Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Departamento de Computação

Programa de Pós-Graduação em Ciência da Computação

Um Portal para o P2PGrid

WILLIAN BORGES LISBOA

São Carlos – SP

Julho 2007

Um Portal para o P2PGrid

Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Departamento de Computação

WILLIAN BORGES LISBOA

UM PORTAL PARA O P2PGRID

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, para obtenção do título de mestre em Ciência da Computação

Orientação: Prof.Dr. Luis Carlos Trevelin

SÃO CARLOS - SP
2007

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

L769pp

Lisboa, Willian Borges.
Um portal para o P2PGrid / Willian Borges Lisboa. -- São Carlos : UFSCar, 2010.
121 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2007.

1. Ciência da computação. 2. Portais da Web. 3. Redes de computadores. I. Título.

CDD: 004 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Um Portal para o P2PGrid"

WILLIAN BORGES LISBOA

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Membros da Banca:

Prof. Dr. Luis Carlos Trevelin
(Orientador – DC/UFSCar)

Prof. Dr. Antonio Francisco do Prado
(DC/UFSCar)

Prof. Dr. Márcio Merino Fernandes
(UNIMEP)

São Carlos
Agosto/2007

A minha esposa que sempre me apoiou nessa jornada!

AGRADECIMENTOS

Agradeço a Deus, por me dar vida, saúde e coragem para transpor barreiras e, ainda, dar-me o norte do entendimento de que o sucesso não vem das conquistas, mas sim das batalhas que levam a estas.

A minha esposa, minha gratidão eterna, pelo apoio, compreensão e por ceder, em beneficio da realização deste sonho, parte do precioso tempo que, por natureza, lhe seria de direito.

Ao meu orientador Trevelin que soube como ninguém me nortear em busca dos caminhos do conhecimento. Sobe ainda aceitar minhas condições peculiares de distância e dedicação para com a pesquisa.

Ao meu amigo Ruy Muniz que me apoiou em busca do conhecimento.

Aos amigos, Roberto, Edmilson, Marcelo e Leo que me ajudaram sempre que solicitados.

A Cristina da secretária da pós-graduação por me ajudar nos momentos que mais precisei de seu apoio e compreensão.

Agradeço ainda aos professores Zorzo, Hélio e Prado que deixaram suas marcas de conhecimento em minha memória.

Agradeço ainda a professor Arlete Borges e Airam Paz pelos momentos perdidos na revisão deste trabalho.

RESUMO

No atual contexto da informática, tecnologias voltadas para aplicações que visam ao poder computacional levaram ao desenvolvimento de Sistemas Distribuídos, cooperativos e compartilhados, também conhecidos como grades computacionais. O crescimento acelerado das redes de computadores e dos meios de comunicação que as interligam favorecem o uso desses sistemas que visam a ocupar o tempo ocioso dos recursos das máquinas que a elas pertencem, com a finalidade de prover um alto poder computacional distribuído. Em face disso, diversas pesquisas nas mais diferentes áreas da Ciência da Computação têm sido estudadas e implementadas exaustivamente, uma delas é o P2PGrid (1) o qual se baseia no desenvolvimento de uma infra-estrutura de suporte para a criação de Grades Computacionais através do compartilhamento de recursos, utilizando mecanismos que tornem transparente a existência dessa infra-estrutura.

A utilização de Portais como mecanismos mediadores entre os usuários da grade e a própria grade vem se tornando cada vez mais comum, preservando a transparência e integridade da mesma. Este trabalho visa o desenvolvimento de um gerador de Portais baseado em componentes utilizando-o para a construção de um Portal para o P2PGrid que atenda aos padrões já existentes de Portais, além de trazer novas abordagens e conceitos para a gerência, desempenho e segurança na submissão de *jobs* no P2PGrid. Em adição ao portal, visando promover uma grade computacional gerenciável e segura, este trabalho apresenta a implementação de um mecanismo de credenciamento de nodos e usuários e um mecanismo de autenticação de nodos distribuídos. Tais propostas visam a aumentar o controle e o nível de segurança na utilização do P2PGrid, bem como facilitar o seu acesso e utilização.

ABSTRACT

In the ongoing informatics context, applications oriented technologies that envisage computational power is leading to the Development of distributed cooperative and shared systems known as Computational Grid. The accelerated growing of Computer networks and its interconnecting communications means make viable the use of such systems that aims to occupy its machines idle time with the objective of providing a high distributed computational power.

Facing this, several researches in Computer Sciences has been conducted and developed exhaustively, one of it is the P2PGrid (1) which is based on a infrastructure Development to support computational grid creation through distributed resource sharing via mechanisms that make this infrastructure transparent.

As computational grid reach its objectives of high computational power through Computer networks idle resource sharing , the need of a high computational power at low costs stimulates new Research projects and computational systems Development to attend this definition.

The use of Portals as middleware mechanisms between the User and Grid is becoming ever more common, preserving the grid transparency and integrity.

This work aims to create a Portal to the P2P Grid environment that could meet the existing standards besides bringing new approaches and concepts for *job* submission Management, performance and security. Also, a users credentialing and authenticating mechanisms implementation is proposed. Such proposals aim to power-up the control and security levels when using the P2P Grid as well as to facilitate its Access and utilization.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 GRIDS, PORTAIS E P2PGRID

Para atender a demanda de processamento e recursos das aplicações com elevado grau de complexidade foram criados diversos conceitos dentre eles destacamos a computação distribuída e paralela de alto desempenho.

Inicialmente a solução adotada foi a utilização de máquinas paralelas com multiprocessadores as quais oferecem alto desempenho, contudo, possuem custo elevado, e deficiência em flexibilidade no que tange escalabilidade e configuração. Atualmente a solução adotada é obter alto desempenho através do gerenciamento do poder computacional de vários componentes de uma rede interligada. Neste contexto temos as aplicações baseadas em aglomerados (*clusters*) e as grades computacionais.

As grades computacionais possuem como característica implícita compartilhar a capacidade de processamento de recursos computacionais, geograficamente distribuídos, através de uma rede interligada.

Neste contexto, apresentamos o ambiente P2PGrid (grade computacional *peer-to-peer*), cujo desenvolvimento difere das características observadas em trabalhos relacionados ao mesmo assunto.

Com sua estrutura básica implementada, percebe-se a necessidade de novos recursos e implementações voltados para maior segurança e utilização da grade, de modo a promover a acessibilidade e ubiqüidade a que a computação distribuída se propõe. Este trabalho foca-se nas grades computacionais e na criação de Portais para elas, abordando o credenciamento de nodos e usuários, bem como a implementação de um mecanismo de autenticação e atualização de nodos distribuídos, cuja utilização não se restringe simplesmente à computação paralela e distribuída.

1.2 MOTIVAÇÃO

A popularidade da Internet e o avanço tecnológico das redes de alta velocidade, aliados ao custo muito mais acessível de computadores se comparados com os supercomputadores, tornam a visão de computação em grade extremamente atrativa.

Esse novo ambiente de programação possibilita ao usuário utilizar um sistema de alto poder computacional, formado por diversos recursos distribuídos geograficamente, através de uma relação de compartilhamento entre os usuários participantes da grade.

A visão original de grade computacional estabelece uma metáfora entre a Rede Elétrica (The Electric Grid (2)), que disponibiliza energia elétrica sob demanda e esconde do usuário detalhes como a origem da energia e a complexidade da malha de transmissão e distribuição. Ou seja, simplesmente conectando um aparelho na tomada, é possível receber energia. A grade de computadores, portanto, seria uma rede na qual o indivíduo se conecta para obter poder computacional (ciclos, armazenamento, software, periféricos etc.).

Tem-se como objetivo aproveitar ao máximo o poder computacional disponível através da colaboração entre os participantes. Com base no relacionamento dos usuários da grade, constrói-se um grande e poderoso sistema computacional, que engloba recursos disponíveis por cada usuário e permite a sua utilização de maneira global e coordenada entre os usuários, obtendo assim a vantagem de aproveitar a capacidade computacional existente.

Assim, o conceito principal de uma grade é o compartilhamento de recursos coordenados entre coleções dinâmicas de indivíduos, instituições e recursos, definidos por regras de compartilhamento as quais são chamadas de organizações virtuais (Virtual Organizations – Vos (2)).

Trata-se não somente de trocas de arquivos entre as VOs, mas no acesso direto a computadores, software, dados e serviços, além da ligação direta com diversos

recursos eletrônicos através da Internet a fim de tornar possível a colaboração entre os usuários.

Porém, o compartilhamento deve ser altamente controlado, com provedores de recursos e usuários claramente definidos, permitindo que cada usuário acesse apenas o que lhe é possível e definindo condições para que o compartilhamento possa ocorrer.

Dessa forma, devem existir mecanismos para expressar uma política de identificação do usuário que requisita um determinado recurso e determinar se uma operação é consistente com o relacionamento de compartilhamento (autorização).

Para isso, padrões e conceitos estão sendo definidos, a fim de proporcionar que ambientes de computação em grades sejam bem desenvolvidos e possam ser amplamente utilizados.

O desenvolvimento de Portais para grades proporciona as seguintes características:

- Acesso ubíquo a grade via Navegador;
- Acesso unificado a todos os recursos;
- Localização e geração automática de *jobs*;
- Contabilização a submissão de *jobs*;
- Parametrização centralizada/colaborativa das execuções;
- Submissão e acompanhamento dos *jobs*;
- Acesso às novas tecnologias da grade.

Tais características foram contempladas na construção do Portal proposto para o P2Pgrid. Nota-se ainda, que o desenvolvimento e o aprimoramento dos Portais para as grades computacionais tendem a nivelar e facilitar a utilização dos recursos da grade tornando sua utilização muito mais acessível aos usuários em geral.

1.3 METODOLOGIA

Uma grade pode ser vista como um ambiente colaborativo de computação em que, os usuários participantes cedem recursos próprios, que podem ser utilizados globalmente por uma variedade de serviços distintos e, em troca, utilizam os recursos disponíveis oferecidos pelos demais usuários.

O estudo de metodologias e tecnologias para desenvolvimento do sistema foi analisado de forma criteriosa e contínua. Neste contexto, serão apresentadas nesta seção, algumas metodologias, técnicas e tecnologias pesquisadas e utilizadas no projeto.

1.3.1 UML (UNIFIED MODELING LANGUAGE)

Com o avanço da computação e a integração dos sistemas distribuídos, observou-se que a complexidade crescia de forma geométrica, forçando assim a comunidade de pesquisadores da computação a estabelecer um novo paradigma de modelagem de sistemas.

Os Pesquisadores começaram a adotar os conceitos da orientação a objetos como uma forma de reduzir o esforço empregado na análise e na modelagem de sistemas. Porém, foram desenvolvidas várias metodologias, cada uma com seu ponto forte e seu ponto fraco, mas sem um padrão estabelecido.

Foi realizado um enorme esforço para que se conseguisse especificar um padrão. Em 1997 a OMG (*Object Management Group* (3)) reconheceu um padrão, sendo este desenvolvido através da integração de três das principais metodologias aplicadas no mercado, o resultado foi a Linguagem UML.

Tratando-se de uma linguagem que utiliza a orientação a objetos de forma concisa e completa esta se encaixa e integra-se perfeitamente com a linguagem de programação e de implementação Java (4) e XML (5).

É importante destacar que a linguagem UML, além de suportar diversas tecnologias de implementação, oferece suporte completo à modelagem e à

especificação de sistemas para ambientes distribuídos, como o caso do sistema proposto.

1.3.2 SQL (STRUCTURE QUERY LANGUAGE)

A linguagem de busca estruturada SQL (6) foi desenvolvida pela IBM (7) em meados dos anos 70 com o nome de SEQUEL. Com o progresso da linguagem e com as constantes atualizações das versões, em 1980 recebeu o nome SQL.

No atual contexto, a linguagem SQL é a mais importante linguagem de manipulação de dados, isto se deve ao fato de ter sido padronizada pela ANSI (*American National Standards Institute* (8)). Esta linguagem é utilizada na como mediadora das informações contidas no banco de dados de SQL e o sistema.

1.3.3 JAVA

A Linguagem Java foi desenvolvida pela *Sun Microsystems* (9) no ano de 1990 no intuito de estabelecer funcionalidade controlada por computador para equipamentos eletrodomésticos, porém, aquele ainda era um projeto avançado demais para aquela época.

Em face disto, passou-se a explorar as áreas de linguagem de programação, em que a Linguagem Java demonstrava ter uma vantagem principal sobre as outras: era multiplataforma. O termo multiplataforma deve-se ao fato de que um programa nesta linguagem pode ser escrito e compilado em qualquer tipo de sistema operacional, bastando para isto uma máquina virtual, denominada JVM (*Java Virtual Machine* (10)).

Devido à grande aceitação da linguagem, ainda mais se tratando da crescente expansão dos sistemas operacionais de código abertos(*open source*) (11) a Linguagem Java tomou diversos rumos no que diz respeito a seus paradigmas de programação e, os utilizados neste projeto, estão citados a seguir:

Applets (12): Programas *client-side* desenvolvidos para gerar funcionalidade em páginas WEB (13), utilizando classes Java da biblioteca *java.awt* (14).

JavaScript (15): Desenvolvida pela *Netscape* (16) também para colocar funcionalidade nas páginas *web*, porém não manipula classes da linguagem Java.

Java-Servlet (17): Programas *server-side* desenvolvidos para geração dinâmica de conteúdo de páginas *web*, o código HTML (18) é colocado dentro do código Java. Os servlets são códigos Java que rodam no servidor e como resposta a uma requisição do cliente geram código HTML dinamicamente e o devolvem como resposta ao cliente. São capazes de receber parâmetros através de dados vindos de um formulário HTML, ou até mesmo via URL e, através do processamento desses dados ou de consultas ao banco de dados uma página HTML de resposta é gerada, personalizada de acordo com cada requisição. Os Servlets podem ser escritos para trabalhar com diversos tipos de protocolo de comunicação, como SMTP (19), HTTP (20) e IRC (21). São bastante semelhantes ao CGI (Common Gateway Interface) (22), mas são preferíveis por possuírem as seguintes vantagens sobre CGI:

- Servlets não executam em um processo separado.
- Servlets são mantidos na memória entre uma requisição e outra.

Há apenas uma instância carregada de cada Servlet no container que serve todas as requisições para ele concorrentemente.

JSP (*Java Server Pages* (23): Programas *server-side* desenvolvidos para geração de conteúdo dinâmico de páginas *web*, o código Java é colocado dentro do código HTML. Esta tecnologia é usada para prover conteúdo dinâmico para o usuário, utilizando o gerenciamento de dados e a lógica de processos no lado servidor. JSP faz parte da plataforma J2EE (24) e, juntamente com *servlets* e *java beans*, pode ser usada para desenvolver aplicações *web* eficientes, escaláveis e seguras. Tem a vantagem da portabilidade de plataforma, podendo ser executada em vários sistemas operacionais, e da compilação das páginas, permitindo que elas possuam um maior desempenho. Além disso, essa tecnologia permite separar a programação lógica (parte dinâmica) da programação visual (parte estática), facilitando o desenvolvimento de aplicações mais robustas, de maneira que o desenvolvedor e o designer possam trabalhar no mesmo projeto, mas de forma

independente. Uma característica importante da tecnologia *jsp* é a produção de conteúdos dinâmicos que podem ser reutilizados.

1.3.4 ESPECIFICAÇÃO JSR 168

Especificação desenvolvida pelo JCP(*Java Community Process*) (25) que objetiva principalmente buscar a interoperabilidade de portais e *portlets* (26), trazendo benefícios, como a simplificação da utilização e desenvolvimento de *portlets*, bem como sua utilização em servidores de portais J2EE. Esta especificação define um conjunto de APIs para *Portlets* baseado em J2EE, cujos principais componentes são (*portal server*, *portlet containers* e *portlets*) descritos na Especificação J2EE (26).

1.3.5 OS MODELOS DE PROJETO

O êxito no desenvolvimento de aplicações orientadas a objetos está diretamente relacionada à arquitetura que será utilizada em sua construção. Uma organização em camadas é a chave para a independência entre os componentes, o que favorece as aplicações no que tange desempenho, portabilidade, escalabilidade, reusabilidade e manutenabilidade.

1.3.6 O PADRÃO MVC

Modelo de desenvolvimento em três camadas físicas onde a lógica de negócios é uma camada intermediária entre a camada de apresentação e a camada base. Fornecendo uma forma de compartilhar a funcionalidade englobada na manutenção e apresentação dos dados a arquitetura MVC (Modelo Visualização Controle) (27) favorece o mapeamento de aplicações *web* multicamadas.

O modelo MVC é composto pelos componentes:

Modelo: composto pelas regras de negócio que gerenciam os dados.

Visualização: composto pelas regras que renderizam o conteúdo da aplicação para o usuário, encaminhando suas interações à camada de controle.

Controle: composto pelas regras que definem o comportamento da aplicação. Possui as regras que interpretam as chamadas do usuário e intermedia o acesso da camada Modelo para a camada Visualização. Uma implementação desta arquitetura está ilustrada na Figura 1.

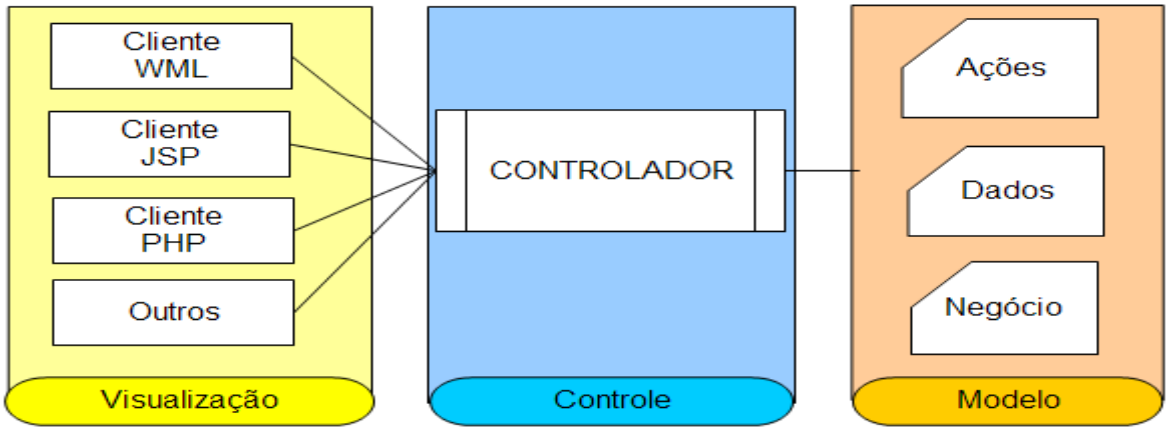

Figura 1. Uma implementação do modelo MVC

Dentre os pontos positivos da utilização do modelo MVC, podemos destacar:

- Gerencia múltiplos visualizadores usando o mesmo modelo e é fácil manter, testar e atualizar sistemas múltiplos;
- Facilita a inclusão de novos clientes apenas incluindo seus visualizadores e controladores;
- Favorece a escalabilidade da aplicação;
- Propicia a modularização do desenvolvimento para as camadas, pois são independentes;

O modelo MVC requer um maior conhecimento e análise durante a modelagem do sistema não sendo aconselhável para pequenas aplicações.

1.4 ESPECIFICAÇÃO E DESENVOLVIMENTO

Nesta seção apresenta-se um levantamento das necessidades de um Portal para o P2Pgrid, bem como algumas considerações no que tange o seu desenvolvimento e especificação.

1.4.1 CONSIDERAÇÕES INICIAIS

Com o crescimento natural da internet e do número de usuários conectados à rede, é normal que o número de tecnologias existente para implementação e especificação seja tão elevado. Porém devemos considerar o processo de definição das tecnologias a serem empregadas no desenvolvimento porque é importante para o êxito da aplicação proposta. Assim são apresentados na arquitetura do sistema alguns mecanismos que visam a minimizar problemas de: portabilidade, recompilação de código, tempo de resposta da aplicação, número de usuários conectados simultaneamente, entre outros fatores.

Por sua vez o mecanismo desenvolvido para a comunicação e autenticação de nodos apresenta característica que visam favorecer a integridade, autenticidade e confidencialidade das informações e dos nodos que a transmitem.

1.4.2 LEVANTAMENTO

É importante destacar que a construção de um Portal deve ser interativa e incremental. Nesta fase apresentamos as tecnologias que serão utilizadas na confecção do Portal. Ressalte-se ainda que a estrutura do P2PGrid sofreu alterações com o intuito de agregar recursos ao Portal e a grade.

1.4.2.1 LEVANTAMENTO DAS NECESSIDADES DO PORTAL

Esta fase descreve o que o Portal deve fazer, quais os seus principais aspectos, conforme descrito a seguir:

Aspectos não funcionais:

- Oferecer um baixo custo de implantação e manutenção. É necessário para alcançar esses objetivos que ele não perca em funcionalidade e qualidade;
- Especificar um Portal multiplataforma;
- Oferecer mecanismos de segurança tais como monitoramento e autenticação;
- Oferecer um bom tempo de resposta;

- Oferecer acesso a um número bastante elevado de usuários conectados simultaneamente.

Aspectos funcionais:

- Promover a integração com o P2PGrid;
- Promover o acesso unificado a todos os recursos;
- Submeter e acompanhar *jobs*;
- Promover a parametrização centralizada das execuções;
- Promover a contabilização dos *jobs*;
- Promover o controle de reputação dos nodos.

É importante ressaltar que com a atual evolução dos Portais para grades computacionais e a variação que os mesmo assumem, deve-se pensar em um Portal que seja flexível tanto quanto se fizer necessário ou algo próximo disso. Assim sendo, propõe-se como parte deste trabalho a criação não de um Portal para um grade específica, mas um Portal que seja adequado a qualquer grade e que possua organização de dados distintos para uma boa parte das formações idealizadas pelos seus administradores. Levanta-se então a necessidade de um Portal que nos possibilite o desenvolvimento de diversos Portais com variações que vão desde o visual chegando ao funcional. Tal Portal possibilitará a seus administradores a facilidade de manutenção, correção e evolução, adicionando ainda funcionalidades de segurança e monitoramento sem que haja a necessidade de programação massiva.

2 REVISÃO BIBLIOGRÁFICA

A computação paralela e distribuída de alto desempenho vem cada vez mais sendo abordada no desenvolvimento de aplicações que demandam alto grau de processamento. Aplicações nas áreas como: física, genética, financeira são exemplos destes tipos de aplicações.

Uma das soluções para atender os requisitos apresentados, é adotar máquinas paralelas multiprocessadoras que elevam o custo, mas oferecem alto desempenho. Outra solução, mais recente, é oferecer alto desempenho por meio da interconexão de computadores, através de uma rede, que possibilita agregar o poder computacional de cada recurso, sem causar dependência entre os mesmos. Esse tipo de solução nos possibilita criar sistemas denominados sistemas de computação paralela e distribuída.

Os aglomerados de computadores (*cluster computing*) (28) constituem-se em um dos tipos de sistemas de computação paralela e distribuída mais utilizados, sendo sistemas computacionais locais, compostos por um conjunto independente de computadores interligados através de uma rede.

Os computadores comuns, normalmente empregados em serviços e aplicações *standalone*, como a estação de trabalho de uma escola, podem compor um nodo de um aglomerado e ainda podem ser nodos os computadores multiprocessados, agindo como uma única máquina (SMP - *symmetric multiprocessor).*

A evolução da Internet e dos aglomerados de computadores traz à tona os sistemas paralelos e distribuídos baseados em grades computacionais. Utilizando a grande abrangência da Internet, esses sistemas ampliam a distribuição geográfica e possibilitam que um número maior de recursos computacionais heterogêneos sejam conectados. A heterogeneidade dos sistemas de grades de computadores (29) refere-se à localização e aos tipos dos recursos, sendo que a localidade está associada ao fato de englobar recursos pertencentes a diferentes proprietários. Pode-se considerar a heterogeneidade de tipos de recursos superior à dos aglomerados de computadores, devido à possível existência de diferentes recursos.

Em adição, a este trabalho de mestrado, visando a aprimorar e prover segurança e gerência de recursos distribuídos em sistemas de computação de grade, são apresentados neste capítulo os principais conceitos e trabalhos relacionados a esta área.

2.1 EM GRIDS

A denominação computação de grade é uma nova nomenclatura dada ao compartilhamento de recursos distribuídos geograficamente através de redes interligadas (30), sendo também denominada de metacomputação, *scalable computing*, computação em malha, computação global e *Internet computing*.

Inicialmente a idéia da computação de grade foi de unir supercomputadores, objetivando o aumento da capacidade computacional. Essa visão, atualmente, abrange a unificação de diferentes tipos de recursos, inclusive os supercomputadores.

A denominação **grade** é uma analogia (2) a grades, ou redes, do sistema de distribuição elétrico, o qual proporciona acesso amplo, transparente e consistente à eletricidade. Desse modo, idealiza-se tornar as grades de computadores um serviço disponível a todos os usuários, os quais podem estar geograficamente distribuídos.

As aplicações em ambientes de grades podem ser categorizadas em cinco classes principais, conforme descrito em (30):

- **Supercomputação Distribuída** - engloba aplicações com necessidade de grande poder de processamento;
- ***On-demand*** - aplicações que utilizam alguns instrumentos e recursos mais específicos em determinados momentos;
- **Colaborativo** - Aplicações que envolvem a participação síncrona ou assíncrona de indivíduos geograficamente distantes;

- **Grande fluxo de dados** - aplicações que possuem fluxo de informações vindo de várias fontes, inviabilizando a concentração em um único computador;
- **Grande quantidade de dados** - aplicações que utilizam grande quantidade de informações, armazenadas em vários meios secundários.

Outra classe de aplicação que pode se beneficiar com as grades de computadores são aquelas orientadas a serviços. Um exemplo dessas aplicações é a utilização de serviços que oferecem adaptação de mídia a usuários. Vários são os motivos que justificam que as aplicações sejam distribuídas em grades de computadores, destacando-se a exploração da natureza distribuída das aplicações que necessitam diminuir o tempo de resposta. Outro fator que favorece o uso dos ambientes de computação de grade, principalmente os orientados a serviços, é a possibilidade de trabalhar com o paradigma orientado a componentes, que facilita o reuso de funções e aplicações.

Serviços computacionais devem prover mecanismos de segurança, para a execução de aplicações individuais ou coletivas, nos recursos distribuídos. É esperado ainda que esses serviços forneçam mecanismos de monitoramento e controle das execuções de tarefas, além de possibilitar o controle da reserva e alocação dos recursos disponíveis.

As grades computacionais provêem esses serviços e são normalmente destinadas à computação de alto desempenho e computação com grande saída de informação (31).

Os serviços de armazenamento objetivam disponibilizar mecanismos seguros de acesso a repositórios de dados e os seus gerenciadores, sendo esta combinação de serviços chamada de *data grids*.

Por sua vez, os serviços de aplicação visam a gerenciar aplicações legadas e prover acesso ubíquo a softwares e bibliotecas de programação, distribuído entre vários recursos. Com base nos serviços de processamento e armazenamento podemos destacar quatro variações:

1. **Serviços de repositório de códigos:** ex.: controle de versões de códigos(CVS);

2. **Serviços de catálogo:** ex.: listagens de banco de dados relacionais indexados.

3. **Serviços de informação:** utilizam serviços de dados, serviços computacionais ou serviços de aplicação. Os serviços desta categoria estão relacionados com o modo que as informações são processadas.

4. **Serviços de conhecimento:** ex.: sistemas de tomada de decisão onde o conhecimento é adquirido, usado, retornado, publicado e mantido para ajudar usuários a alcançar seus objetivos particulares.

Ao avaliar e desenvolver sistemas em grade, alguns princípios diferem dos já observados ao construir aglomerados e outros sistemas distribuídos, outros são comuns às diferentes categorias.

O desenvolvimento básico de um ambiente de grade resume-se basicamente na integração de componentes independentes de *hardware* e *software* em um recurso combinado de comunicação no desenvolvimento:

- de um *middleware* de baixo nível que responde ao acesso ubíquo e seguro entre os recursos;
- de um *middleware* com ferramentas no nível do usuário para a agregação dos recursos distribuídos e o desenvolvimento de aplicações;

Além da otimização das aplicações que usufruem da infra-estrutura dos recursos distribuídos disponíveis.

2.1.1 ARQUITETURA DE UMA GRADE

Uma grade possui uma arquitetura dividida em quatro camadas (32): fábrica, *middleware* base, *middleware* do nível do usuário e aplicações, conforme apresentado na Figura 2.

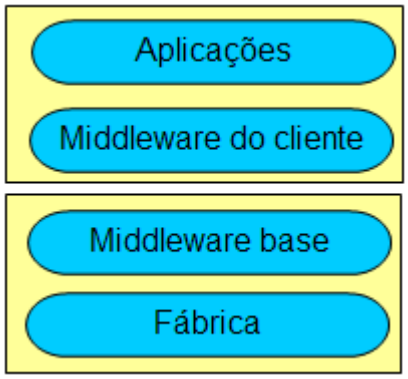

Figura 2. Arquitetura de uma grade

Fábrica: composta pelos recursos propriamente ditos e pelos componentes complementares dos mesmos além dos componentes para o funcionamento individual, quando não associados à grade. Costumeiramente os itens participantes dessa camada não são administrados pelo administrador da grade, mas sim escolhidos dentre os já existentes, uma vez que é desejado criar ambientes que funcionem com recursos e padrões amplamente utilizados. Esse fator diferencia as grades de outros sistemas distribuídos, por exemplo, *clusters*, onde algumas vezes *hardware*, protocolos e bibliotecas específicas são criadas com o intuito de ganhar desempenho ou alcançar alguma melhoria nos resultados.

***Middleware* base**: serviços essenciais para a ligação entre os nós componentes da grade. Sua formação possui uma subcamada de segurança que media todas as transferências de informações entre a camada inferior e os serviços responsáveis pela negociação, gerenciamento de informação, armazenamento de dados, execução de tarefas e demais serviços específicos da grade.

***Middleware* do cliente**: ferramentas utilizadas pelos usuários do ambiente para administração de recursos, submissão de trabalhos, desenvolvimento de aplicações, monitoração da situação dos recursos e demais funções que exigem participação do usuário. Utilizam serviços oferecidos pela camada base para executar operações de gerenciamento e seleção de recursos.

Aplicações: composto pelas aplicações com finalidades específicas de solucionar problemas. Em geral as aplicações utilizam os recursos da grade para alcançar suas metas.

2.1.2 COMPUTAÇÃO PEER-TO-PEER

Um modelo de comunicação entre os componentes de uma rede, de modo a promover o mesmo potencial operacional entre eles, define-se como computação *peer-to-peer*, conceito amplamente utilizado em sistemas de compartilhamento de arquivos (Napster (33), Emule (34), Kazaa (35) etc.), também é utilizado em aplicações chamadas de Internet Computing, como os sistemas SETI@Home (36) e Parabon (37). Tal modelo generaliza o modelo cliente-servidor, cujos nodos são clientes e servidores, de modo que este modelo seja dequado à uma implementação de um ambiente de grade computacional.

É notável que esta convergência seja a menos encontrada em relação às demais tecnologias, o fato é que o uso de tecnologias e implementações existentes não é interessante, por fornecer características inadequadas aos ambientes de grade desejados, por exemplo, compartilhamento de arquivos sem controle de acesso (Kazaa e afins) e compartilhamento computacional com servidor centralizado (SETI@Home).

Com base nisto, e na evolução das tecnologias, é interessante investir na convergência (32) (38) entre grades, peer-to-peer e Internet Computing.

2.1.3 GRID COMPUTING

Um novo conceito que explora os potenciais das redes de computadores é o Grid Computing(39), que objetiva, especificamente disponibilizar camadas virtuais, permitindo a um usuário acessar aplicações de alto nível de exigência, bem como aderir à comunidades virtuais de grande escala, com elevada diversidade de recursos de computação e de repositórios de informações.

A computação em grade e a computação distribuída diferem-se pelo fato de que a computação distribuída é um conceito que surgiu em meados dos anos 80 e 90 e refere-se à possibilidade de resolver um determinado problema computacional através da utilização de diferentes recursos distribuídos geograficamente. Por sua vez, a computação em grade faz-se quando a computação distribuída possui uma infra-estrutura física e uma infra-estrutura lógica (software), que possibilita

coordenar as tarefas que serão processadas garantindo ainda sua qualidade de serviço.

