InGriDE: um ambiente integrado e extensível de desenvolvimento para computação em grade

Eduardo Leal Guerra

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título de Mestre
em
Ciências

Área de Concentração: Ciência da Computação
Orientador: Prof. Dr. Alfredo Goldman

Durante o desenvolvimento deste trabalho o autor recebeu financiamento de um *IBM Eclipse Innovation Grant*

São Paulo, fevereiro de 2007

Agradecimentos

A entrada e a conclusão do mestrado não teria sido possível sem a ajuda de pessoas muito especiais. Primeiramente gostaria de agradecer, mais do que este espaço permite, à minha família de sangue: meu pai Augusto, minha mãe Rosa, minha irmã Lucinha e meu irmão caçula Serginho. Vocês são a base de tudo, obrigado pelo indescritível apoio, ligações, cobranças ("cadê o mestrado ?") e amor que vocês me deram no decorrer do mestrado. Obrigado por serem tão especiais.

Agradeço com amor à minha noiva Melina. Pela paciência, dedicação, amor e até ajuda nas correções desse trabalho. Obrigado por ter estado ao meu lado em exatamente todos os momentos.

Agradeço aos que estiveram ao meu lado, muitas vezes, sem que eu soubesse. Agradeço a Deus. Agradeço aos meus avós Tom Leal e Américo Guerra pelo exemplo deixado por eles, os quais me serviram de grande motivação, e por terem estado ao meu lado em espírito.

Gostaria de agradecer também às pessoas que me ajudaram e incentivaram na entrada no mestrado. Primeiramente aos meus pais, que me incentivaram desde o início da graduação, ao professor Afonso Guedes, orientador de iniciação científica e amigo, ao grande amigo Leonardo Gatti, por todos os momentos pré-mestrado e "gás"nas inscrições nos programas de mestrado. Agradeço também toda à torcida de meus amigos de Belém.

Em São Paulo formei uma família que foi muito importante para suportar os momentos mais difíceis. Minha irmã Lucinha, minha nova irmã (loira) Lú, meus amigos e irmãos Thiago e Vladi, meu tio Gerson, meus primos Pedro, Bruna, Carol e Fabrício. Obrigado pelos momentos felizes e força nos momentos difíceis.

Em especial, gostaria de agradecer ao meu orientador, conselheiro, incentivador e amigo, professor Alfredo Goldman. Obrigado pelas experiências e aprendizados no decorrer deste mestrado.

Agradeço também aos professores e colegas do projeto InteGrade com quais tive contato nesses

anos, tanto no laboratório quanto nas famosas reuniões quinzenais. Agradeço ao Raphael, Braga, Marco, Ricardo e Giuliano Mega, pela ajuda em correções de artigos, textos e outras tarefas do mestrado. Agradeço a estes mesmos e às outras "figuras"do LCPD como Arlindo, Vidal e Celina, pelos momentos de descontração.

Gostaria de agradecer também aos amigos formados durante o mestrado. Os peruanos Christian, Karina, Jesus e Gordito. Agradeço também aos demais professores, funcionários e colegas do IME com os quais tive contato nesses anos.

Agradeço à IBM pela ajuda financeira sem a qual o desenvolvimento desse projeto não teria sido possível. Esse tipo de iniciativa é fundamental para alunos que vêm de fora. Agradeço aos professores Alfredo Goldman e Fabio Kon por terem acreditado em mim no momento de concessão da bolsa.

Não poderia deixar de fazer um agradecimento especial ao meu grande amigo Vladi, amigo de todas as horas, que me apoiou de forma sem igual na reta final do mestrado, trabalhando por ele e por mim na empresa.

Resumo

Recentes avanços proporcionaram às grades computacionais um bom nível de maturidade. Esses sistemas têm sido implantados em ambientes de produção de qualidade na comunidade de pesquisa acadêmica e vêm despertando um grande interesse da indústria. Entretanto, desenvolver aplicações para essas infra-estruturas heterogêneas e distribuídas ainda é uma tarefa complexa e propensa a erros. As iniciativas de facilitar essa tarefa resultaram, na maioria dos casos, em ferramentas não integradas e baseadas em características específicas de cada grade computacional.

O presente trabalho tem como objetivo minimizar a dificuldade de desenvolvimento de aplicações para a grade através da construção de um ambiente integrado e extensível de desenvolvimento (IDE) para computação em grade chamado InGriDE. O InGriDE fornece um conjunto único de ferramentas compatíveis com diferentes sistemas de *middleware*, desenvolvidas baseadas na interface de programação *Grid Application Toolkit* (GAT). O conjunto de funcionalidades do InGriDE foi desenvolvido com base na plataforma Eclipse que, além de fornecer um arcabouço para construção de IDEs, facilita a extensão do conjunto inicial de funcionalidades. Para validar a nossa solução, utilizamos em nosso estudo de caso o *middleware* InteGrade, desenvolvido no nosso grupo de pesquisa.

Os resultados obtidos nesse trabalho mostraram a viabilidade de fornecer independência de *middleware* para IDEs através do uso de uma interface genérica de programação como o GAT. Além disso, os benefícios obtidos com o uso do Eclipse como arcabouço para construção de IDEs indicam que os recursos fornecidos por esse tipo de arcabouço atendem de forma eficiente as necessidades inerentes ao processo de desenvolvimento de aplicações para a grade.

Palavras-chave: computação em grade, ambiente integrado de desenvolvimento, IDE, interface de programação, API.

Abstract

Computational grids have evolved considerably over the past few years. These systems have been deployed in production environments in the academic research community and have increased the interest by the industrial community. However, developing applications over heterogeneous and distributed infrastructure is still a complex and error prone process. The initiatives to facilitate this task, in the majority of the cases, resulted in isolated, middleware-specific tools.

This work has the objective of minimizing the difficulty of developing grid applications through the construction of an integrated and extensible development environment for grid computing, called InGriDE. InGriDE provides a unique set of tools, compliant with different middleware systems, based on the Grid Application Toolkit (GAT). We developed the InGriDE set of features, based on the Eclipse platform, which provides both a framework for building IDEs and the possibility to extend the initial set of features. To validate our solution we used the InteGrade middleware, developed in our research group, as our case study.

The results obtained from our work showed the viability of providing middleware independence to IDEs through the use of a generic application programming interface like GAT. Moreover, the benefits obtained through the use of Eclipse as our framework for building IDEs indicates that this kind of framework satisfies the requirements inherent to the grid application development process in a efficient way.

Keywords: grid computing, integrated development environment, IDE, application programming interface, API.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O crescimento da Internet e os avanços nas tecnologias dos computadores e das redes vem mudando a forma como cientistas e engenheiros fazem computação e como a sociedade gerencia a informação. Essas novas tecnologias permitem a interligação de vários recursos geograficamente distribuídos, com o objetivo de serem utilizados como um único e poderoso recurso. Este novo paradigma é popularmente chamado computação em grade (*Grid Computing*) [25, 54, 56]. A computação em grade e a utilização da infra-estrutura global da grade apresentam desafios significativos em todos os níveis incluindo modelos conceituais e de implementação, projeto e desenvolvimento de aplicações, sistemas de programação, infra-estrutura e serviços, gerenciamento de recursos, redes e segurança.

O rápido posicionamento da computação em grade como o paradigma dominante na computação distribuída vem acontecendo principalmente com o desenvolvimento de infra-estrutura e serviços oferecidos pelos sistemas de computação em grade, ou grades computacionais, ou sistemas de *middleware* para grades. Uma grade computacional é uma infra-estrutura de software capaz de interligar e gerenciar diversos recursos distribuídos geograficamente, com o objetivo de oferecer ao usuário da grade acesso transparente a estes recursos, independente de sua localização. Os avanços tanto na implementação quanto na padronização das grades computacionais vêm alcançando um nível de robustez que possibilita a implantação de ambientes de produção de qualidade na comunidade de pesquisa acadêmica e desperta um grande interesse da indústria.

Apesar dos avanços feitos na área de infra-estrutura, o desenvolvimento de aplicações que possam explorar o potencial da grade ainda é bastante complexo [23]. O nível de abstração fornecido pelas APIs dos sistemas de *middleware* para grades é muito baixo, fazendo com que os desenvolvedores de

aplicações tenham que se preocupar com detalhes da complexa estrutura da grade. A grade, como exposta por suas APIs, consiste em um conjunto de serviços que precisam ser combinados para que o objetivo desejado possa ser atingido. Essa visão é bem diferente da idéia proposta pela analogia com a rede de energia elétrica, na qual o poder computacional é comparado à energia elétrica e os sistemas de *middleware* à rede de distribuição [37]. Na rede de energia elétrica, o uso da energia pelos usuários é feito de forma simples, ligando seu equipamento em uma tomada sem preocupações com detalhes da transmissão e geração da energia. O atual uso da grade pelas aplicações equivale a ter-se uma parede cheia de tomadas e um equipamento com vários cabos de força, onde o funcionamento do equipamento só é alcançado com a combinação correta de cabos e tomadas.

Visando facilitar o uso da grade, várias ferramentas foram desenvolvidas para oferecer abstrações de mais alto nível do que as oferecidas pelas APIs dos sistemas de *middleware*. Apesar dos esforços da comunidade do Global Grid Forum (GGF) [72], a falta de padronização em diversas áreas da computação em grade (por exemplo, nos modelos de programação) fez com que essas ferramentas fossem desenvolvidas baseadas em características específicas de cada *middleware*. Isso resultou em um grande conjunto de ferramentas com diferentes funcionalidades e normalmente compatíveis com um *middleware* de grade específico. Na maior parte dos casos, a utilização dessas ferramentas pelos desenvolvedores é feita de forma não integrada, mesmo quando utilizando o mesmo *middleware*. Quando existe a necessidade de utilização de sistemas de *middleware* diferentes, a situação se complica ainda mais, pois as ferramentas precisam ser substituídas e as aplicações reescritas.

A diversidade de ferramentas e sistemas de *middleware* a serem utilizados em diferentes situações pelo desenvolvedor de aplicações torna a sua tarefa extremamente complicada. Nesse trabalho trataremos dois problemas que visam facilitar o desenvolvimento de aplicações para a grade. O primeiro é a obtenção de independência de plataforma de *middleware* para ferramentas e aplicações. O segundo é facilitar o uso das diferentes ferramentas necessárias no desenvolvimento de aplicações unificando-as em um único ambiente.

Considerando que a dificuldade de uso da grade tem impedido a sua adoção por vários cientistas [23], e que o desenvolvimento pleno dessa tecnologia depende de sua utilização pelos mais variados grupos de cientistas, questões relacionadas à facilidade de uso são fundamentais. Nesse sentido, as questões mencionadas acima podem ser consideradas importantes por: (1) oferecer transparência, simplicidade e unificação semântica através da independência de plataforma de *middleware*, e (2) facilitar o uso de várias ferramentas através da unificação dessas ferramentas.

