*. Para evitar que o mesmo servidor seja consultado mais de uma vez caso esteja na lista de mais de um usuário, o campo *updating* é colocado com valor igual a "1"

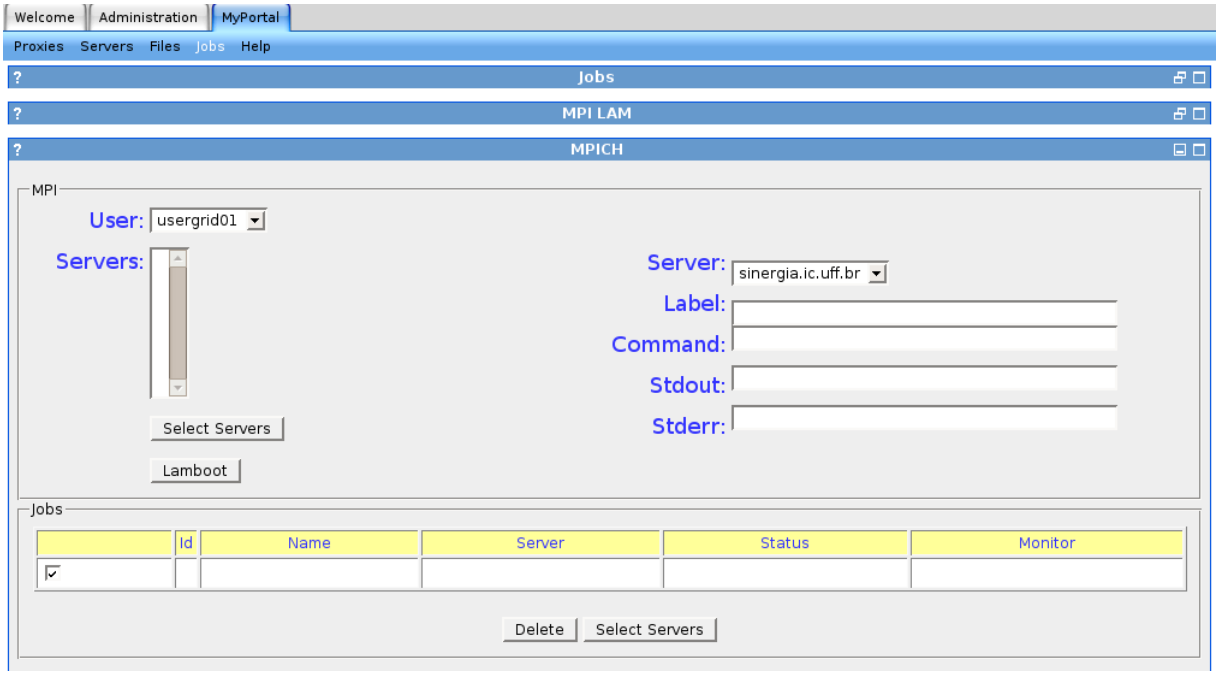


Figura 4.24: Página inicial do *MPICH portlet*

no início da verificação do *daemon*, sinalizando que este ainda deve ser atualizado, e é passado para "0" pela primeira atualização que for feita para aquele servidor, assim, o número máximo de consultas realizadas é igual ao número de servidores diferentes cadastrados no portal.