As grade computacionais nasceram da comunidade de Processamento de Alto Desempenho (PAD). O conceito foi apresentado pelos pesquisadores Ian Foster e Carl Kesselman, sendo composto por uma infra-estrutura de hardware e software que permite a eles acesso a grandes capacidades computacionais geograficamente distribuídas, de forma confiável, consistente, econômica e persistente.

Ian Forster traduz os conceitos de Grid de duas formas clássicas:

- "Compartilhamento de recursos coordenados e resolução de problemas em organizações virtuais multi-institucionais dinâmicas" e
- "Grids Computing são sistemas de suporte à execução de aplicações paralelas que acoplam recursos heterogêneos distribuídos, oferecendo acesso consistente e barato aos recursos, independente de sua posição física".

A tecnologia de Grids Computing possibilita agregar recursos computacionais variados e dispersos em um único 'supercomputador virtual', acelerando a execução de várias aplicações paralelas.

Um aspecto essencial dos serviços de Grid é que eles estão disponíveis uniformemente através dos ambientes distribuídos na Grid. Os serviços são agrupados em um sistema integrado, também chamado de middleware. Exemplos de ferramentas atuais de Grid incluem Globus (40), Legion (41) e ProGrid (42).

A grade permite ainda o uso de técnicas de programação paralela por passagem de mensagens, sendo o ambiente MPI (43) disponível para essa comunicação através da versão MPICH-G2 (44). O padrão MPI define uma biblioteca de rotinas que implementam uma comunicação ponto a ponto, em que a operação "*send*" é usada para iniciar uma transferência de dados entre dois programas concorrentes, e a operação "*receive*" é usada para obter dados do sistema. Existem ainda operações coletivas envolvendo múltiplos processos. Em decorrência do alto número de comunicação entre processos, as aplicações devem ser construídas com uma granularidade bem projetada de modo a comunicarem-se o mínimo

possível. Por conseguinte, as aplicações mais adequadas às grades são as que possuem tarefas independentes (*bag of tasks*), uma vez que as tarefas não trocam informações entre si.

2.1.3.1 BENEFÍCIOS DE UM GRID

A seguir detalhamos alguns dos benefícios, conforme apresentados por (45), que podem ser obtidos com a utilização de um Grid:

Organizações podem agregar recursos: a computação em grade possibilita agregar recursos não importando a sua localização global.

Poderosa plataforma de suporte a Organizações Virtuais (46)**:** organizações podem melhorar a qualidade de sua grade mediante a colaboração ubíqua dos recursos entre as organizações.

Acesso distribuído a diversos tipos de recursos: possibilita que as organizações compartilhem a base de dados.

Colaboração entre centro de pesquisas: possibilita a dispersão das organizações viabilizando a troca de experiências, colaboração em projetos e troca de dados e aplicações.

Melhor utilização de largura de banda**:** pode-se criar a mais robusta e resistente infra-estrutura de informações.

Aproveitamento de recursos ociosos**:** possibilita o aproveitamento dos ciclos de processamento disponíveis dos computadores não dedicados que se encontram distribuídos geograficamente.

Desafios operacionais e de pesquisa a serem vencidos:

- A localização dos recursos;
- A reserva de recursos;
- A adaptabilidade a mudanças no ambiente;
- A criação e submissão das tarefas;
- A autonomia de cada participante em definir suas políticas pessoais de segurança;

- A distribuição geográfica de recursos requisitados;
- A qualidade de serviço das aplicações.

A padronização da tecnologia de grades vem ocorrendo há bastante tempo, porém atualmente passaram a ter uma maior prioridade devido a sua utilização por diversas organizações.

Então, basicamente, uma grade computacional se resume a:

- Computadores geograficamente distribuídos;
- Computação de alto-desempenho;
- Ambiente colaborativo composto por diferentes organizações;
- Grande quantidade de dados;
- Agregação e compartilhamento de recursos computacionais como: Computadores e supercomputadores; Dispositivos especialistas como microscópios e telescópios; Sistemas de armazenamento e bancos de dados.

As dificuldades encontradas são muitas, e os estudos são incessantes na área de grades computacionais, destacando-se:

- Localização e reserva de recursos;
- Capacidade para adaptar-se a mudanças no contexto do ambiente;
- Criação e submissão das tarefas;
- Autonomia de cada grupo participante para definir suas próprias políticas de segurança;
- Recursos requisitados podem estar em diferentes localidades;
- Qualidade de serviço exigida por cada aplicação.

2.1.4 GLOBAL GRID FORUM

O *Global Grid Forum* (GGF) foi criado com o intuito de definir uma padronização mundial das tecnologias de grades. Resultado da união do *European Grid Forum*

(e-Grid), a comunidade *Pacific-Asia Grid* e o *Grid Forum,* o GGF agrega atualmente diversas indústrias e centros de pesquisa, com representação em diversos países.

Objetivando o desenvolvimento de um padrão que provenha um modelo computacional aberto de domínio público o GGF objetiva ainda:

- Definir uma especificação padrão para integrar os diversos padrões existentes;
- Desenvolver uma comunidade mundial de colaboração e troca de idéias, experiências e práticas inovadoras.

O GGF compôs alguns grupos de pesquisa e desenvolvimento de modo a direcionar o foco e as áreas de interesse, sendo eles (47):

Infra-estrutura: Explora a relação entre o *middleware* de grades e as camadas de recursos inferiores.

Dados: Responde pelo acesso, gerenciamento e transporte dos dados nas grades.

Computacional: Descreve e executa tarefas computacionais nas grades, como também escalona e negocia os recursos.

Arquitetura: Define e estuda a arquitetura padrão para grades.

Aplicação: Responsável pelos desafios relacionados ao desenvolvimento de aplicações de grades.

Gerenciamento: Visa a padronizar um modelo para o gerenciamento dos componentes operacionais essenciais nas grades, como: políticas, equipamentos, processos e dados.

Segurança: Responde pelas questões relacionadas à segurança, abrangendo os requisitos essenciais, como: privacidade, integridade e confiabilidade dos dados, das informações e dos usuários.

Ligação: Este grupo visa a aperfeiçoar a troca de informações promovendo um ambiente de colaboração entre os usuários e pesquisadores das tecnologias de grades e seus portais.

2.1.5 GRID ECONOMY

Grid Economy(48) é a definição da utilização de modelos econômicos aplicados na modelagem dos mecanismos de interação entre as organizações que provêem recursos e usuários. Fazer uso da teoria econômica é um modo comum de lidar com a oferta e demanda gerada pelas grades e as organizações que as utilizam.

Nos modelos econômicos aplicados a grades encontramos duas formas de alocar recursos entre agentes competidores: a *economia baseada em trocas* e a *economia baseada em preços* (49). A primeira se baseia no contexto onde cada organização possui um conjunto de recursos e permuta os mesmos com outras organizações. A segunda abordagem baseada em preços (50) utiliza uma moeda para expressar os custos dos recursos. Cada organização possui uma importância utilizando-a para consumir recursos de outras organizações. A tabela de valores praticados nas duas abordagens é predefinida, podendo ser automatizada, entre as partes envolvidas, regulando a oferta e a procura. Ambas as abordagens apresentadas precedem a existência de mecanismos de contabilização, cobranças e segurança.

2.1.6 PRINCIPAIS PROJETOS DE GRID

Nesta seção apresenta-se alguns dos projetos de Grids estudados durante o desenvolvimento deste trabalho. São eles: Globus, Legion, ProGrid e o P2PGrid.

2.1.6.1 GLOBUS

Iniciado em 1997, o projeto Globus possui como objetivo o desenvolvimento e promoção de protocolos padrões de modo a permitir a interoperabilidade entre infra-estruturas.

Um conjunto de ferramentas desenvolvidas de código aberto compõe o Globus Toolkit, desenvolvidos por Ian Forster, Carl Kesselman e Steve Tuecke que objetivaram facilitar a computação em grade por meio de APIs e SDKs. Na sua mais recente versão (4.x) o Globus agrega serviços e bibliotecas para monitoração, descoberta e gerenciamento de recursos, segurança e gerenciamento de arquivos.

2.1.6.2 LEGION

A grade computacional Legion desenvolvida em 1993 pela Universidade de Virginia foi uma das pioneiras em computação em grade. Atualmente seus idealizadores desenvolvem e comercializam o ambiente através de uma empresa, fundada em 2001, chamada Avaki.

O Legion foi desenvolvido utilizando-se o paradigma de orientação a objetos, sendo que todos os elementos da Grade são representados por objetos. A comunicação entre os objetos se dá por chamadas de métodos assíncronas, e suas interfaces são definidas por um tipo de IDL. É responsabilidade de suas classes a criação de objetos, ativação/desativação e agendamento da execução.

Preocupado com o suporte a aplicações paralelas, o Legion possui uma implementação das bibliotecas MPI (Message Passing Interface) e PVM (Parallel Virtual Machine). Para que um programa escrito em PVM ou MPI execute na grade basta que seja recompilado com as bibliotecas fornecidas pelo Legion, permitindo que a migração entre as versões de infra-estruturas Legion seja praticamente instantânea.

Além de fornecer as bibliotecas, o Legion disponibiliza ainda suporte nativo a algumas linguagens de programação paralela, possibilitando a utilização de linguagens como: MPL (Mentat Programming Language, uma extensão de C++ para programação paralela), BFS (Basic Fortran Support) e Java.

Para aplicações legadas, que não utilizam nenhuma das linguagens ou bibliotecas acima, podemos encapsular a aplicação dentro de objetos Legion. Para tanto é preciso registrar o programa legado através do comando *legion_register_program* e será construído um objeto Legion que encapsulará o programa legado. Tal recurso garante que qualquer programa possa ser executado no ambiente do Legion. Caso o programa realize algum tipo de comunicação, será necessário que seja desenvolvido um adaptador legado que medie as chamadas da biblioteca da aplicação com as chamadas do Legion.

2.1.6.3 PROGRID

O projeto ProGrid (42) foi concebido com o objetivo de desenvolver uma infra-estrutura para a construção de uma grade computacional com suporte para aplicações independentes e aplicações MPI. A infra-estrutura desenvolvida é composta por serviços que buscam a minimização de esforços do usuário tanto para a utilização como para a configuração da grade.

A arquitetura do ProGrid é composta por três elementos principais: um Portal de grade baseado na web, servidores *Proxies* e *Hosts*. Cada um desses elementos apresenta funcionalidades independentes e interagem entre si, de forma transparente ao usuário, possibilitando a composição de um ambiente seguro e reduzindo esforços de configuração e utilização dos recursos e serviços, afastando o usuário da complexidade do sistema.

Hosts correspondem aos recursos computacionais disponibilizados pelos usuários para a execução de aplicações. Um servidor *Proxy* é responsável por intermediar a comunicação entre as diferentes sub-redes que compõem a grade, atuando como um *gateway* e reduzindo o número de comunicações necessárias entre os *hosts* de sub-redes distintas. É sua responsabilidade intermediar tais comunicações, facilitando, com isso, a passagem por *firewalls*, pois realiza a multiplexação de várias conexões locais sobre uma única conexão externa.

Dessa forma, um servidor *Proxy* aumenta a segurança na comunicação inter-grid, provendo suporte para autenticação de *hosts* e usuários, e na troca de mensagens entre as diferentes redes da grade. A figura 3 ilustra a arquitetura da grade ProGrid e a posição do *proxy* nessa arquitetura.

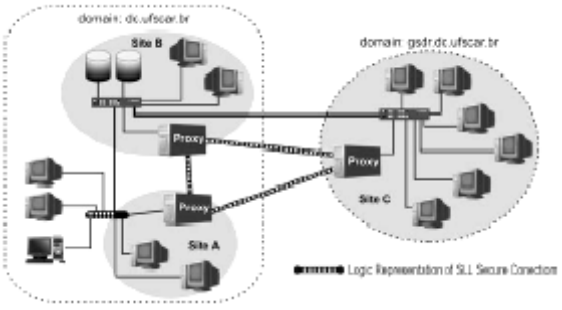

Figura 3. (42) Visão geral da arquitetura ProGrid

O terceiro elemento na grade ProGrid é um Portal (51) de grade, baseado na *web* que provê uma interface amigável ao usuário, disponibilizando todas as funcionalidades necessárias para o gerenciamento e monitoramento dos recursos e aplicações pertencentes à grade.

2.1.6.4 P2PGRID

Observando a possibilidade de convergência entre grades computacionais e sistemas *peer-to-peer*, foi desenvolvido o P2PGrid (1) tomando como base na sua implementação algoritmo de balanceamento de carga descrito em (51). Consiste em um ambiente de grade computacional que gerencia recursos distribuídos e interligados por uma rede, seguindo o modelo de comunicação *peer-to-peer*, que define uma rede composta por componentes com igual capacidade de oferecer e utilizar serviços. Em outras palavras, assemelha-se a uma rede, seguindo o modelo cliente-servidor, cujos participantes são clientes e servidores dos demais.

No P2PGrid foram implementados alguns componentes responsáveis por mecanismos que gerenciam operações em recursos da grade. Esses componentes são agrupados em uma camada intermediária de controle do ambiente, a qual utiliza serviços da camada inferior denominada base e oferece serviços à camada superior composta por aplicações dos usuários.

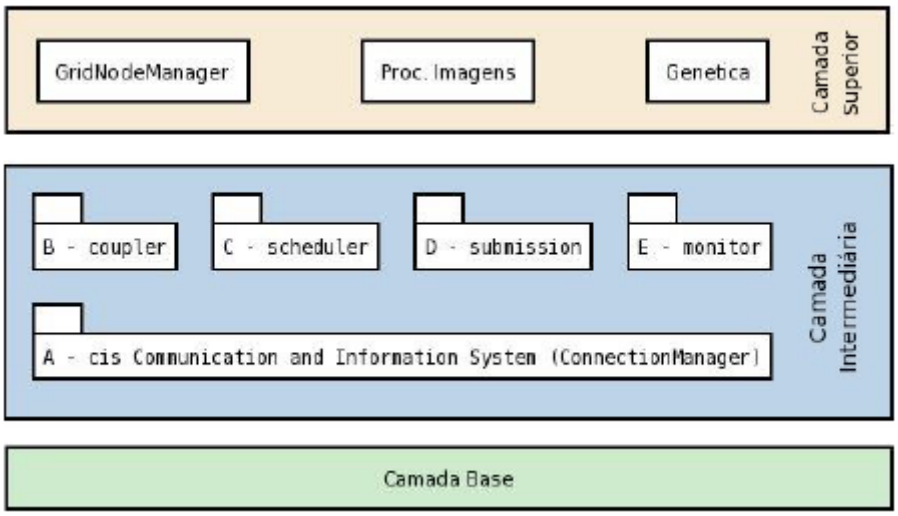

Figura 4. (1)Arquitetura em camadas do P2PGrid

Na camada intermediária (ver figura 4) concentram-se os componentes responsáveis por todos os serviços de controle necessários para manter o ambiente de grade computacional ativo.

Os componentes da camada intermediária abstraem mecanismos gerenciadores específicos, incluindo acoplador, escalonador, sistema de submissão (disparador e executor) e monitor.

Há ainda o componente que representa o sistema de comunicação e informação do ambiente, responsável por centralizar todas as informações das conexões adjacentes.

O sistema de comunicação e informação (CIS) é intermediário de todas as mensagens trocadas entre componentes equivalentes dos recursos. Essa equivalência é dada pela correspondência entre serviços, definindo um ator e um receptor para cada ação; por exemplo, os escalonadores trocam mensagens entre si e os disparadores de tarefas trocam mensagens com executores das mesmas. Esse sistema tem a finalidade de centralizar todas as informações relativas às conexões existentes, além de disponibilizar um meio único e seguro para a troca de mensagens entre os demais componentes.

O componente denominado acoplador (*coupler*) é responsável pelos serviços que mantêm as interconexões lógicas dos recursos. Exemplos desses serviços são os mecanismos de controle do ingresso de novos nós à grade e da reestruturação em caso de falhas de recursos. Ele está diretamente associado ao modelo de comunicação entre os recursos utilizados pelo ambiente distribuído desenvolvido.

O componente escalonador (*scheduler*) compreende o mecanismo responsável por alocar um trabalho em algum recurso disponível capaz de executá-lo. Ele implementa todas as funcionalidades necessárias para garantir o escalonamento, seguindo o algoritmo proposto. Por exemplo, para efetuar o balanceamento de carga, o escalonador deve gerenciar a captação e a troca das informações de carga dos recursos.

O componente de submissão de trabalhos incorpora componentes para disparar e executar remotamente trabalhos enviados pelos usuários. O disparador é o

componente responsável por enviar todas as informações necessárias ao executor, que executa os serviços associados ao trabalho remoto e responsabiliza-se por retornar o resultado do processamento. Além disso, o sistema de submissão de trabalhos é responsável por utilizar serviços do escalonador para descobrir os melhores recursos, e do disparador para executar e controlar processos e tarefas remotamente, genericamente chamados de trabalhos. Nesse momento, não há distinção entre processos e tarefas, que definem apenas um conjunto de instruções com finalidade específica, mas que serão diferenciados pelos componentes disparador e executor.

Os monitores representam mecanismos que monitoram determinadas características de cada recurso e enviam os valores aos demais recursos, com o objetivo de tornar disponíveis informações relacionadas ao estado do ambiente. Esses mecanismos abstraem os procedimentos de trocas de mensagens para a criação de qualquer sistema de monitoração, incluindo sistemas para mapear recursos ativos, buscar falhas dos recursos e listar execuções em andamento. A formação da grade do P2PGrid é baseada em uma árvore balanceada com N nodos adjacentes, sendo que o primeiro nodo da grade é o nodo raiz e os demais são adjacentes desse nodo. A distribuição de nodos é balanceada a partir do nodo entrante, ou seja, se todas as requisições de ingresso à grade forem realizadas ao nodo raiz teremos uma árvore balanceada.

2.2 EM PORTAIS

A utilização de portais *web* como ponto de acesso aos recursos de uma grade, tornou-se uma idéia altamente atrativa, pois isola os usuários de grande parte dos detalhes necessários para a execução de uma aplicação sobre recursos remotos compartilhados.

Grande parte dos projetos de grade já inclui portais como parte em suas arquiteturas. Embora algumas soluções ainda adotem o modelo de sistemas em desktop, a solução web vem sendo a mais indicada, tornando-se uma tendência.

Para efeito de ilustração, apresenta-se o ProGrid. Os serviços oferecidos pelo Portal ProGrid(51) têm como objetivo principal a minimização de esforços do usuário na realização de tarefas básicas, tais como transferência de arquivos, execução de aplicações, gerenciamento e monitoramento de recursos (carga, fila etc.), juntamente com mecanismos de segurança e confiabilidade, que são realizados de forma transparente.

Entre os serviços disponíveis destacam-se:

• Composição da grade: Serviços para criação de contas de usuários e registro de hosts e servidores Proxies.

• Gerenciamento de recursos: Serviços para inserção, remoção e atualização de informações sobre recursos.

• Suporte para submissão: Mapeamento automático dos requisitos necessários para a execução de uma aplicação sobre os hosts.

• Composição de um ambiente de execução de aplicações: Usuários pré-definem os recursos que atendem aos requisitos das aplicações (previamente escalonados pelo serviço anterior) para a execução de suas aplicações.

• Serviços de monitoramento para recursos e aplicações: Apresentação do estado atual do histórico de aplicações e do uso de recursos.

• Execução de aplicações: Transferência de arquivos necessários para a execução da aplicação, determinação de parâmetros e acesso aos dados de resultados.

• Mecanismos de segurança: Definição de controle de acesso (ACLs) sobre os recursos, utilização de protocolos de comunicação segura.

• Suporte para inserção de modelos econômicos: Contabilização de recursos cedidos e utilizados por cada usuário.

Dessa forma, o Portal ProGrid provê grande parte das funcionalidades necessárias para controlar uma grade, desde aquelas que permitem a composição de um ambiente seguro, até funções para execução e monitoramento de aplicações.

Diferentemente da maioria das abordagens existentes na literatura, o uso do ProGrid não se inicia pela configuração dos recursos e serviços para estabelecer uma infra-estrutura e posteriormente realizar o mapeamento dessas funcionalidades no portal.

2.2.1 SUBMISSÃO DE TAREFAS

A Submissão de tarefas em grades computacionais geralmente é realizada através de aplicações mediadoras (*stand-alone*, Portais, etc.) ou de chamadas diretas via comando através da console de um *Daemon*.

A submissão de tarefas consiste basicamente no processo de associar uma tarefa a algum recurso disponível capaz de processá-la. Para uma mesma estrutura ou conjunto de recursos é possível utilizar diferentes mecanismos de submissão, compostos por algoritmos e políticas, os quais podem ser classificados de diversas maneiras comparando suas características.

Uma das classificações mais abrangente e aceita é a proposta em (52), cuja taxonomia divide-se em classificação hierárquica apresentada em (1) e propriedades distintivas lineares.

Além das características da classificação hierárquica, é possível encontrar algumas propriedades distintivas nas diferentes categorias de escalonadores. Essas propriedades são definidas analisando os algoritmos quanto à adaptabilidade, existência de distribuição de carga, disponibilização de recursos ociosos, utilização de informações probabilísticas e execução de realocação de processos.

2.3 SEGURANÇA

Quando se trabalha com sistemas que utilizam controles distribuídos, alguns detalhes quanto à segurança devem ser especialmente tratados. Garantia de integridade, por exemplo, deve ser fornecida pelo sistema de comunicação. O Acesso a um Grid deve ser autorizado e validado segundo critérios e políticas de segurança, previamente estabelecidos. Tais critérios devem abranger confidencialidade, integridade, autenticidade e identidade. Cada nó pertencente a

uma grade deve possuir autorização de conexão e validar seu acesso junto a um servidor de credenciamento/validação antes de se juntar à grade.

Aspectos como a segurança do Portal também devem ser considerados tendo seu desenvolvimento voltado às especificações de segurança básica de sites, bem como às especificações propostas pelo GGF (47). O Portal proposto neste trabalho visa a atender aos requisitos de autenticação recomendados pelo OGSA-WG (53), OGSA-Authz WG (54) (55).

2.3.1 NO P2PGRID

A segurança no P2PGrid, atualmente, encontra-se focada na política de uso dos recursos. A política de uso dos recursos determina quais operações podem ser utilizadas na computação das tarefas remotas, por exemplo, fazer acesso a disco, estabelecer comunicação por rede, executar processos do sistema, entre outros. Por ser o P2PGrid desenvolvido totalmente em java, fez-se uso de alguns recursos de segurança inerentes à linguagem, como o recurso, nomeado *SecurityManager* responsável por gerenciar permissões sobre operações executadas pelo programa em execução na máquina virtual.

Das possíveis operações supervisionadas pelo *SecurityManager*, pode-se destacar a prevenção da instalação de novos *ClassLoaders* pela aplicação remota, com o objetivo de garantir isolamento entre as computações; proteção das threads e dos grupos de threads para que um não interfira em outro; controle da execução de aplicações do sistema operacional; controle da finalização da JVM pelas tarefas; controle do acesso a recursos do sistema incluindo, fila de impressão, *clipboards*, fila de eventos, propriedades do sistema e janelas de programas; restrição das operações sobre o sistema de arquivo, como leitura, escrita e remoção; controle de operações na rede de comunicação, incluindo *connect* e *accept*, e controle de acesso a pacotes Java ou grupo de classes, incluindo acesso a classes destinadas à segurança.

Tendo em vista estes possíveis controles, é possível criar e associar um *SecurityManager* capaz de adequar-se às necessidades de diferentes grades,

sendo flexível na parametrização das características, que devem ser uniformes em todos os recursos, para a garantia da homogeneidade dos serviços prestados. Neste caso, o importante não é só criar um gerenciador que permita a adaptação conforme a necessidade, mas sim um sistema que garanta a igualdade em todos os recursos. Por exemplo, se desejar criar uma grade que permita acesso a disco pelos processos remotos, basta configurar parâmetros no *SecurityManager*, cujos valores serão iguais em todos os seus nós.

2.3.2 EM PORTAIS

Promover a segurança de Portais que gerenciam grades computacionais se faz necessário a medida que desejamos prover controle de acesso, autenticação, integridade de mensagem e privacidade. Outros aspectos de segurança e recomendações para implementações são descritos em (58).

- Autorização: Certifica-se os usuários têm a permissão apropriada para acesso a um módulo ou determinada operação.
- Privacidade: Os dados trocados entre o navegador e o servidor não podem ser lidos por terceiros.
    - Privacidade nas transações: Deve-se garantir que o conteúdo das transações HTTP efetuadas (*request* e *response*), não possam ser acessíveis a terceiros, mesmo possuindo acesso indevido aos dados.
- Autenticidade: Obtida através da utilização de assinaturas digitais que visam garantir que tanto o remetente quanto o destinatário das mensagens sejam quem dizem ser.
    - Autenticação de servidores e serviços: Um servidor HTTP deve-se autenticar perante os clientes para garantir privacidade e para que os mesmos possam confiar nos dados recebidos do servidor.
    - Autenticação de clientes: Um cliente deve autenticar-se perante os servidores, ou seja, apresentar credenciais válidas, a fim de que esses procedam ao acesso especificado na sua parametrização.

- Integridade: Integridade de mensagem garante que os dados recebidos numa comunicação são consistentes e íntegros, ou seja, não sofreram alteração por terceiros.
- Gerenciamento de sessões: gerenciar as sessões criadas por cada usuário após a autenticação definindo dentre outras informações o prazo de expiração das mesmas após um período de inatividade.
- Auditoria: Mecanismos que possibilitem a depuração das operações dos usuários e levantamento de suas operações devem ser providos com a finalidade de mapear todas as operações realizadas no Portal.

A segurança é fundamental no desenvolvimento de Portais, visto que os dados que trafegam na rede e que serão atualizados no banco de dados são críticos e sigilosos. Algumas vulnerabilidades em portais são descritas em (58) e devem ser consideradas durante a especificação e implementação de um portal para uma grade computacional.

3 PORTALMAKER - O PORTAL PROPOSTO

O portal proposto para integração com o P2PGrid foi implementado em *JSP* e *servlets* conectado ao banco de dados *MySQL* (56) via *JDBC* (57). Rodando sobre uma conexão segura *https*, o Portal contém um applet de autenticação, que dificulta o processo de captura da senha do usuário por terceiros. Todo o portal foi desenvolvido reutilizando o *framework VirtualClass* (61), apresentado na Figura 5. Esse *framework* possibilita agregar recursos para o Portal e geri-los de maneira centralizada mediante especificações e implementações definidas nela. A partir desse *framework* especificou-se um Portal denominado PortalMaker utilizado para a construção de Portais.

O *PortalMaker* é um sistema *web* que permite modelar portais, adicionando funcionalidades tanto visuais quanto funcionais sem a necessidade de implementação para o mesmo. O *PortalMaker* trabalha com *plugins* e *templates* que adicionam recursos aos portais geridos pelo *PortalMaker*. Assim, a criação de um portal para uma grade qualquer dependerá apenas de as especificações de dados do mesmo serem lançadas no sistema e que um *plugin* de comunicação do portal para a grade seja criado e adicionado ao *PortalMaker*, que com base nisso possibilitará a geração de diversas formas visuais e funcionais do portal escolhido.

Sendo assim, apresentaremos a estrutura do *framework* utilizado para a criação do *PortalMaker*, em seguida, a estrutura do *PortalMaker* e, para finalizar, um estudo de caso de um Portal para submissões, acompanhamento de *jobs* e de visualização do *P2PGrid*. É importante ressaltar que o *PortalMaker* é um *middleware* para a criação de portais para os usuários tratando-se de um portal meio e não de um portal fim.

3.1 FRAMEWORK VIRTUALCLASS

Nesta seção é apresentado a arquitetura e os componentes do *framework* VirtualClass reutilizado e estendido no desenvolvimento do *PortalMaker*.

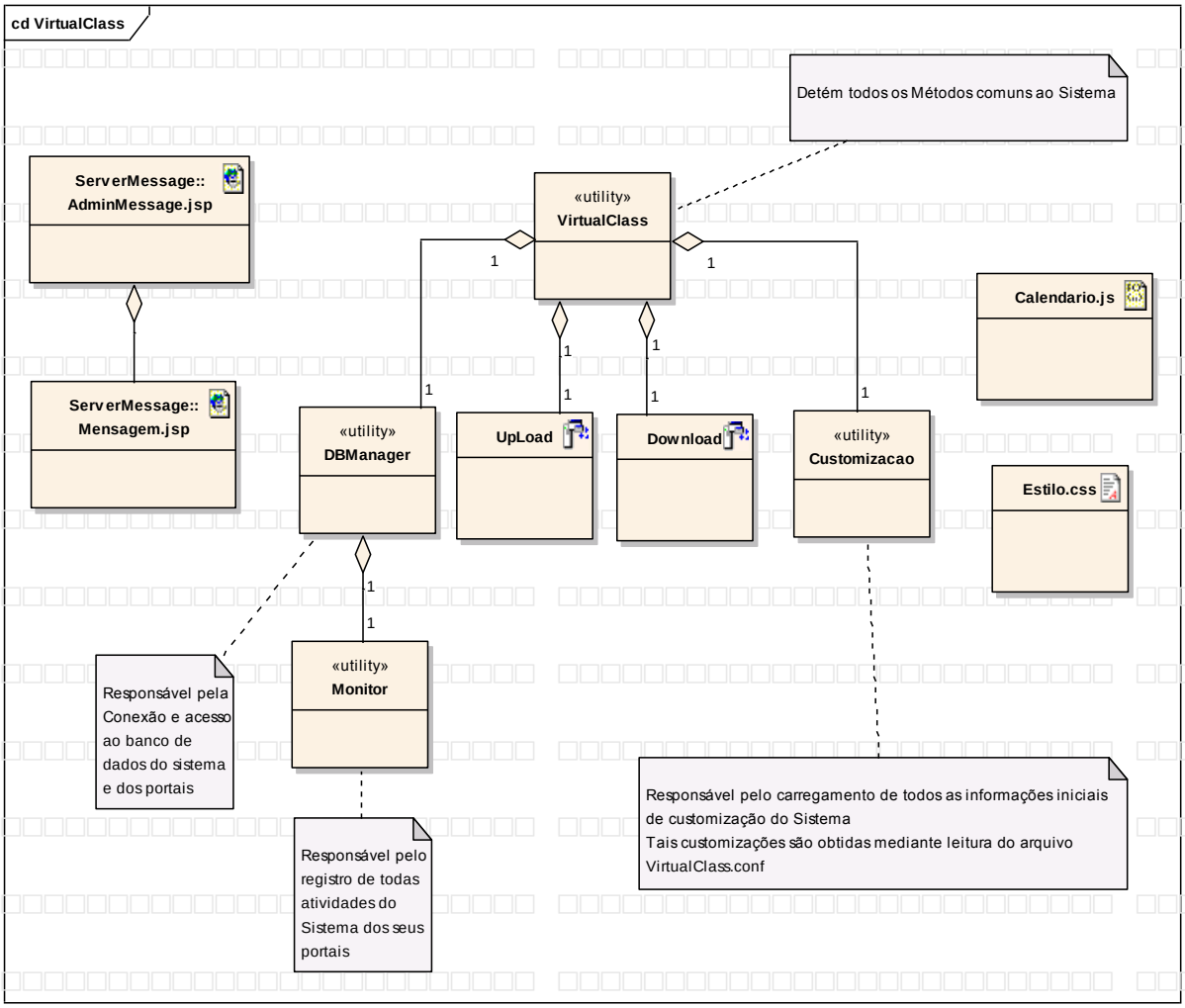

Figura 5. Framework VirtualClass do PortalMaker

<ServerMessage>: Pacote de recurso JSP que gerencia todas as mensagens de retorno aos usuários do *PortalMaker*. São componentes desse pacote o *AdminMessage.jsp* e o *Mensagem.jsp*.

<ServerMessage::AdminMessage.jsp> Aplicação JSP, que gerencia as requisições de todo o *PortalMaker,* de seus portais e do *framework VirtualClass*, a Lógica de abertura e fechamento de janelas; bem como a de registro de ocorrências e de atualização de *frames* e janelas *popups*.