1.1 Objetivos

O principal objetivo desse trabalho é o desenvolvimento de um ambiente integrado de desenvolvimento (IDE) para computação em grade extensível e independente de plataforma de *middleware*. Para atingir esse objetivo principal, os seguintes objetivos específicos foram considerados:

- Identificar um mecanismo para obtenção de independência de *middleware* para ser utilizado pela ferramenta a ser desenvolvida;
- Projetar a integração do mecanismo de independência de *middleware* ao IDE;
- Projetar o IDE de forma extensível;
- Implementar o IDE atendendo um conjunto de requisitos funcionais restrito, mas o mais abrangente possível.

1.2 Organização do Trabalho

Logo após a Introdução, o Capítulo 2, apresenta uma revisão dos principais projetos de grades Computacionais e o estado da arte das ferramentas disponíveis.

O Capítulo 3 apresenta as principais alternativas disponíveis para obtenção da independência de *middleware* e as localiza na pilha de software da grade. Em seguida, a ferramenta escolhida para ser utilizada em nosso projeto é detalhada.

O Capítulo 4 apresenta a ferramenta desenvolvida no contexto desse trabalho. São apresentados os requisitos funcionais e não funcionais considerados no projeto da ferramenta, as tecnologias utilizadas na sua construção, sua arquitetura e implementação. No final do capítulo são feitas algumas considerações de lições aprendidas durante o desenvolvimento do projeto.

O Capítulo 5 apresenta as principais ferramentas que se relacionam com o nosso trabalho pelo fornecimento de IDEs para computação em grade baseados em arcabouços como Eclipse ou Netbeans.

O Capítulo 6 apresenta as conclusões obtidas com o desenvolvimento desse trabalho, e suas principais contribuições. Finalmente, são apresentadas sugestões para trabalhos futuros a serem desenvolvidos como continuação da pesquisa realizada nesse trabalho.

Capítulo 2

Grades Computacionais e suas Ferramentas

Os benefícios e o potencial da computação em grade já foram comprovados de forma convincente através de demonstrações de sua utilização por aplicações de grande porte implantadas em plataformas de grade [54]. Além destas, diversas outras aplicações vêm sendo adaptadas para explorar o potencial oferecido pelas grades [21].

Entretanto, para muitos usuários a utilização da grade continua sendo um mistério. Dúvidas de como adaptar ou executar suas aplicações permanecem quase sem resposta. As grades computacionais oferecem serviços fundamentais, mas de nível de abstração muito baixo para os cientistas desenvolvedores de aplicações. Isso faz com que esses usuários tenham que conhecer detalhes da complexa arquitetura da grade para atingir seus objetivos.

No intuito de diminuir a distância entre as grades e sua comunidade de usuários, várias ferramentas foram desenvolvidas. Neste capítulo, apresentamos o estado da arte dessas ferramentas e usamos estudos de caso representativos para ilustrar como essas ferramentas funcionam na prática. Antes disso, revisamos os principais projetos de grades computacionais.

2.1 Grades Computacionais

A disseminação da computação em grade propiciou o surgimento de diversos sistemas desenvolvidos tanto pela indústria quanto pela comunidade acadêmica [136]. Nesta seção, apresentamos alguns dos principais sistemas de computação em grade existentes, em ordem cronológica de criação, analisando suas principais características. Mais detalhes sobre os projetos aqui apresentados podem ser encontrados na dissertação de mestrado de Andrei Goldchleger [75] e nos *websites* dos projetos,

utilizados como referência nessa seção.

2.1.1 Condor

Condor [40, 154] é um sistema distribuído para computação intensiva que fornece mecanismos de gerenciamento de tarefas, escalonamento, esquema de prioridades e monitoramento e gerenciamento de recursos. O sistema é o mais antigo aqui apresentado, tendo seu início em 1988 na Universidade de Wisconsin-Madison como derivado de um sistema ainda mais antigo, RemoteUnix [119], que possibilitava a integração e o uso remoto de estações de trabalho. O sistema vem sendo utilizado em centenas de organizações tanto no meio acadêmico quanto na indústria.

O principal objetivo do Condor é utilizar os recursos computacionais ociosos disponíveis em estações de trabalho para executar programas (Computação Oportunista [121]), preservando o proprietário do recurso de perdas de desempenho. Esta característica requer do sistema mecanismos flexíveis suficientes que permitam a adaptação do sistema às freqüentes mudanças do ambiente.

Um desses mecanismos é o chamado de *ClassAds* [142], um conjunto de expressões que são usadas para descrever tanto requisitos de execução de uma aplicação, quanto ofertas de recursos. Essas expressões são pares do tipo (atributo, valor) e podem incluir números, *strings*, intervalos, entre outros. Essas informações são utilizadas no processo de execução de uma aplicação onde um componente do Condor emparelha os *ClassAds* de requisições com *ClassAds* de recursos disponíveis, identificando as compatibilidades (*Matchmaking*). Através deste mecanismo, Condor pode identificar um recurso adequado para a execução remota da aplicação.

Em virtude da disponibilidade intermitente dos recursos compartilhados utilizados pelo Condor, faz-se necessário um mecanismo que garanta o progresso das aplicações mesmo que as máquinas com aplicações em execução tornem-se indisponíveis. Condor utiliza técnicas de *checkpointing* [120] para resolver este problema. Este mecanismo salva periodicamente o estado da aplicação e quando uma máquina torna-se indisponível, todo o estado da aplicação pode ser recuperado a partir de um *checkpoint* prévio. Este mecanismo permite também que uma tarefa seja migrada de uma máquina para outra.

Um dos problemas que ocorrem na execução de aplicações em grades é a dificuldade no gerenciamento de recursos relacionados à execução, tais como arquivos de entrada e saída. A solução do Condor para este problema é a utilização de um mecanismo de chamadas remotas de métodos,

que permite que operações de entrada e saída efetuadas na máquina na qual a aplicação executa sejam de fato efetuadas na máquina que solicitou a execução. Uma biblioteca reimplementa parte das chamadas de sistema (I/O), fazendo-as serem executadas remotamente.

A arquitetura original do Condor foi projetada para gerenciar um pequeno grupo de máquinas (*Condor Pools*) localizadas em uma rede local, pertencentes a um único domínio administrativo. Entretanto, com a proliferação dos aglomerados Condor, surgiu a necessidade de interconectar esses aglomerados. A primeira solução desenvolvida foi denominada *Gateway Flocking* [48]. Nessa solução, em cada aglomerado Condor é adicionado um novo módulo, o *gateway*, responsável pela integração do aglomerado com os demais. Esta solução não se mostrou tão prática por dificultar a manutenção dos protocolos do Condor, sendo substituída pelo *Direct Flocking*, uma solução bem mais simples na qual uma máquina de um aglomerado pode enviar requisições diretamente a outros aglomerados, sem passar pelo gerenciador de seu aglomerado. Apesar de adotada na prática, esta solução apresentou problemas de escalabilidade. Outra solução surgiu com a popularização das grades Globus (descrito na Seção 2.1.3): o projeto Condor desenvolveu a ferramenta Condor-G [61], que é o casamento das tecnologias de Condor com Globus. Do projeto Globus foi usado o protocolo seguro de comunicação inter-domínio e o acesso padronizado a uma variedade de sistemas remotos de processamento. Do Condor foi usada a parte de alocação e submissão de tarefas, recuperação de erros e a criação de um ambiente de execução amigável. Uma desvantagem desta última solução é a ausência de algumas facilidades presentes no Condor, como migração e *checkpointing*.

Uma solução normalmente adotada para conciliar as funcionalidades de Condor com a disponibilidade de recursos gerenciados por Globus é a técnica denominada *Glide In* [40], que consiste em criar um aglomerado Condor *ad hoc* sobre os recursos gerenciados por Globus. Para isso utiliza-se Condor-G para submeter cópias do *daemon* de Condor, que são executados nas máquinas como uma aplicação qualquer. Um *Collector* [1] previamente iniciado pelo usuário recebe informações dos *daemons*, criando assim um aglomerado Condor padrão.

Além dessas características, Condor disponibiliza mecanismos para execução de aplicações paralelas baseadas em MPI [125] e PVM [50]. No entanto, o suporte a aplicações MPI é limitado a execução em recursos dedicados [41, 161].

[1] Collector é o módulo responsável por manter informações do aglomerado Condor e receber requisições de execução. Tipicamente existe um Collector em cada aglomerado.

2.1.2 Legion

Legion [90,114] é um sistema com o objetivo de integrar vários recursos computacionais espalhados na rede e fornecer para os usuários a ilusão de estarem utilizando um único e poderoso computador. O projeto teve início em 1993 na Universidade de Virgínia, com desenvolvimento iniciado em 1996 e a primeira implantação ocorrendo em 1997.

A construção do Legion passou por uma longa fase de análise de requisitos e projeto do sistema. Isto ocorreu principalmente pela estrutura de sua arquitetura: uma camada básica que fornece serviços para uma camada de serviços de mais alto nível. Esta estrutura contrasta com o projeto do Globus Toolkit versão 2, que baseava-se em um conjunto de serviços razoavelmente independentes, e se parece com a versão 3 deste mesmo sistema, que baseava-se na arquitetura OGSA/OGSI (apresentada na próxima seção).

A principal característica do Legion é sua arquitetura ser baseada no paradigma de orientação a objetos, onde todas as entidades do sistema, desde computadores até serviços são representados por objetos. Estes objetos utilizam os serviços da camada básica fornecida pelo Legion. Essa arquitetura resultou em um modelo elegante e flexível que oferece, dentre outras coisas, escalabilidade (milhões de nós), interoperabilidade entre objetos escritos em diferentes linguagens de programação, etc.

O crescimento do uso do Legion unido ao sucesso do Globus Toolkit deu origem ao projeto chamado Legion-G [44]. Este projeto consiste basicamente no sistema Legion utilizando o Globus como uma infra-estrutura de baixo nível. A integração dos dois sistema era interessante para o aproveitamento dos pontos positivos de cada um. Em dezembro de 2001 foi feita uma demonstração de uma aplicação usando a infra-estrutura MPI do Globus e o LegionFS [160], um componente do sistema de arquivos distribuído de Legion.

O projeto Legion foi encerrado em 2001, quando a empresa Avaki adquiriu os direitos legais do Legion. O projeto mudou de nome para o mesmo da empresa, que removeu alguns serviços mas manteve a arquitetura original. Em 2005, a Sybase adquiriu a Avaki com intenção de expandir sua atuação no mercado de integração de dados, e hoje comercializa uma solução chamada *Sybase Avaki Enterprise Information Integration* [2] que fornece uma visão integrada de dados distribuídos, através de uma camada única.