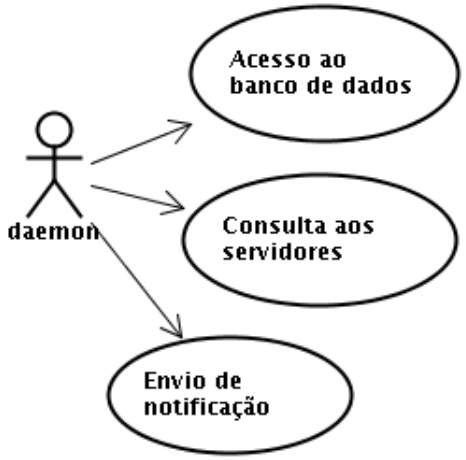


Figura 4.25: Casos de uso do *daemon "Job Monitor"*

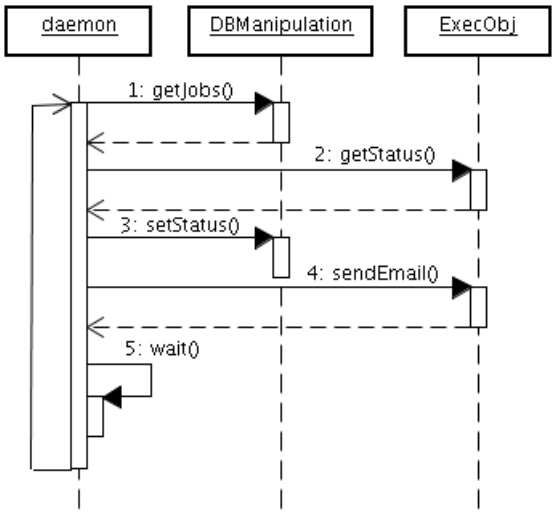


Figura 4.26: Diagrama de sequência do *daemon "Job Monitor"*

Capítulo 5

Conclusão e trabalhos futuros

A computação paralela e distribuída é uma tendência em função da evolução tecnológica e redução dos preços da tecnologia de sistemas computacionais. Dia após dia, os computadores pessoais tornam-se cada vez menores fisicamente e com poder de processamento e armazenamento cada vez maiores.

Com a distribuição física de recursos computacionais, crescem também os desafios tecnológicos para sua integração e utilização de forma eficiente e transparente, trazendo seus benefícios aos usuários sem conhecimentos específicos de computação. No entanto, atualmente a operação e utilização de uma grade computacional ainda requer conhecimentos específicos sobre seu funcionamento, dificultando a utilização por parte de comunidades científicas que necessitam de poder computacional, seja de processamento ou armazenamento de dados.

O *MyEasyGridPortal* contribui como ferramenta de gerência básica para acesso a grades, implementado de forma que o resultado de sua operação, armazenando em banco de dados e interagindo diretamente com o sistema operacional da máquina que o hospeda, possa ser utilizado por outros aplicativos, como por exemplo um escalonador de tarefas, de forma totalmente independente da linguagem em que foi escrita. Ainda há muito o que se fazer até que a total transparência e simples utilização da grade computacional seja atingida. Abaixo são citadas algumas sugestões sobre aspectos que podem ser implementados.

Durante a pesquisa realizada para este trabalho e também ao longo de sua implementação, várias outras funcionalidades foram encontrados. Devido aos prazos para sua conclusão e objetivando ter uma ferramenta prática e funcional ao término, alguns desses aspectos puderam ser implementados e outros não.

O principal objetivo de um portal para acesso a grades é facilitar a utilização destas, ocultando os detalhes de sua operação ao ponto de permitir que usuários com o mínimo ou nenhum conhecimento específico de grades possa tirar proveito de sua utilização, conforme abordado em capítulos anteriores. As interfaces gráficas, sem dúvida nenhuma, facilitam muito a utilização de sistemas computacionais, sendo responsáveis inclusive pela popularização dos computadores pessoais, desta forma, as informações que podem ser exibidas de forma gráfica normalmente são preferidas pelos usuários. Dos "trabalhos futuros" sugeridos, muitos se relacionam com essa abordagem de visualização gráfica. A estrutura de um portal, por sua própria definição, permite que vários aplicativos, em forma de *portlets*, sejam adicionados e que cada usuário personalize seu espaço, não só na sua forma de exibição mas também quais *portlets* utilizar. Isso permite que *portlets* especializados sejam utilizados por determinados grupos que têm interesses específicos como por exemplo aplicativos de simulações ou manipulação de imagens.

A forma como o *MyEasyGridPortal* foi implementado, permite que outros portlets utilizem sua estrutura sem a necessidade de comunicação direta com os *portlets* do próprio portal, o que modulariza e torna os novos desenvolvimentos independentes. O fato das credenciais, ou *proxies* dos usuários serem armazenados sob o sistema operacional da máquina que está rodando o portal, além de todos os dados dos servidores e *jobs* que estão sendo executados estarem disponíveis em tabelas de banco de dados, faz com que qualquer outro aplicativo, como por exemplo um escalonador de tarefas que tome como base informações do estado dos servidores da grade, possa ser escrito em qualquer linguagem de programação diferente da que foi usada para a implementação do portal.