<ServerMessage::Mensagem.jsp> Arquivo JSP, que contém a apresentação visual de todas as mensagens do Sistema. Este arquivo é utilizado em janelas *popups* ou *frames* conforme especificação do *PortalMaker*. Sua chamada é exclusivamente feita pelo *AdminMessage*, que, após efetuar a lógica de negócios da mensagem, redireciona a requisição tratada para este arquivo.

<utility DBManager> detentor de todos os métodos primários de manipulação de banco de dados. Esse pacote gerencia e media todas as requisições de manipulação do banco de dados, bem como o gerenciamento das conexões através de um *pool* de conexões.

<UpLoad> Servlet responsável por efetuar as operações de *Upload* do *PortalMaker* e dos portais gerenciados, seguindo as especificações dos mesmos.

<Download> Servlet responsável pelo gerenciamento dos downloads. Gerencia e restringe o acesso aos conteúdos restritos disponibilizados pelo *PortalMaker* e seus portais aos usuários.

<Customizacao> Pacote responsável pelo carregamento das customizações mediante a leitura de um arquivo de configuração do *PortalMaker*, denominado *VirtualClass.conf*.

<*VirtualClass*.conf> Arquivo em formato XML, que contém todas as configurações do *PortalMaker*. Exemplos de informações contidas nesse arquivo: parâmetros de conexão ao banco de dados remoto ou local; parâmetros de configuração de diretórios de *Upload* e *Download*; especificação de gerenciamento de *logs* de utilização do *PortalMaker*; Definição de restrição e localização de recurso e módulos(alguns recursos do *PortalMaker* podem estar em servidores distintos);

<Monitor> Detém os métodos de gerenciamento de *logs* de todo o *framework VirtualClass* e do *PortalMaker*. Os métodos contidos neste pacote possibilitam o registro de ocorrências de todo o sistema: Manipulação de Dados, acesso a páginas, operações de *upload* e *download*, mecanismos de *backups* de *logs*, são alguns dos recursos pertinentes a este pacote.

<VirtualClass> Detém todos os métodos reutilizáveis do sistema. Responsável também pelo: carregamento e instanciamento dos objetos das classes: DBManager e Customização;

<Calendario.js> Responsável pela geração de um calendário ilustrado de acompanhamento de agendamento e notificações do Sistema para o usuário.

<Estilo.css> Detém todas as configurações de apresentação visual do sistema. Este arquivo pode ser manipulado de acordo com a necessidade dos usuários do *PortalMaker* de forma a promover a customização visual do mesmo. Tal arquivo pode possuir variações e ambas anexadas ao sistema de modo que o usuário possa escolher a que lhe é mais conveniente.

O *PortalMaker* baseia-se nesse framework reutilizando e estendendo seus recursos e conseqüentemente provendo essa reutilização e extensão a todos os portais que gerencia. A definição e a adoção deste *framework* objetiva favorecer o melhor entendimento e desenvolvimento do *PortalMaker*, uma vez que todos os recursos visuais e de programação serão geridos pelo *framework*, o que resultará em facilidade ao se pensar em manutenções de portais sejam elas: evolutiva, adaptativa e corretiva. A manutenção evolutiva garante que os Portais tenham seu desenvolvimento realizado de acordo com as novas necessidades e pretensões do administrador. Por sua vez, a manutenção adaptativa define a possibilidade de os portais atenderem a mudanças estruturais de maneira que novas interações e regras de negócios sejam atendidas. Por fim, a manutenção corretiva busca eliminar erros de operações do Sistema. É primordial, no desenvolvimento de portais, que a redundância de código seja mínima, de forma a aperfeiçoar e reduzir a quantidade de códigos de programação.

3.2 A ESTRUTURA DO PORTALMAKER

Trata-se de um Portal *middleware* que possui, em sua estrutura básica, funcionalidades que possibilitam gerar e administrar qualquer Portal a partir dele. Podemos considerar o *PortalMaker* como um embrião de portais onde, sem muito conhecimento de programação, podem ser gerados e, através da gestão de

plugins e *templates*, adicionar funcionalidades e otimizações para todos os portais administrados.

O *PortalMaker* possui uma estrutura básica de serviços e recursos inerentes à criação e à facilitação de atividades como: criação de portais, *uploads*, suporte e comunicação entre usuários. Assim sendo o *PortalMaker* possui um gama de recursos implementados com esse propósito conforme apresentado na Figura 6.

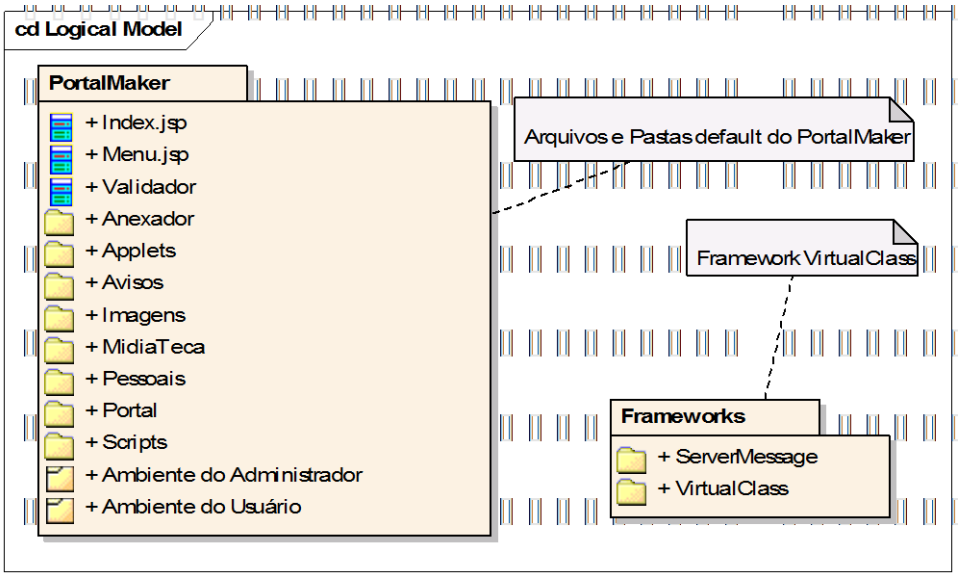

Figura 6. Estrutura do *PortalMaker*

3.2.1 OS MÓDULOS

O *PortalMaker* foi desenvolvido com base em uma estrutura modular e independente de modo a disponibilizar recursos que podem ser agregados durante o desenvolvimento dos portais. A figura 7 apresenta o organograma de entrada do *PortalMaker*.

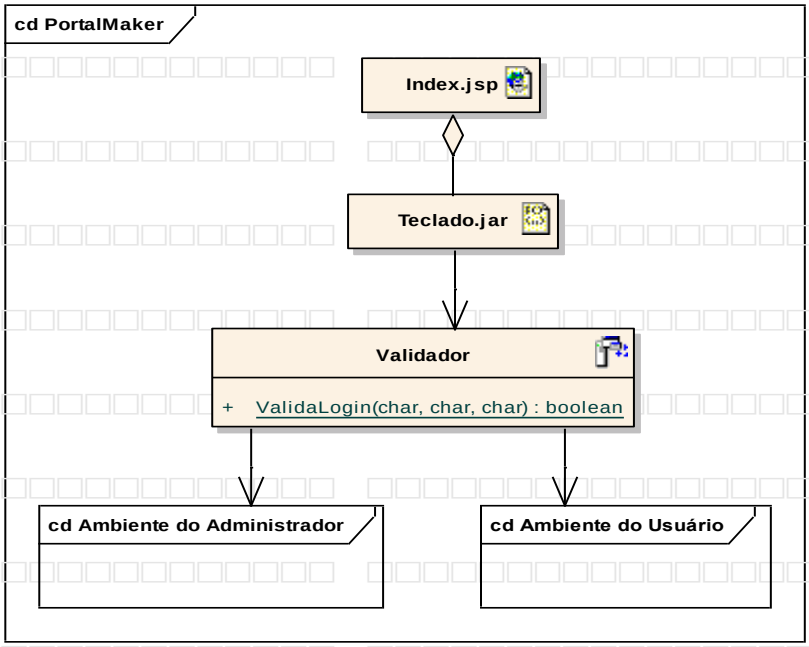

Figura 7. Organograma de Entrada do *PortalMaker*

Para o acesso ao *PortalMaker* basta que seja digitada a URL que hospeda o sistema. O arquivo *Index.jsp* carrega o applet *teclado.jar* a partir do qual e submetido a requisição de autenticação dos módulos administrador ou usuário. Neste *applet* são fornecidas informações como: módulo desejado, usuário e senha de acesso. Esta requisição é encaminhada ao *Validador.jsp* que valida o acesso e redireciona o usuário para o módulo desejado (em caso de êxito na autenticação) ou para a tela de *login* com mensagem da falha ocorrida durante a autenticação.

3.2.1.1 MÓDULO DO ADMINISTRADOR

O módulo administrador é responsável pela configuração e manutenção dos portais a figura 8 apresenta a organização dos arquivos JSP que compõem esse módulo. O detalhamento das telas de cadastro e procedimentos para a configuração e utilização do *PortalMaker* encontra-se em manual especifico disponível em (61).

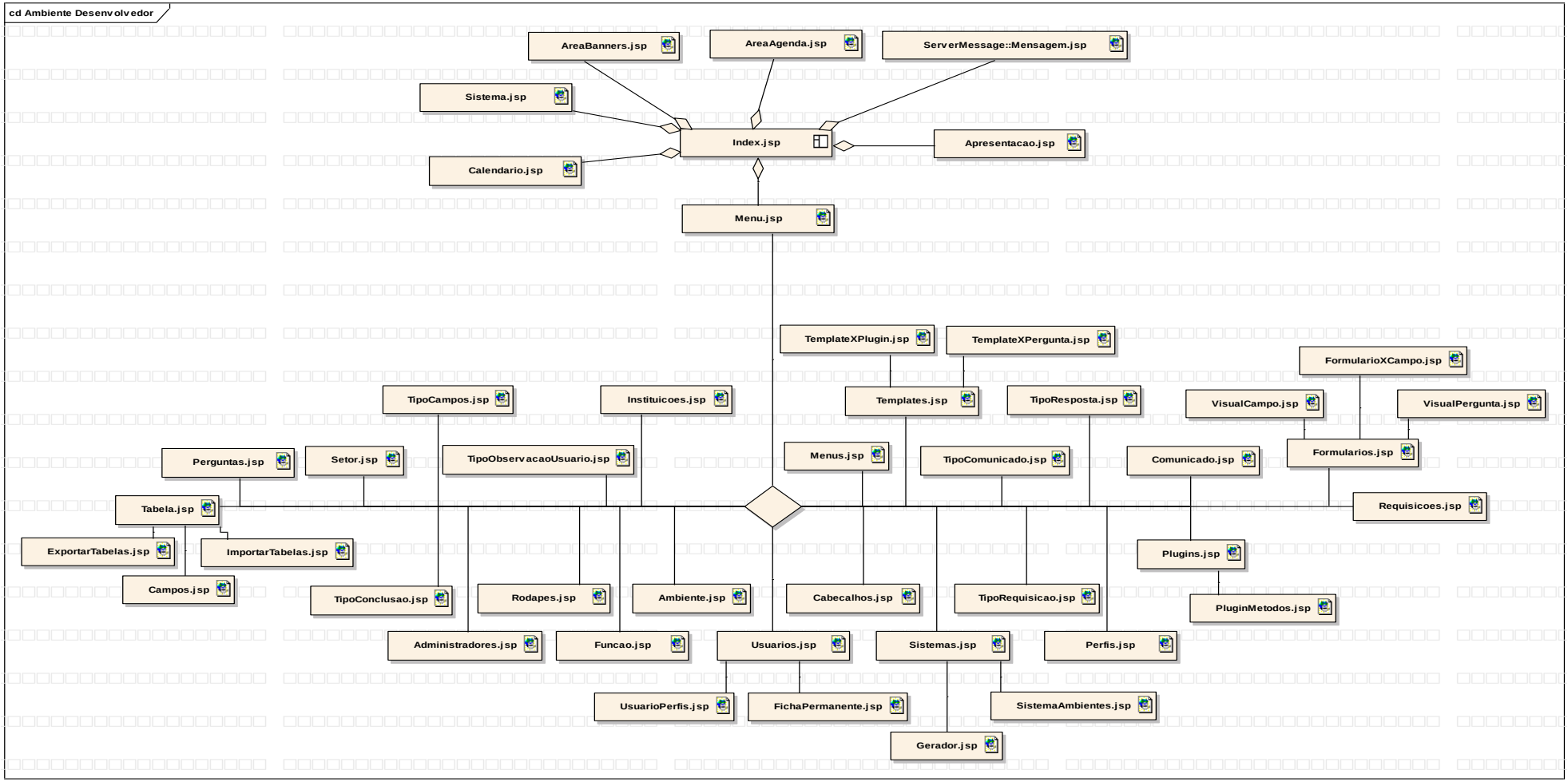

Figura 8. Organograma do módulo do Administrador

3.2.1.2 MÓDULO DO USUÁRIO

Responsável por organizar o acesso dos usuários aos portais hospedados. Composto basicamente por um *frameset* que possui quatro *frames* (superior direito, superior esquerdo, superior central, e inferior). Os três primeiros *frames* são padrões do *PortalMaker* já o *frame* inferior contém a estrutura do portal podendo agregar desde um único arquivo JSP até um outro *frameset* com mais *frames*.

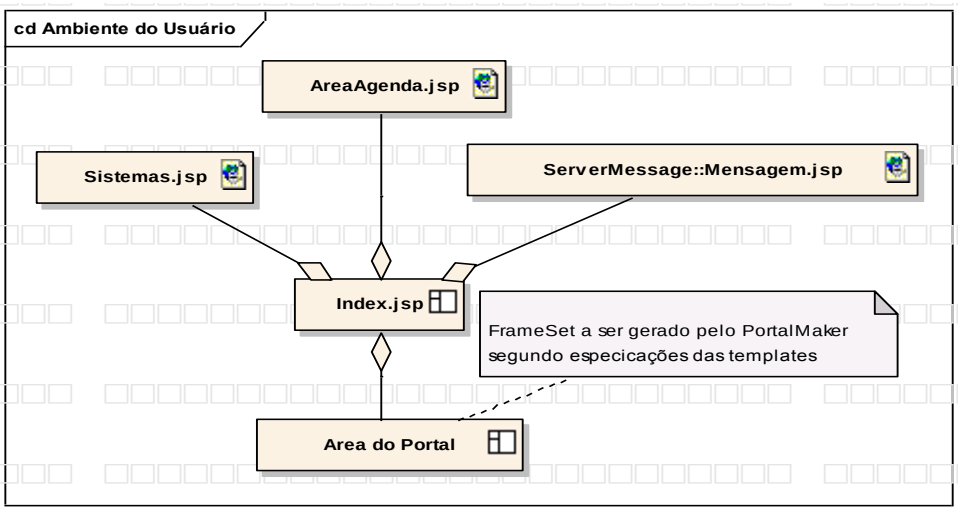

Figura 9. Organograma do módulo do Usuário

3.2.2 O PROCESSO DE GERAÇÃO DOS PORTAIS

Para que um portal seja gerado é preciso conhecer alguns conceitos abordados pelo sistema como: Configuração e Criação dos portais, os conceitos que envolvem a criação e utilização das *templates* e *plugins*, bem como a criação dos mesmos. Os portais gerados utilizam os recursos que o sistema contém e podem evoluir de acordo com a adição e manutenção dos recursos utilizados em sua geração. Tais recursos englobam o *framework VirtualClass*, as *templates*, os *plugins* e os serviços provenientes do *PortalMaker* como os Sistemas de Avisos, Suporte, *Download* e *Uploads*.

3.2.2.1 A PUBLICAÇÃO

O *PortalMaker* possui um banco de dados primário(Figuras 10), disponível em (61), com acesso *web* para o desenvolvedor(lê-se administrador). Tais tabelas são abordadas em detalhes juntamente com os arquivos *jsp* que as gerenciam em um manual de titulo "*PortalMaker* – Gerador de Portais web baseado em templates e plugins".

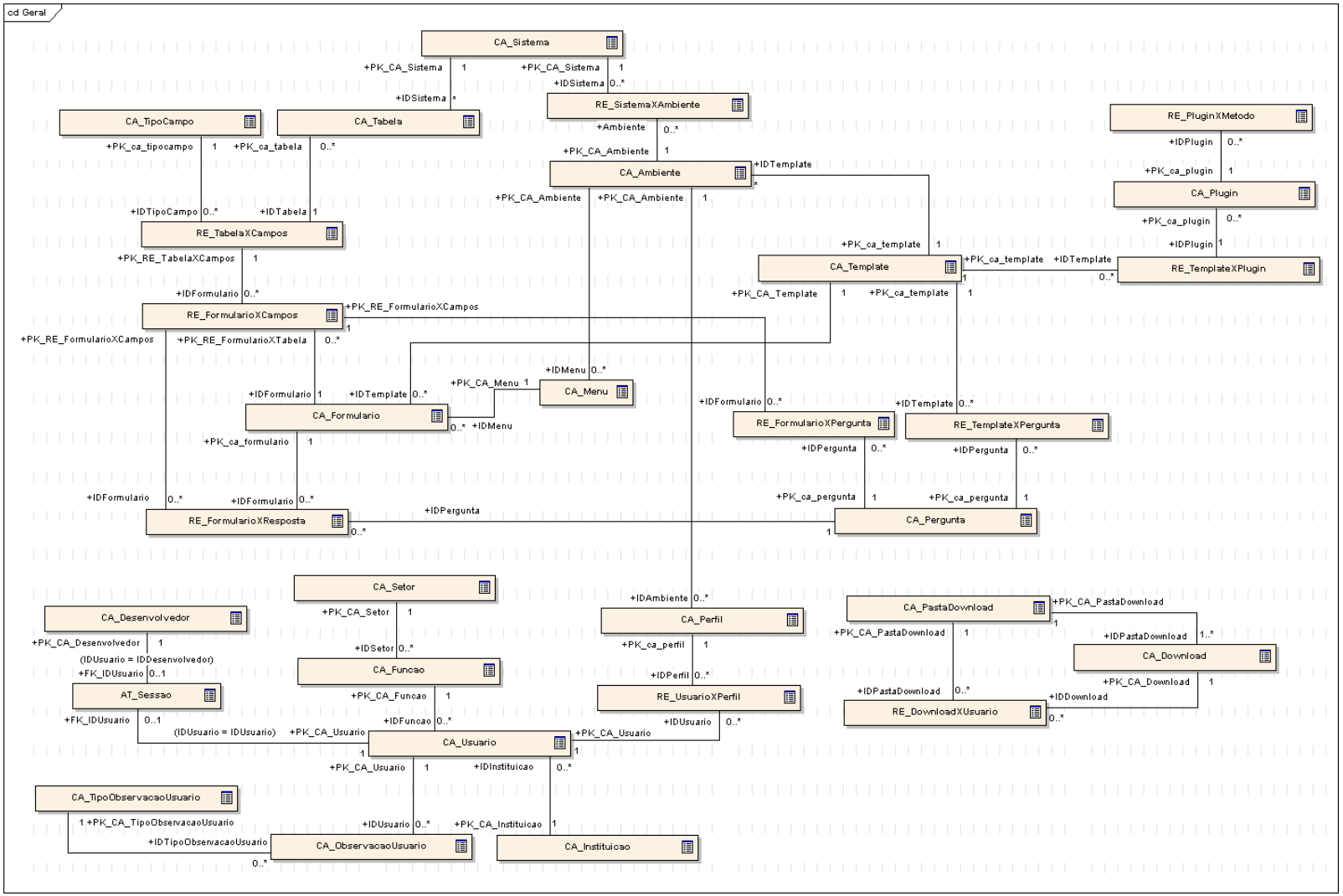

Figura 10. (62)Visão geral do banco de dados do *PortalMaker*

Para a publicação de um portal, o administrador deve efetuar os procedimentos, de modo que o *PortalMaker* trabalha como um "*container*" de portais. Tais procedimentos são apresentados a seguir:

- Acessar o *PortalMaker* com a permissão de Desenvolvedor;
- Cadastrar um portal com o preenchimento dos campos exigidos e da conexão com o banco de dados (exemplo: URL de conexão *jdbc*, usuário, senha, número de conexões dedicadas).
- Cadastrar os módulos (lê-se Ambientes) de cada portal, exemplo: Contabilizador, Cliente, Administrador etc.;
- Cadastrar os Menus de cada módulo, exemplo: Serviços, Cadastro, Relatórios;
- Cadastrar os Perfis (tipos de classificação de usuários) de cada Módulo, exemplo: Administrador Geral, Administrador restrito etc.;
- Cadastrar os usuários que terão acesso ao portal e vincular a eles um ou mais dos perfis cadastrados (o vínculo possui validade e restrição de IP para o acesso).

Executados esses procedimentos teremos um Portal "vazio" que possibilita autenticar usuários e exibir os respectivos perfis após a devida autenticação.

3.2.2.2 O DESENVOLVIMENTO

A definição do banco de dados do portal poderá ser feita de duas maneiras:

1. Cadastrando as tabelas e campos no Sistema;
2. Importando as tabelas e campos de um banco de dados existente quando deve-se configurar uma conexão *jdbc* do portal para o banco de dados em questão.

Para ambos os casos é permitido no cadastro e na manutenção dos campos, definir o título que será apresentado visualmente para o usuário, bem como definir um texto explicativo sobre o preenchimento dos mesmos. Tais textos poderão ser usados durante o processo de geração do portal, utilizando-se as *templates*.

Após a geração das tabelas o desenvolvedor poderá gerar os formulários correspondentes a elas. Esta definição abrange informações do tipo:

- A qual Perfil pertence? Ex.: Administrador Geral;

- A qual Menu pertence? Ex.: Cadastro;
- Qual *Template* utilizar?
- Dependendo da *template* escolhida poderá responder perguntas do tipo:
    - A tabela possuirá tela de cadastro, Consulta e manutenção?
    - Que campos serão de preenchimento do usuário?
    - Quais possuem restrição de valores e modo de apresentação diferente?
    - Que campos devem ser importados de outra tabela e que tabela é essa?

Enfim, definirá todas as informações pertinentes à geração de um formulário que irá gerir as informações das tabelas solicitadas.

3.2.2.3 OS PLUGINS

Referencia-se como plugin todos os recursos (pacotes java, *applets*, *servlets*, *javascripts*, *flash* etc.) que aprimorem ou adicionem novas funcionalidades ao portal. Para adicionar plugins ao portal bastará que o desenvolvedor adicione os mesmos via *PortalMaker* definindo quais as assinaturas (entende-se por assinaturas os métodos que invocam os recursos) e parâmetros necessários para a utilização destes.

Após anexá-los e definir as assinaturas publicadas os *plugins* estarão disponíveis para os formulários a serem gerados e os que já foram (desde que os parâmetros e assinatura sejam os mesmos) usarão a nova implementação do *plugin* sem que nada seja feito no seu código.

Exemplo: um *plugin* poderá ser um *servlet* (*package*, *applet*, *jsp*, ou *portlet*) que invoca a submissão de um *job* a uma grade computacional. Para tanto, bastará anexar esse servlet ao *PortalMaker* e os métodos que o mesmo disponibiliza para os formulários dos portais.

Para comprovar a utilização dos *plugins* foram cadastrados no *PortalMaker* o *framework* do *VirtualClass* e o *P2PGridPlugin* do Portal. Ambos foram utilizados na elaboração do estudo de caso do Portal para a grade computacional P2PGrid.

3.2.2.4 AS TEMPLATES

As *templates,* abordadas neste trabalho, são arquivos HTML com *markups* especiais que são substituídas, durante a geração do portal, por códigos como: *jsp*, valores e

chamadas de *plugins*, sendo que a criação delas devem seguir as especificações de *markups* que o *PortalMaker* utiliza. Caso haja a demanda de novos *markups*, bastará adicionar o tratamento para os mesmos dentro do *JSPMaker*, que é responsável pelo processo de geração do portal, efetuando a fusão dos formulários com as *templates* através da conversão das *markups*. As *templates* do *PortalMaker* podem ser classificadas em 3 tipos:

head

São as *templates* que possuem os códigos iniciais para a geração dos formulários. Geralmente possuem códigos de segurança para o controle da sessão e das variáveis de comunicação com o banco de dados. Podemos definir nessas *templates* as variáveis globais a serem utilizadas no escopo de cada formulário.

body

Em geral, possui o conteúdo de apresentação visual, funcional ou ambos. Corresponde ao corpo do formulário a ser gerado.

bottom

Essas *templates* possuem os códigos ou tratamentos que finalizam os formulários.

Ao compor um formulário baseamos na junção de uma *template* de cada um dos tipos apresentados anteriormente, conforme mostra a figura 11. Podendo haver *templates* vazias com a finalidade de simplesmente compor o trio que gera o formulário.

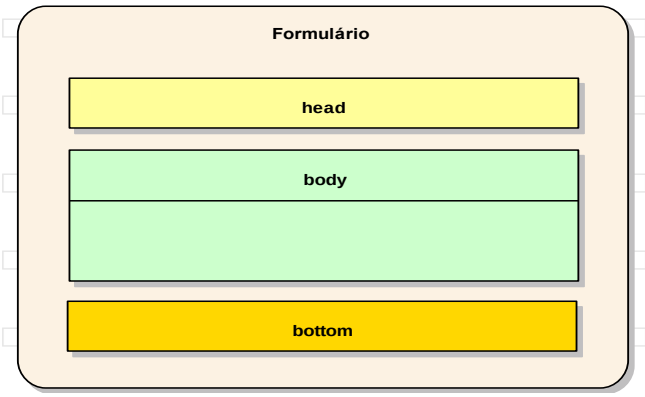

Figura 11. Conjunto de *templates* que compõem um formulário

As templates criadas podem possuir diferentes níveis de abstração de código, permitindo que sejam classificadas de acordo com esse nível:

Templates abstratas

São aquelas que não possuem quase nenhuma linha de código JSP em seu conteúdo. A maior parte do código é gerada a partir dos markups interpretados pelo *JSPMaker* (exemplo: as telas de cadastro e manutenção das informações do portal). Essas *templates* possuem um nível de abstração elevado o que possibilita que sejam utilizadas na criação de diversos portais. As *templates* do tipo *body* seguem esse padrão apesar de existirem situações em que as mesmas não são abstratas.

Templates não abstratas

Sua composição é em maior parte de códigos JSP, geralmente focados em operações específicas para os portais ou seus *plugins*. Essas *templates* possuem um baixo nível de abstração, o que dificulta a sua utilização em outros portais, por serem as suas funcionalidades específicas para um determinado portal. As implementações das *templates* do tipo *head* e *bottom* seguem esse padrão, pois se utilizam quase em sua totalidade apenas de códigos JSP.

Podemos ainda criar *templates* que possuam estruturas onde parte do código é abstrato e parte do código é não-abstrato. Como exemplo, citamos uma situação em que seja necessária uma tela de cadastro e manutenção de informações, e que ao final do cadastro seja enviado um e-mail para um determinado endereço. A parte do cadastro e manutenção fica abstrata a qualquer tabela do portal enquanto a parte de envio do e-mail não.

É importante ressaltar que qualquer *template* adicionada ao *PortalMaker* poderá ser utilizada por todos os portais gerenciados. Assim sendo devemos tomar cuidado ao alterar ou excluir uma *template* existente, pois poderemos estar excluindo uma template que esteja em uso por um ou mais portais.

Implementações futuras

Para uma implementação futura sugerimos que seja realizado um processo de informação ao administrador indicando em que quantidade as *templates* estão sendo utilizadas. Recomenda-se ainda impedir operações de exclusão de *templates* que já se encontram em utilização.

3.2.2.5 A CONSTRUÇÃO DE UM ARQUIVO JSP

A construção de um arquivo JSP pelo *PortalMaker* só é possível mediante o preenchimento das seguintes informações pelo administrador:

1- Cadastros das templates;

2- Cadastros dos formulários e da template que o mesmo utiliza;

3-Cadastros das Tabelas e Campos que o formulário irá trabalhar;

4-Definição da ordem de apresentação e interação dos campos da Tabela para o formulário;

Após a definição destas informações o *PortalMaker* irá sempre que solicitado pelo administrador, encaminhar a solicitação de construção dos arquivos JSPs do portal para o *JSPMaker*. Tal solicitação é apresentada no diagrama de seqüência da figura 12;

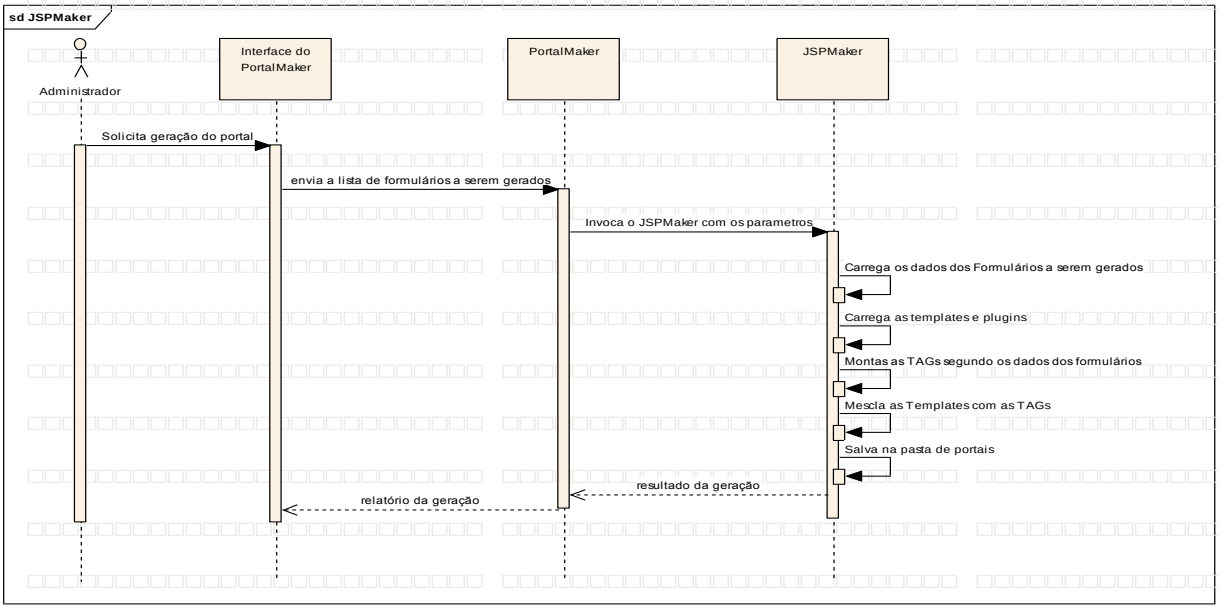

Figura 12. Diagrama de Seqüência da geração de portais

Após o cadastro do formulário o administrador deve especificar que tabelas e campos o formulário irá utilizar. Em seguida especifica os critérios a serem seguidos por cada campo (isso visualmente). No cadastro do formulário já estará definida a *template* a ser utilizada. O *PortalMaker* registra o formulário em uma tabela de *logs*. (gerando um *idlog* para a tabela o qual será utilizado no registro das operações de banco de dados). O *idlog* é utilizado juntamente com a *template* sendo que para cada chamada de inserção no banco de dados temos uma chamada para registro do *log* com o identificador daquele formulário.

Quando solicitado pelo administrador o *PortalMaker* invoca uma instância do *JSPMaker* que por sua vez efetua a geração do portal. Durante esta geração são adicionados às *templates* definidas para o cabeçalho, o corpo e o rodapé dos formulários, mesclando

os seus *markups* pelos respectivos códigos *JSPs* e, ao final do processo, gerando o arquivo *JSP*, publicando-o na pasta específica do portal.

É importante ressaltar que o *PortalMaker* nos possibilita gerar portais para qualquer Grid. Para isso necessitamos de um *plugin* que saiba "conversar com a grade". A manutenção dos portais, bem como suas atualizações, podem ocorrer sem que haja necessidade de conhecer os códigos utilizados e, nem sequer possuir conhecimento sobre programação. Em adição, podemos considerar que a facilidade de administrar, adaptar e gerar novos recursos e funcionalidades para os portais é enorme. O "programador" só será necessário para criar novas *templates* e *plugins*, sendo que o nível de abstração para *templates* é muito grande, o que exige do desenvolvedor apenas noções de algoritmo e conhecimento dos *markups* utilizados pelos portais. Mesmo havendo necessidade do desenvolvimento de novos *markups* a estrutura para programação e o *framework* utilizado facilitam e aperfeiçoam esse processo.