[2] http://www.sybase.com/products/developmentintegration/avakieii

2.1.3 Globus

O sistema Globus [53, 74] é um sistema de computação em grade que propõe um tratamento vertical integrado entre aplicações, *middleware* e rede. A idéia é disponibilizar uma infra-estrutura básica que possa ser usada na construção de serviços de mais alto nível. O Globus é atualmente o maior e mais importante projeto na área de computação em grade. Sua comunidade, chamada Globus Alliance, é formada por diversas instituições de pesquisa, além de contar com o apoio da indústria através de empresas como IBM e Microsoft.

O sistema de computação em grade mantido pela Globus Alliance é denominado *Globus Toolkit*. Iniciado em 1998 com sua versão 1.0, passando pela 2.0 em 2002, e pela 3.0 em 2003, atualmente encontra-se na versão 4.0. A versão 2 do Globus Toolkit (GT2) foi a primeira versão mais sólida e amplamente usada. Esta versão caracterizava-se por um conjunto de serviços para a construção de sistemas e aplicações para a grade. Os principais serviços incluíam o serviço de informação, denominado MDS (*Monitoring and Discovery Service*), o serviço de gerenciamento de recursos, representado principalmente pelo GRAM (*Globus Resource Allocation Manager*) e o serviço de segurança, o GSI (*Globus Security Infrastructure*).

A versão 3 do Globus Toolkit (GT3) apresentou uma profunda mudança na concepção do sistema. O toolkit deixou de ser uma coleção de ferramentas e serviços definidos apenas no âmbito do projeto Globus, e passou a implementar a arquitetura OGSA/OGSI. A *Open Grid Services Architecture* (OGSA) e a *Open Grid Services Infrastructure* (OGSI) [55], definidas pelo comitê formado pelo Global Grid Forum [72] e algumas outras empresas, têm o objetivo de definir uma arquitetura padrão que permita a construção de sistemas de computação em grade, possibilitando a interoperabilidade entre diferentes grades. A arquitetura proposta fundamenta-se em serviços implementados em *Web Services* [29] que disponibilizam funcionalidades a usuários e a aplicações através da troca de mensagens. Em GT3, alguns serviços do GT2 estão disponíveis baseados neste novo padrão.

A versão 4 do Globus Toolkit (GT4) [52, 93], disponível a partir de abril de 2005, representou um avanço significativo em relação à implementação das funcionalidades de Web Services disponível no GT3 em termos de componentes fornecidos, funcionalidades, conformidade com padrões, usabilidade e qualidade da documentação. Alguns desses avanços ocorreram graças à coordenação entre as comunidades de computação em grade e de Web Services, que identificaram um conjunto de requisitos comuns, o que levou à formulação da especificação conhecida como *Web Service Resource Framework*

(WSRF) [162], que pode ser considerada um refinamento da OGSI. A especificação WSRF é seguida pelo GT4 e define como fornecer serviços para a grade usando implementações em Web Services. Atualmente, o GT4 inclui serviços e bibliotecas para monitoramento, busca e gerenciamento de recursos, segurança, gerenciamento de arquivos, dentre outros.

2.1.4 SETI@home

SETI@home [15, 146] é um projeto que utiliza computadores pessoais espalhados pelo mundo, interconectados pela Internet, para busca de inteligência extra-terrestre através da análise de ondas de rádio capturadas por um rádio-telescópio instalado no observatório de Arecibo, Porto Rico. O projeto foi proposto por David Gedye em 1995 e desenvolvido na Universidade da Califórnia, em Berkley. Originou-se da necessidade de um poder computacional que dificilmente seria obtido através do uso de supercomputadores dedicados. SETI@home foi originalmente colocado em produção em maio de 1999.

A idéia do sistema é dividir uma enorme quantidade de dados em pequenas unidades de 350 KB chamadas de unidades de trabalho (*workunits*). Os clientes processam uma unidade de trabalho por vez. Na comunicação entre clientes e o servidor é usado o protocolo HTTP, que facilita a passagem por *firewalls*. Não há comunicação entre os clientes, e estes podem operar no modo desconectado.

O projeto alcançou um grande sucesso atingindo mais de 4,5 milhões de usuário, sendo 600 mil ativos [146]. É considerado o problema que recebeu mais tempo de computação na história. Alguns fatores contribuíram para este sucesso tais como a facilidade de instalação e o fornecimento de informações aos usuário sobre o andamento do projeto e sobre resultados obtidos. Um ponto negativo do sistema é sua limitação à apenas um tipo de aplicação.

BOINC

O sistema BOINC (*Berkeley Open Infrastructure for Network Computing*) [14, 28] foi criado com o intuito de utilizar os pontos positivos do projeto SETI@home e apresentar soluções para suas limitações. Desenvolvido pelo mesmo grupo de pesquisa, BOINC é uma plataforma de sistemas distribuídos para uso de recursos computacionais de terceiros (*Public-Resource Distributed Computing*) que possibilita a construção de diversas aplicações, ao contrário de SETI@home. Essas aplicações que executam em BOINC devem ser do tipo *Bag-of-Tasks* (BOTs) acompanhando a idéia de seu antecessor. Uma aplicação *BOT* é composta por uma ou mais tarefas que podem ser executadas de

forma independente, ou seja, não existe comunicação entre as tarefas.

A estrutura do BOINC aproveitou algumas idéias de SETI@home e introduziu o conceito de *Projeto*, que é um conjunto de programas tanto do lado servidor quanto do lado cliente, que visam resolver um determinado problema. Cada projeto tem seus servidores próprios, geram as suas unidades de trabalho e fornecem componentes para verificação de resultados e tratamento de unidades processadas. Um usuário fornecedor de recurso pode participar de vários projetos simultaneamente, especificando quanto de seus recursos deseja compartilhar com cada um.

O projeto aprimorou diversos aspectos que eram deficientes em SETI@home. O mecanismo de *checkpointing* foi introduzido permitindo que o estado da aplicação fosse salvo e retomado posteriormente. Diversos mecanismos de segurança foram desenvolvidos tais como: (1) redundância para impedir falsificação de resultados, (2) assinatura de código para impedir a distribuição forjada de aplicações e (3) limite do tamanho máximo do arquivo de saída para impedir ataques do tipo negação de serviço.

2.1.5 MyGrid

O MyGrid [39, 135] é um ambiente para a execução de aplicações em recursos computacionais distribuídos. A idéia do MyGrid é simplificar o processo de implantação da grade, a ponto de permitir que os usuários possam implantar uma grade nos seus recursos disponíveis sem a necessidade de um administrador. O sistema vem sendo desenvolvido pela Universidade Federal de Campina Grande, com o apoio da empresa Hewlett-Packard.

O foco deste projeto restringi-se apenas às aplicações do tipo *Bag-of-Tasks* (BOTs). O fraco acoplamento deste tipo de aplicação simplifica a execução na grade pois reduz consideravelmente problemas de coordenação. Apesar de ser uma solução minimalista se comparada às abordagens de outros sistemas de grades, esse tipo de aplicação é extremamente útil em diversas áreas tais como biologia computacional, processamento de imagens, entre outros.

O escalonador de tarefas do MyGrid não utiliza informações sobre a disponibilidade detalhada dos recursos e sobre as necessidades das aplicações. Ele trabalha apenas com duas informações: a quantidade de máquinas disponíveis e a quantidade de tarefas de uma aplicação. Neste esquema, não é possível especificar um intervalo com a quantidade mínima de memória ou de CPU a serem utilizados na execução.

OurGrid

OurGrid [16,38] é uma grade *peer-to-peer* na qual os *peers* doam os seus recursos computacionais ociosos em troca do acesso aos recursos ociosos de outros *peers* quando precisarem. Criado pelo mesmo grupo de MyGrid, o objetivo do projeto acompanha a idéia de permitir a usuários de aplicações BOTs obterem fácil acesso e uso a recursos computacionais, mas com a diferença de fornecer um mecanismo para a utilização de recursos de terceiros. O sistema é software livre, distribuído sob licença GPL, e está em produção desde dezembro de 2004. Atualmente conta com 163 máquinas, sendo considerada uma das maiores grades computacionais no Brasil.

A arquitetura de OurGrid beneficia-se dos componentes desenvolvidos no projeto MyGrid, tendo acrescentado estruturas chamadas *peers*, que são responsáveis por implementar a lógica de compartilhamento dos recursos na rede *peer-to-peer*. OurGrid utiliza um modelo econômico de *rede de favores* no qual um *site* doa os seus recursos ociosos como favor, esperando ser priorizado quando necessitar de favores da comunidade. Este modelo foi projetado de forma descentralizada através dos *peers*, o que fornece maior simplicidade para implantação e escalabilidade.

2.1.6 Outras grades computacionais

Atualmente existem vários projetos de grades computacionais espalhados pelo mundo. Os projetos apresentados nas seções anteriores foram selecionados por serem considerados mais importantes e por representar o que se tem feito com relação a desenvolvimento de grades computacionais. Vários projetos, não menos importantes que os apresentados, ficaram de fora desta lista. Alguns exemplos desses projetos incluem o Unicore [157], o GridLab [9, 86], o GLite [73] e o próprio InteGrade, desenvolvido no nosso grupo de pesquisa e apresentado com detalhes na Seção 4.3.1.

Uma lista mais completa de projetos de grades computacionais com os respectivos ponteiros para seus *websites* pode ser encontrada em [82].

2.1.7 Resumo

Os sistemas de computação em grade vistos podem ser analisados em dois grupos que têm propósitos diferentes. No primeiro grupo estão os sistemas chamados oportunistas, que utilizam recursos computacionais ociosos disponíveis em estações de trabalho. Neste grupo estão o Condor, projeto mais antigo que se destaca pelo seu suporte às aplicações paralelas e os seus mecanismos de

ClassAds e *checkpointing* para lidar com o dinâmico ambiente da grade. O projeto BOINC, sucessor de Setti@Home, também entra nessa grupo apresentando um interessante modelo para atender aplicações BOTs que visam utilizar recursos de terceiros.

O segundo grupo, formado pelos sistemas considerados grades computacionais tradicionais, traz como carro chefe o Globus. Este sistema, entre todos, é o que tem o uso mais difundido, é o único que implementa o padrão OGSA na versão GT3 e WSRF na versão GT4, e também oferece suporte à aplicações paralelas. Em virtude da sua ampla disseminação, o Globus Toolkit tem um conjunto enorme de ferramentas desenvolvidas, fato que é de especial interesse para o nosso trabalho. Ainda neste grupo estão o Legion, que apresentou uma elegante e flexível arquitetura orientada a objetos antes do projeto acabar; o MyGrid/OurGrid, com um modelo que fornece simplicidade para implantação e escalabilidade atendendo aplicações BOTs.