Um recurso que aumentaria a flexibilidade e praticidade em se utilizar os comandos já implementados por um middleware para o acesso e operação da grade, é o mapeamento, através de arquivos em *xml*, relacionando o comando dado pelo portal, com o comando que realmente será chamado pelo sistema operacional, assim, por exemplo, supondo que o portal utlize o comando "*getCredential*" para solicitar um *proxy* ao *MyProxy Server*, e que dois *middlewares* diferentes, genericamente denominados "A" e "B", utilizem comandos diferentes para a mesma função, um arquivo em *xml* conforme mostrado pela Figura 5.1 poderia mapear qual comando realmente utilizar na realização da função desejada.

Pela Figura 5.1 observamos que o comando "*getCredential*" é mapeado para "*commandAtoGetProxy*" para o *middleware* A, e "*commandBtoGetProxy*" para o *middleware* B.

Pelo exposto, trabalhos futuros que visem a especialização da utilização da grade po-

```
1. <portalCommand>
2.   <command>getCredential</command>
3.   <middleware>A</middleware>
4.   <osCommand>commandAtoGetProxy</osCommand>
5. </portalCommand>
6.
7. <portalCommand>
8.   <command>getCredential</command>
9.   <middleware>B</middleware>
10.  <osCommand>commandBtoGetProxy</osCommand>
11. </portalCommand>
```

Figura 5.1: Arquivo *xml* de configuração sugerido

dem ser agregados de forma simples, uma vez que toda estrutura básica necessária já está implementada, criando inclusive o conceito de serviços em que determinadas máquinas já possuam os programas exigidos para executar uma aplicação específica, como uma simulação por exemplo, e que o *portlet* agregado só transfira os dados necessários ao processamento em questão. Este tipo de especialização facilita muito a vida do usuário leigo em computação que precise utilizar a grade. Pela estrutura utilizada, algoritmos podem consultar as tabelas de dados referentes aos servidores pertencentes àquele usuário e tomar decisões de quais máquinas utilizar segundo critérios pré estabelecidos como carga de processamento destas máquinas ou quantidade de memória disponível.

Para a utilização do portal, o primeiro *portlet* a ser utilizado é o dos *Proxies*, ou seja, antes de qualquer interação com a grade, um *proxy* tem que ser criado para garantir o acesso seguro aos recursos oferecidos. Também por uma questão de segurança, os *proxies* tem seu tempo de validade limitados, sendo 12h o prazo padrão quando não é especificado nada diferente disso, conforme abordado na subseção 3.2. Visualizar de forma gráfica o tempo restante de validade do proxy, facilitaria o usuário tomar uma decisão sobre renová-lo antes de utilizar a grade. Adicionar um recurso de revalidação automática, preocupando-se em armazenar a senha do usuário de forma segura no banco de dados também pode ser implementado e facilitaria sua utilização.

Os dados visualizados pelo *Server portlet* também podem ser mostrados de forma gráfica, assim, a ocupação de memória e carga de processamento da *CPU* são mais facilmente interpretados. Filtros que ordenem os servidores por parâmetros escolhidos pelo usuário também facilitam a visualização e decisão de qual máquina utilizar, uma vez que os aplicativos possuem necessidades diferentes de processamento e memória disponível. Uma representação gráfica da posição física do servidor também pode ser utilizada, baseando-

se em mapas geográficos reais como os disponíveis pelo *Google Maps*, assim, por exemplo, um servidor pode ser marcado sobre o mapa em cores diferentes de acordo com seu estado atual de disponibilidade de recursos. Agregar outros parâmetros de desempenho ou de custo aos servidores também ajudariam os algoritmos que avaliem em qual máquina devem alocar um *job*.