Implementações futuras

Para uma implementação futura, sugerimos que exista para cada *template* um *JSPMaker* específico que compõe seus *markups*; caso o não seja encontrado tratamento para todos eles o *JSPMaker* utiliza o gerador da *markups p*adrão (o atual *JSPMaker*). Tal abordagem favorece a simplicidade e flexibilidade da criação de templates e do *parser* dos novos *markups*.

3.3 PROJETO PROPOSTO PARA O P2PGRID

O P2PGrid foi concebido com enfoque no algoritmo de balanceamento de carga, sendo necessário ainda serem desenvolvidos e aprimorados muitos componentes que o compõem, adicionando a ele várias funcionalidades de segurança e gerência dos recursos. Baseado nessa premissa, este projeto visa a trabalhar com alguns desses tópicos não abordados pelo P2PGrid. São eles:

- Gerenciamento e registro das submissões;
- Contabilização de submissões;
- Controle de reputação e refutação;
- Controle de acesso à grade;

- Credenciamento de nodos e usuários;
- Mecanismo de autenticação distribuído;
- Mecanismo de monitoramento de carga;
- Avaliação da melhor formação de nodos adjacentes para a grade;
- Mecanismos de segurança na comunicação entre portal, nodos e processos.

Com base nos itens apresentados anteriormente este trabalho optou por abordar os tópicos de gerenciamento e registro de submissões, controle de acesso à grade e os mecanismos de segurança na comunicação entre o portal e os nodos, deixando para futuros trabalhos os outros tópicos.

3.3.1 A ARQUITETURA

A construção do P2PGrid foi realizada mediante o desenvolvimento de alguns componentes que gerenciam operações em recursos da grade. Esses se encontram distribuídos em camadas (Superior, Intermediária e Base) conforme descrito em (1)(pág 24).

Camada Superior - componentes responsáveis pelos serviços de controle que mantêm a grade ativa.

Camada Base – componentes que possibilitam o funcionamento da grade, como: máquina virtual java, recursos do sistema operacional, protocolos de comunicação, dispositivos de redes, computadores, dentre outros.

Camada intermediária – componentes que gerenciam serviços específicos, como: *acoplador*, *escalonador*, *sistema de submissão* (disparador e executor) e *monitor*.

Nesta camada encontram-se os componentes que compõem o P2PGrid, sendo eles:

A - CIS (Sistema de Comunicação e Informação) é o mediador da troca de mensagem entre os nodos.

B - Acoplador (coupler) responde pelos serviços que mantêm as ligações lógicas entre os recursos.

C - Escalonador (scheduler) responde pela alocação um *job a* algum nodo disponível capaz de executá-lo.

D - Submissor responde por disparar e executar remotamente trabalhos enviados pelos usuários.

E – Monitor responde pela avaliação da carga do nodo bem como da atualização desses valores para todo o Grid.

Os componentes e camadas abordados estão representados na Figura 13, onde foram adicionados neste trabalho dois novos componentes na camada intermediária: o *Configurator* e o *Updater*, descritos a seguir:

F – Configurator responde pelo carregamento dos dados de inicialização do nodo a partir de um arquivo de configuração assinado. Informações como serviços ativos, a que nodos podem solicitar autenticação e quais pode submeter *jobs*.

G – Updater responde pela atualização e notificação de informações do nodo e o Portal. Atualizações de dados e da composição da grade são informadas ao Portal através deste componente que se comunica com o *ServerUpdater* do Portal. O *ServerUpdater* é um componente do Portal que recebe as requisições dos nodos e as trata devolvendo as informações solicitadas. Este mecanismo é descrito posteriormente.

A Arquitetura do P2PGrid sofre ainda uma subdivisão na camada superior onde agora temos duas sub-camadas sendo elas: *Application* e *PluginManager*, descritas a seguir:

PluginManager comporta os sistemas dos desenvolvedores que trabalham como *middleware* entre a aplicação final do usuário e a grade. Como exemplo citamos o *PortalMaker*, proposto e desenvolvido neste trabalho.

Application contém as aplicações dos usuários finais que utilizam a grade. Como exemplo desenvolvemos um Portal do P2PGrid utilizando o *PortalMaker* da camada inferior.

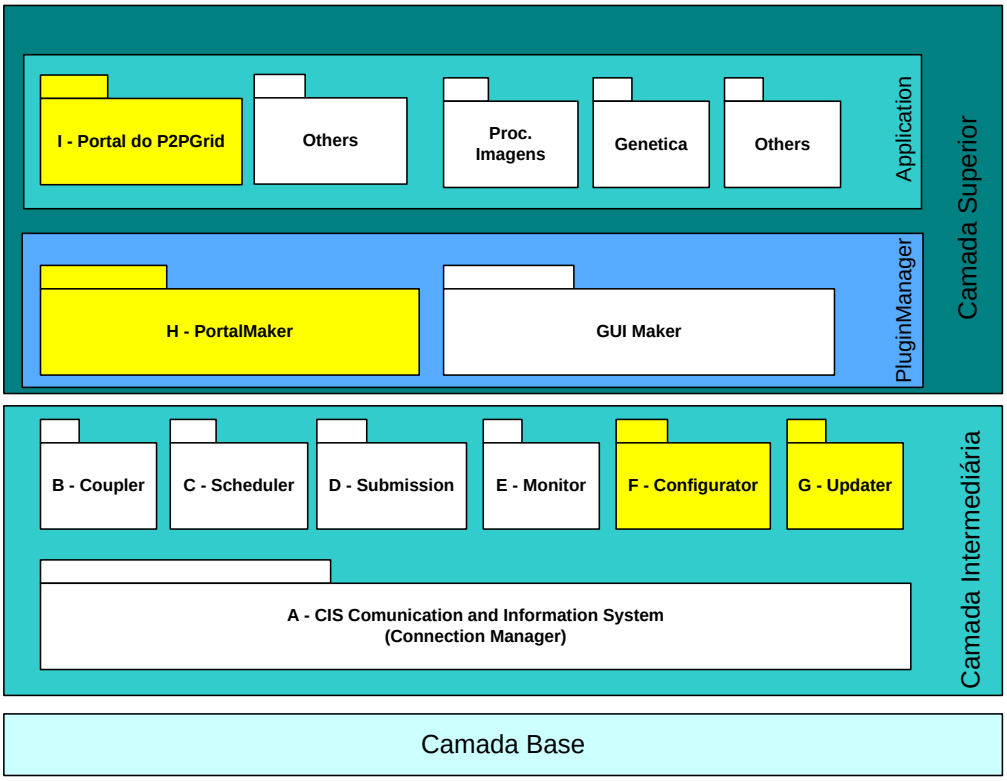

Figura 13. A nova arquitetura em camadas do P2PGrid

3.3.2 O SISTEMA DE COMUNICAÇÃO

O Sistema de comunicação do P2PGrid descrito em (1) (pág 28), sofreu alterações no componente CIS, onde foram adicionados dois componentes: *SSLConnectorFactory* que possibilita a criação de conexões SSL e o *Comunicator* que possui métodos de abertura e gerenciamento de conexões SSL ilustrados na figura 14.

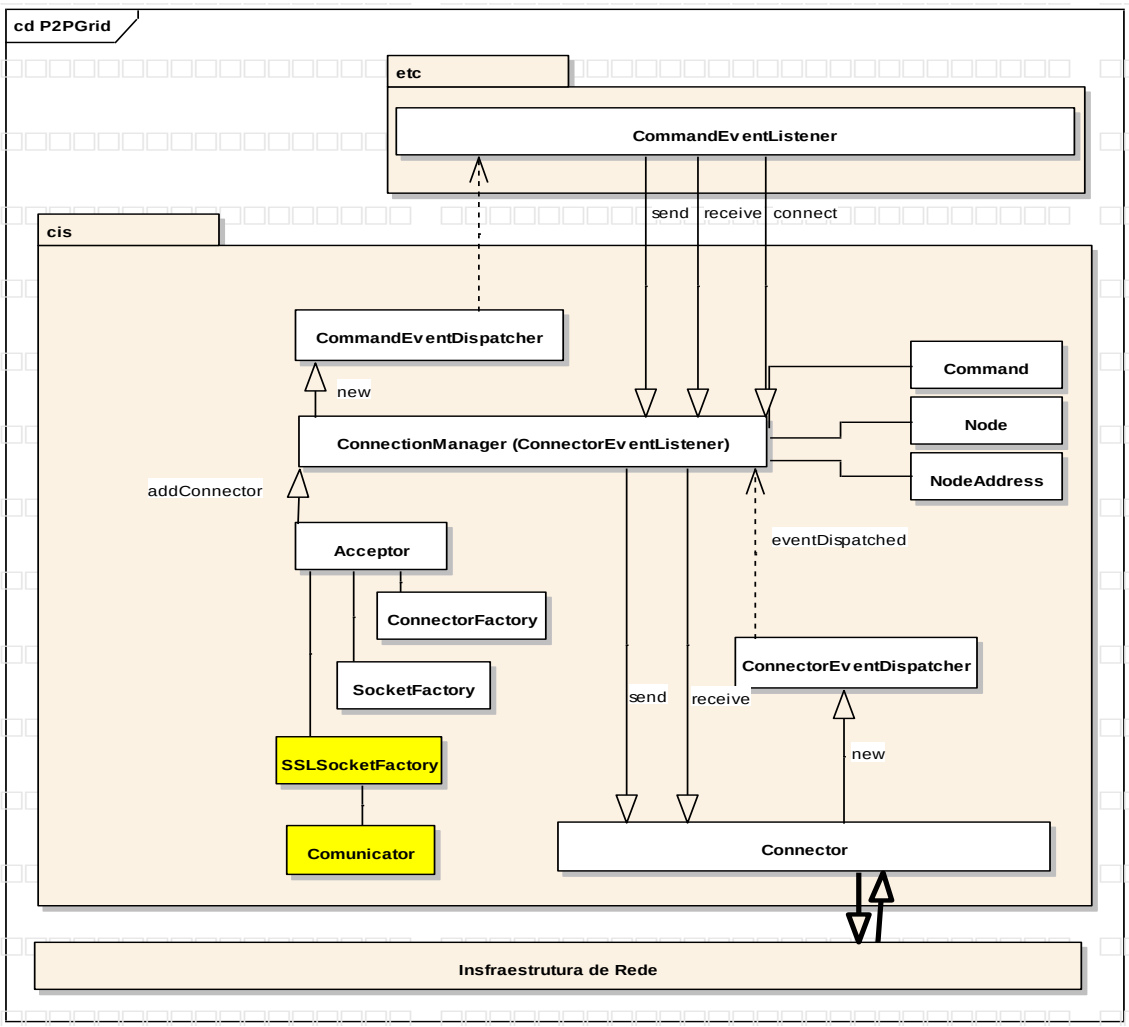

Figura 14. Sistema de Comunicação do P2PGrid

3.3.3 MECANISMO DE CREDENCIAMENTO

A criação ou a adoção de um mecanismo de credenciamento para o P2PGrid é de singular importância para o controle da expansão da grade de forma controlada. Sendo assim, especificamos alguns mecanismos e procedimentos de credenciamento para o P2PGrid, englobando tanto o credenciamento dos nodos da grade, como o credenciamento dos usuários do Portal.

A seguir detalharemos cada um destes mecanismos.

3.3.3.1 CREDENCIAMENTO DE NODOS

Por se tratar de uma grade *peer-to-peer*, muitos fatores de segurança devem ser considerados ao credenciar seus nodos. Assim sendo, modelamos um mecanismo de credenciamento que suporta um posterior processo de autenticação distribuída de

modo a favorecer a computação *peer-to-peer*, sem promover gargalos e dependências centralizadas o que comprometeria o conceito do P2PGrid.

Com base nos estudos feitos sobre as técnicas de credenciamento utilizadas pelas grades optamos pela utilização de um processo de credenciamento via Portal, mediado por uma senha de administrador da grade, onde os nodos pertencentes à mesma serão cadastrados e armazenados em um banco de dados centralizado.

Procedimentos para o credenciamento de nodos:

- Acessar o Portal do P2PGrid;
- Validar o acesso à área de administração do Portal;
- Selecionar a opção de credenciamento de um novo nodo;
- Preencher a ficha cadastral no nodo;
- Confirmar a solicitação de credenciamento.

A partir daí o Portal irá:

- Registrar o nodo no banco de dados;
- Disponibilizar o *download* de um *pack* com o programa do P2PGrid e os arquivos necessários para sua execução.

É importante ressaltar que o Portal possui uma chave privada com a qual assina todos os arquivos de configuração do nodo de modo a impedir que suas informações sejam alteradas evitando possíveis fraudes e credenciamentos indevidos por terceiros.

O *Pack* montado para o nodo contém:

1. **Chave pública do Portal**

   Esta chave é utilizada para criptografar as mensagens e informações de credenciais enviadas ao Portal.

2. **Arquivo de Configuração**

   Trata-se de um arquivo XML que é carregado durante a ativação do P2PGrid no nodo. Contém informações de serviços ativos e a configuração a ser adotada pelo nodo. É composto por dois *markups*: *config* e *sign*. O *markup config* contém os *markups* de configuração, são eles:

<CONFIG> - *Markup* que contém os *markups* de configuração do nodo.

Conteúdo do *markup config*:

<DEBUG> - indica se o nodo está em modo procurar de erros. Esse *markup* contém *true* ou *false* indicando se deve ou não apresentar na tela os *logs* de execução.

<TYPE> - indica o tipo de chaves que é utilizado no nodo. Padrão:JKS.

<IDNODE> - Contém o identificador único do nodo na grade. Tal identificador é obtido através da fórmula: P+<ID do Portal no banco de dados>+N+<ID do nodo no banco de dados>. Cada nodo possui um identificador único fornecido automaticamente. Já o identificador do Portal também é único, porém sua distribuição é manual. É utilizado como *alias* para referenciar a chave pública no nodo na grade.

<IP> - Contém o IP do nodo cadastrado no Portal.

<PORT> - Contém a porta de execução do nodo.

<PORTAL_IP> - IP de acesso ao *ServerUpdater* do Portal que gerencia o nodo.

<PORTAL_PORT> - Porta de acesso ao *ServerUpdater* do Portal que o gerencia.

<PORTAL_PUBLIC_KEY> - Identificador do Portal na grade, obtido através da fórmula: P+<ID do Portal no banco de dados>. Esse identificador deve ser único entre os portais que gerenciam a grade. Utilizado para referenciar o *alias* da chave pública do Portal.

<AUTENTICATOR> - indica se o nodo pode autenticar os demais nodos da grade. Valores: *true* ou *false*.

<SUBMIT> - indica se o nodo pode receber solicitações de submissão de *jobs*. Valores: *true* ou *false*.

<AUTENTICATION_MUTUAL> - indica se as conexões estabelecidas entre os nodos devem ser autenticadas mutuamente ou apenas pelo nodo requisitado. Valores: *true* ou *false*.

<THRESHOLD> - indica o valor de variação padrão do monitor de carga do nodo, quando o valor ultrapassar o estabelecido a atualização de carga do nodo e realizada. Padrão:20.

<MONITOR> - indica o tempo, em milissegundos, de reavaliação da carga do nodo.

<KEYSTORE> - indica o arquivo que contém o *keystore* de chaves do nodo. Neste arquivo encontram-se as chaves públicas do Portal e dos demais nodos além da chave privada do nodo em questão. Padrão <IDNODE>+".ks".

<AUTENTICATORS> - indica o caminho do arquivo assinado pelo Portal que contém a lista de nodos que podem autenticar na grade. Este arquivo é composto por uma estrutura XML de dois markups: *Autenticators* e *Sign*.

<PUBLIC_KEYS> - indica onde serão alojados os certificados com as chaves públicas recebidas a serem incorporadas no *keystore* do nodo.

<URL_RESULTS> - local onde o resultado das submissões fica armazenado.

<MAX_ADJACENTS> - indica o número máximo de nós adjacentes ao nodo em questão. Valor Padrão: 3.

<SIGN> - contém a assinatura do conteúdo da TAG *config* pela chave privada do Portal.

3. P2PGrid.jar

Arquivo que contém todos os componentes necessários para o funcionamento do nodo.

4. P2PGrid.bat

Arquivo de execução em lote para ambiente Windows. Realiza as operações de:

- Geração do *keystore* do nodo;
- Geração da chave privada do nodo;
- Exportação da chave pública do nodo;
- Inicialização do nodo do P2PGrid.

As duas primeiras operações são realizadas sempre que o nodo necessita ser configurado antes de ser ativado. Em geral trata-se da primeira execução de um nodo.

5. P2PGrid.sh

Funcionalidade idêntica ao do P2PGrid.bat porém o script foi elaborado para rodar em ambiente Linux.

6. **P2PGrid.ks**

   Arquivo em formato XML, não assinado, composto por dois markups: *keystore_password* e *private_key_password*.

   Após a criação do *keystore* e da chave privada do nodo é preciso adicionar a senha de acesso a ambos neste arquivo.

Fazem parte do *pack* outros arquivos que são criados e carregados pelo nodo durante a inicialização do mesmo, são eles:

1. **Keystore do Nodo**

   Arquivo que contém a chave privada do nodo, a chave pública do Portal e as chaves públicas dos demais nodos.

2. **Chave privada do nodo**

Esta chave será utilizada para descriptografar as mensagens de comunicação e transmissão enviadas pelos demais nodos, cuja criptografia tenha sido realizada com a chave pública correspondente.

3. **Chaves públicas dos nodos já cadastrados**

Estas chaves servem para criptografar e validar as mensagens enviadas entre os nodos. São obtidas sempre que o nodo é inicializado através da chamada do *Updater* ao *ServerUpdater* do Portal que o gerencia. Essas chaves são transmitidas via conexão segura e mutuamente autenticada onde o nodo envia a requisição de atualização e o *ServerUpdater* verifica junto ao banco de dados se existem chaves a serem repassadas ao nodo requisitante. As chaves, caso existam, são repassadas e armazenadas no diretório configurado e adicionadas no *keystore* do nodo com o alias dos nodos aos quais pertencem.

4. **Lista dos nodos autenticadores**

Esta lista é assinada pelo Portal que a gerencia e contém o endereço dos nodos que podem efetuar o processo de validação e autenticação da grade. A estrutura desse arquivo é formada por dois *markups* XML: *autenticators* e *sign*. O *markup autenticators* contém a lista de nodos autenticadores organizada, sendo que para cada nodo autenticador temos uma entrada do *markup autenticator*. Cada *markup autenticador* é

composto por *markups*: *idnode*, *ip* e *port*. O *markup sign* possui a assinatura do conteúdo do *markup autenticators* pelo do Portal que gerencia o nodo.

Sempre que o nodo é inicializado uma requisição de atualização através do *Updater* para o *ServerUpdater* do Portal. O *ServerUpdater* compõe, caso exista, a lista de nodos autenticadores que o nodo ainda não possui e envia ao nodo juntamente com a nova assinatura da lista completa.

É importante ressaltar que uma grade deve possuir pelo menos um nodo autenticador. Uma grade pode também ser formada apenas por nodos autenticadores, seguindo a política de autenticação descrita a seguir:

- O início de uma grade poderá ser feito única e exclusivamente por um nodo autenticador;
- Todo e qualquer nodo que se conecte à grade deverá passar pelo processo de autenticação de um nodo autenticador.
- Um nodo autenticador efetua sua autenticação através de outro nodo autenticador, exceto quando o nodo em questão for o primeiro da grade;

5. P2PGrid.autr

Arquivo XML que registra a reputação dos nodos autenticadores os sucessos e as falhas. A reputação é medida pela diferença entre as autenticações bem sucedidas e as falhas de autenticação. Sua estrutura é formada por *dois markups*: *reputations* e *sign*. O *markup reputations* possui, para cada incidência do *markup autenticator*, no arquivo P2PGrid.aut, um registro do *markup* com o *idnode*, o qual é composto pelos *markups*:*sucess*, *failure* e *reputation*. Esse arquivo é carregado durante o processo de inicialização do nodo, onde é montada uma lista de autenticadores ordenados, com base no índice de reputação registrado nele. É assinado pela chave privada do nodo por questões de integridade. O Conteúdo dos *markups sucess* e *failure* corresponde ao número de vezes que o nodo tentou autenticar e o processo teve êxito ou falha para com um dado autenticador.

3.3.3.2 CREDENCIAMENTO DE USUÁRIOS

O processo de credenciamento de usuários deve prover confidencialidade, autenticidade, integridade e controle. A Confidencialidade, ou sigilo, é "garantida"

criptografando as mensagens enviadas com a chave pública do Portal em que a informação é dirigida. Deste modo, somente este Portal, de posse da chave privada correspondente (o destinatário correto), é que consegue descriptografar a mensagem. A Autenticidade consiste em garantir a veracidade das informações recebidas pelo Portal. Por meio da autenticação é possível confirmar a identidade de um usuário ou entidade que utiliza o Portal. A Integridade sinaliza a conformidade dos dados transmitidos pelo usuário com os recebidos pelo Portal. A manutenção da integridade pressupõe a garantia de não violação dos dados, com intuito de alteração, gravação ou exclusão, seja ela proposital ou acidental. O Controle é provido mediante o gerenciamento dos usuários do Portal, bem como de seus acessos e utilização dos serviços do P2PGrid. Com base no Controle podemos, ainda, mapear todas as atividades dos usuários como também quantificá-las e qualificá-las.

A adoção da política de que apenas os usuários credenciados poderão utilizar a grade, se faz necessária, para alcançar maior controle e permitir o gerenciamento das submissões. A idéia inicial do P2PGrid era de que todo nodo poderia submeter *jobs* ao Grid, porém esta idéia torna o controle das submissões um tanto quanto complexo e mais suscetível a falhas de segurança, uma vez que a dependência das informações fornecidas pelos nodos será maior. Sendo assim, a idéia de centralizar as submissões através de um Portal, eleva a segurança e o controle do uso da grade, além de seguir a tendência das grades em popularizar e facilitar a sua utilização através dos Portais.

Sendo assim, propõe-se um mecanismo centralizado de credenciamento de forma semelhante ao proposto para o credenciamento de nodos.

Procedimentos para o credenciamento de usuários:

- Acessar o Portal do P2PGrid;
- Validar o acesso à área de administração do Portal;
- Selecionar a opção de credenciamento de um novo usuário;
- Preencher a ficha cadastral do usuário;
- Confirmar a solicitação de credenciamento.

A partir daí o Portal irá:

- Validar a existência do usuário no banco de dados;

- Registrar o usuário no banco de dados;
- Avaliar a ficha do usuário deferindo ou indeferindo o cadastro. Esse processo é realizado pelo usuário administrador ou outro usuário que possua essa função.
- Enviar e-mail para o usuário informando a ativação da senha.

Com a centralização do controle de usuários espera-se obter um grau de segurança elevado, porém criamos um *fail point* para a grade no que engloba a submissão de *jobs*. É importante ressaltar que uma falha de acesso ao Portal não impede a formação da grade, mas apenas a submissão de novos *jobs* e o acompanhamento dos já submetidos. Uma única grade pode conter inúmeros Portais o que pode favorecer a implantação de um serviço de Portal redundante de alta disponibilidade, reduzindo a possibilidade de inacessibilidade do Portal.

3.3.4 MECANISMO DE AUTENTICAÇÃO

O P2PGrid, na sua atual composição, não possui um mecanismo seguro de autenticação de nodos. Sendo que, para um nodo fazer parte do P2PGrid, basta que o mesmo tenha o programa da grade e saiba o endereço de um nodo que esteja autenticado à mesma. Existem vários métodos de autenticação que podem ser aplicados ao P2PGrid, porém todos convergem em algum momento à centralização de alguma parte do processo. Como a idéia do P2PGrid é de uma estrutura *peer-to-peer*, não poderíamos utilizar mecanismos de autenticação que resultem na criação de *fail points*. Sendo assim, este trabalho propõe um mecanismo de autenticação distribuído, cuja possibilidade de existir *fail points* de autenticação estará diretamente ligada à composição da grade estabelecida pelo administrador do Portal do P2PGrid. A seguir detalharemos as abordagens proposta para a autenticação de nodos e usuários.

3.3.4.1 AUTENTICAÇÃO DE NODOS

Um mecanismo de autenticação *peer-to-peer*, permitirá que os nodos se autentiquem através de outros nodos da grade. Para que isto seja possível definimos uma característica específica que possibilita a determinados nodos efetuarem o processo de autenticação. A proposta deste mecanismo reorganiza os nodos da grade em três grupos: os nodos autenticadores, os nodos comuns e os nodos submissores. É

importante lembrar que um nodo pode pertencer a um ou mais grupos conforme a formação pretendida pelo administrador da grade.

Os nodos comuns: Nodos que provêem recursos à grade com a única finalidade de cedê-los, aumentando seu poder computacional. Tais nodos não podem submeter *jobs*, uma vez que este trabalho propõe que a submissão de *jobs* seja mediada por Portais e não pelos nodos.

Os nodos autenticadores: Nodos comuns que além de prover recursos à grade são capazes de autenticar as entradas de novos nodos.

Os nodos de submissão: Nodos que possuem o serviço de submissão ativo à grade. Este nodo aceita requisições de submissão oriunda do Portal que o gerencia.

A autenticação de nodos só poderá ser estabelecida a partir da existência de pelo menos um nodo autenticador pertencente à grade. A requisição de autenticação pode partir de um nodo comum ou autenticador. O nodo requisitante solicita autenticação a um dos nodos da lista, de nodos autenticadores, fornecida e atualizada pelo Portal. O nodo autenticador requisitado validará a autenticação do nodo requisitante através da tentativa de criação de uma conexão SSL, mutuamente autenticada, de forma que tanto o nodo requisitante como o requisitado devem validar a chave pública do outro. Caso a conexão seja criada com sucesso, subentende-se que ambos os nodos são quem dizem ser, e autoriza a entrada do nodo solicitante à grade. É importante a existência da autenticação mútua para que o nodo autenticador também seja avaliado e para que o requisitante saiba que o nodo ao qual solicitou autenticação é autorizado pela grade.

3.3.4.2 ORGANIZAÇÃO DOS NODOS

A existência da classificação dos nodos não indica que a grade não seja formada por nodos iguais, mas sim que alguns nodos dela irão possuir serviços ativos e outros não. Para um nodo se tornar um autenticador, bastará ao Portal que o gerencia anunciar este serviço ao nodo que receberá a notificação de seu novo serviço, e terá início o processo de autenticação, sem sequer precisar instalar outro programa ou extensão do P2PGrid.

Um P2PGrid pode ser formado por diversas combinações de nodos e Portais. Essas combinações possibilitam várias composições de grades com diferentes características

no que tange à segurança, à disponibilidade da mesma e até mesmo do Portal. Algumas das diferentes formações do P2PGrid são apresentadas a seguir. Para a ilustração das formações apresentadas os nodos foram configurados para aceitarem no máximo dois nodos adjacentes.

FORMAÇÃO 1P1S1ANC

Essa formação compõe uma grade com autenticação centralizada o que cria um *fail point*. Outros *fail points* existentes estão no Portal e no nodo Submissor que concentram respectivamente a atualização dos nodos, e suas submissões. Esta não é uma formação recomendada para uma grade *peer-to-peer,* porém recomenda-se utilizar essa formação quando não se tem nodos confiáveis para autenticação. Essa formação é identificada pela sigla 1P1S1ANC (1 Portal, 1 nodo submissor, 1 nodo Autenticador e N nodos Comuns). Para melhor entendimento apresentamos na figura 15 uma ilustração dessa formação.

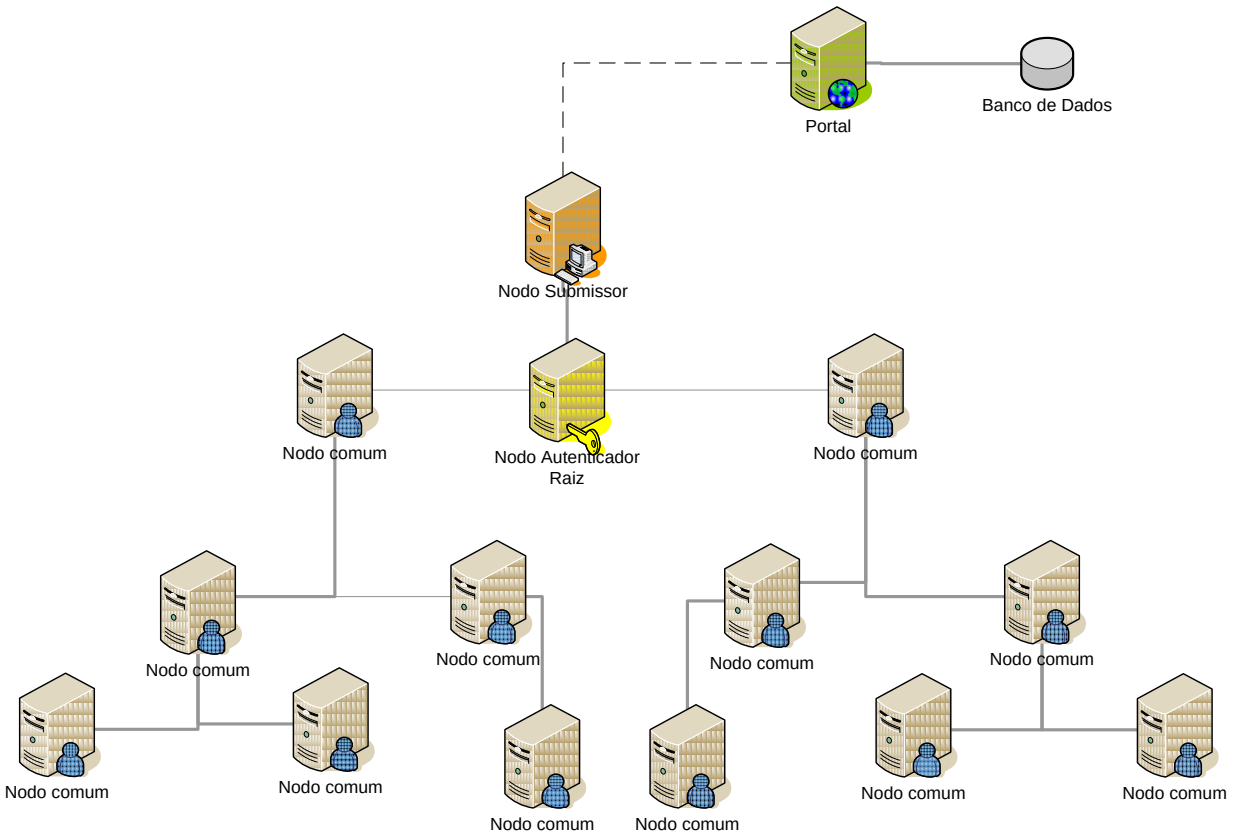

Figura 15. Formação 1P1S1ANC

Na formação da figura 15 podemos perceber que o Portal possui acesso à grade através da comunicação, sob demanda, com o nodo Submissor. A não existência de um nodo submissor impede que o Portal submeta *jobs* à grade.

FORMAÇÃO NP1SMAPC

Essa formação compõe uma grade com ponto de submissão centralizada o que cria um *fail point*. A alta disponibilidade dos Portais favorece a grade no que tange à integração dos recursos como ilustrado na figura 16. O processo de autenticação se torna distribuído devido à existência de vários autenticadores que garantiram a disponibilidade do serviço de autenticação. Esta não é uma formação recomendada para uma grade *peer-to-peer,* porém recomenda-se utilizar essa formação quando não se tem nodos confiáveis para o processo de submissão. Essa formação é identificada pela sigla NP1SMAPC (N Portais, 1 nodo Submissor, M nodos Autenticadores e P nodos Comuns).