2.2 Ferramentas

Uma ferramenta para a grade, ou *Grid application-level tool*, é um software construído baseado na infra-estrutura básica do *middleware* de grade para disponibilizar novas funcionalidades e abstrações de alto nível que permitam aos usuários escrever e executar suas aplicações na grade [21]. Portanto, essas ferramentas estão situadas acima dos serviços da grade na pilha de software ilustrada na Figura 2.1, que mostra as ferramentas no contexto da arquitetura da grade definida por Foster et al. [56]. Apesar da figura apresentar apenas uma camada de ferramentas (*Grid application-level tools*), algumas ferramentas são construídas baseadas nas abstrações fornecidas por outras ferramentas, formando várias camadas de ferramentas na arquitetura ilustrada. É importante lembrar ainda que a atuação das ferramentas no ambiente distribuído da grade pode ocorrer apenas no lado cliente, apenas no lado servidor ou em ambos.

A necessidade por ferramentas incentivou o desenvolvimento de vários projetos focados nas diferentes grades computacionais disponíveis. Projetos como GridRPC [147], MPICH-G2 [105], Triana [152] e GridSphere [131] são alguns poucos exemplos. Em virtude da grande quantidade de ferramentas disponíveis, estruturamos nossa discussão com base na taxonomia definida por Bal et al. [21] com uma pequena modificação. A taxonomia de Bal et al. modificada e traduzida está ilustrada na Figura 2.2. A modificação realizada foi a inclusão da subcategoria de *Ambientes de resolução de problemas*.

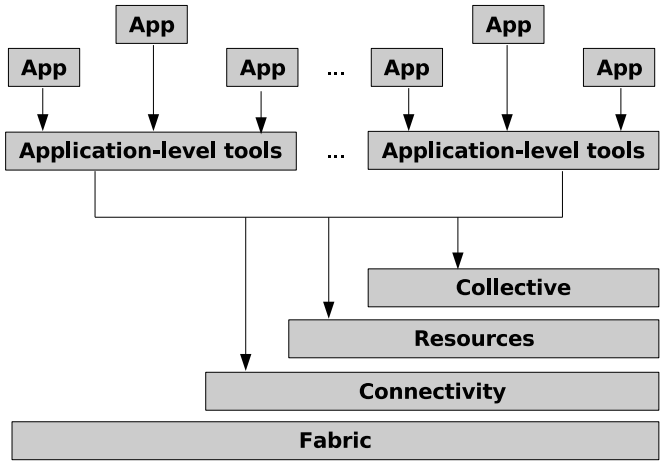

Figura 2.1: Ferramentas na pilha de software da grade.

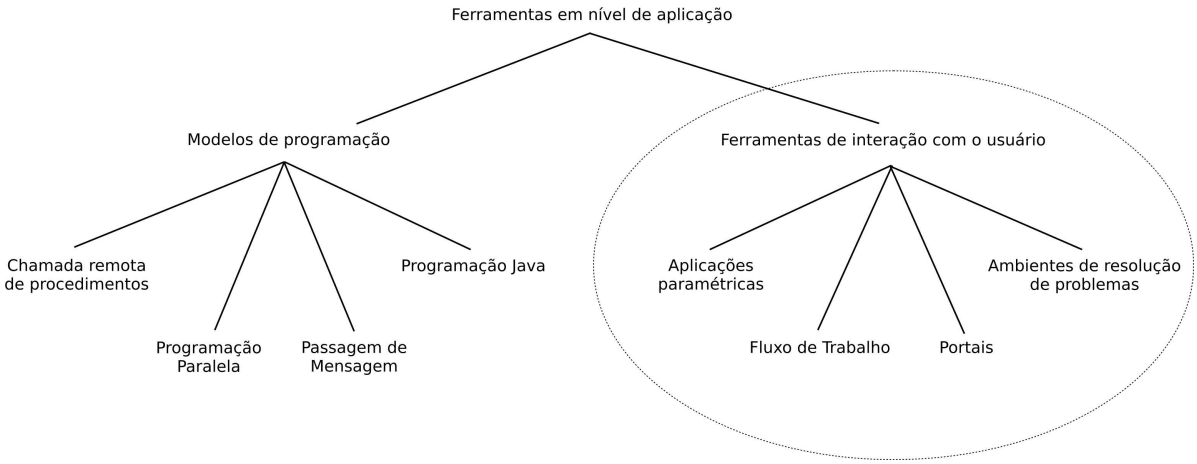

Figura 2.2: Taxonomia das Ferramentas.

Bal et al. dividem as *ferramentas em nível de aplicação* (*Grid application-level tools*) em *modelos de programação* (*Grid programming models*) e *ferramentas de interação com o usuário* (*Grid application execution environments*), e ainda as subdivide em subcategorias.

A categoria de *modelos de programação* engloba as ferramentas que oferecem abstrações de alto nível a serem usadas para construir aplicações, disponibilizando suporte em tempo de execução para

essas abstrações. Modelos de programação tradicionais como chamada remota de procedimentos (RPC), passagem de mensagem e *master-worker* possuem ferramentas específicas para adaptar os modelos à plataforma da grade. Exemplos dessas ferramentas são GridRPC e MPICH-G2. Ferramentas dessa categoria caracterizam-se por oferecer interfaces de programação e ambiente de execução, não tendo como seu foco as interfaces de interação com o usuário.

A categoria de *ferramentas de interação com o usuário*, também conhecidas na literatura como *Grid Computing Environments* [58, 60], são ferramentas através das quais os usuários podem acessar a grade de suas estações de trabalho e terem a ilusão da grade ser um supercomputador dentro de sua máquina. A idéia dessas ferramentas é fazer com que a grade se aproxime da praticidade de utilização existente nas redes elétricas. Ao contrário da primeira categoria descrita, as ferramentas desta têm forte presença de interfaces de interação com o usuário.

Em nosso estudo, enfatizamos as ferramentas que estão mais próximas dos usuários, as *ferramentas de interação com o usuário* (em destaque na Figura 2.2). Essas ferramentas oferecem aos usuários dois grupos de funcionalidades [60]: (1) ambiente de programação, que normalmente está integrado às bibliotecas de *runtime* dos sistemas de computação em grade específicos ou das ferramentas dos *modelos de programação*; (2) ferramentas para interação com os serviços da grade.

Em nossa análise vamos considerar principalmente as funcionalidade fornecidas pelas ferramentas, deixando de lado o modelo computacional utilizado na integração com o sistema de grade. Na categoria a qual nosso trabalho tem interesse, as *ferramentas de interação com o usuário*, três subcategorias foram definidas por Bal et al.: ambientes para construção de *Fluxos de Trabalho* (*Workflows*), *Portais* (*Portals*) e ambientes específicos para *Aplicações Paramétricas* (*Parameter Sweep Applications* (PSAs)). Apresentados por Bal et al. dentro da categoria de *Fluxos de Trabalho*, os *Ambientes de Resolução de Problemas* (*Problem Solving Environments* (PSEs)) serão apresentados no nosso trabalho como uma categoria separada para ressaltar suas características específicas. A classificação das ferramentas foi feita de acordo com a funcionalidade mais marcante da mesma, portanto uma ferramenta pode incluir (e quase sempre inclui) características de mais de um grupo.

2.2.1 Fluxos de Trabalho

Um fluxo de trabalho consiste de um conjunto de módulos de software com entrada e saída bem definidas, conectados de forma ordenada para alcançar um determinado objetivo [21]. Atualmente os

fluxos de trabalho têm uma grande importância tanto na área acadêmica quanto na indústria, pois as aplicações de hoje não são mais entidades monolíticas, e sim fluxos de trabalho complexos. A grade, pela adequação de sua arquitetura a este modelo, torna-se uma candidata natural para o ambiente de execução destas aplicações. Este casamento permite que um módulo da aplicação execute em um nó da grade e a saída desta execução sirva de entrada para outro módulo que executará em outro nó.

Em virtude desse casamento, vários projetos na área de grades desenvolveram interfaces gráficas para a criação de aplicações de fluxo de trabalho. Um levantamento não acabado de várias dessas ferramentas está sendo feito por Slominski [149]. Apesar da quantidade considerável de projetos, ainda não existe uma linguagem padrão para a especificação dos fluxos de trabalho para as aplicações para a grade. O grupo de pesquisa *Workflow Management* (WFM-RG) do *Global Grid Forum* [71] tem trabalhado nesse sentido, focando em vários aspectos do gerenciamento do fluxo de trabalho no ambiente de grade. Atualmente, as linguagens usadas na especificação dos fluxos de trabalho nas ferramentas direcionadas a ambientes de grade são a *Grid Services Flow Language* (GSFL), *Service Workflow Language* (SWFL) e a *Grid Workflow Execution Language* (GWEL) [123]. Todas estas linguagens foram definidas baseadas nas linguagens de composição de Web services, a *Web Services Flow Language* (WSFL) [115] ou a *Business Process Execution Language for Web Services* (BPEL4WS) [17].

O projeto TENT [144] é um exemplo de ferramenta que fornece um ambiente para a criação e execução de aplicações de fluxo de trabalho. A ferramenta permite que o usuário desenvolva a sua aplicação, submeta-a para execução e a controle, além de permitir a visualização dos resultados. A ferramenta foi escrita na linguagem Java na sua maior parte, fornece uma interface gráfica, e gerencia a computação em recursos distribuídos usando os serviços de CORBA [134]. TENT utiliza os serviços de Globus (Seção 2.1.3) para a execução na grade.

DAGman [42] é um sistema meta-escalonador bastante popular que possibilita a execução de aplicações de fluxo de trabalho no sistema Condor (Seção 2.1.1). DAGMan gerencia as dependências das tarefas de acordo com um grafo dirigido acíclico (Directed Acyclic Graph - DAG), otimizando a ordem da execução das tarefas.

Seguindo a tendência do padrão OGSA, o projeto XCAT [109] utiliza um modelo de programação baseado em componentes chamado *Commom Component Architecture* (CCA) [18] que propõe a construção de aplicações através da composição de componentes de software. XCAT usa uma abordagem

na qual um componente CCA é modelado como um conjunto de serviços de grade, permitindo que componentes CCA sejam acessíveis para o cliente da grade. O projeto é baseado no sistema Globus (Seção 2.1.3) e na API Cog Kit [111], e usa RMI sobre XSOAP para a comunicação. Para interação com usuário fornece uma interface de linha de comando.

A seguir apresentamos P-Grade, que apesar de ser uma ferramenta proprietária, foi escolhida para representar os ambientes para construção de fluxos de trabalho por ser a ferramenta desta categoria mais rica em funcionalidades. Além disso, P-Grade utiliza tanto Globus quanto Condor como *middleware* de grade.