A transferência de arquivos pode ser facilitada pela utilização do portal. O processo, ou *job* a ser executado pela grade deve ser transferido para a máquina na qual será executado, assim como os seus parâmetros e dados de entrada, se necessários. Caso seja necessário a transferência para mais de uma máquina, deve ser implementado uma forma de se verificar se estas máquinas possuem e fazem parte do mesmo *NFS*, o que torna necessária a transferência para somente uma delas. Para saber se uma máquina possui algum diretório montado em *NFS*, e em qual servidor o *NFS* está montado, algumas campos na tabela "*default_servers*" podem ser criados e preenchidos de forma manual uma vez que a inserção de mais um servidor é feita somente pelo administrador do portal, ou, esses dados podem ser obtidos de forma dinâmica através do arquivo *fstab*. A linha em negrito abaixo foi retirada do arquivo citado e mostra que o diretório */home* da máquina "abacate" (abacate:/home), está montado como "*/home*" localmente à máquina a que o arquivo *fstab* pertence.

abacate:/home /home nfs defaults 0 0

Uma tabela que liste as transferências pode ser implementada da mesma forma que a tabela de *jobs* do *Job portlet* e do *MPI LAM portlet*, com o objetivo de permitir um encadeamento da execução das transferências e da submissão de tarefas. A execução de um processo em segundo plano pelo sistema operacional retorna uma identificação do processo, também conhecida como *PID*, como saída. Este *PID* pode ser inserido em uma tabela do banco de dados que tem seu estado atualizado regularmente por um *daemon* criado para esta função, desta forma, uma submissão de *job* pode ser condicionada ao final de uma transferência de arquivo que tenha sido colocado em segundo plano.

A execução de um job, ou processo, pode levar horas, ou até dias, assim, o sistema de notificação da mudança do estado das tarefas em execução através do envio de e-mail é bastante útil para seu acompanhamento, o usuário é notificado praticamente em tempo real. Além deste acompanhamento automático por envio de e-mail, o portal também oferece o acesso remoto para este mesmo tipo de monitoramento através de dispositivos

móveis, o que se diferencia do primeiro basicamente pela interface, já que estes dispositivos em geral possuem interface limitada em tamanho físico e capacidade de recursos gráficos. A implementação deste recurso no *MyEasyGridPortal* é feito basicamente através de uma consulta ao banco de dados a partir da entrada do login do usuário do portal, não exigindo nenhuma senha uma vez que o acesso não permite nenhuma modificação no estado das tarefas. As funcionalidades deste acesso remoto podem ser estendidas para permitir maior interação com o portal, como por exemplo permitir a renovação de certificados digitais, ou a submissão de jobs previamente colocados em uma fila de espera aguardando um simples comando para serem iniciados.

Uma outra facilidade que o portal deve oferecer é, através do acesso por dispositivos portáteis, o monitoramento de todas as tarefas que estejam na lista do usuário. O acesso através de dispositivos móveis é destacado pelo fato destes dispositivos, em geral, possuírem uma interface com o usuário limitada, normalmente um display pequeno, sendo necessário que o portal identifique-os e utilize páginas apropriadas, como *wml* ou *HTML* limitado.

As interfaces gráficas são preferidas pelos usuários em geral, como já foi abordado, existindo uma tendência natural para as linguagens gráficas. Neste sentido, a utilização da grade através de gráficos que apresentem uma sequência de execução de ações como um fluxograma é bastante útil. Este tipo de execução é conhecida como *workflow* e pode ser implementado com a ajuda de ferramentas relativamente simples.

A implementação do *MyEasyGridPortal* utilizando a tecnologia *portlet* atingiu os objetivos incialmente almejados. A utilização do ambiente *Web* como interface gráfica *frontend*, chamando em *background* os comandos de um *middleware* totalmente independente da linguagem de programação em que foi escrito, se mostrou bastante funcional e flexível, aproveitando todo o desenvolvimento realizado até hoje nas duas áreas, acesso *Web* e *middleware* para grades computacionais.

Referências

[1] http://www.servogrid.org/slide/iSERVO/NMI/Year3AnnualReport/NMI-OGCE-Annual-Report-2006.pdf.

[2] http://www.gridsphere.org/gridsphere.

[3] http://www.globus.org/toolkit/data/rft/.

[4] Allcock1, W., Kettimuthu1, J. B. R., Link1, M., Dumitrescu, C., Raicu, I., and Foster, I. The globus striped gridftp framework and server. http://www.globus.org/alliance/publications/papers/gridftp$_final.pdf$.