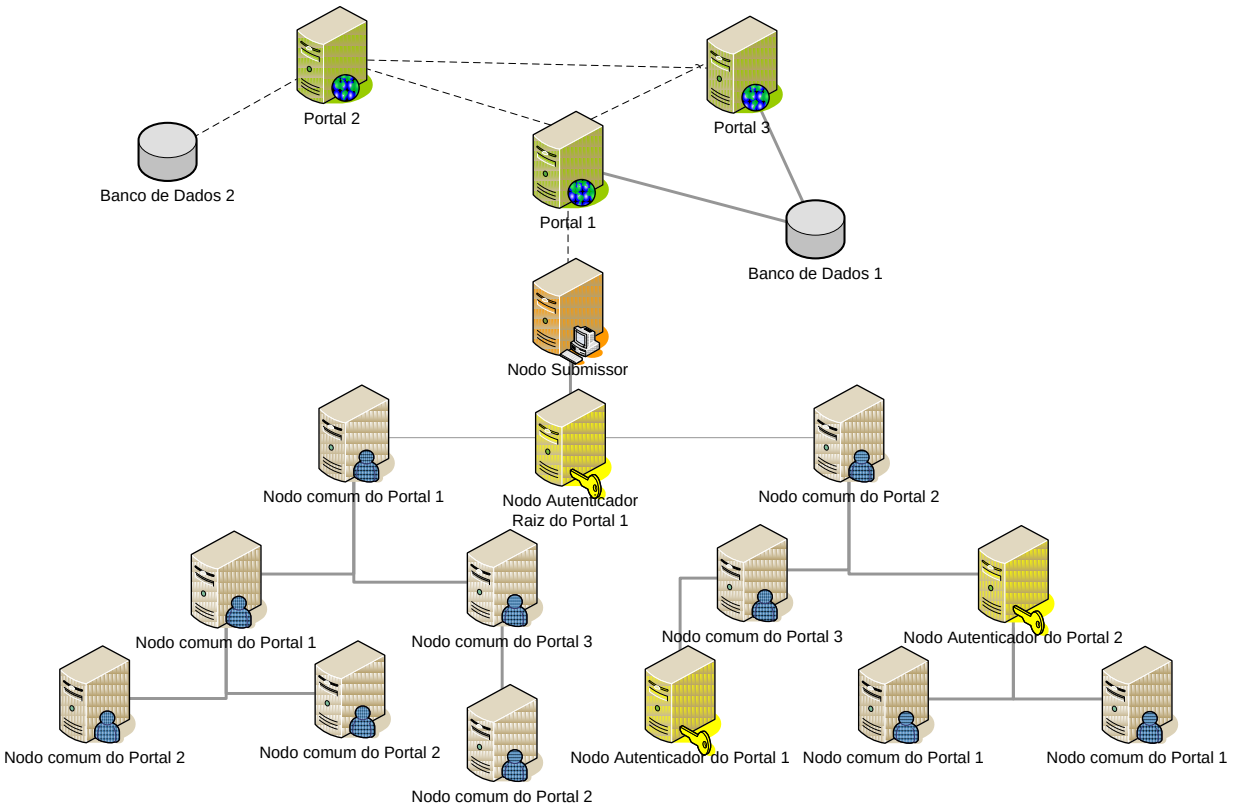

Figura 16. Formação NP1SMAPC

Nesta formação percebe-se a existência de ligações esporádicas entre os Portais. As ligações são realizadas pelo *ServerUpdater* sempre que um Portal possui mudança em sua estrutura e necessita repassar suas informações para os demais nodos. Essas informações compartilhadas são: Mudança do *status* dos nodos que gerencia, envio de chaves e notificação de novos nodos e portais cadastrados. O "Portal 2" possui um banco de dados exclusivo onde fica o registro de todos os nodos que foram registrados através dele. Já os Portais 1 e 3 compartilham um único banco de dados.

FORMAÇÃO 1P1SMANC

A formação, ilustrada na figura 17, cria dois *fail points* em decorrência da centralização do processo de submissão e do acesso ao Portal, porém provê uma maior disponibilidade de autenticação devido à existência de diversos nodos autenticadores.

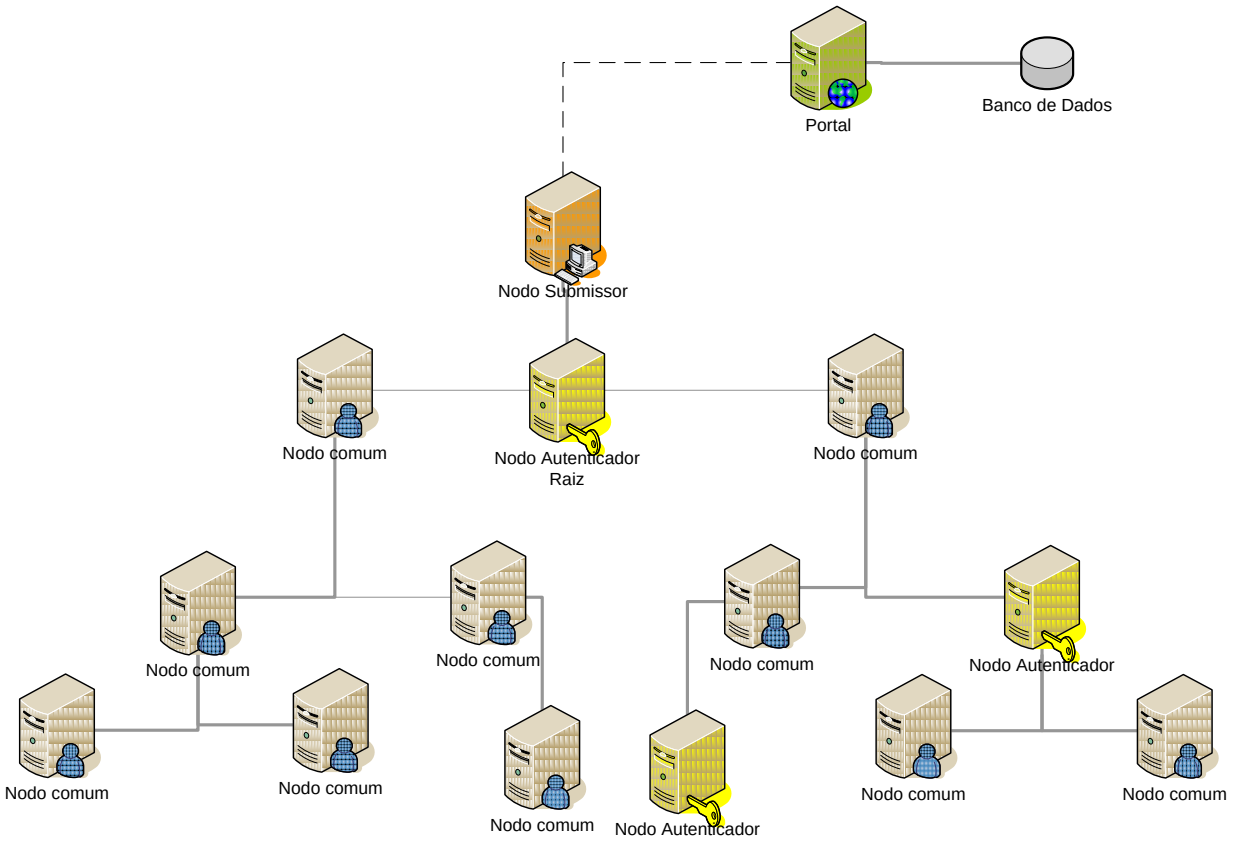

Figura 17. Formação 1P1SMANC

FORMAÇÃO MPNSPAQC

A formação da Figura 18 ilustra a convergência e integração de diversos Portais e seus recursos. Tal formação possibilita estender o poder computacional da grade e reduzir os *fail points* de autenticação, submissão e de acesso ao Portal. Essa formação é obtida ao promover a integração de grades como, por exemplo, as de instituições diferentes. Costuma surgir quando há a aplicação da *Economy Grid* no processo de composição da grade.

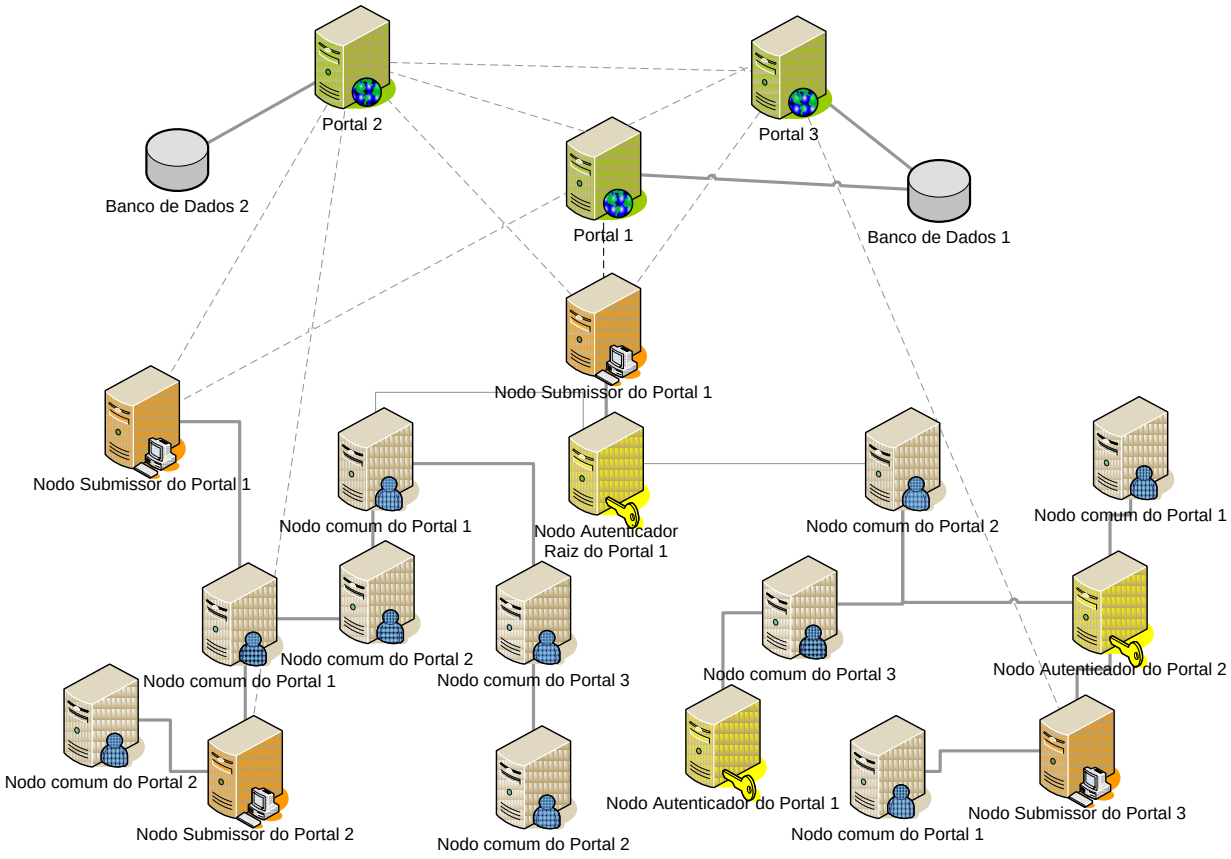

Figura 18. Formação MPNSPAQC

É importante ressaltar que essa formação pode também ser obtida por uma única grade, com a finalidade de gerenciar sub-grades dentro da mesma instituição, de modo a promover controle e alta disponibilidade da grade.

FORMAÇÃO MPNSPA0C

Essa formação é a que mais promove a disponibilidade da grade. É adotada quando todos os nodos são confiáveis o suficiente para serem nodos autenticadores e

submissores e quando a necessidade de alta disponibilidade é atendida à medida que agrega mais Portais à grade. A figura 19, apresentada a seguir, ilustra a formação de uma grade com vários Portais e apenas nodos com o serviço de autenticação e submissão ativo, o que reduz consideravelmente a possível inacessibilidade dos nodos à grade e da submissão dos *jobs* à mesma.

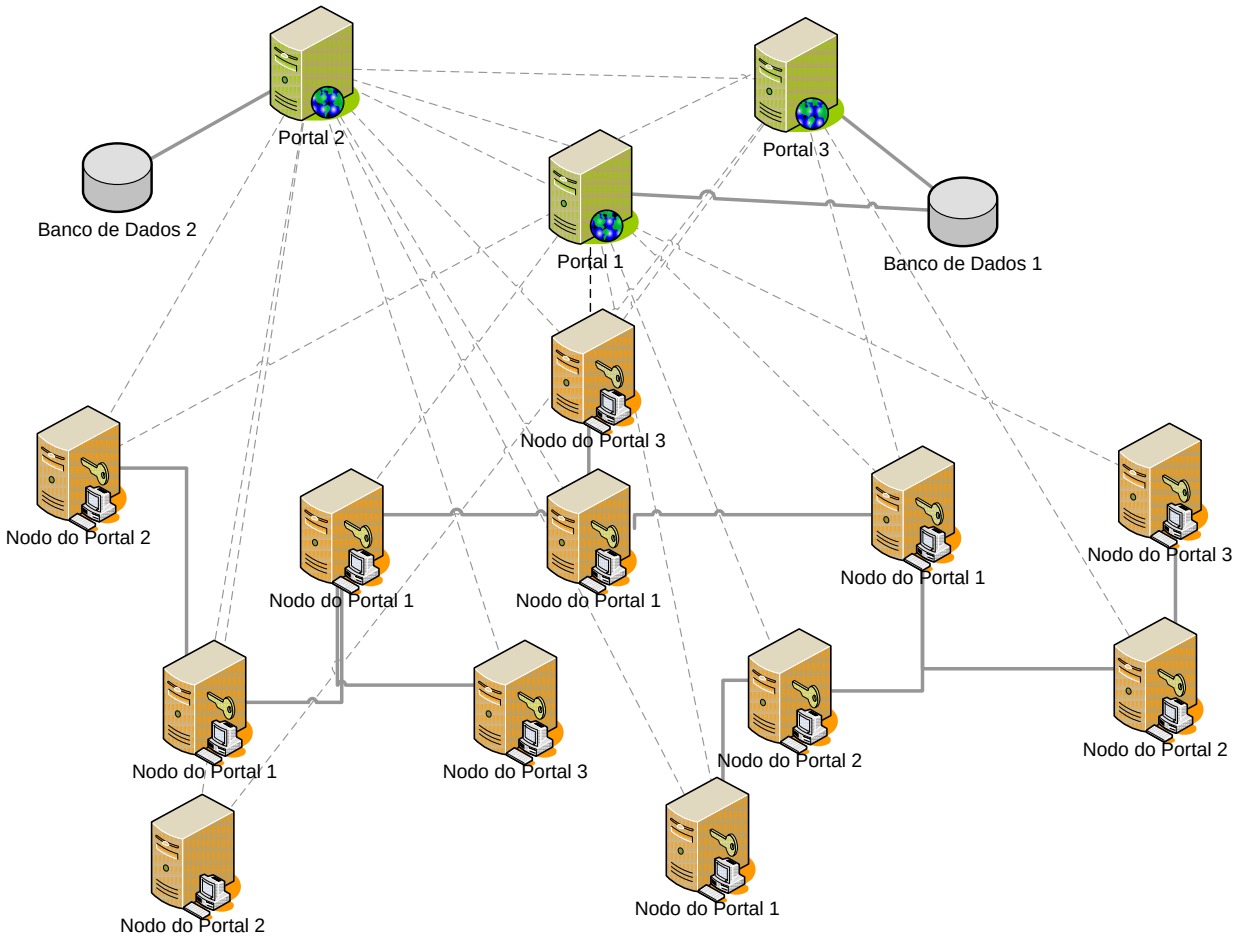

Figura 19. Formação MPNSPA0C

Essa formação minimiza consideravelmente a possibilidade de *fail points*, sejam eles de acesso do usuário, de autenticação dos nodos, de submissão de *jobs*, porém exige uma confiabilidade plena nos nodos que compõem a grade, pois caso um único nodo esteja corrompido, a integridade da grade ficará comprometida. As linhas tracejadas que ligam os Portais a todos os nodos ilustram a possibilidade de submissão de *jobs* a qualquer nodo da grade.

3.3.4.3 AUTENTICAÇÃO DE USUÁRIOS

O Processo de autenticação de usuários proposto é semelhante ao utilizado em diferentes serviços restritos da *web*, uma vez que os mesmos poderão efetuar sua autenticação única e exclusivamente via Portal P2PGrid. Após o credenciamento do usuário e a ativação do seu acesso, ele pode acessar o P2PGrid via o Portal, mediante o fornecimento dos campos de usuário e senha requeridos pelo sistema. Os procedimentos a serem tomados para a autenticação dos usuários são:

- Acessar um dos Portais P2PGrid via *web*;
- Enviar os dados de acesso para validação do usuário;
- Em caso de êxito na autenticação, o Portal carrega as políticas de acesso para aquele usuário e o encaminha à parte restrita do Portal.

Em caso de falha, o Portal registra a tentativa de acesso indevido, e redireciona o usuário a um tutorial de procedimentos para recuperação de senhas.

3.3.5 PLUGINS DE INTEGRAÇÃO ENTRE O GRID E O PORTALMAKER

3.3.5.1 PLUGIN DE SUBMISSÃO

O mecanismo de submissão do P2PGrid é composto por dois arquivos *P2PGridPlugin*, um pertencente ao *PortalMaker* e o outro pertencente à grade. O Processo de submissão de um *job* é iniciado pelo Portal que instancia um *thread* que efetua a chamada do *plugin* da grade que, por sua vez, interpreta o arquivo XML do BoT(*Bag of Task*), apresentado na figura 20, da submissão e cria uma *thread Task* para cada *job* encontrado no XML com suas devidas configurações e cada instancia de *Task* dispara o processo padrão do P2PGrid.

XML STARTER

Este arquivo no atual trabalho é elaborado pelo usuário, porém sua estrutura é de fácil compreensão. É composto por um *markup* BOT que contém todas as *tasks* a serem criadas para uma dada submissão. O markup BOT possui N markups TASK que representam os *jobs* da submissão. Os parâmetros de cada *task* estão contidos dentro do *markup* PARAMETERS de cada *markup* TASK.

```
<?xml version="1.0" ?>
- <BOT timeout="120" delay="12000">
  - <TASK id="0" job="Multiplica" timeout="1000" delay="1000">
    - <PARAMETERS>
        <PARAMETER type="number" value="2" />
        <PARAMETER type="task" value="1" />
      </PARAMETERS>
    </TASK>
  - <TASK id="1" job="Soma">
    - <PARAMETERS>
        <PARAMETER type="number" value="1 2 3 4 5 6 7 8 9 10" />
        <PARAMETER type="number" value="10 11 12 13 14 15 16 17 18 19 20" />
      </PARAMETERS>
    </TASK>
  - <TASK id="2" job="Potencia">
    - <PARAMETERS>
        <PARAMETER type="task" value="0" />
        <PARAMETER type="number" value="1" />
      </PARAMETERS>
    </TASK>
  </BOT>
```

Figura 20. Exemplo de um Arquivo XML *starter*

Os parâmetros de cada markup do XML *starter* são apresentados a seguir:

- <BOT delay=value timeout=value >

  Timeout -> expresso em segundos, corresponde ao tempo máximo para que o BoT conclua o processamento de suas *tasks*, caso contrário a finalização será forçada.

  Delay -> expresso em milissegundos, corresponde ao intervalo de tempo entre o disparo de cada *task*.

- <TASK id=value *job*=class timeout=value delay=value>

  Id -> expresso em número inteiro corresponde ao identificador de cada task deve seguir a numeração de 0 a N, tal que (N +1) seja igual ao número máximo de tasks do BoT.

  Job -> Corresponde ao nome da classe java que chama a aplicação a ser executada no *job*.

  Timeout -> expresso em segundos, corresponde ao tempo máximo para que a *task* conclua o seu processamento, caso contrário a finalização será forçada.

  Delay -> expresso em milissegundos, corresponde ao intervalo de tempo a ser esperado antes de disparar a *task*.

- <PARAMETER type={task,number,file} value={xxx}>

  Type ->corresponde ao tipo de parâmetro passado a *task*. As opções são:

number -> passagem de parâmetro de valores numéricos ou literais.

File -> passagem de referência de um arquivo que contém os dados.

Task -> a ser implementada, corresponde à referência de um resultado de outra *task* do BoT. Após a finalização das dependências do *task* os dados são substituídos pelos valores correspondentes e a *task* é executada.

Value -> Informação que identifica o conteúdo dos parâmetros passados em *type*.

PLUGIN

O *plugin* do P2PGrid sempre que instanciado cria uma instância da classe *DBManager* responsável por abrir e compartilhar as conexões abertas com as submissões em execução. Para a integração do Portal com outras grades é necessária a elaboração dos *plugins* que irão mediar esse processo. As submissões são registradas pelo Portal e o *Upload* dos arquivos que compõem os *jobs* (aplicações) dos usuários para o Portal é mediado pelo pacote *Upload* da *framework VirtualClass*. A Figura 21 apresenta o diagrama de seqüência do processo de submissão entre o usuário e o Portal.

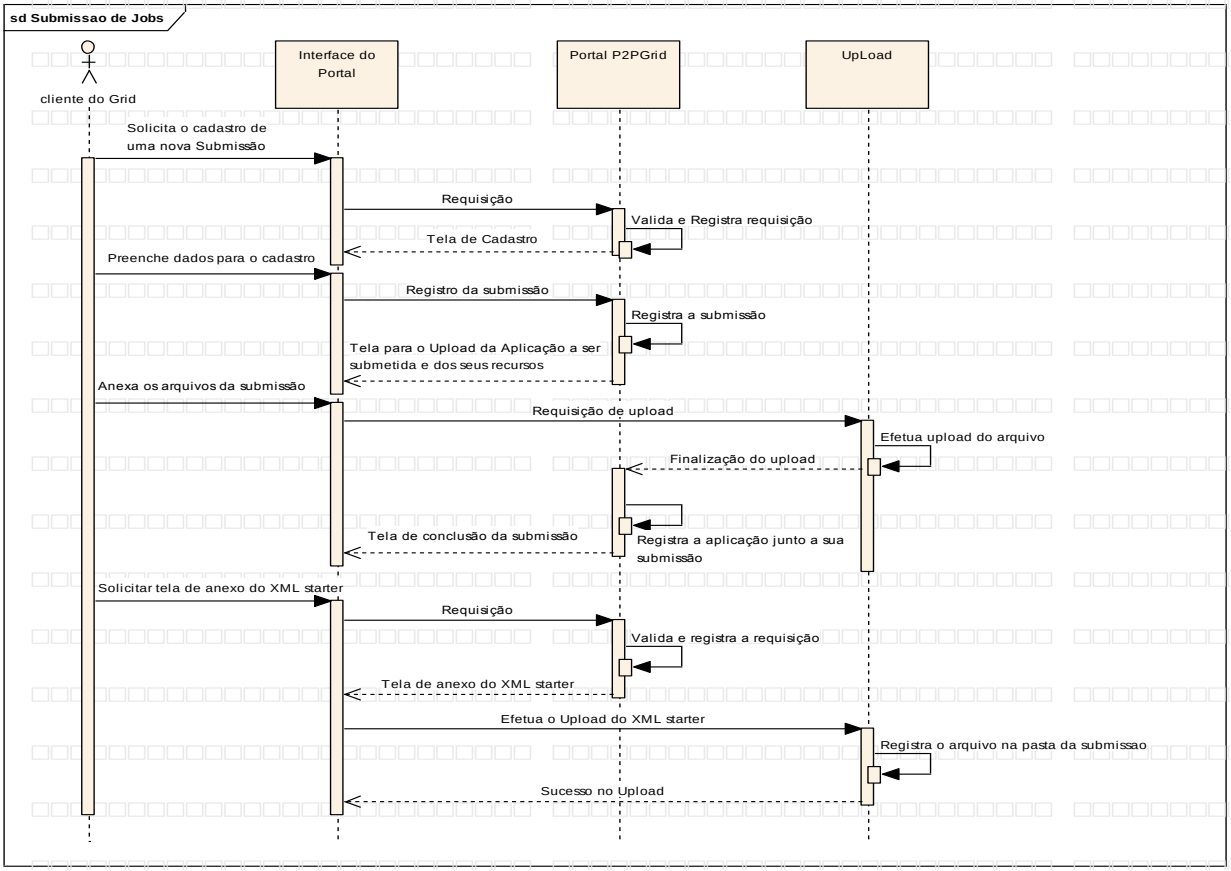

Figura 21. Diagrama de seqüência da submissão de *Jobs* pelo Portal P2PGrid

O processo de execução de uma submissão é realizado através da comunicação entre os *plugins* de execução do Portal e da grade. A solicitação da execução de uma submissão é realizada manualmente. No nosso estudo de caso, a figura 22 apresenta o diagrama de seqüência que ilustra esse processo de execução.

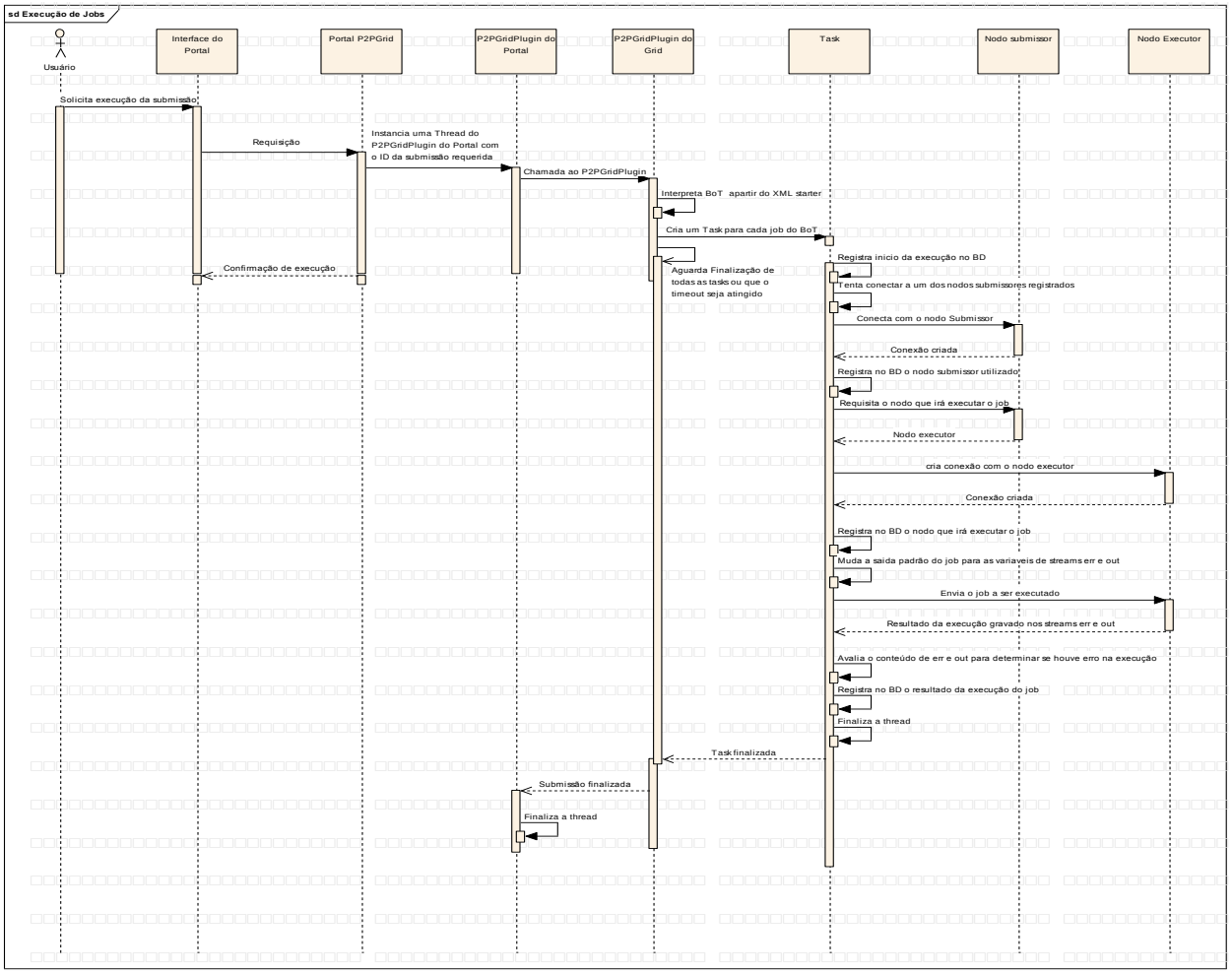

Figura 22. Diagrama de seqüência da execução de *jobs* pelo Portal P2PGrid

A implementação dos *plugins* de comunicação do Portal com a grade foi desenvolvida com a finalidade de promover a submissão assistida dos *jobs*, sendo que novas variações desses componentes podem propiciar um melhor desempenho na submissão e no registro das operações. O atual *plugin* da grade busca os parâmetros para a execução, diretamente do banco de dados do Portal, e o resultado de suas submissões fica registrado em dois *streams* de dados da instância submissora do *job*, sob o nome "err" e "out". As saídas padrões, de resultado e erro, do *job* submetido, são redirecionadas para essas variáveis. Após finalizado a execução o conteúdo desses *streams* são avaliados. Se a variável "err" possuir conteúdo é porque houve erro durante a execução e o erro é registrado no banco de dados do Portal. Caso a variável

"err" esteja vazia e a "out" não esteja o êxito na submissão ocorreu e o conteúdo desse arquivo é registrado no banco de dados com a notificação de sucesso na execução. Caso ambas as variáveis estejam vazias é adotado que houve um erro na execução. A figura 23 apresenta uma tela que mapeia a execução de um *job* submetido pelo Portal. Essá tela foi desenvolvida no estudo de caso do Portal do P2PGrid para o *PortalMaker*.


SOMA DE NRS

ACOMPANHAMENTO DA SUBMISSÃO

Soma 100 11000

RASTREAMENTO

NR	Data	Hora	Status	Opções
1	2007-04-23	11:48:05	INICIADO	
2	2007-04-23	11:48:05	SUBMISSOR SELECIONADO	Detalhar
3	2007-04-23	11:48:06	EXECUTOR SELECIONADO	Detalhar
4	2007-04-23	11:48:06	FINALIZADO COM SUCESSO!	Detalhar

Figura 23. Tela acompanhamento de uma submissão

3.3.6 MECANISMO DE ATUALIZAÇÃO

Com o mecanismo de comunicação proposto para o P2PGrid que se baseia em uma comunicação e autenticação baseada na infra-estrutura de chaves públicas, surge a necessidade de estabelecer um mecanismo que mantém os nodos e portais atualizados com as chaves públicas compartilhadas. Tal processo é realizado através de dois componentes específicos para esta função, são eles: *ServerUpdater* e *Updater*, respectivamente para o Portal e para o Nodo.

3.3.6.1 SERVERUPDATER

Aplicação que inicia dois *threads* com *sockets* seguros que visam a receber conexões de nodos e outros Portais. O primeiro *socket* é iniciado na porta especificada no arquivo de configuração *P2PGrid.conf*. Esse *socket* aceita conexões de todos que possuam a chave pública do Portal e que estejam efetuando sua conexão a partir do endereço IP cadastrado no Portal. O segundo *socket* é iniciado na porta subseqüente ao primeiro, porém aceita apenas conexões mutuamente autenticadas e que originem a partir do endereço IP cadastrado no Portal. O primeiro *socket* recebe apenas a primeira requisição de atualização do nodo que envia sua chave pública para o Portal. O segundo *socket* recebe as requisições de atualização de chaves e autenticadores, além das notificações de *status* do nodo ou de Portais remotos.

Os Comando de retorno enviados pelo *ServerUpdater* são:

1. **<Comando>,<Identificador do Nodo ou Lista XML>,<Conteúdo>**

   Comandos:

   k- indica o envio de uma chave pública;

   a - indica o envio de um complemento da lista de autenticadores.

   Identificador ou Lista XML:

   No caso de envio de chaves públicas entre o nodo e o Portal este campo recebe o identificador único do nodo composto pela fórmula: P+<IDPortal>+N+<IDNodo>.Na comunicação entre Portais este campo é representado pelo identificador do Portal obtido pela formula: P+<IDPortal>.

   Para o comando de envio da lista complementar de autenticadores este campo possui uma estrutura XML que contém a lista de autenticadores e da nova assinatura da lista completa. Conforme descrito no Conteúdo.