P-Grade

P-Grade (*Parallel Grid Runtime and Application Development Environment*) [3,102] é uma ferramenta que fornece um conjunto integrado de ferramentas de programação para o desenvolvimento de aplicações a serem executadas tanto em sistemas de computação distribuídos homogêneos quanto heterogêneos, tais como supercomputadores, *clusters* e grades. P-Grade foi desenvolvido pelo instituto de pesquisa húngaro *MTA SZTAKI Computer and Automation* tendo como principal pesquisador Peter Kacsuk. O sistema já foi testado no *Hungariann Supercomputing Grid* e têm sido intensamente usado em várias universidades no Reino Unido, Hungria e Polônia.

A idéia de P-Grade é fornecer um ambiente gráfico de alto nível que omita os detalhes de baixo nível de várias APIs de programação dos sistemas distribuídos e seja capaz de gerar código PVM, MPI ou GAT [10] de acordo com a plataforma de execução. Para alcançar tal objetivo fornece várias ferramentas que permitem, entre outras coisas, a criação de fluxos de trabalho. A criação das aplicações é feita usando a linguagem GRAPNEL (GRAphical Process NEt Language) [101] junto com o editor gráfico GRED (Figura 2.3). GRAPNEL é uma linguagem híbrida onde a parte gráfica é usada para definir as atividades paralelas da aplicação e a parte textual é usada para descrever as atividades seqüenciais. Na parte gráfica os processo são representados por componentes gráficos no formato de retângulos e a troca de mensagens entre os processos são representadas por setas. A parte textual é usada para fornecer a lógica dos processos através de codificação nas linguagens C/C++ ou Fortran.

As funcionalidades de P-Grade fornecem suporte às fases de projeto, edição, execução e monitoramento da aplicação, e baseiam-se em três aspectos do ponto de vista do usuário final da grade.

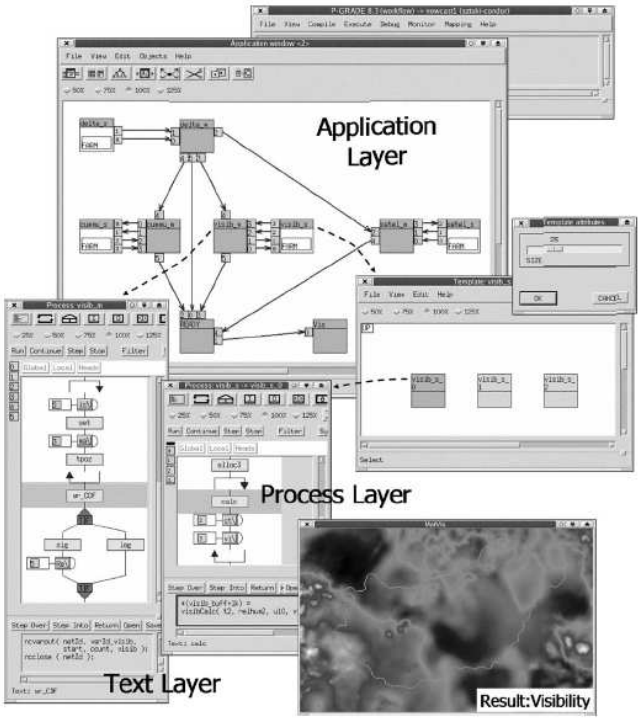

Figura 2.3: Diferentes visões do editor GRED.

O primeiro aspecto é a criação de aplicações para a grade, levando em consideração a portabilidade do programa escrito. Para este aspecto P-Grade fornece um compilador que gera código PVM, MPI e futuramente GAT a partir da notação gráfica definida pelo usuário. O segundo aspecto está relacionado à execução de programas paralelos na grade. P-Grade gera processos para a execução nos sistemas Condor, Condor-G e Globus-2. O outro aspecto considerado é a migração de processos de aplicações paralelas. P-Grade fornece um mecanismo de *checkpointing* e migração para programas PVM quando estes estão sendo executados como processos Condor ou Condor-G.

Diante da necessidade de monitorar a migração de aplicações paralelas na grade, o grupo de pesquisa do P-Grade desenvolveu uma infra-estrutura para ambientes heterogêneos adaptando a já existente GRM/PROVE [22], antes restrita ao monitoramento em ambientes homogêneos. O resultado disso foi a criação de uma ferramenta genérica para monitoramento em grades chamada Mercury [2], que foi desenvolvida e inserida no projeto GridLab [9]. O uso desta ferramenta possibilita ao P-Grade, por exemplo, disponibilizar a visualização do gráfico de barras horizontais da Figura

2.4. Neste gráfico cada barra equivale a um processo da aplicação paralela e as setas entre as barras representam a troca de mensagens entre os processos.

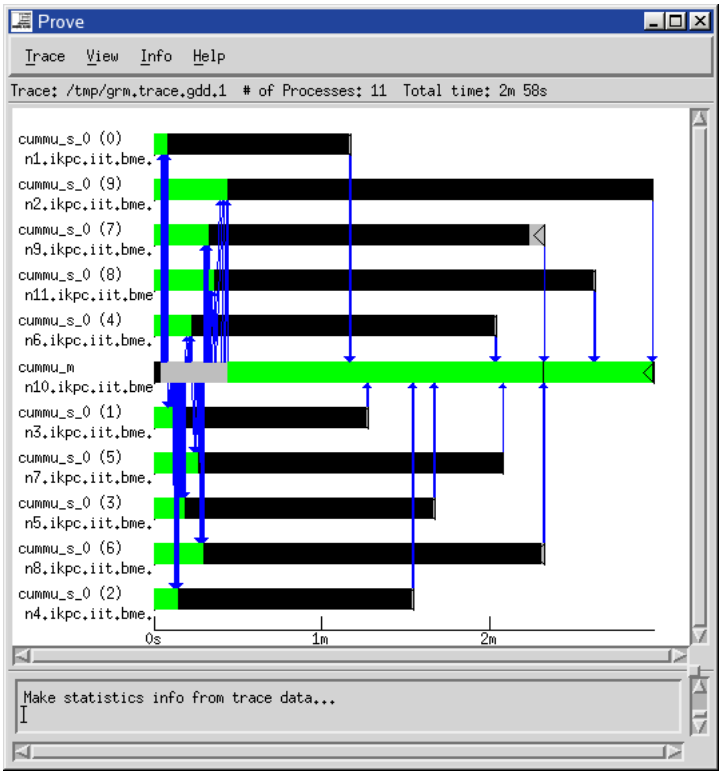

Figura 2.4: Visualização da execução de uma aplicação paralela.

Atualmente o mecanismo de *runtime* de fluxo de trabalho utilizado pelo P-Grade é o Condor DAGMan. Entretanto, está prevista a criação de um *Grid Application Manager* genérico que possibilite a execução de uma aplicação de fluxo de trabalho em diferentes grades.

Um portal P-Grade [4] foi desenvolvido para o acesso via Web às funcionalidades da camada de fluxo de trabalho, bem como à submissão de aplicações.

2.2.2 Ambientes de Resolução de Problemas

Um ambiente de resolução de problema é um sistema que disponibiliza todas as facilidades computacionais necessárias para resolver uma determinada classe de problemas [64]. No contexto de

computação em grade, os ambientes de resolução de problemas são ferramentas com um conjunto de funcionalidades específicas para um determinado domínio de aplicação tais como química, biomedicina ou astrofísica. Esse tipo de ferramenta tem se mostrado extremamente eficiente na disseminação da computação em grade junto à comunidade científica [21].

Um exemplo de ambiente de resolução de problema é o projeto Cactus [8], que através de uma arquitetura modular, fornece um arcabouço para construção de aplicações para diversas áreas da ciência e engenharia tais como modelagem climática, engenharia química e astrofísica. A arquitetura modular de Cactus permite o uso de Globus (Seção 2.1.3) ou GridLab [9] como *middleware* de grade. Além disso, o projeto fornece uma interface Web para interação do usuário com a grade.

O projeto Proteus [34] é um ambiente de resolução de problema para aplicações de Bioinformática. Esse projeto define uma metodologia baseada em ontologias para a descrição das aplicações de bioinformática no formato de fluxos de trabalho de componentes de software distribuídos. A arquitetura do Proteus é baseada em um sistema de grade que usa como base o sistema Globus (Seção 2.1.3). Os usuários do Proteus podem utilizar o ambiente visual VEGA [35] para projetar e executar suas aplicações na forma de fluxos de trabalho.

A seguir apresentamos Triana como a ferramenta representante dos ambientes de resolução de problemas. Triana, junto com Cactus, são os ambientes de resolução de problemas mais populares e utilizados pela comunidade, além de ambos serem software livre. Triana foi escolhido para ser apresentado em virtude da sua maior quantidade de funcionalidades disponíveis.

Triana

Triana [152,153] é um ambiente de resolução de problema, distribuído como software livre, desenvolvido na Universidade de Cardiff. O sistema fornece uma interface gráfica bastante intuitiva para a composição de aplicações. A arquitetura de Triana é composta por componentes plugáveis chamados *units* que oferecem as funcionalidades do ambiente. O desenvolvedor constrói suas aplicações compondo os *units* disponíveis através da interface gráfica da ferramenta que funciona da mesma forma que as ferramentas de fluxo de trabalho apresentadas na Seção 2.2.1. Em virtude do seu uso por cientistas de diversas áreas tais como análise de dados, processamento de imagens e texto, já existe uma biblioteca com mais de 500 *units* abrangendo diversas aplicações. Além disso, Triana pode ser estendido através da criação de *units* com novas funcionalidades.

A interface gráfica do Triana (Figura 2.5) é uma aplicação Java e consiste de quatro componentes principais que juntos compõem o ambiente de desenvolvimento. Neste ambiente o desenvolvedor monta a sua aplicação utilizando as funcionalidades disponíveis no ambiente.