[5] Andronico, G., Ardizzone, V., Barbera, R., Catania, R., Falzone, A., Giorgio, E., Rocca, G. L., Re, G. L., Passaro, G., Platania, G., Pulvirenti, A., and Rodolico, A. The genius grid portal: Architecture and applications. www.teragrid.org/programs/sci$_gateways/docs/genius_position_paper.doc$.

[6] Cirne, W., Brasileiro, F., Suavé, J., Andrade, N., Paranhos, D., Neto, E. S., and Medeiros, R. Grid computing for bag of tasks applications. *Third IFIP Coference on e-Commerce, e-Business and e-Goverment* (2003), 591–609.

[7] Deitel, H. M., and Deitel, P. J. *JAVA Como programar*, 4th ed. Bookman, 2003.

[8] Developer Network (SDN) Sun. The java servlet api white paper. http://java.sun.com/products/servlet/whitepaper.html.

[9] DOE Science Grid research and development. Grid portal development kit. http://doesciencegrid.org/projects/GPDK/.

[10] Ferreira, E. C. Ferramentagridlncc. http://grade01.lncc.br/~lrodrigo/arquivos/referencias_00/sistemaDistribuido/instalacoes/IntroConstGradesLNCC-1-2.pdf.

[11] Foster, I., Kesselman, C., and Tuecke, S. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications 15*, 3 (August 2001), 200–222.

[12] Globus Alliance. About the globus toolkit. http://www.globus.org/toolkit/about.html.

[13] Hepper, S., and Lamb, M. Best practices: Developing portlets using jsr 168 and websphere portal v5.02. http://www-128.ibm.com/developerworks/websphere/library/techarticles/0512_hepper2/0512_hepper2.html.

[14] IBM. *IBM WebSphere Portal V4.1 Handbook Volume 2*. e-book, 2003.

[15] Java Community Process. Jsr168 portlet specification. http://jcp.org/aboutJava/communityprocess/final/jsr168/index.html.

[16] Martins, A. Autoridade certificadora para acesso seguro. www.lockabit.coppe.ufrj.br/downloads/academicos/CA.pdf.

[17] National Grid Service. The globus x509 proxy certificate. http://www.grid-support.ac.uk/content/view/116/68/.

[18] Netto, G. O. Infra-estrutura de chaves públicas certificação digital. http://www.serpro.gov.br/publicacao/tematec/1999/ttec46.

[19] Network Working Group. Internet x.509 public key infrastructure operational protocols: Ftp and httpe. http://www.ietf.org/rfc/rfc2585.txt.

[20] Patil, S. What is a portlet. http://www.onjava.com/pub/a/onjava/2005/09/14/what-is-a-portlet.html.

[21] Peixoto, D. M., Barros, L. C., de Albuquerque, M. P., and de Albuquerque, M. P. Instalação e configuração do globus toolkit para computação em grid. http://mesonpi.cat.cbpf.br/grid/PDF/nt01203.pdf.

[22] Pitanga, M. Computação em grade - uma visão introdutória. http://www.clubedohardware.com.br/artigos/124.

[23] Prentice Hall and Sun Microsystems. *Overview of Servlets and JavaServer Pages*, 2th ed. e-book.

[24] Rodrigues, H., Trannin, H., Sena, A., and Rebello, V. Usando grades computacionais para a avaliação de heurísticas de escalonamento de tarefas. *VI Workshop em Sistemas Computacionais de Alto Desempenho (WSCAD 2005)* (2005), 218–221.

[25] The Globus Alliance. Myproxy. http://dev.globus.org/index.php?title=MyProxy.

[26] The National Center for Supercomputing Applications (NCSA). Common gateway interface. http://hoohoo.ncsa.uiuc.edu/cgi/intro.html.

[27] Thomas, M. NPACI AHM03 Tutorial 10 Grid Computing Portals. https://portal.tacc.utexas.edu/userguides.html.

[28] UK Lancaster University. What is grid computing. http://www.lancs.ac.uk/postgrad/jains1/what_is_Grid_Computing.html.