   Conteúdo:

   No caso de envio de chaves públicas, segue neste campo a chave pública do nodo identificado no parâmetro anterior. Para o comando de envio da lista complementar de autenticadores, este campo possui a assinatura do Portal para a lista completa dos autenticadores a ser substituída no nodo.

3.3.6.2 UPDATER

Componente responsável pela solicitação de atualização e notificação de status, do nodo ao Portal e entre Portais. O *Updater* é chamado na inicialização de um nodo antes do processo de ingresso a grade e sempre que um pedido de conexão a um nodo falha informando "*refused*" ou "*certificate invalid*". Após a falha na autenticação, o nodo requisitante e o requisitado, iniciam o processo de atualização e ao final do mesmo reenviam o pedido que outrora falhou. Na comunicação entre Portais o *Updater* é chamado sempre que for recebido algum evento pelo *ServerUpdater* seja ela de troca de chaves ou de notificação. Assim sendo para cada Portal remoto cadastrado no banco de dados será criada uma instância do *Updater* para sua atualização.

Comandos enviados pelo *Updater* do nodo para o primeiro *socket* do *ServerUpdater*:

1. **<Comando>,<Identificador do Nodo>, <Chave do Nodo>**

   Comando:

   UP - representa a solicitação de Atualização;

   Identificador:

   Na comunicação entre o nodo e o Portal é representado pelo identificador único do nodo composto pela fórmula: P+<IDPortal>+N+<IDNodo>;

   Na comunicação entre Portais este campo é representado pelo identificador do Portal obtido pela formula: P+<IDPortal>.

   Chave do Nodo:

   Enviado somente quando o nodo é iniciado pela primeira vez, com a finalidade de registrar no Portal a chave pública do nodo.

Essa chamada ocorre uma única vez no nodo, ou seja, quando o nodo necessita compartilhar sua chave pública logo após sua geração.

Comandos enviados pelo *Updater* do nodo para o segundo *socket* do *ServerUpdater*:

2. **<Comando>,<Identificador>,[<Adjacentes>]**

   Comando:

   UP - Solicitação de Atualização.

   ON - Notificação de ativação do Nodo.

   OFF - Notificação de desativação do próprio ou de outros nodos adjacentes;

   Identificador:

   Na comunicação entre o nodo e o Portal é representado pelo identificador único do nodo composto pela fórmula: P+<IDPortal>+N+<IDNodo>;

   Na comunicação entre Portais este campo é representado pelo identificador do Portal obtido pela formula: P+<IDPortal>.

   Adjacentes (opcional):

Representa os nodos adjacentes ao nodo separados por ponto e vírgula (;).

3.3.6.3 EM NODOS

O processo de atualização do nodo é realizado através do *Updater* que se conecta ao *ServerUpdater* do Portal que por sua vez consulta o banco de dados listando os autenticadores e chaves públicas a serem atualizadas no nodo. Tal lista é possível devido à existência de uma tabela no banco de dados que indica todos os nodos com o status de recebimento de suas chaves pelos outros nodos. Um processo semelhante é realizado para o envio dos nodos autenticadores. O diagrama de seqüência desta comunicação é representado na figura 24.

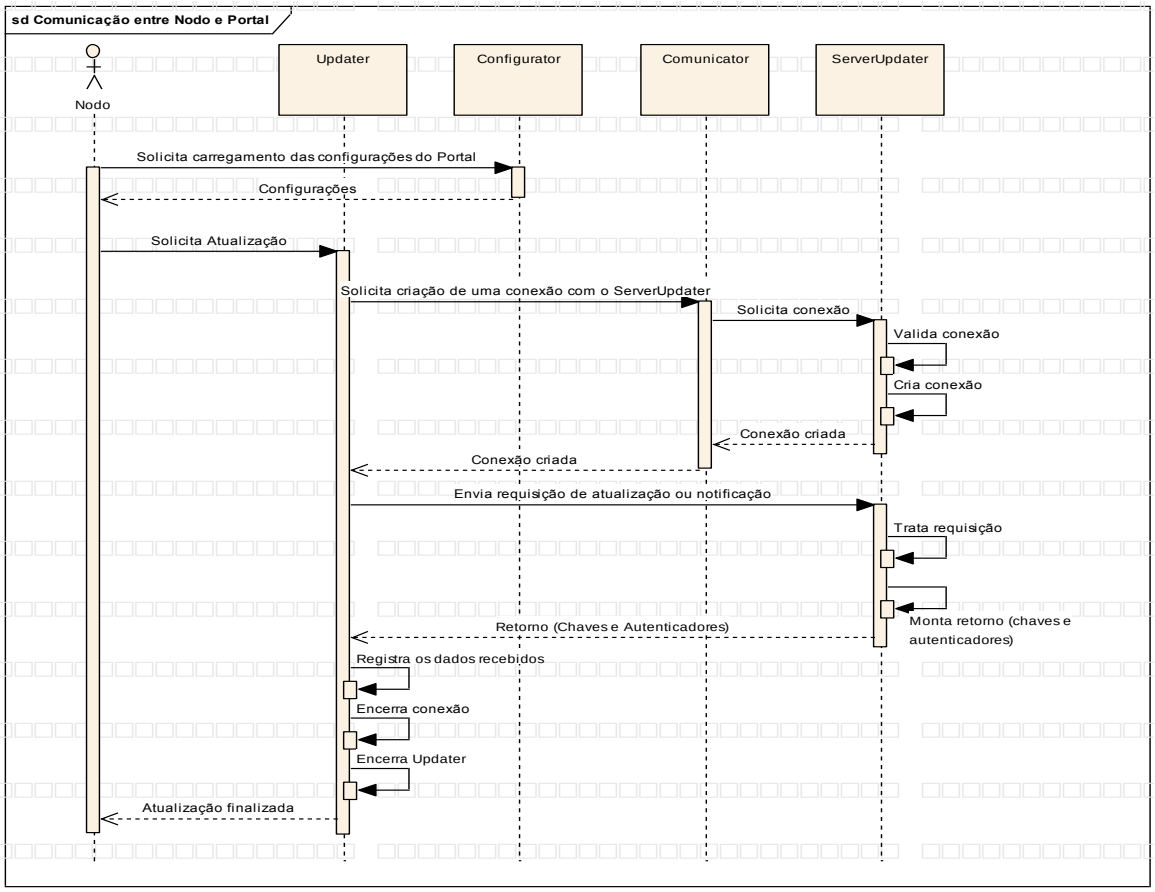

Figura 24. Diagrama de seqüência da comunicação entre o nodo e o portal.

3.3.6.4 EM PORTAIS

O processo de atualização entre Portais é realizado através do *Updater* que se conecta ao *ServerUpdater* dos Portais remotos cadastrados no seu banco de dados. Neste processo é efetuada uma consulta no banco de dados listando os autenticadores e chaves públicas a serem atualizadas em cada um dos Portais remotos. Tais listas são possíveis devido à existência de uma tabela no banco de dados que indica todos os nodos com o status de recebimento de suas chaves pelos Portais remotos. Um processo semelhante é realizado para o envio dos nodos autenticadores. Após o recebimento das chaves e dos autenticadores pelo Portal remoto, o mesmo registra o repasse dessas informações aos nodos que gerencia, de modo que os nodos nas próximas chamadas de atualização recebam essas informações. O diagrama de seqüência desta comunicação entre Portais é semelhante ao do nodo com o Portal. Sendo que a única diferença é o campo identificador da requisição que é composto apenas do identificador do Portal.

3.3.7 MECANISMO DE INICIALIZAÇÃO

Para a inicialização dos nodos e dos Portais é preciso que alguns procedimentos sejam executados. Assim sendo, definimos neste trabalho o fluxograma de inicialização de nodos e de Portais, de modo a facilitar a compreensão deste processo e de como cada um deles deve se comportar em falhas que podem, ou não, ocorrer.

3.3.7.1 DO NODO

O Processo de inicialização do nodo é representado pelo fluxograma da figura 25.

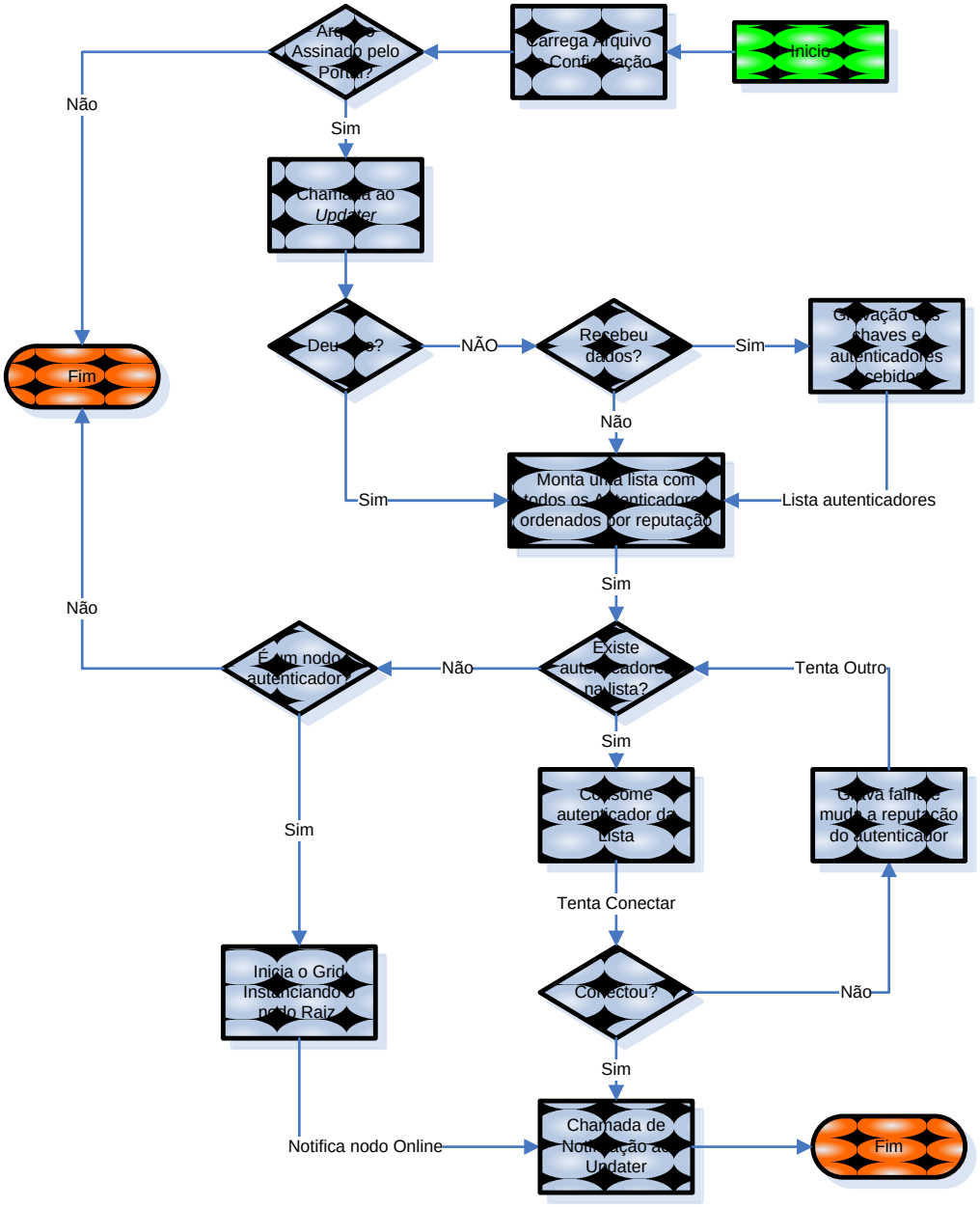

Figura 25. Fluxograma de inicialização do nodo

3.3.7.2 DO PORTAL

O Processo de inicialização do *ServerUpdater* pelo Portal é representado pelo fluxograma da figura 26.

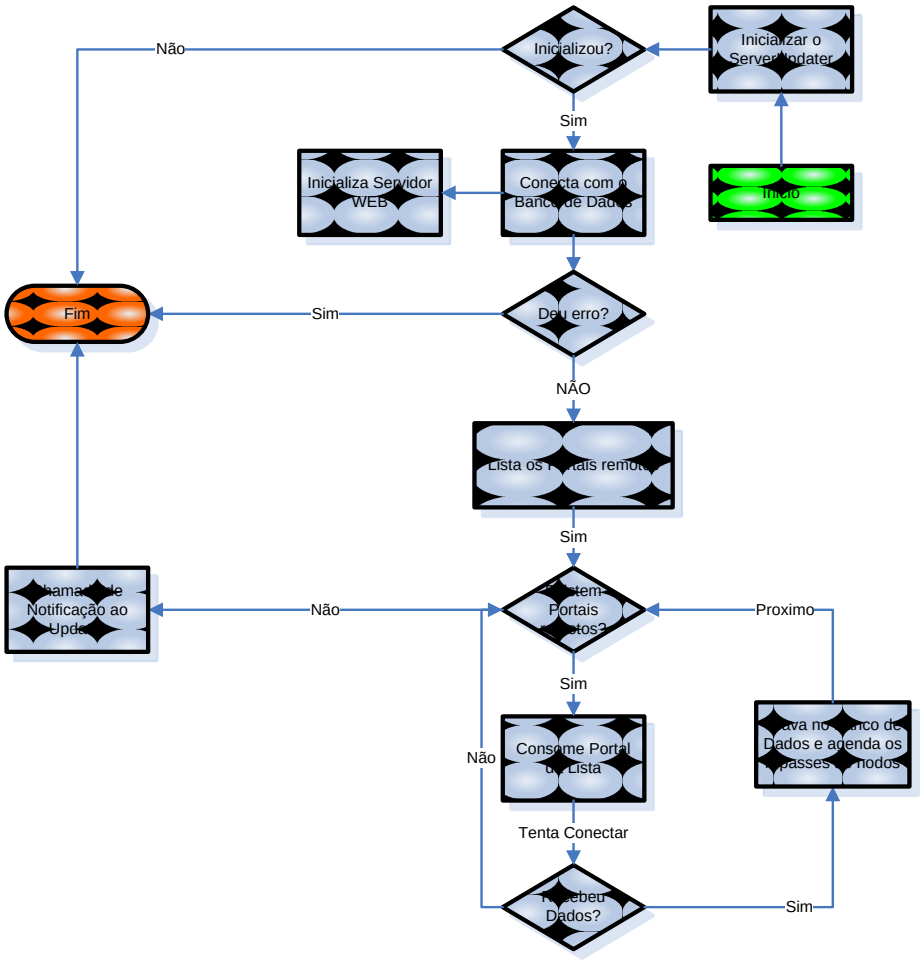

Figura 26. Fluxograma de inicialização do Portal

4 RESULTADOS E CONCLUSÕES

É preciso avaliar a capacidade do *PortalMaker* em construir e gerenciar portais, bem como avaliar seu desempenho. Para o P2PGrid sua avaliação já realizada em (1)(pág 98) foi refeita com uma nova aplicação mediada pelo portal gerado pelo *PortalMaker* proposto para o P2PGrid. Tal avaliação definiu o desempenho da grade após as mudanças estruturais e os novos processos de comunicação adotados, bem como avaliar a capacidade dos *plugins* e do portal de submeter e acompanhar *jobs*.

Para avaliar a eficiência do *plugin* de submissão desenvolvido para o Portal aferiu-se o tempo de resposta de uma *job* pelo método tradicional do P2PGrid e comparar com o tempo aferido pelo *plugin* criado para o Portal.

Avaliamos ainda o *PortalMaker* (O Middleware para geração de portais) efetuando para tanto a geração de um portal para o P2PGrid. Em seguida efetuou-se o teste de caixa preta para avaliar a eficácia e eficiência do Mecanismo de notificação e atualização de chaves da grade (*ServerUpdater* e *Updater*). Por fim avaliamos a eficiência do P2PGrid após as novas implementações de segurança e comunicação.

4.1 DESEMPENHO DO FRAMEWORK NAS CONSULTAS SQL

O *PortalMaker* faz uso do *framework VirtualClass* que gerencia e compartilha o acesso ao banco de dados mantendo um *pool* de conexões abertas, ou seja, sempre que é feita uma solicitação ao banco de dados, o *framework* reserva uma das conexões previamente aberta para a requisição durante sua execução. Caso uma solicitação seja feita e o *framework* não encontre conexões para reservar, ele entra em espera até que uma conexão seja liberada e possa ser executada. A Disponibilidade das conexões está diretamente relacionada ao número de conexões a serem compartilhadas definida no *PortalMaker*.

A Comparação realizada neste processo é a de um arquivo JSP que abre conexão e fecha a cada solicitação e do sistema de compartilhamento de conexões do *framework VirtualClass* gerenciado pelo componente *DBManager*.

A Figura 27 apresenta uma tabela comparativa de tempo aferido durante a execução de consultas SQL através do componente *DBManager* do *framework VirtualClass* e o processo simples de abertura e fechamento de conexão para cada consulta chamado de

processo padrão. A consulta SQL executada para o teste foi: "select count(*) from CA_Sistema" no banco de dados do *PortalMaker*. A Tabela "CA_Sistema" possuía nesta avaliação três registros. Devido a fatores externos como: variação do uso da memória e disponibilidade do processador, os valores das avaliações variaram a cada vez que a avaliação foi aferida, porém mantiveram um padrão de oscilação nos valores obtidos. Para a avaliação foram obtidas dez amostras com a finalidade de comprovar a eficiência do *DBManager* sem que ocorra o risco de a avaliação ser violada pelo uso de *cache* do computador e dos recursos de processamento disponíveis no momento da avaliação. A aplicação que afere o desempenho dos métodos encontra-se disponível no *PortalMaker* na sub-pasta *test*.

Amostra 1		
Número de Consultas	DBManager	Padrão
10 consulta(s)	0ms	16ms
100 consulta(s)	15ms	172ms
1000 consulta(s)	140ms	1813ms

Amostra 2		
Número de Consultas	DBManager	Padrão
10 consulta(s)	0ms	16ms
100 consulta(s)	16ms	187ms
1000 consulta(s)	156ms	1984ms

Amostra 3		
Número de Consultas	DBManager	Padrão
10 consulta(s)	0ms	16ms
100 consulta(s)	15ms	203ms
1000 consulta(s)	172ms	1860ms

Amostra 4		
Número de Consultas	DBManager	Padrão
10 consulta(s)	0ms	32ms
100 consulta(s)	16ms	187ms
1000 consulta(s)	156ms	1828ms

Amostra 5		
Número de Consultas	DBManager	Padrão
10 consulta(s)	0ms	15ms
100 consulta(s)	15ms	188ms
1000 consulta(s)	156ms	1907ms

Amostra 6		
Número de Consultas	DBManager	Padrão
10 consulta(s)	0ms	15ms
100 consulta(s)	15ms	188ms
1000 consulta(s)	156ms	1844ms

Amostra 7		
Número de Consultas	DBManager	Padrão
10 consulta(s)	0ms	16ms
100 consulta(s)	16ms	187ms
1000 consulta(s)	172ms	1875ms

Amostra 8		
Número de Consultas	DBManager	Padrão
10 consulta(s)	0ms	16ms
100 consulta(s)	15ms	297ms
1000 consulta(s)	156ms	1860ms

Amostra 9		
Número de Consultas	DBManager	Padrão
10 consulta(s)	0ms	15ms
100 consulta(s)	15ms	188ms
1000 consulta(s)	156ms	1860ms

Amostra 10		
Número de Consultas	DBManager	Padrão
10 consulta(s)	0ms	31ms
100 consulta(s)	0ms	188ms
1000 consulta(s)	156ms	1875ms

Figura 27. Amostras de desempenho do *DBManager* em chamadas SQL

Perceba que o *framework* não perde em desempenho para o processo padrão em nenhuma das amostras em que o número de consultas é superior ou igual a 10 conexões. Nas situações em que o número de consultas é inferior a 10 temos algumas amostras com o valor zero "0" o que identifica que o tempo de execução foi menor do que 1 ms impedindo a comparação entre os métodos, porém é perceptível que quanto maior o número de consultas melhor é o desempenho do *DBManager* em relação ao método padrão. Essa avaliação foi realizada de maneira seqüencial e não de consultas concorrentes. Para atender às conexões concorrentes com melhor eficiência o *DBManager* inicia um *pool* de conexões com o banco de dados juntamente com a primeira requisição JSP do Servidor que o hospeda. Assim sendo todas as solicitações SQL são realizadas através das conexões abertas pelo *DBManager* sem que haja perda de desempenho no processo de abertura e fechamento de conexões. Esse *pool* de conexões é alocado de acordo com a demanda do portal.

4.2 PLUGIN DE SUBMISSÃO DE *JOBS*

O *plugin* de submissão de *jobs* denominado *P2PGridPlugin* passou por uma seqüência de testes que avaliaram a resposta do *plugin* para as seguintes situações de pedido de submissão:

- Sem nodo submissor disponível;

  O *plugin* lista no banco de dados todos os nodos submissores *online*. A partir dessa lista inicia o processo de conexão até que um dos nodos aceite a execução ou que a lista se esgote registrando a falha e o seu motivo no banco de dados.

- Sem conexão com o banco de dados;

  No caso em que a conexão com o banco de dados não é concedida o *plugin* não submete o *job* à grade por não poder registrar seu resultado no banco de dados. A Falha na submissão também não e registrada.

- Falha ao conectar ao nodo trabalhador

  Caso ocorra erro ao conectar ao nodo trabalhador o *plugin* registra no banco de dados o erro.

- Com falha durante o envio do *job;*

Caso o envio de um *job* falhe o erro é registrado pelo *plugin* no banco de dados.

- Com falha durante a execução do *job;*

  Caso a execução de um *job* falhe a variável *stream* "err" receberá a mensagem de erro da execução, mas se caso o erro ocorrer devido à perda de comunicação com a grade o *plugin* registra no banco de dados o erro com a mensagem referente ao mesmo.

É importante ressaltar que a execução de *jobs* pelo *plugin* é tolerante a falhas de disponibilidade do Portal, uma vez que todas as submissões disparadas são gerenciadas e registradas pelo *plugin* que possui total autonomia sobre as mesmas. O fluxograma da Figura 28 ilustra o processo de submissão pelo *P2PGridPlugin* já o fluxograma da figura 29 apresenta o fluxograma de submissão de cada *job* de um BoT.

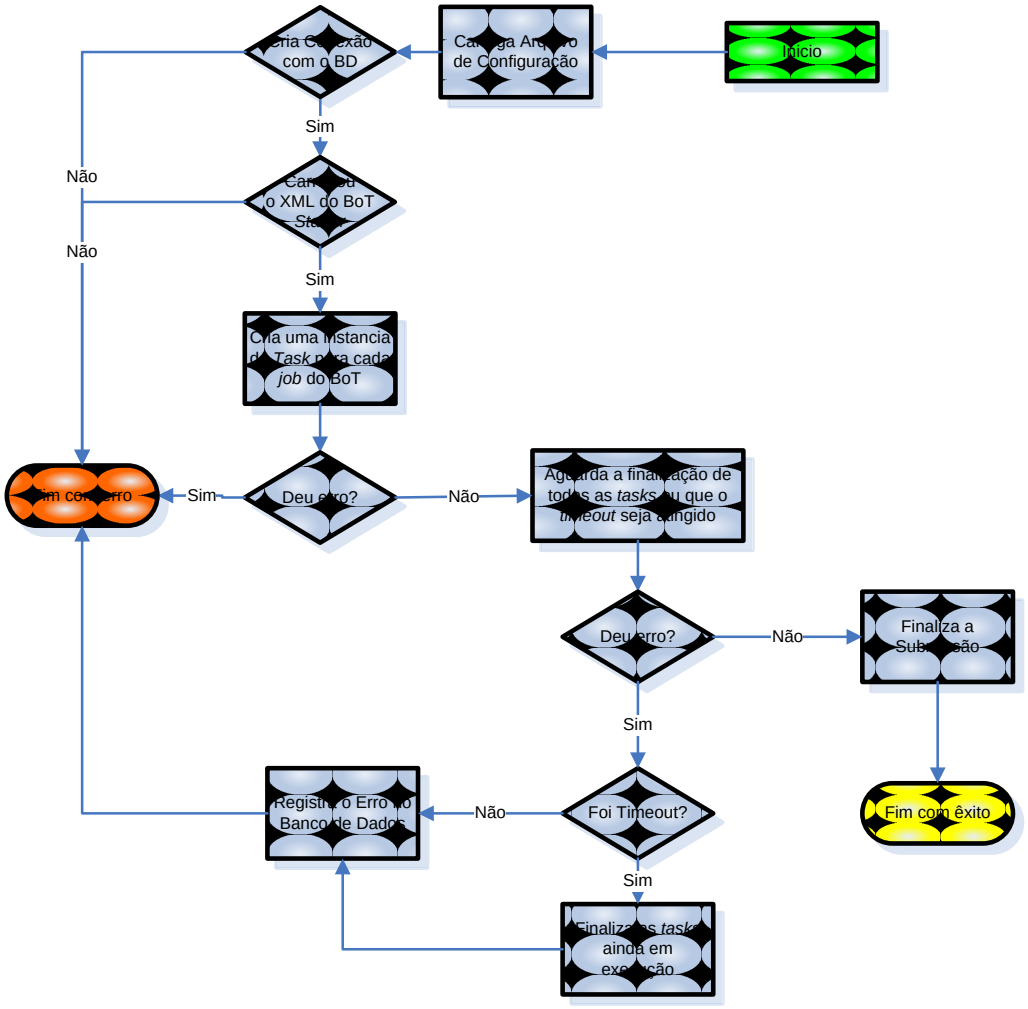

Figura 28. Fluxograma do *P2PGridPlugin*

O *P2PGridPlugin* interpreta um arquivo XML denominado *starter* o qual gerencia e determina todos os *tasks* de um BoT. A figura 28 ilustra o funcionamento com os testes avaliados neste *plugin*. Por sua vez a figura 29 apresenta o processo de execução e tratamento de erros de cada *task* criado pelo *P2PGridPlugin*.

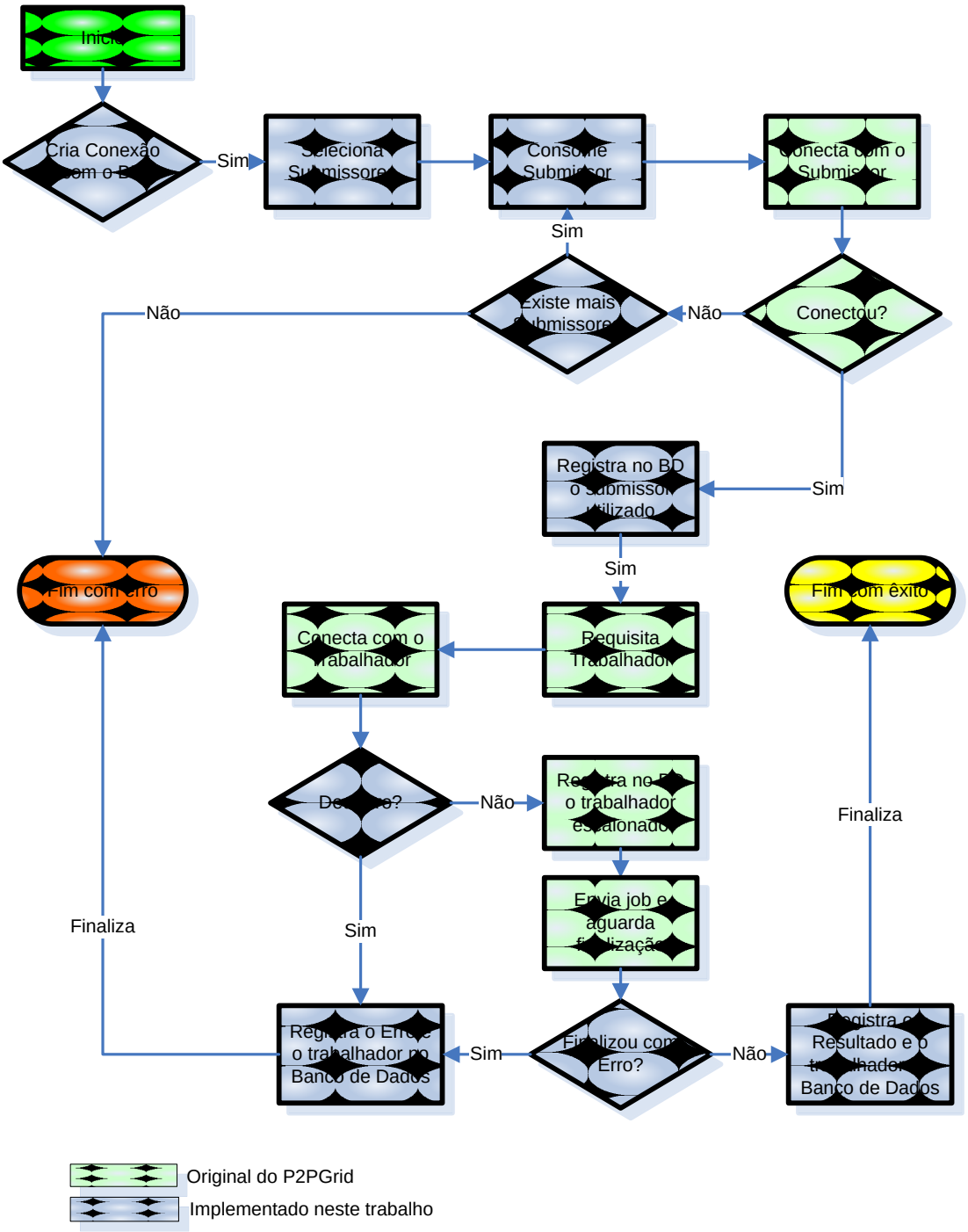

Figura 29. Fluxograma de um *task* do *P2PGridPlugin*

4.3 AVALIAÇÃO DE DESEMPENHO DO PLUGIN DE SUBMISSÃO

Avaliamos o overhead gerado pela adição dos processos de registro no banco de dados e interpretação de disparo dos *jobs* do XML *starter*. Para aferir os valores foram consideradas as situações onde o primeiro submissor encontra-se apto a receber requisições. Aferimos o tempo de resposta de 2 processos:

1. Tempo de execução desde a inicialização do *plugin*, a interpretação do XML *starter*, a alocação dos *jobs*, e as operações de registro no banco de dados.
2. Tempo de execução da inicialização de um *job*, conexão com submissor, escalonamento, envio do *job* e registro no banco de dados.

Comparamos os 2 processos com o tempo de resposta do processo original de um único *job* onde são efetuados apenas os processos de conectar, escalonar, carregar e executar. No algoritmo original caso *job* de um *BoT* é realizado mediante a chamada individual de cada *job*. A avaliação afere o tempo de resposta da submissão pelo *plugin* para um único *job* no XML *starter*. São apresentados respectivamente em cada amostra:

- O tempo de carregamento do *plugin* agregado ao tempo de disparo dos *jobs* e ao tempo utilizado pelas conexões e comandos ao banco de dados;
- O tempo do disparo do *job* desconsiderando o do *plugin* e do banco de dados;
- O tempo consumido com as criações das conexões de banco dados e operações *sql*.

Avaliações (milissegundos)											
1ª amostra			2ª amostra			3ª amostra			4ª amostra		
plugin	task	BD	plugin	task	BD	plugin	task	BD	plugin	task	BD
109 ms	16 ms	15ms	102 ms	16 ms	8 ms	125 ms	16 ms	25 ms	109 ms	15 ms	11 ms

Tabela 1. Tabela de consumo de tempo do plugin de submissão

Os valores aferidos consideraram uma máquina com as seguintes características:

- Conexão com o banco de dados local;
- XML *starter* com apenas um *job*;

- Máquina Pentium 4 com 3.06Ghz, HT com 550 MB de memória livre (PC 333);
- Máquina virtual *java* versão 1.6.0_01;
- Grade com apenas um nodo local;
- Portal Local ao nodo da grade;
- Sistema operacional Windows XP Professional;

Assim sendo os valores apresentados na Tabela 1 correspondem ao tempo adicional levado no processo de submissão devido à adição das funcionalidades do *plugin* ilustradas na figura 28 para o *plugin* e na figura 29 para os *jobs* instanciados pelo mesmo.