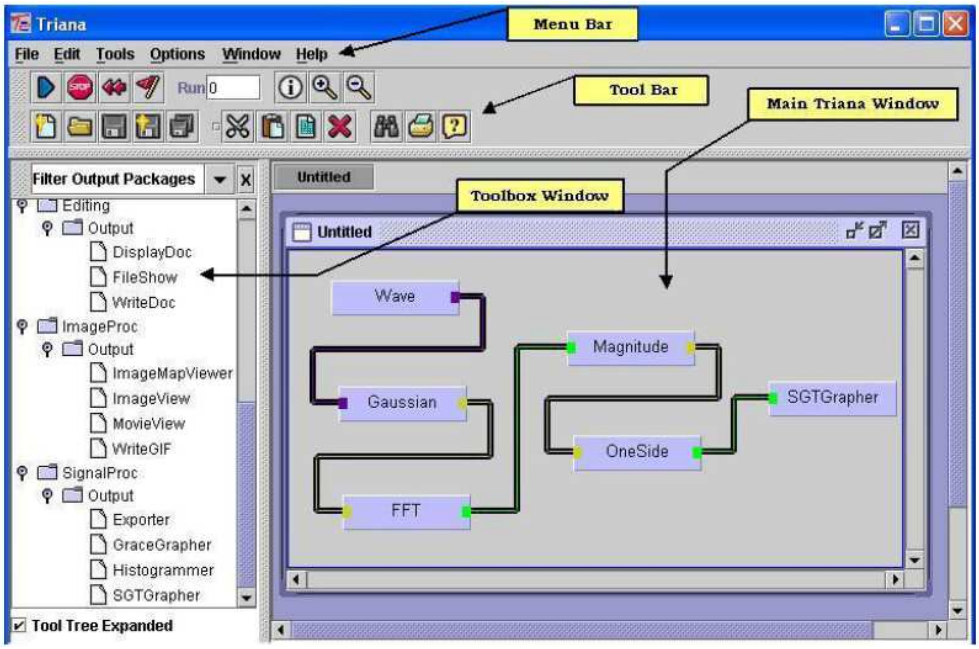

Figura 2.5: Interface gráfica do Triana.

Triana oferece também facilidades para computação distribuída, nosso principal interesse. Os componentes para computação distribuída estão divididos em duas categorias: orientados a serviço (*service-oriented*) e os orientados a grade (*grid-oriented*). A primeira categoria é formada pelos componentes de serviços *Peer-to-Peer* ou *Web services*, que são acessíveis remotamente. A segunda inclui os componentes utilizados na interação com os serviços das grades computacionais, que serão objeto de nosso estudo.

Os componentes *grid-oriented* utilizam a API GridLab GAT [10] para acessar os componentes distribuídos das grades computacionais. O uso dessa API permite que os componentes de Triana sejam desenvolvidos independente da grade computacional. O estado atual de desenvolvimento de GAT permite que a submissão de tarefas no ambiente do Triana possa ser feita tanto para o serviço GRMS de GridLab ou para o GRAM de Globus. Quando o suporte de GAT à submissão de tarefas

para o Condor estiver disponível, Triana já estaria pronto para submeter tarefas para esta grade.

O uso desses componentes *grid-oriented* permite ao usuário submeter uma aplicação para execução na grade, monitorar o seu estado e até interromper uma execução. Esse monitoramento pode ser retomado inclusive após o usuário desativar a aplicação cliente e reativá-la. O ambiente oferece também componentes que facilitam múltiplas submissões com diferentes parâmetros.

2.2.3 Portais

O termo *portal* ainda não está uniformemente definido na comunidade da Ciência da Computação. Algumas vezes representa um conjunto de computadores integrados, espaços de comércio eletrônico ou de informação [57,59,150]. O uso do termo na computação em grade é feito com o significado mais usado: um serviço para uma comunidade, com um único ponto de acesso a um sistema integrado, fornecendo informações, dados, aplicações e serviços.

Um *Portal de grade* (*Grid Portal*) é um portal especializado, direcionado a usuários de grades em produção [112]. Um portal deste tipo fornece informações sobre o estado de recursos e serviços da grade. A interface do portal oferece transparência aos usuários de grande parte da complexidade relacionada aos serviços da grade. Isso permite que algumas tarefas como a implantação de uma aplicação em uma grade em produção seja bastante facilitada.

Os *Portais de grade* não se restringem apenas às tecnologias usadas em portais Web comuns. Alguns módulos específicos para a interação com a grade são necessários para atender às necessidades dos usuários. Em virtude disso, algumas tecnologias específicas para o desenvolvimento de portais de grade surgiram. Li e Baker [116, 122] fizeram uma revisão de vários portais (Web) disponíveis para a grade. Eles destacaram três gerações de tecnologias de portal que são descritas a seguir.

A primeira geração de portais de grade focou seus esforços na definição das funcionalidades da interface gráfica e os serviços necessários para atendê-las tais como autenticação de usuário, submissão de tarefas, monitoramento de recursos e transferências de dados. Outra característica marcante dos portais desta geração foi o uso restrito do Globus como *middleware*. As limitações destes portais eram a falta de personalização das funcionalidades (eram fixas) e o uso dos serviços de grade restrito aos de Globus. Alguns arcabouços para criação de portais chegaram a ser criados como o GridPort 2.0 [87] e o Grid Portal Development Kit (GPDK) [81], mas era complicado para os usuários finais construírem portais que atendessem às necessidades dos seus requisitos específicos fazendo uso destes

arcabouços.

A segunda geração de portais faz uso do conceito de *Portlets* [5], que são serviços personalizáveis que rodam no servidor Web. O conteúdo de um *portlet* é normalmente agregado ao conteúdo de outros *portlets* para formar a página de um portal. Através do uso desta tecnologia é possível construir portais com funcionalidades personalizáveis, reutilizar *portlets* de terceiros, entre outras vantagens. Atualmente os *portlets* têm recebido uma crescente atenção tanto da comunidade de computação em grade quanto da indústria. Com o intuito de definir padrões para a interoperabilidade entre *portlets* de diferentes fornecedores, dois grupos têm trabalhos neste sentido. O primeiro, o JCP (Java Community Process) [97] tem trabalhado na JSR168 [3] [100], uma API específica para portais baseados na linguagem Java. O segundo, o OASIS (Organization for the Advancement of Structured Information Standards) [132] tem trabalhado na definição da WSRP (Web Services for Remote Portlets) [31], uma API universal que permite a portais de qualquer tipo usar *portlets* de qualquer tipo. Atualmente, a especificação JSR168 é a mais usada em virtude do predomínio da linguagem Java em ambientes servidores.

Um *portlet* de um *portal de grade* não é apenas um *portlet* integrado ao portal, mas é um componente associado a um serviço de grade na sua retaguarda (Figura 2.6). A este tipo especial de *portlet* chamamos de *Portlet de grade* (*Grid Portlet*). Um *portal de grade* construído a partir de *portlets de grade* pode fornecer ao usuário a possibilidade de integrar serviços de grade fornecidos por diferentes sistemas de *middleware*. Esta geração de portais é o atual estado da arte em portais de grade, sendo GridSphere (apresentado a seguir) uma das ferramentas mais utilizadas na construção de *portlets de grade*. A terceira geração, ainda em fase de pesquisa, estende o uso de *portlets de grade* associando-os a serviços semânticos de grade. PortalLab [117], por exemplo, oferece mecanismos para relacionar necessidades de usuários e aplicações a conhecimento sobre os recursos computacionais disponíveis e outras características da grade.

A emergência de OGSA como padrão na construção de grades computacionais orientadas a serviço faz com que o desenvolvimento de *portlets* leve em consideração este padrão. Os futuros *portlets* devem ser compatíveis com OGSA, o que significa que estes *portlets* devem ser associados a serviços de grade desenvolvidos e implantados através de um *middleware* compatível com OGSA.

[3] JSR: Java Specification Request

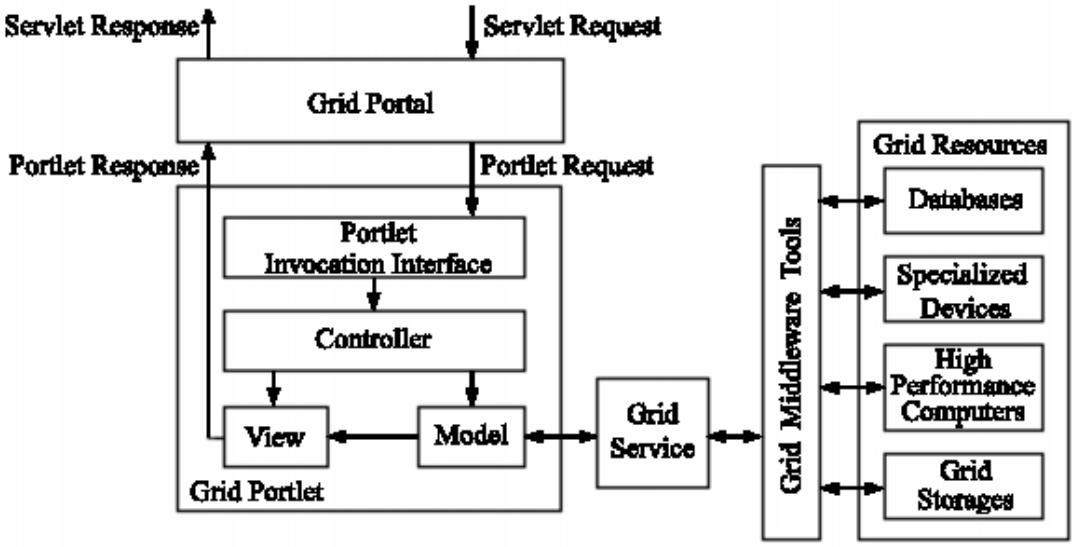

Figura 2.6: Arquitetura de um portal de segunda geração.

GridSphere

GridSphere [89, 131] é um *portal de grade* de segunda geração que fornece a implementação de um arcabouço de *portlet* baseado na API de *portlets* da IBM e compatível com a especificação da JSR168. GridSphere permite que desenvolvedores criem aplicações Web baseadas em *portlets*, não restritas a grades, que podem ser executadas e administradas no *container* de *portlets* do Gridsphere.

GridSphere foi desenvolvido baseado nas lições aprendidas em projetos de portais de grade anteriores, tais como Grid Portal Development Kit (GPDK) [81] e Astrophysics Simulation Collaboratory (ASC) [1]. O projeto GridLab [9] vem sendo usado como protótipo inicial para o levantamento das necessidades dos usuários do portal GridLab. Entretanto, a arquitetura de GridSphere foi projetada como um arcabouço *caixa-branca* e com intenso uso de padrões de projeto. Isso resultou em uma arquitetura modular o suficiente para atender às necessidades de diferentes comunidades não pertencentes ao projeto GridLab.

O arcabouço fornece um conjunto de *portlets* básicos que oferecem as funcionalidades básicas necessárias para o uso de um portal. Os *portlets* básicos oferecem funcionalidades como o autenticação, pedido de contas de usuário, gerenciamento de contas de usuário, gerenciamento de per-

missões de usuário e gerenciamento de *portlets*, permitindo ao usuário a adição ou remoção de *portlets* à sua área de trabalho.

O projeto GridSphere oferece suporte a funcionalidades relacionadas à grade através de um módulo chamado *Grid Portlets* [143]. *Grid Portlets* foi disponibilizado em junho de 2005 e vem sendo construído a partir dos *portlets* básicos do arcabouço GridSphere. O objetivo é fornecer um arcabouço para o desenvolvimento de *portlets de grade*. *Grid Portlets* fornece como implementação base o suporte ao Globus Toolkit 2 (GT2) e, distribuída separadamente, uma implementação para o GT3 e GT4. A arquitetura modular facilita o desenvolvimento de implementações que atendam a outras grades computacionais. Os principais *portlets* de *Grid Portlets* são descritos a seguir:

- Gerenciamento de credenciais e autenticação única: fornece suporte à delegação de credenciais a um portal através do *Credential Retrieval Portlet* e permite que usuários façam a autenticação na grade usando essas credenciais;

- Acesso a Recursos: acesso aos recursos da grade através do *Resource Browser Portlet* (Figura 2.7), que permite visualização dos serviços, softwares e contas disponíveis nestes recursos;

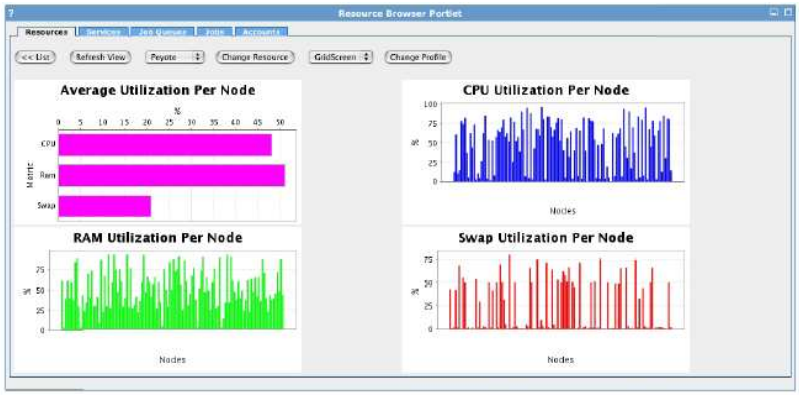

Figura 2.7: Resource Browser Portlet exibindo detalhes de um recurso.

- Submissão de tarefas: permite a submissão e monitoramento das tarefas em recursos computacionais remotos. O *Job Submission Portlet* disponibiliza também a visualização do histórico das tarefas e o recebimento de notificações no término das tarefas;

- Gerenciamento de arquivos: o *File Browser Portlet* (Figura 2.8) permite a navegação em sis-

temas de arquivos remotos. Além de fornecer suporte a comandos básicos (listar, renomear, criar pasta, etc.), este *portlet* pode ser usado para *upload* e *download* de arquivos em recursos remotos. O *File Activity Portlet* permite ao usuário monitorar o progresso de tarefas com arquivos como *upload*.

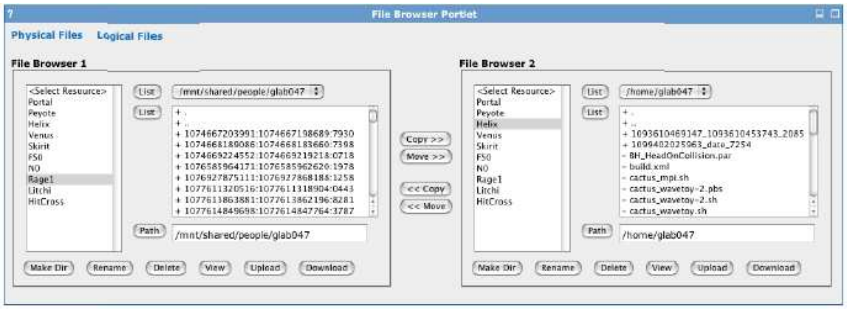

Figura 2.8: Transferência de arquivos usando o File Browser Portlet.

O uso do GridSphere tem se disseminado rapidamente pela comunidade de computação em grade. O motivo disso é a sua facilidade de instalação, suporte a personalizações e sua boa documentação. Vários projetos adotaram GridSphere como a sua plataforma para desenvolvimento de portal de grade além de GridLab, dentre eles estão o UK E-Science Program [49], D-Grid [45] e P-Grade (Seção 2.2.1).

2.2.4 Ferramentas para Aplicações Paramétricas

Ferramentas para Aplicações Paramétricas, também chamadas de *estudo paramétrico* (*parametric study* ou *parameter study*), são aplicações geralmente da área científica ou de engenharia, onde a execução da aplicação com diferentes conjuntos de parâmetros de entrada é importante para o problema. Essas aplicações são fáceis de implementar na grade pois as diferentes execuções iniciadas com diferentes parâmetros são completamente independentes [103]. Este tipo de aplicação está presente em vários áreas da ciência e da engenharia tais como bioinformática, simulação de eventos discretos, computação gráfica, astronomia, etc. [21].

O gerenciamento da implantação dessas aplicações na grade possui diversas tarefas repetitivas que podem ser automatizadas. A necessidade de analisar o resultado das execuções com seus respectivos conjuntos de parâmetros, fazer pequenas mudanças e submeter novamente são exemplos de

necessidades dos usuários deste tipo de aplicação.

Várias ferramentas foram desenvolvidas e utilizadas com sucesso com o objetivo de gerenciar aplicações paramétricas em ambientes de grade. Nimrod/G [7] e AppLeS Parameter Sweep Template (APST) [36] representam um grupo que se concentrou em aplicações com apenas um processo e suas funcionalidades priorizaram mecanismos de escalonamento e instrumentos para implantação na grade.

Apesar do sucesso desses projetos um grande desafio para área de pesquisa de *estudos paramétricos* é a execução de aplicações complexas como fluxos de trabalho (Seção 2.2.1). ILab (apresentado a seguir) e SEGL [118] atenderam a essa demanda com abordagens diferentes. ILab permite a criação de fluxos de trabalho graficamente onde o usuário pode definir explicitamente como distribuir e replicar os parâmetros de entrada na grade e quantos processos independentes devem ser disparados para cada entrada. Essa abordagem é muito estática restringindo a exploração na natureza dinâmica da grade, que permite a alocação dinâmica de recursos. A abordagem de SEGL coloca mais ênfase na exploração do dinamismo da grade, permitindo a seleção dinâmica de conjuntos de parâmetros baseada em resultados intermediários. SEGL também dispõe de interface gráfica para a criação dos fluxos de trabalho.

O projeto P-Grade, apresentado na Seção 2.2.1, publicou recentemente um mecanismo para suporte a *estudos paramétricos* em fluxos de trabalho com objetivo de disponibilização em ambiente de portal, o que não é oferecido pelas ferramentas anteriores. A proposta desse projeto difere dos objetivos de ILab e SEGL, pois nestes últimos o fluxo de trabalho é formado por componentes que representam os estágios de processamento do estudo paramétrico. P-Grade propõe a realização de *estudos paramétricos* em aplicações de fluxo de trabalho DAG já existentes. Neste caso a aplicação em si é o fluxo de trabalho. Esta funcionalidade ainda não está disponível na atual versão 2.4 do portal, constando apenas como funcionalidade planejada no website do projeto [3].

A seguir apresentamos ILab, escolhido para representar a categoria em virtude da sua interface gráfica com o usuário (GUI) rica e de alto nível.

ILab

ILab [96,163] é uma ferramenta para a criação, execução e monitoramento de *estudos paramétricos*. O desenvolvimento desta ferramenta foi motivada pelas necessidades dos cientistas do *Ames Research Institute* da NASA em automatizar tarefas relacionadas à execução de aplicações que necessitavam de

estudos paramétricos. ILab foi projetado para ser uma ferramenta amigável e fácil de usar, fornecendo uma interface gráfica com o usuário (GUI) de alto nível. O sistema foi desenvolvido na linguagem Perl [137] e Tk [155], e usa atualmente o sistema Globus como *middleware* de grade.

As funcionalidades fornecidas por ILab direcionadas para *estudos paramétricos* incluem: a parametrização de valores de arquivos de entrada, que facilita a construção de um conjunto de arquivos de entrada (Figura 2.9); a possibilidade de mascarar algumas execuções, ou seja, evitar a execução de determinadas combinações de parâmetros; visualização de resultados; geração de scripts; geração de arquivos específicos para submissão no sistema Globus; dentre outras.

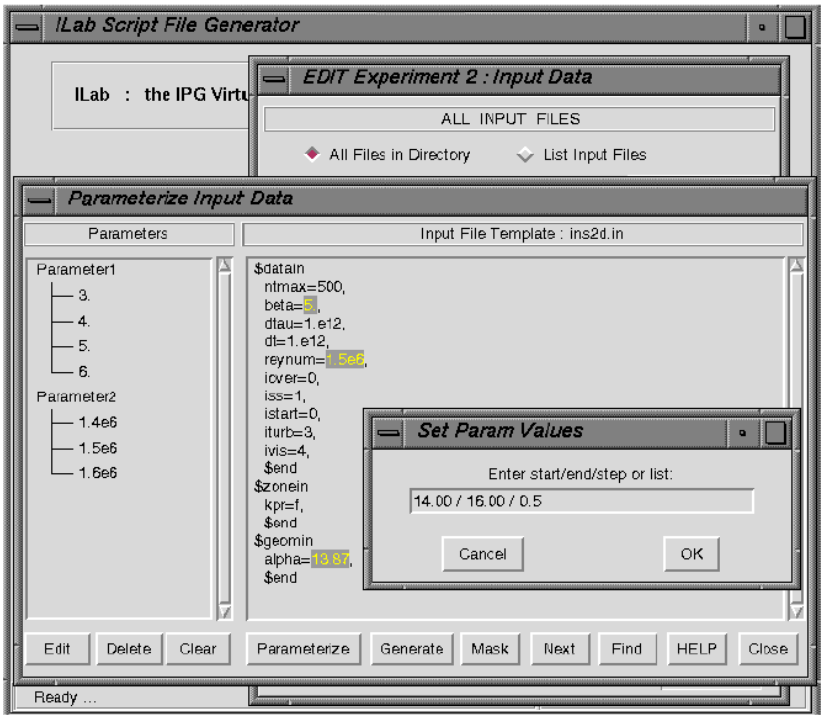

Figura 2.9: Tela de parametrização do ILab.

2.2.5 Resumo

Nesta seção vimos as principais funcionalidades disponíveis nas atuais ferramentas de computação em grade. Essas funcionalidades, conforme mencionado anteriormente, podem ser divididas em funcionalidades que são direcionadas para a programação das aplicações da grade ou para a interação

com os serviços da grade.

Para a programação das aplicações da grade vimos a utilização marcante de componentes gráficos tanto nos *Fluxos de Trabalho* (Seção 2.2.1) quanto nos ambientes de resolução de problemas (Seção 2.2.2). As abstrações de alto nível oferecidas pelos ambiente gráficos e seus componentes, unidas à possibilidade de reutilização de software, facilitam a aproximação dos cientistas desenvolvedores de aplicações à tecnologia de computação em grade. Entretanto, ainda não há uma linguagem padronizada entre as ferramentas para a especificação dos componentes gráficos e grande parte das ferramentas são dependentes de um *middleware* de grade específico. Apesar disso, os significativos esforços que têm sido feitos nesta área nos fornecem indícios para acreditar que as ferramentas para programação de aplicações para a grade serão fortemente baseadas na utilização de componentes gráficos e prometem crescer bastante utilizando pesquisas da área de programação baseada em componentes da Engenharia de Software.