4.4 MECANISMO DE ATUALIZAÇÃO E NOTIFICAÇÃO

O mecanismo de atualização e notificação do P2PGrid possui dois componentes principais o *ServerUpdater* e o *Updater*. Para avaliar esse processo de comunicação validamos as seguintes situações dos componentes:

- Do *ServerUpdater*
    - Sem conexão com o banco de dados;

      Ao iniciar o *ServerUpdater* instancia um objeto da classe *DBManager* que cria um *pool* de conexões com o banco de dados para atender as requisições do *Updater* dos nodos e portais. Caso o *pool* de conexões não seja criado o *ServerUpdater* finaliza sua execução.
    - *Criação dos ServerSockets* não efetuada;

      O *ServerUpdater* finaliza sua execução informando o motivo de sua falha.
    - Conexão recusada;

      Acontece quando o requisitante não possui a chave pública do Portal ou quando a chave pública apresentada não corresponde à do Portal.
    - IP não autorizado;

      Acontece quando a conexão SSL é realizada, porém o IP do requisitante difere do cadastrado no Portal.

- Do *Updater*
    - Arquivo de configuração inválido ou com assinatura divergente;

    Quando o *Updater* é chamado ele obtém o endereço de acesso do Portal através do componente *Configurator* que por sua vez carrega essa informação do arquivo de configuração do nodo. O carregamento só é possível após a verificação da assinatura desse arquivo com a chave pública do Portal. Caso a assinatura seja inválida o *Updater* finaliza sua execução.

    - Conexão com o *ServerUpdater* recusada;

    Acontece quando a chave do Portal apresentada difere da esperada ou quando o *ServerUpdater* não está em execução ou ao alcance do requisitante. Nessa situação o *Updater* finaliza sua execução.

    - Chave pública recebida inválida;

    Durante o processo de atualização, as chaves recebidas são armazenadas e adicionadas no *keystore* do requisitante. Caso exista erro na recepção ou na agregação de uma chave ao *keystore*, o *Updater* notifica na tela o erro e não adiciona a chave.

4.5 COMUNICAÇÃO ENTRE PORTAIS E ENTRE NODO E PORTAL

O processo de comunicação e notificação entre portais é semelhante ao de comunicação entre nodos e o portal. A diferença está nas tabelas de controle de distribuição que foi implementada para notificar os repasses de informações, alguns comando do *Updater* só existem para a comunicação entre portais os quais possibilitam que os nodos de outros portais sejam cadastrados no Portal. A figura 30 apresenta as tabelas de controle de distribuição de chaves e autenticadores entre os Portais.

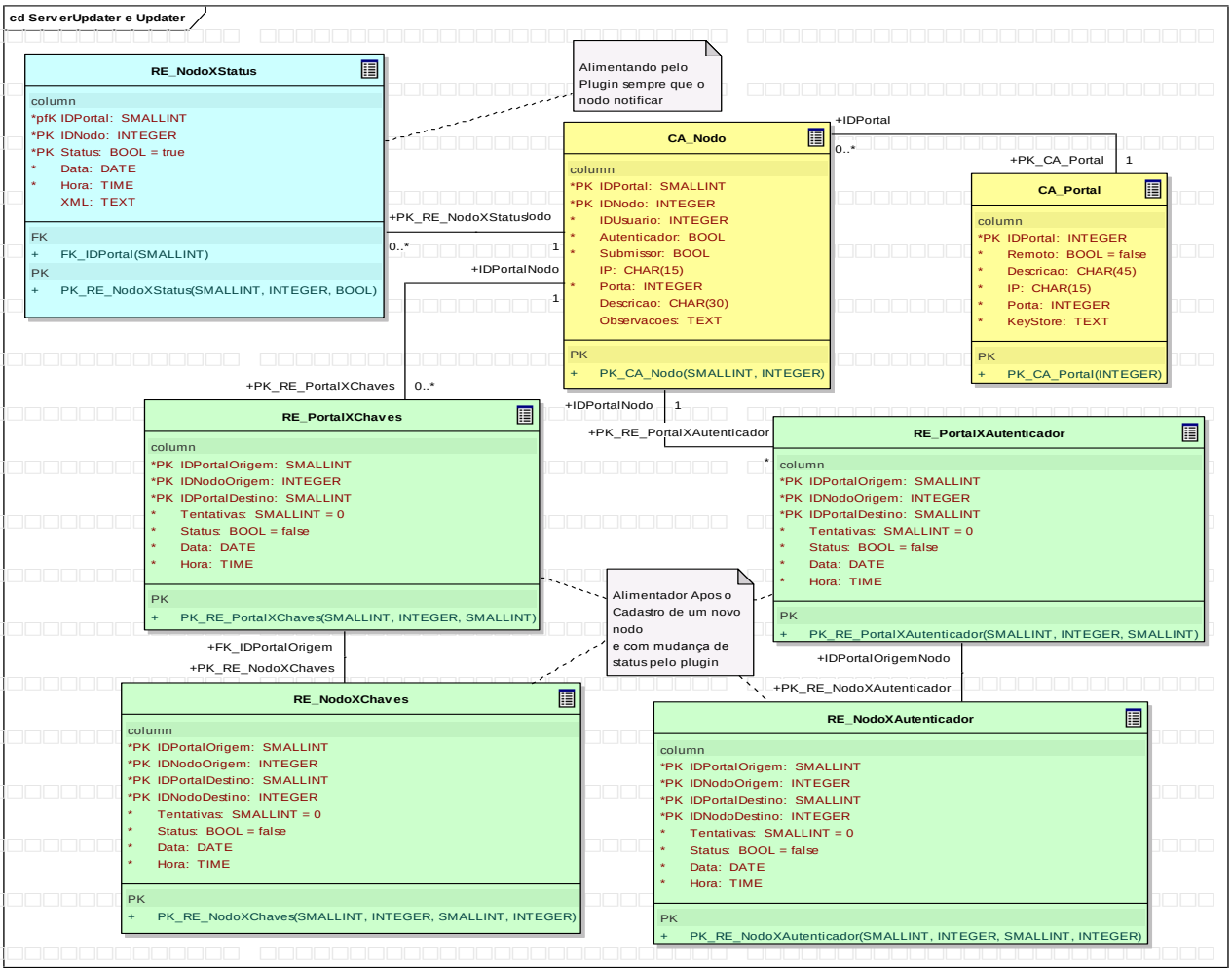

Figura 30. Banco de dados de troca de chaves e autenticadores

Os comandos de notificação de nodos são registrados na tabela *RE_NodoXStatus* nesta tabela temos o identificador do Portal (IDPortal)e do Nodo (IDNodo) sua situação é informada pelo campo *Status* (*true* para online e *false* para *off-line*). Os campos Data e Hora são utilizados para notificar a hora da última notificação recebida. O campo XML, no nosso estudo de caso, contém os nodos adjacentes do nodo em questão, cada nodo adjacente é separado por ponto e vírgula (;).

As tabelas *CA_Nodo* e *CA_Portal* são alimentadas pelo Portal e pelo *ServerUpdater* na comunicação entre portais a fim de registrar os nodos recebidos pelos portais remotos.

Durante o recebimento de nodos remotos e suas chaves o *ServerUpdater* registra o repasse dessas informações a todos os nodos que gerencia na tabela *RE_NodoXChaves*. Os nodos que forem autenticadores também são registrados para a atualização da lista de autenticadores dos nodos na tabela *RE_NodoXAutenticador.* As tabelas *RE_PortalXChaves* e *RE_PortalXAutenticador* são preenchidas sempre que o nodo do

Portal registra um novo nodo, agendando o repasse de suas informações aos portais remotos, caso existam.

4.6 P2PGRID

Para a avaliação de desempenho do P2PGrid fizemos dois testes que visam o impacto das mudanças ocorridas em sua estrutura de comunicação. A primeira avalia a configuração e os serviços disponíveis de cada nodo e a possibilidade de burlar esse processo. A segunda avaliação afere o desempenho de uma aplicação seqüencial que realiza operações matemáticas para a obtenção dos números primos encontrados em um dado intervalo e de sua execução distribuída no P2PGrid.

4.6.1 AVALIAÇÃO DA CONFIGURAÇÃO DO NODO

Nesta avaliação testamos os meios de burlar o arquivo de configuração do nodo bem como analisar o impacto das mudanças dos arquivos do P2PGrid. Assim sendo levantamos as seguintes situações no nodo:

- Alteração das informações do arquivo de Configuração(*P2PGrid.conf*)

  No caso de alteração dessas informações o nodo irá verificar a assinatura do arquivo que será diferente da informada impedindo a ativação do nodo. Um meio de burlar esse processo seria a substituição da assinatura desse arquivo com uma chave privada gerada pelo nodo e substituída no *keystore* do mesmo. Assim o arquivo de configuração é carregado com a assinatura aceita, porém ao iniciar o processo de comunicação com o Portal a chave não será aceita. O processo de atualização de chaves e notificação falha e durante o processo de carregamento dos nodos autenticadores(*P2PGrid.aut*) a assinatura é verificada e falha. Caso esse arquivo possua sua assinatura substituída o processo de autenticação continuará, porém os serviços burlados no nodo não serão publicados aos demais por não estar notificado no Portal que distribui as informações da grade.

- Múltiplas instâncias de um mesmo nodo

  A tentativa de executar múltiplas instâncias de um mesmo Pack é reduzida à medida que o *ServerUpdater* aceita conexões oriundas de nodos que sejam originadas pelo endereço IP cadastrado no Portal. As demais situações são

recusadas impedindo que o nodo se atualize e notifique sua situação ao Portal. A única possibilidade de múltiplas instâncias ocorre quando as instâncias se originam aparentemente do mesmo endereço IP. Essa situação ocorre quando o nodo se encontra atrás de um NAT ou *Proxy*.

- Alteração do arquivo(*P2PGrid.aut*) de nodos autenticadores

  Durante o carregamento da lista de autenticadores a assinatura do arquivo é verificada com a chave pública do Portal. Em caso de falha o nodo finaliza sua execução.

- Alteração do arquivo(*P2PGrid.autr*) de reputação dos autenticadores

  Durante o carregamento da lista de autenticadores os mesmos são ordenados de acordo com a reputação registrada neste arquivo. Esse arquivo é assinado pela chave do nodo e caso a verificação falhe a lista de autenticadores é carregada sem avaliar a reputação.

- Exclusão de Arquivos

  O arquivo *P2PGrid.autr* é o único arquivo que pode ser excluído, do *pack* compõem o nodo, sem que impossibilite o seu funcionamento. A exclusão de qualquer um dos demais arquivos impede o funcionamento do nodo e requer a reinstalação do *pack* do mesmo.

4.6.2 COMPARAÇÃO DE DESEMPENHO ENTRE APLICAÇÕES SEQÜENCIAIS E PARALELAS

Para avaliar o desempenho do P2PGrid em relação a uma aplicação seqüencial iremos efetuar a seguinte avaliação: Obter todos os números primos existentes entre 100.000 e 150.000. Para tanto utilizaremos uma grade composta por 19 máquinas para a avaliação paralela, sendo elas:

NR	Localização	Configuração
1	Laboratório Informática/Alano	AMD Atlon XP 1.6, 256 MB de Ram, HD 40GB
2	Laboratório Informática/Alano	Intel Celeron 2.5, 256 MB de Ram, HD 40GB
3	Laboratório Informática/Alano	AMD Atlon XP 1.6, 128 MB de Ram, HD 40GB
4	Laboratório Informática/Alano	AMD Atlon XP 1.6, 128 MB de Ram, HD 40GB
5	Laboratório Informática/Alano	AMD Atlon XP 1.6, 128 MB de Ram, HD 40GB
6	Laboratório Informática/Alano	AMD Atlon XP 1.6, 128 MB de Ram, HD 40GB
7	Laboratório Informática/Alano	AMD Atlon XP 1.6, 128 MB de Ram, HD 40GB
8	Laboratório Informática/Alano	AMD Atlon XP 1.6, 128 MB de Ram, HD 40GB
9	Laboratório Informática/Alano	AMD Atlon XP 1.6, 128 MB de Ram, HD 40GB
10	Laboratório Informática/Alano	AMD Atlon XP 1.6, 128 MB de Ram, HD 40GB
11	Laboratório Informática/Alano	AMD Atlon, 128 MB de Ram, HD 20 GB
12	Laboratório Informática/Alano	Pentium 4 1.7, 256 MB de Ram, HD 40GB
13	Laboratório Informática/Alano	Pentium 4 1.7, 128 MB de Ram, HD 40GB
14	Laboratório Informática/Alano	Pentium 4 1.7, 128 MB de Ram, HD 40GB
15	Laboratório Informática/Alano	Pentium 4 1.7, 128 MB de Ram, HD 20GB
16	Laboratório Informática/Alano	Pentium 4 1.7, 128 MB de Ram, HD 40GB
17	Laboratório Informática/Alano	Pentium 4 1.7, 128 MB de Ram, HD 40GB
18	Laboratório Informática/Alano	Pentium 4 1.7, 128 MB de Ram, HD 20GB
19	Laboratório Informática/Alano	Pentium 4 1.7, 128 MB de Ram, HD 40GB

Tabela 2. Computadores da grade do estudo de caso

Por sua vez a avaliação seqüencial será realizada na seguinte máquina:

- Pentium 4HT – 3.06GHz, 1.256MB RAM, HD 80 GB.

A Aplicação a ser submetida é exatamente a mesma sendo que a variação encontra-se na chamada da mesma.

A aplicação nomeada *NumeroPrimo* implementada na linguagem *java* possui sua chamada seqüencial efetuada pelo console de emulação do sistema operacional através do comando: "java NumeroPrimo 100000 150000". Essa chamada retornará todos os números primos do intervalo bem como o tempo consumido durante a operação.

Para a avaliação paralela pela grade foi cadastrado no Portal P2PGrid a aplicação *NumeroPrimo.class* bem como o arquivo XML que rege a inicialização da aplicação. Tal arquivo efetuará as chamadas da aplicação com diferentes definições de intervalos e distribuição, de modo a permitir que determinemos o melhor *throughput* de execução para a grade e sua relação com a execução seqüencial.

Os resultados obtidos na avaliação da execução da aplicação são apresentados na figura 32. O arquivo XML *starter* utilizado é o mesmo em ambos os casos. Ele distribui o intervalo a ser avaliado em 20 *tasks* com a divisão de tarefas por igual entre eles, ou seja, cada *task* processa 2500 números.

NR	**Máquinas**	**Avaliações (milissegundos)**				
		1	2	3	4	5
1	1 (local)	805,989ms	829,541ms	855,532ms	815,089ms	773,823ms
2	1 nodo(20 tasks)	451,312ms	455,578ms	447,531ms	444,297ms	445,062ms
3	2 nodos(20 tasks)	336,875ms	296,062ms	329,719ms	312,047ms	294,609ms
4	5 nodos(20 tasks)	282,469ms	268,250ms	240,297ms	305,109ms	272,312ms
5	10 nodos(20 tasks)	289,079ms	330,407ms	297,812ms	326,078ms	327,172ms
6	19 nodos(20 tasks)	320,828ms	319,984ms	313,656ms	303,390ms	325,859ms

Tabela 3. Relação Comparativa de Desempenho Nodos/*jobs*

Foram aferidas cinco amostras de execução de modo a demonstrar a permanência aproximada dos valores a cada avaliação.

Para a avaliação do primeiro teste executamos a aplicação seqüencial sem utilização da grade na máquina de número 3 da tabela 2, ou seja, temos um único thread rodando a aplicação descrita no Apêndice A. Por sua vez a segunda avaliação foi realizada com uma grade composta de apenas um nodo (o nodo em questão é a mesma máquina da primeira avaliação), porém devido à divisão do trabalho em 20 *tasks* proporcionou um melhor *throughput*. Isso se deve ao fato de existirem 20 processos concorrentes o que ocasiona maior uso da CPU para o calculo em questão. A terceira avaliação foi aferida a partir de uma grade composta pelas maquinas 3 e 12 da tabela 2. Nesta avaliação obtivemos um maior desempenho e foi percebido que o processo de submissão de *jobs* com base no intervalo especificado no XML *starter* apresentado no Apêndice B promoveu uma melhora no balanceamento da carga uma vez que o intervalo entre as submissões possibilita a atualização de carga na grade. Na quarta avaliação com uma grade de 5 nodos (respectivamente as máquinas 3, 4, 12, 13 e 14 da tabela 2) obtivemos o melhor desempenho de toda a avaliação. Apesar de a grade ser composta por 5 nodos a distribuição se fez presente em 3 nodos durante os 5 aferimentos. Na

quinta (máquinas 3, 4, 5, 12, 13, 14, 15, 16, 17, 18 na tabela 2) e sexta (todas as máquinas da tabela 2) avaliação aferiu-se um desempenho menor do que o obtido na grade com 5 nodos. Em uma verificação mais detalhada percebemos que a distribuição dos *jobs* se fez presente em apenas 3 nodos desde a grade de 5 nodos até a de 19 nodos. Já que a distribuição se tornou a mesma nas formações da grade de 5, 10 e 19 nodos, conclui-se que o desempenho deveria ser o mesmo para grades a partir de 5 nodos, porém um fator deve ser considerado, o *overhead* de atualização de carga. Esse *overhead* é maior à medida que a grade cresce sendo esse o fator que tornou o desempenho menor para avaliações das grades de 10 e 19 nodos da tabela 3.

4.7 ESTUDO DE CASO

4.7.1 IMPLEMENTANDO UM PORTAL PARA O P2PGRID

Para avaliar o processo de criação do *PortalMaker*, o *plugin* de submissão e as *templates* criadas implementamos um portal para o P2PGrid cujas as funcionalidades são:

- Submissão de *jobs*;
- Controle de acesso ao portal;
- Acompanhamento de *jobs;*
- Grafo de visualização do Grid;
- Montar *pack* de download para os nodos;

Foram criados, neste estudo de caso, dois módulos:

- O módulo Administrador que possui permissão de cadastro de nodos e Portais;
- O módulo Cliente que possibilita o download do *pack* dos nodos do cliente, possibilita ainda a submissão e o acompanhamento dos *jobs*.

Ambos os módulos podem acessar o *applet* que gera o grafo de visualização da grade. Alguns serviços inerentes ao *PortalMaker* são habilitados aos módulos, serviços como: avisos, repositório de objetos multimídia e área de configuração de perfil e acesso.

A modelagem do banco de dados proposto para esse portal é apresentada na figura 31. Perceba que além das tabelas apresentadas na figura 30 temos outras tabelas que

correspondem ao registro e acompanhamento da submissão. As tabelas *CA_TipoSubmissao* e *CA_Submissao* são utilizadas respectivamente para categorizar e registrar as submissões. O acompanhamento das submissões é registrado, pelo *ServerUpdater*, na tabela *RE_SubmissaoXStatus* registrando a data e hora da ocorrência. Os valores que o campo *Status* dessa tabela pode assumir são: E - erro; I - iniciado; S - submissor selecionado; W - trabalhador escalonado; F- finalizado. Entradas na tabela *RE_SubmissaoXResultado* são realizadas sempre que é realizado um registro na tabela *RE_SubmissaoXStatus* onde o valor do campo *Status* não seja o I (iniciado);

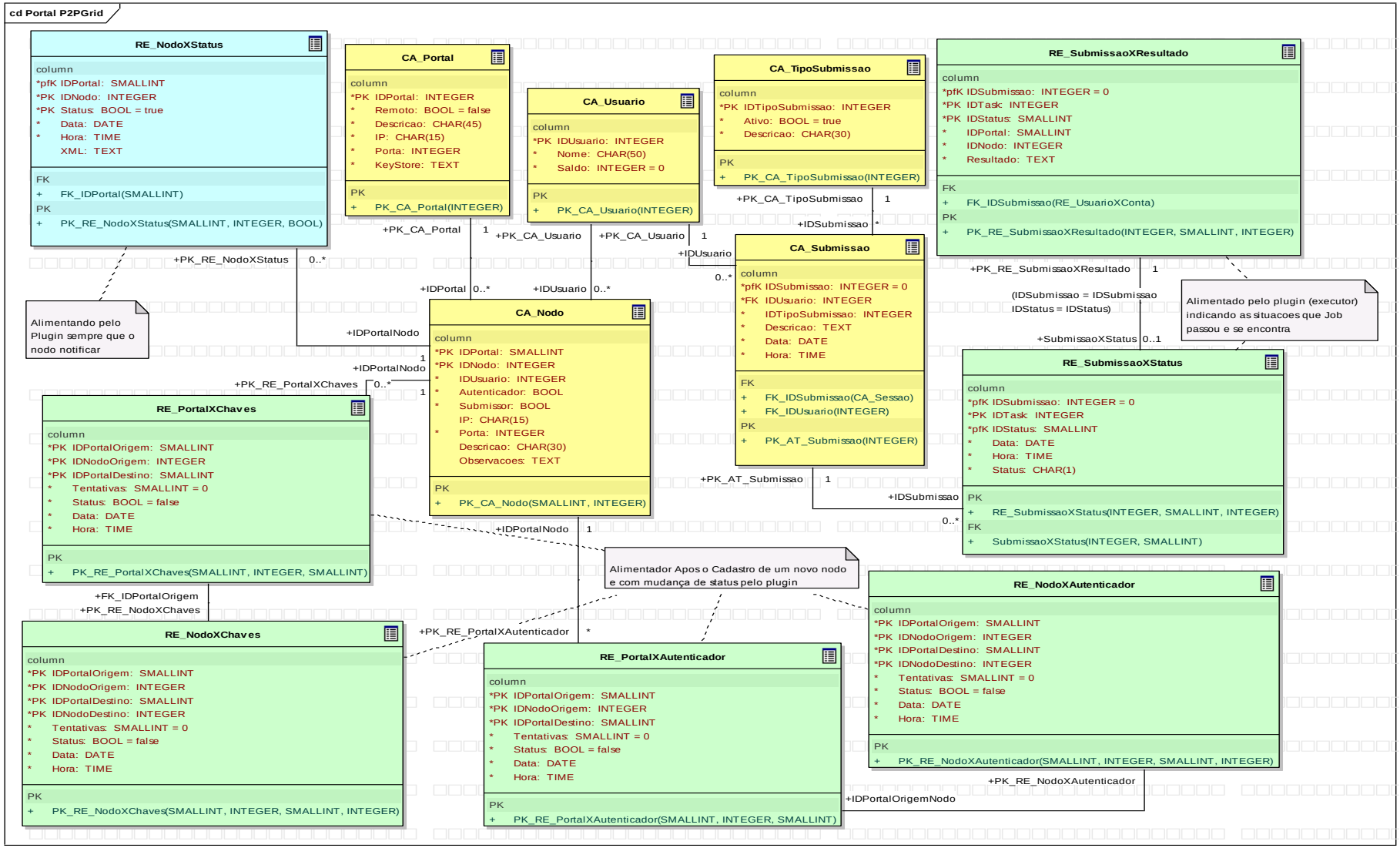

Figura 31. Banco de dados do portal P2PGrid

Algumas telas desse portal são ilustradas com a finalidade de contemplar os diferentes tipos de interações existentes. São elas:

- Figura 32 - Tela de manutenção de portais;
- Figura 33 - Tela de manutenção de nodos;
- Figura 34 - Tela de manutenção de nodos;
- Figura 35 - Tela de cadastro de submissões;
- Figura 36 - Tela de anexos da submissão;
- Figura 37 - Tela de acompanhamento da submissão;
- Figura 38 - Grafo de visualização da grade.

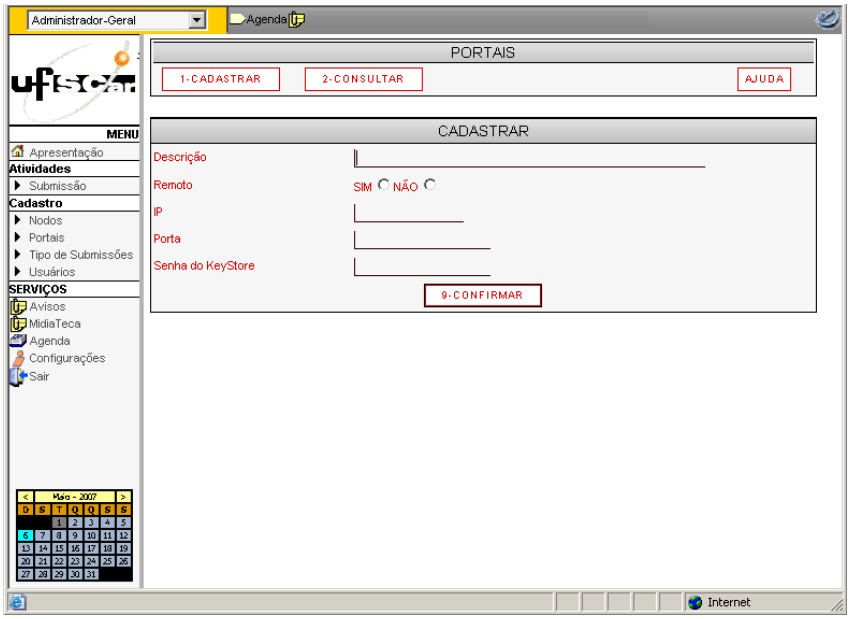

Figura 32. Tela de manutenção de Portais

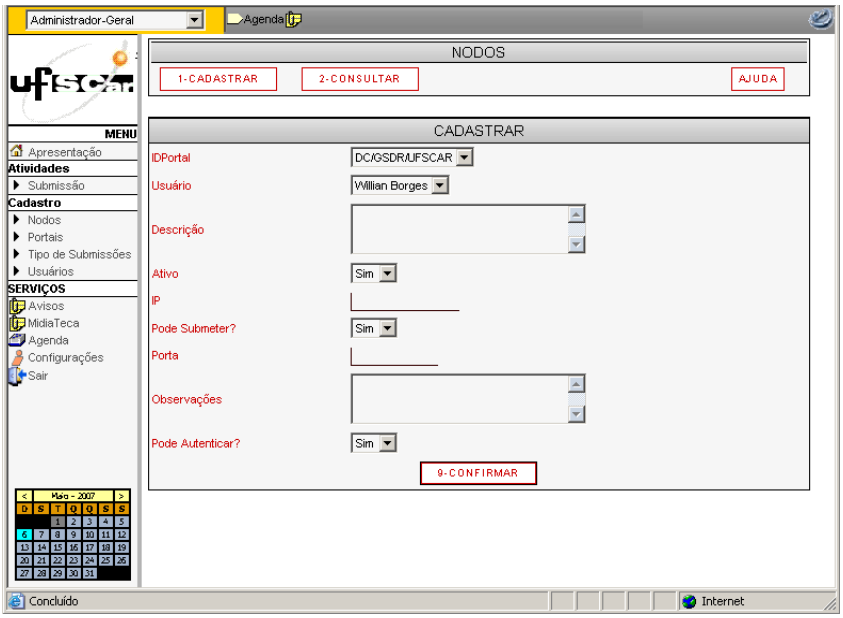

Figura 33. Tela de manutenção de nodos

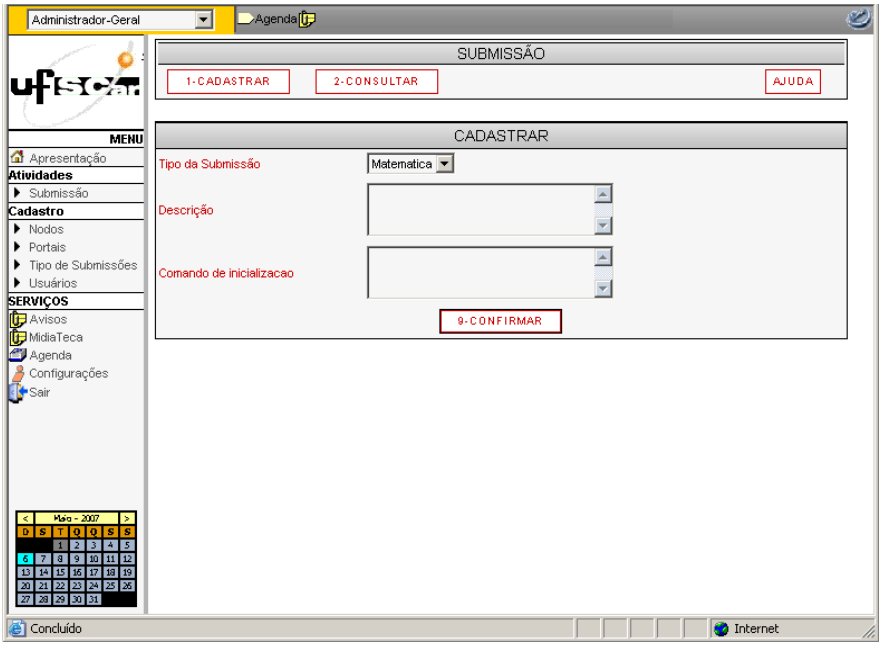

Figura 34. Tela de Cadastro de Submissões

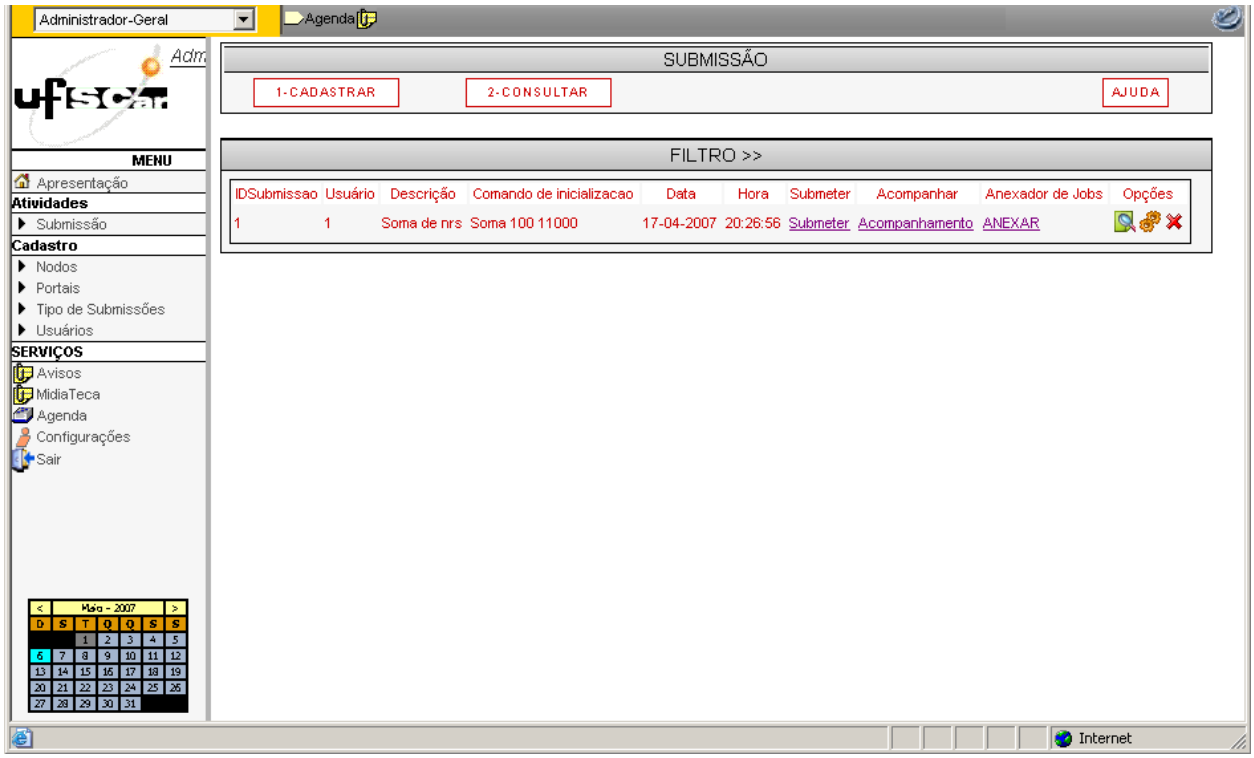

Figura 35. Tela de opções da submissão

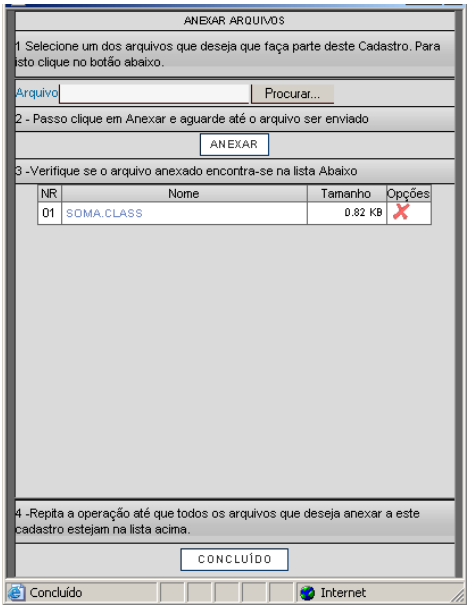

Figura 36. Tela de Anexos da submissão

Figura 37. Tela de Acompanhamento da Submissão

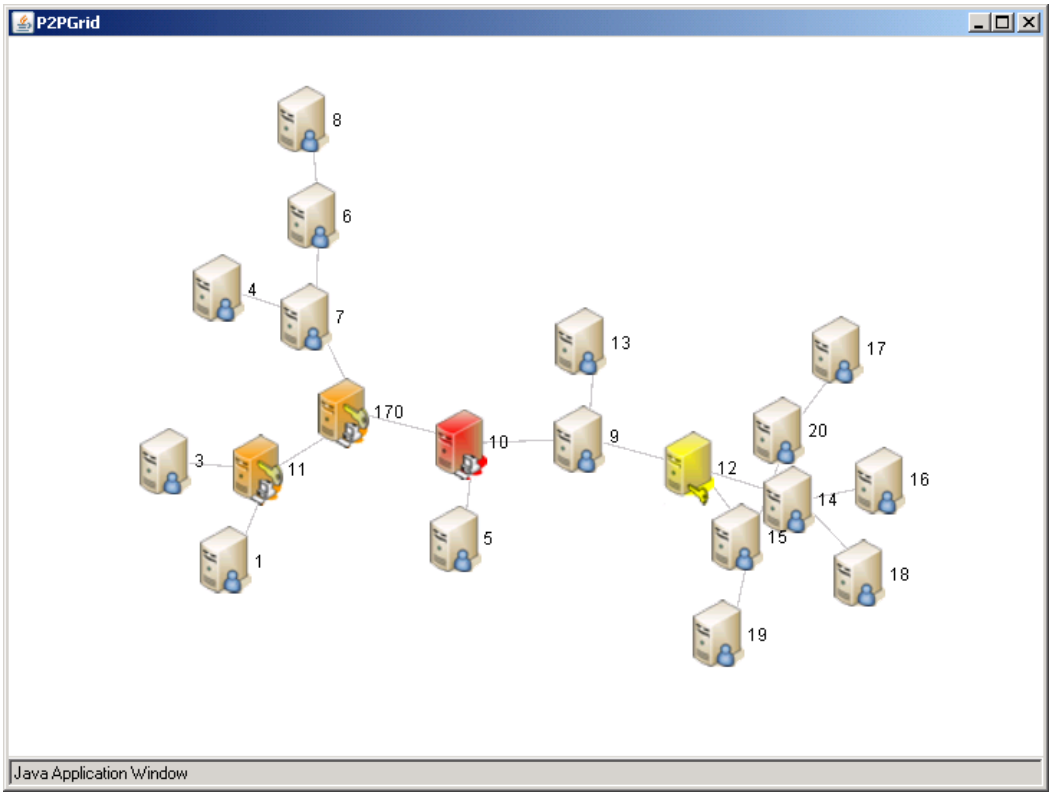

Figura 38. Applet que gera o grafo de visualização do Grid

5 CONSIDERAÇÕES FINAIS

A criação de um Portal que seja extensível e suscetível às constantes mudanças nos conceitos e às abordagens de Portais para grades computacionais é de extrema importância para sua popularização e expansão entre a comunidade científica e a sociedade. É uma grande oportunidade de desenvolvermos um Portal que seja compatível com os anseios do projeto P2PGrid. Existem vários motivos que levaram à definição deste trabalho de mestrado, com grande destaque para a necessidade e o empenho em criar e desenvolver um Portal para o P2PGrid de forma a tornar sua utilização simples, eficiente e segura em quaisquer máquinas. Com o objetivo de popularizar o acesso às grades este projeto busca conceber muito mais do que um simples Portal; conceber um mecanismo para utilização ubíqua do P2PGrid por instituições de ensino e pesquisa, bem como para qualquer usuário credenciado no Portal.

O P2PGrid possibilita a construção de grades computacionais, com utilização de recursos que são facilmente encontrados nas empresas e escolas, que possuem um conjunto de máquinas individuais que permanecem ociosas durante boa parte do tempo, mesmo quando possuem demanda de processamento. Como exemplo pode-se ter os laboratórios existentes na maioria dos departamentos de uma Universidade ou de Institutos de pesquisas, ou os computadores de uma escola de informática, ou ainda em Lan Houses e Cyber Cafés e até mesmo em casa.

Nesse universo de recursos ociosos existe uma grande heterogeneidade de hardware e software o que faz necessária a utilização de ambientes de grades computacionais capazes de se adaptar às mudanças e variações existentes. Essa característica fundamenta a importância para a decisão de se desenvolver e popularizar o ambiente P2PGrid utilizando a linguagem de programação Java, a qual fornece portabilidade para diferentes plataformas de hardware e diferentes sistemas operacionais. A criação do Portal P2PGrid só vem a agregar mais recurso e facilidades para à popularização das grades computacionais em especial o próprio P2PGrid.

Com o objetivo de dar continuidade ao projeto, todas as implementações realizadas no P2PGrid e no Portal foram modeladas e documentadas o que possibilita e facilita essa

continuidade bem como favorece o desenvolvimento de trabalhos futuros que venham a agregar valores ao Portal e ao P2PGrid.

5.1 IMPLEMENTAÇÕES FUTURAS

5.1.1 P2PGRID

Nesta seção são apresentadas algumas recomendações de desenvolvimento para o P2PGrid.

5.1.1.1 TIMEOUT PARA *TASKS* REMOTAS

Durante a avaliação de submissões no P2PGrid deparamos com situações em que, por um motivo ainda não determinado, que algumas submissões não retornam informação alguma e que o timeout do XML *starter* é obedecido. Porém a aplicação remota continua com os *sockets* de resposta para "err" e "out" abertos, consumindo memória e possivelmente executando a aplicação em *loop* ou que a mesma esteja "travada". Assim sendo é fundamental que a informação de *timeout* da submissão, que é realizada no lado do requisitante, seja propagada para a aplicação remota de modo que após um tempo pré-determinado a execução remota e seus *sockets* de retorno sejam finalizados.

5.1.1.2 AVALIADOR DE CARGA

A atual avaliação de carga do nodo é aferida com base no tempo de resposta de uma chamada ao sistema operacional do comando "*java -version*". É prudente para uma real avaliação do recurso de nodo que um mecanismo que afere a carga de uma máquina com base em diversas variáveis de sistema como poder computacional, capacidade de memória, tipo do S.O., dentre outras deva ser considerado.

5.1.1.3 NOVO ALGORITMO DE RECONSTRUÇÃO

O Atual algoritmo de reconstrução considera todos os nodos iguais e a atual formação do P2PGrid os classifica em três tipos distintos de nodos, os autenticadores, os submissores e os trabalhadores conforme descrito no capitulo 3.3. O novo algoritmo de reconstrução deve levar em considerações algumas formações e tratá-las, são elas:

- Grade formada apenas de nodos trabalhadores;

Esta grade é inacessível pela política proposta, uma vez que não pode estender seu crescimento devido à inexistência de um nodo autenticador. Torna-se inútil em utilização, pois não possui nenhum submissor em sua formação.

- Grade formada apenas de submissores ou de submissores e trabalhadores

  Temos uma grade que não pode ser estendida devido à inexistência de autenticadores, porém pode ainda ser utilizada para executar *jobs* devido à existência de submissores.

- Grade formada apenas de autenticadores ou de autenticadores e trabalhadores;

  Esta formação possibilita que a grade cresça, porém impede a submissão de *jobs*.

É importante avaliar a formação que a grade obtém após uma reconstrução se uma dessas formações foi conseguida informando ao Portal de modo que o administrador/usuários possa ter acesso à informação de restrição de submissão por ausência de nodos com o serviço ativo. Ainda devemos considerar a elaboração de um algoritmo que avalie a formação das grades que não podem ser acessadas nem estendidas e tome uma situação de aguardar até que um autenticador esteja online ou outra técnica para solucionar o problema.

5.1.1.4 PROTOCOLO DE COMUNICAÇÃO PARA A GRADE

O atual processo de comunicação entre os nodos do P2PGrid é realizado via serialização de objetos via *sockets*, tais objetos são recebidos pelo destinatário e comparados com a classe que os originou de modo a reconstruir o respectivo objeto remotamente. Acontece que esse processo de comparação, reconstrução e serialização de objetos torna o entendimento, a eficiência e a estensibilidade do P2PGrid limitada. Adicionar informações do *payload* da comunicação sugere que uma nova classe seja definida e que novos objetos para a mesma sejam criados. A definição de um módulo de comunicação que utilize um protocolo extensível e adaptável à realizada do P2PGrid, irá prover uma maior eficiência e adaptabilidade possibilitando, além de tudo, uma maior compreensão do código implementado.

5.1.1.5 MECANISMO DE RE-SUBMISSÃO DE *JOBS*

Durante a avaliação de submissão de *jobs* foi detectado que às vezes alguns *jobs* não finalizam sua execução sendo necessário re-submeter o *job* em questão a fim de obter êxito no processamento do BoT. Assim sendo é importante que seja estendido o P2PGridPlugin e P2PGrid de modo a detectarem os *jobs* não concluídos e que possam ser re-submetidos a fim de obter o resultado esperado. Tal solução promove um aumento considerável na eficácia na submissão de *jobs* à grade.

5.1.2 PORTALMAKER

Nesta seção são apresentadas algumas recomendações de desenvolvimento para o *PortalMaker*.

5.1.2.1 NOVAS TEMPLATES

O atual *PortalMaker* possui templates básicos que foram utilizadas para o desenvolvimento do Portal proposto no Estudo de Caso. É importante que as *templates* sejam estendidas e criadas de modo a adicionar novas funcionalidades aos Portais e fornecer uma maior gama de opções ao administrador. Adicionar *templates* favorece a elaboração do Portal pelo administrador que poderá em alguns casos optar por *templates* equivalentes que trabalham a mesma informação de um modo visual diferente.

5.1.2.2 ESTENDER O JSPMAKER PARA UMA OU MAIS TEMPLATES.

O *PortalMaker* trabalha com um único *parser*, o *JSPMaker*, que mescla as templates com os dados do usuário. É importante estender o *JSPMaker* de modo que o criador de uma templates possa não apenas criar as *templates* como também o *parser* correspondente a ela favorecendo o desenvolvimento e facilitando a integração com o *PortalMaker* que por sua vez irá apenas chamar o respectivo *parser* para cada template durante o processo de construção de um Portal.

5.1.2.3 PROPAGAR A ATUALIZAÇÃO DOS PORTAIS

Após mudar uma *template* ou *plugin* é preciso solicitar manualmente a atualização dos Portais que utilizam os recursos alterados. Uma proposta é de que após a alteração de um recurso o *PortalMaker* mapeie os Portais que utilizam o recurso e solicite ao administrador a autorização para gerá-los novamente.

5.1.2.4 PROPAGAR MUDANÇAS DE BANCO DE DADOS

Após uma mudança da estrutura de uma tabela, campo ou banco de dados o *PortalMaker* sugere a atualização automática dos Portais que utilizam a tabela manipulada. A atual formação somente atualiza o banco de dados após solicitação do administrador.

5.1.3 PORTAL P2PGRID

Nesta seção são apresentadas algumas recomendações de desenvolvimento para o Portal gerado para o P2PGrid.

5.1.3.1 CONFIGURAR A ESTRUTURA PARA CONTABILIZAÇÃO

O estudo de caso proposto para o *PortalMaker* implementou um Portal com funcionalidades básicas de submissão e controle de acesso à grade do P2PGrid. É importante estender as funcionalidades adicionando recursos para contabilização, reputação e auditoria da utilização da grade.

5.1.3.2 RELATÓRIOS DE ACOMPANHAMENTO E AUDITORIA

Para a utilização real do Portal do P2PGrid é importante que sejam desenvolvidos alguns relatórios para acompanhamento e levantamento estatístico da grade. Relatórios que monitoram e determinam erros de execução são essenciais para a determinação e aprimoramento da grade. Para a criação desses relatórios que devem ser adaptados ao contexto de cada administrador, sugerimos a implementação de um *middleware*, que auxilie a geração de relatórios, de modo a favorecer o desenvolvimento e customização dos mesmos.

5.1.3.3 MECANISMO INTERATIVO DE CRIAÇÃO DO XML STARTER

Elabora um *plugin* (ex.: applet, javascript, flash) que possibilite a elaboração do XML starter de modo interativo e visual. Tal abordagem tende a facilitar a criação do *BoT* e minimizar a ocorrência de erros na elaboração do XML *starter*. Este *plugin* além de auxiliar na criação visual, valida também a estrutura do arquivo minimizando os erros.

5.1.4 PLUGIN DE SUBMISSÃO

Nesta seção são apresentadas algumas recomendações de extensão para o *plugin* que media a submissão entre o Portal e a grade P2PGrid.

5.1.4.1 EXTENSAO DO XML STARTER

Elabora um mecanismo para que, a partir do arquivo do *BoT*, o usuário possa definir qual o formato do arquivo de saída de cada *job* e do arquivo que venha por ventura receber a concatenação dos arquivos gerados pelos *jobs*. Com essa extensão, o resultado da submissão poderá ser registrado ou não no banco de dados, sendo o local de gravação uma definição do usuário, podendo também combinar várias formas de saída dos resultados dos *jobs*.

5.1.4.2 ESTENDER O PLUGIN DE SUBMISSÃO PARA OUTRAS GRADES

Uma elaboração futura é a criação de uma extensão do *plugin* de submissão que possibilite a interação com outras grades computacionais, tornando o *plugin* de submissão um submissor para as mais variadas grades. Para tanto, é necessário apenas que seja implementada uma nova instância da classe *Task* para cada grade que for integrada ao plugin.

6 REFERÊNCIAS BIBLIOGRÁFICAS

1. MATTOS, Erico C. T. de. ***Estudo e construção de um ambiente de grade computacional peer-to-peer com ênfase no balanceamento de carga***. São Carlos - SP : Dissertação de Mestrado, Departamento de Computação - UFSCar, 2005.

2. Chetty, Madhu e Buyya, Rajkumar. **Weaving Computational Grids: How analogous are they with electrical grids?** *Journal of Computing in Science and Engineering (CiSE).* [Online] 2002. [Citado em: 15 de Agosto de 2006.] http://portal.acm.org/citation.cfm?id=615952&dl=ACM&coll=portal.

3. **Object Management Group – UML**. *UML® Resource Page.* [Online] [Citado em: 10 de 08 de 2005.] http://www.uml.org.

4. ***Java Technology***. [Online] [Citado em: 10 de 08 de 2005.] http://www.java.sun.com.

5. **Extensible Markup Language (XML)**. *W3C.* [Online] [Citado em: 10 de 08 de 2005.] http://www.w3.org/XML/.

6. Hoffman, James. **Introduction to Structured Query Language**. [Online] 04 de 09 de 2001. [Citado em: 10 de 08 de 2005.] http://riki-lb1.vet.ohio-state.edu/mqlin/computec/tutorials/SQLTutorial.htm.

7. **ibm.com Brasil - Sobre a IBM - Brasil**. *IBM.com Brasil.* [Online] [Citado em: 10 de Agosto de 2005.] http://www.ibm.com/ibm/br/.

8. ***American National Standards Institute - ANSI***. [Online] [Citado em: 10 de 08 de 2005.] http://www.ansi.org/.

9. **Sun Microsystems, Inc.** Sun Microsystems. [Online] [Citado em: 10 de Agosto de 2005.] http://br.sun.com/.

10. **Java Virtual Machine**. [Online] [Citado em: 10 de Agosto de 2005.] http://www.java.com/pt_BR/download/index.jsp.

11. Nascimento, Marcelo Martins. **Saiba mais sobre Open Source**. *OpeNetwork Solutions - Saiba mais sobre Open Source.* [Online] 05 de Outubro de 2005. [Citado em: 10 de Dezembro de 2005.] http://www.openetwork.com.br/index.php?option=com_content&task=view&id=24&Itemid=2.

12. Sun Microsystems, Inc. **Applets**. *The Java Tutorials.* [Online] [Citado em: 10 de Agosto de 2005.] http://java.sun.com/docs/books/tutorial/deployment/applet/index.html.

13. Consortium, World Wide Web. **About The World Wide Web**. [Online] 24 de Janeiro de 2001. [Citado em: 10 de Agosto de 2005.] http://www.w3.org/WWW/.

14. **Package java.awt**. [Online] [Citado em: 10 de Agosto de 2005.] http://java.sun.com/j2se/1.5/docs/api/java/awt/package-summary.html .

15. **JavaScript Guide**. *Netscape Communications Corporation.* [Online] [Citado em: 10 de Agosto de 2005.] http://wp.netscape.com/eng/mozilla/3.0/handbook/javascript/.

16. ***Netscape.*** [Online] [Citado em: 10 de Agosto de 2005.] http://www.netscape.com/about/.

17. **The Java Servlet API White Paper**. [Online] [Citado em: 10 de Agosto de 2006.] http://java.sun.com/products/servlet/whitepaper.html.

18. **W3C HTML Home Page**. [Online] [Citado em: 10 de Agosto de 2006.] http://www.w3.org/MarkUp.

19. Postel, Jonathan B. ***Simple Mail Transfer Protocol*** . [Online] Agosto de 1982. [Citado em: 10 de Agosto de 2006.] http://www.ietf.org/rfc/rfc0821.txt.

20. **Hypertext Transfer Protocol**. *W3C.* [Online] [Citado em: 10 de Agosto de 2006.] http://www.w3.org/Protocols/.

21. Oikarinen, Jarkko. **Internet Relay Chat**. [Online] 1988. [Citado em: 10 de Agosto de 2006.] http://www.irc.org/history_docs/jarkko.html.

22. **Common Gateway Interface** . [Online] [Citado em: 10 de Agosto de 2006.] http://www.w3.org/CGI/.

23. **JavaServer Pages White Paper**. *JavaServer Pages[tm] Technology.* [Online] [Citado em: 15 de Agosto de 2006.] http://java.sun.com/products/jsp/jspguide-wp.html.

24. ***Java EE at a Glance.*** [Online] [Citado em: 15 de Agosto de 2006.] http://java.sun.com/javaee/.

25. **Introducing the Java Community Process (JCP) Program**. *Java Comunity Process white papers.* [Online] 2004. [Citado em: 15 de setembro de 2006.] http://www.jcp.org/en/press/whitepapers.

26. **JSR 168: Portlet Specification.** *The Java Community Process(SM) Program - JSRs: Java Specification Requests.* [Online] [Citado em: 15 de março de 2007.] http://www.jcp.org/en/jsr/detail?id=168.

27. DEACON, J. **Model-View-Controller (MVC) Architecture**. [Online] [Citado em: 10 de março de 2006.] http://www.jdl.co.uk/briefings/MVC.pdf.

28. BAKER, M. **Cluster Computing White Paper**. *University of Portsmouth - UK.* [Online] 2000. [Citado em: 15 de Agosto de 2006.] http://arxiv.org/ftp/cs/papers/0004/0004014.pdf.

29. FOSTER, I., KESSELMAN, C. e TUECKE, S. **The Anatomy of the Grid: Enabling Scalable Virtual Organizations**. *International Journal of High Performance Computing Applications.* 2001, Vol. 15, pp. 200-222.

30. Baker, Mark, Buyya, Rajkumar e Laforenza, Domenico. **Grids and Grid technologies for wide-area distributed computing**. [Online] 2002. [Citado em: 15 de Agosto de 2006.] http://www.gridbus.org/papers/gridtech.pdf.

31. Krauter, Klaus, Buyya, Rajkumar e Maheswaran, Muthucumaru. **A taxonomy and survey of grid resource management systems for distributed computing**. [Online] 2002. [Citado em: 15 de Agosto de 2006.] http://www.gridbus.org/papers/gridtaxonomy.pdf.

32. Foster, Ian. **Internet Computing and the Emerging Grid**. [Online] 7 de dezembro de 2000. [Citado em: 5 de setembro de 2005.] http://www.nature.com/nature/webmatters /grid/grid.html.

33. **ABOUT NAPSTER**. *Napster - Company Information.* [Online] [Citado em: 15 de março de 2006.] http://www.napster.com/about_napster.html.

34. eMule.org - **Official eMule Site**. [Online] [Citado em: 15 de março de 2006.] http://www.emule.org/.

35. **Kazaa - About Us.** [Online] [Citado em: 10 de março de 2006.] http://www.kazaa.com/us/about/index.htm.

36. **About SETI@home**. [Online] [Citado em: 10 de março de 2006.] http://setiathome.berkeley.edu/sah_about.php.

37. **About Parabon**. [Online] [Citado em: 10 de março de 2006.] http://www.parabon.com/about.jsp.

38. Foster, Ian e Iamnitchi, Adriana. **On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing**. [Online] 2003. [Citado em: 5 de março de 2006.] http://people.cs.uchicago.edu/~anda/papers/foster_grid_vs_p2p.pdf.

39. Taurion, Cezar. ***Grid Computing: Um Novo Paradigma Computacional.*** Brasil : Brasport, 2004.

40. **The Globus Alliance**. [Online] [Citado em: 10 de março de 2006.] http://www.globus.org/alliance/.

41. **Legion: A Worldwide Virtual Computer**. [Online] [Citado em: 5 de março de 2006.] http://legion.virginia.edu/.

42. Costa, Paulo Capellotto, Guardia, Hélio e Zorzo, Sérgio Donizetti. **ProGrid: A Proxy-Based Architecture for Grid Operation and Management**. *SBAC - PAD 2003.* 10 de novembro de 2003.

43. **Message Passing Interface**. [Online] [Citado em: 2 de abril de 2006.] http://www.llnl.gov/computing/tutorials/mpi/#What.

44. **MPICH-G2**. [Online] [Citado em: 2 de abril de 2006.] http://www3.niu.edu/mpi/.

45. Baird, Ian. **Understanding grid computing.** ***Daily news and information for the global grid community****.* [Online] 1 de julho de 2002. [Citado em: 10 de abril de 2006.] http://www.gridtoday.com/02/0701/100060.html.

46. Agustini, Ana Patricia M. Vilha di. **Organização Virtual - um novo paradigma organizacional para o século XXI**. *Infotec.* [Online] [Citado em: 20 de abril de 2006.] http://www.ccuec.unicamp.br/revista/infotec/artigos/anapatr.html.

47. **Global Grid Forum**. [Online] [Citado em: 30 de setembro de 2005.] http://www.ggf.org.

48. Buyya, Rajkumar, et al. **Economic models for resource management and scheduling in Grid computing**. *The Journal of Concurrency and Computation: Practice and Experience.* [Online] 2002. [Citado em: 10 de outubro de 2005.] http://citeseer.ist.psu.edu /buyya02economic.html.

49. Donald F. Ferguson, Christos Nikolaou,Jakka Sairamesh, Yechiam Yemini. **Economic models for allocating resources in computer systems**. [Online] 1996. [Citado em: 5 de novembro de 2005.] http://citeseer.ist.psu.edu/cache/papers/cs/3502/http:zSzzSzwww. cas.ibm.cazSzpresentationszSzeconomic-models.pdf/ferguson96economic.pdf.

50. Cirne, Walfredo e Marzullo, Keith. **The Computational Co-op: Gathering Clusters into a Metacomputer**. [Online] 1999. [Citado em: 5 de novembro de 2006] http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=760453

51. ***Portal Progrid: Um sistema web para monitoramento e controle de aplicações e recursos em um grade****.* Zorzatto, Jessica Antonieta. São Carlos : s.n., 2005. Congresso de Pós-graduação.

52. Mello, R.F., et al. **Proposal of a Tree Load Balancing Algorithm to Grid Computing Environments**. *IEICE Trans. Inf. & Syst.* julho de 2004.

53. **A taxonomy of scheduling in general-purpose distributed computing systems.** *IEEE Transactions on Software Engineering.* [Online] 1988. [Citado em: 21 de janeiro de 2006.] http://ieeexplore.ieee.org/xpl/abs_free.jsp?arNumber=4634.

54. Welch, V., et al. **OGSI Authorization Requirements**. [Online] [Citado em: 10 de novembro de 2005.] http://www.ggf.org/documents/GFD.67.pdf.

55. Thompson, M., et al. **Attributes used in OGSI Authorization**. [Online] [Citado em: 10 de novembro de 2005.] http://www.ggf.org/documents/GFD.57.pdf.

56. Mullen, Shawn e Crawford, Matt. **Grid Authentication Authorization and Accounting Requirements**. [Online] [Citado em: 10 de novembro de 2005.] https://forge.gridforum.org/projects/saaa-rg/document/Draft_5_of_Requirements_Doc/en/1.

57. Vecchio, David Del, Hazlewood, Victor e Humphrey, Marty. **Evaluating Grid Portal Security**. *Proceedings of Supercomputing 2006.* [Online] Novembro de 2006. [Citado em: 5 de 12 de 2006.] http://www.cs.virginia.edu/~humphrey/papers/EvaluatingGridPortalSecurity.pdf.

58. **MySQL Enterprise Enterprise-Class Open Source Software**. [Online] [Citado em: 10 de março de 2006.] http://www.mysql.com/why-mysql/white-papers/mysql_wp_mysql-enterprise.php.

59. **J2EE Connector Architecture White Paper**. [Online] [Citado em: 10 de agosto de 2006.] http://java.sun.com/javaee/overview/whitepapers/connector.jsp.

60. Lisboa, W. B. e Farias, A. L. F. **Especificação e Implementação de uma Arquitetura Distribuída Multiplataforma utilizando: Orientação a Objeto, XML e Consciência de Contexto**. Brasil : Monografia de graduação, Departamento de Computação, UNIFENAS, 2001.

61. Borges, Willian. **Manual de Utilização do PortalMaker**. *PortalMaker.* [Online] 2007. http://portalmaker.virtualclass.com.br/Downloads.html.

APÊNDICE A – APLICAÇÃO DOS NÚMEROS PRIMOS

```
public class NumeroPrimo{public static void main (String[] Args)
{
String IN,FN; // primeiro e ultimo numero da seqüência
long in,fn; // primeiro e ultimo numero da seqüência
long counter = 0; //contador do loop
long iPrimos=0;//contador de números primos
in = Long.parseLong(Args[0]); //  recebe o valor inicial do loop
fn = Long.parseLong(Args[1]); // recebe o valor final do loop
long A=System.currentTimeMillis(); // recebe o tempo inicial em milissegundos
// estrutura de repetição - percorre do primeiro ao ultimo nr do intervalo
for (long j=in;j <= fn;j++)
        {
        for (long i=1; i <= in;i++)
                {
                // testa possivel primo
                if(in%i == 0)
                        {counter = counter+1; }
                }
        // condição de verificação do numero primo
        if(counter == 2) // se for divisivel por apenas  2 numeros
                {
                iPrimos++;
                System.out.println(in);// armazena em buffer o numero primo
                }
        in++; // Pega próximo número para teste
        counter = 0; // zera contador de verificação
        }
long B = System.currentTimeMillis(); // termino da contagem
System.out.println(iPrimos+" numeros primos encontrados em "+(B-A)+" "+"ms");
}}
```

Uma otimização dessa aplicação sugere a adição de um teste após o contador de divisões superar 2 indicando que já não é um número primo. Outra sugestão é que o processo do teste da divisão comece o processo de divisão iniciando da metade do número calculado indo ate o número 2 uma vez que nenhum número é divisível por um

valor maior que sua metade e todos os números são divisíveis por 1 e por ele mesmo. Assim o contador *counter* recebe 2 de imediato.

APÊNDICE B – EXEMPLO DO XML STARTER

XML *starter* do BoT da aplicação do estudo de caso para avaliação do desempenho do P2PGrid.

```
<?xml version="1.0" ?>
<BOT timeout="12644" delay="5500">
<TASK id="0" job="NumeroPrimo"><PARAMETERS>
        <PARAMETER type="number" value="100001" />
        <PARAMETER type="number" value="102500" />
    </PARAMETERS></TASK>
<TASK id="1" job="NumeroPrimo"><PARAMETERS>
        <PARAMETER type="number" value="102501" />
        <PARAMETER type="number" value="105000" />
    </PARAMETERS></TASK>
<TASK id="2" job="NumeroPrimo"><PARAMETERS>
        <PARAMETER type="number" value="105001" />
        <PARAMETER type="number" value="107500" />
    </PARAMETERS></TASK>
<TASK id="3" job="NumeroPrimo"> <PARAMETERS>
        <PARAMETER type="number" value="107501" />
        <PARAMETER type="number" value="110000" />
    </PARAMETERS> </TASK>
<TASK id="4" job="NumeroPrimo"> <PARAMETERS>
        <PARAMETER type="number" value="110001" />
        <PARAMETER type="number" value="112500" />
    </PARAMETERS> </TASK>
<TASK id="5" job="NumeroPrimo"> <PARAMETERS>
        <PARAMETER type="number" value="112501" />
        <PARAMETER type="number" value="115000" />
    </PARAMETERS> </TASK>
<TASK id="6" job="NumeroPrimo"> <PARAMETERS>
        <PARAMETER type="number" value="115001" />
        <PARAMETER type="number" value="117500" />
    </PARAMETERS> </TASK>
<TASK id="7" job="NumeroPrimo"> <PARAMETERS>
        <PARAMETER type="number" value="117501" />
        <PARAMETER type="number" value="120000" />
    </PARAMETERS> </TASK>
<TASK id="8" job="NumeroPrimo"><PARAMETERS>
```

```
            <PARAMETER type="number" value="120001" />
            <PARAMETER type="number" value="122500" />
    </PARAMETERS> </TASK>
<TASK id="9" job="NumeroPrimo"> <PARAMETERS>
            <PARAMETER type="number" value="122501" />
            <PARAMETER type="number" value="125000" />
    </PARAMETERS> </TASK>
<TASK id="10" job="NumeroPrimo"><PARAMETERS>
            <PARAMETER type="number" value="125001" />
            <PARAMETER type="number" value="127500" />
    </PARAMETERS> </TASK>
<TASK id="11" job="NumeroPrimo"><PARAMETERS>
            <PARAMETER type="number" value="127501" />
            <PARAMETER type="number" value="130000" />
    </PARAMETERS></TASK>
<TASK id="12" job="NumeroPrimo"><PARAMETERS>
            <PARAMETER type="number" value="130001" />
            <PARAMETER type="number" value="135000" />
    </PARAMETERS></TASK>
<TASK id="13" job="NumeroPrimo"><PARAMETERS>
            <PARAMETER type="number" value="132501" />
            <PARAMETER type="number" value="135000" />
    </PARAMETERS></TASK>
<TASK id="14" job="NumeroPrimo"><PARAMETERS>
            <PARAMETER type="number" value="135001" />
            <PARAMETER type="number" value="137500" />
    </PARAMETERS></TASK>
<TASK id="15" job="NumeroPrimo"><PARAMETERS>
            <PARAMETER type="number" value="137501" />
            <PARAMETER type="number" value="140000" />
    </PARAMETERS></TASK>
<TASK id="16" job="NumeroPrimo"><PARAMETERS>
            <PARAMETER type="number" value="140001" />
            <PARAMETER type="number" value="142500" />
    </PARAMETERS></TASK>
<TASK id="17" job="NumeroPrimo"><PARAMETERS>
            <PARAMETER type="number" value="142501" />
            <PARAMETER type="number" value="145000" />
    </PARAMETERS></TASK>
```

```
<TASK id="18" job="NumeroPrimo"><PARAMETERS>
  <PARAMETER type="number" value="145001" />
  <PARAMETER type="number" value="147500" />
     </PARAMETERS></TASK>
<TASK id="19" job="NumeroPrimo"> <PARAMETERS>
                <PARAMETER type="number" value="147501" />
                <PARAMETER type="number" value="150000" />
     </PARAMETERS></TASK>
     </BOT>
```