Para a interação com os serviços da grade, os Portais (Seção 2.2.3) apresentam-se como a principal ferramenta utilizada. O ambiente Web, familiar ao usuário, facilita o seu uso e disseminação além de ser muito prático, pois não necessita de instalação do lado do usuário. Os portais personalizáveis que usam a tecnologia de *portlets* permitem uma boa flexibilidade para utilização de serviços de diferentes sistemas de *middleware*. Entretanto, é importante salientar a adequação do uso dos Portais à situação na qual o usuário já dispõe da aplicação pronta para ser executada, pois estes não fornecem recursos para o desenvolvimento das aplicações. As ferramentas para aplicações paramétricas (Seção 2.2.4) também apresentam funcionalidades para interação com a grade. Estas funcionalidades são específicas, mas importantes por atenderem uma grande comunidade de usuários realizadores de *estudos paramétricos*. As funcionalidades dessas ferramentas já estão sendo incorporadas em portais [103]. Acreditamos que os Portais devam se firmar como as ferramentas responsáveis pela interação com os serviços da grade, quando se tratando de usuários não desenvolvedores. Para desenvolvedores, acreditamos na utilização de ambientes integrados de desenvolvimento (IDEs) nos moldes do proposto neste trabalho.

No próximo Capítulo, analisamos os principais projetos que visam tornar aplicações e ferramentas menos dependentes e acopladas aos sistemas de *middleware*. Nesse mesmo capítulo apresentamos os detalhes da alternativa escolhida para ser utilizada no nosso trabalho.

Capítulo 3

Independência de Plataforma em Grades Computacionais

Virtualização é o propósito predominante de toda grade computacional [106]. Assim como sistemas operacionais virtualizam um único computador, grades computacionais virtualizam os recursos computacionais disponíveis em um conjunto, possivelmente muito grande, de computadores para oferecer um único e poderoso sistema ao invés de uma coleção de nós individuais. A arquitetura da grade definida por Foster et al. [56] (Figura 2.1) apresenta diferentes camadas que representam diferentes níveis de abstração. Cada camada disponibiliza sua interface e em cada nível o uso dos recursos torna-se mais simples e uniforme. Entretanto, a obtenção desta simplicidade pode provocar uma perda de parte do controle sobre esses recursos, o que pode causar problemas com aspectos não-funcionais, como desempenho e segurança.

As ferramentas de grade mostradas no Capítulo 2 trazem abstrações de mais alto nível do que as oferecidas pelos sistemas de *middleware*. No referido capítulo comentamos a diversidade de ferramentas existentes e seus diferentes propósitos. As ferramentas apresentadas se concentram em funcionalidades para interação com o usuário. Outras, são utilizadas como modelo de programação. Existem ainda as que tem como objetivo tão somente oferecer abstrações para outras ferramentas de mais alto nível. Todas essas diferentes alternativas de programação para a grade dificultam a escolha da ferramenta correta por parte dos desenvolvedores de aplicações e de ferramentas de alto nível.

Uma discussão feita por Kielmann [106] sobre os diferentes tipos de aplicações e ferramentas de grade mostra que não existe uma única ferramenta que atenda a todas as diferentes necessidades de programação para a grade. O estudo conclui que problemas diferentes necessitam de abordagens diferentes para atender aos aspectos não-funcionais da programação, os quais de fato determinam

a escolha das ferramentas apropriadas. No contexto da computação em grade, que apresenta diferentes propriedades não-funcionais adicionais como tolerância a falhas, segurança e independência de plataforma, a escolha da ferramenta a ser utilizada torna-se ainda mais difícil. A escolha deve ainda considerar o equilíbrio de fatores antagônicos como simplicidade (abstrações) e desempenho.

Um requisito não-funcional de especial interesse para o nosso trabalho é a independência de plataforma. Essa é uma propriedade importante para aplicações e ferramentas de alto nível por possibilitar que o código desses programas sejam independentes dos detalhes da plataforma de grade específica [107]. O atendimento dessa propriedade permite que aplicações e ferramentas de alto nível possam ser utilizadas integradas a diferentes sistemas de *middleware* de grade sem que haja a necessidade de reescrever o software. Além disso, alternativas para obtenção dessa portabilidade podem trazer um desacoplamento que possibilite que estes softwares não sofram os impactos das freqüentes mudanças nas interfaces dos sistemas de *middleware*.

Apesar dos benefícios oferecidos pela independência de plataforma, a maioria das ferramentas disponíveis não apresenta esta característica. As principais ferramentas direcionadas para a construção de aplicações, como ambientes para construção de fluxos de trabalho, aplicações paramétricas e resolução de problemas apresentados no Capítulo 2 são normalmente dependentes de um ambiente de execução. Algumas exceções como os projetos SEGL e Triana utilizam algum mecanismo para obter independência do *middleware* de grade. Os portais também oferecem independência de plataforma através do uso da tecnologia de *portlets*, mas são limitados à interação com os serviços da grade.

Neste capítulo, discutimos onde se encaixam e quais são as principais alternativas disponíveis para obtenção da independência de *middleware*. Em seguida, detalhamos a ferramenta escolhida para ser utilizada em nosso projeto.

3.1 Independência de Plataforma para Ferramentas e Aplicações

A necessidade de independência do *middleware* reflete-se diretamente nas ferramentas de baixo nível que precisam oferecer APIs que abstraiam o *middleware* de grade utilizado. A implementação dessas APIs deve mapear recursos em nível de aplicação, a seus correspondentes físicos na grade. O acesso a essas APIs é feito de acordo com o tipo de "cliente" da grade. Para a grade podemos considerar os seguintes principais clientes: (1) ferramentas de interação com o usuário, (2) aplicações

não cientes da grade (*Grid-unaware applications*) e (3) aplicações cientes da grade (*Grid-aware applications*).

O primeiro grupo já foi descrito e exemplificado no Capítulo 2. As aplicações não cientes da grade são aplicações as quais seus desenvolvedores não precisam se preocupar com detalhes da infraestrutura de grade, a idéia é executar suas aplicações com o mínimo esforço possível. Para os desenvolvedores desse tipo de aplicação, normalmente cientistas desenvolvedores, as abstrações devem ser de mais alto nível, ao custo da perda de controle dos serviços da grade já discutidos no início deste capítulo. As aplicações cientes da grade são aplicações que precisam acessar explicitamente os serviços da grade para explorar todo o seu potencial. Em virtude da proximidade com o ambiente de grade, o desenvolvimento dessas aplicações necessita de um grande esforço por parte de seus desenvolvedores [106], normalmente especialistas em computação em grade.

A Figura 3.1 mostra uma hierarquia de virtualização das abstrações oferecidas para esses três clientes da grade. A base da figura mostra os recursos e serviços da grade. Essa camada representa a API dos diferentes sistemas de *middleware* de grade como Globus ou Condor. Acima dessa camada, está uma camada opcional na qual estão as ferramentas de baixo nível que fornecem funcionalidades adicionais aos serviços dos sistemas de *middleware*, é aqui que estão localizadas as principais APIs que fornecem independência de plataforma para os clientes da grade e que serão estudadas nesse capítulo. Acima desta, do lado direito, estão as ferramentas de interação com usuário e as aplicações cientes da grade, do lado esquerdo estão as aplicações não cientes da grade com uma camada intermediária formada pelas ferramentas de modelos de programação, já apresentados no Capítulo 2 (Figura 2.2).

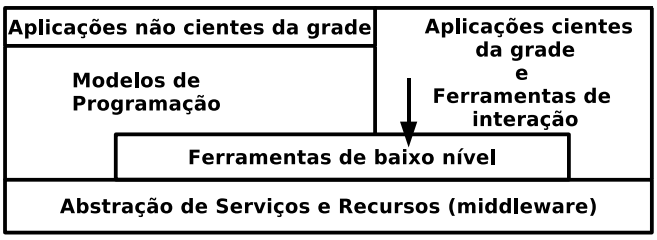

Figura 3.1: Hierarquia de virtualização das ferramentas da grade.

As aplicações não cientes da grade, conforme ilustrado na Figura 3.1, não devem fazer acesso direto às APIs dos sistemas de *middleware*, nem às APIs das ferramentas de baixo nível. Portanto,

a sua independência de plataforma depende exclusivamente do modelo de programação utilizado. Estes, por sua vez, podem utilizar diretamente as APIs fornecidas pelos sistemas de *middleware* ou uma camada adicional de baixo nível para abstraí-los. Os projetos ProActive [20], Ibis [130], Satin [128], Higher-order components [12] e Superscalar [19] são exemplos de ferramentas de modelos de programação que oferecem mecanismos para obter independência de plataforma.

As ferramentas de interação com o usuário e as aplicações cientes da grade, podem obter a independência de plataforma através da utilização direta de ferramentas de baixo nível. Conforme mencionado no início deste capítulo, a grande maioria das ferramentas atuais acessa as APIs dos sistemas de *middleware* diretamente. Como o principal objetivo deste trabalho é a construção de uma ferramenta de interação com o usuário independente de *middleware* de grade, nosso estudo se concentra nas alternativas disponíveis que utilizam as ferramentas de baixo nível, enfatizadas pela seta na Figura 3.1.

3.2 Principais Projetos

Os principais projetos com foco na abstração do *middleware* de grade foram criados com o propósito comum de oferecer aos desenvolvedores abstrações de programação de mais alto nível no intuito de simplificar o uso da grade. Esses projetos estão situados na camada denominada "ferramentas de baixo nível" da hierarquia de virtualização apresentada na Figura 3.1.

Java Cog Kit (CoG) [110] foi um dos projetos precursores tendo se originado no projeto Globus, no qual iniciou como um arcabouço de programação específico para este *middleware*. Desde seu início, o projeto vem sendo utilizado com sucesso por diversas ferramentas para acessar os serviços de Globus através de sua API. A partir de 2005, o projeto reconheceu a necessidade de desacoplar as aplicações da plataforma de grade e apresentou um modelo de abstração para diferentes sistemas de *middleware* [13]. Esse modelo baseia-se em diferentes abstrações oferecidas para as aplicações tais como *Task*, *Task Graph*, *Resource-Execution Pattern*, *Broker-Execution Pattern* e *Handler-Execution*. O modelo permite a utilização de diferentes sistemas de *middleware* através de um mecanismo extensível no qual componentes *handlers* são os únicos do modelo que possuem implementação dependente do *middleware*. A versão 4 de Java Cog Kit inclui a implementação de um protótipo das abstrações incluindo *handlers* para os sistemas Globus (versões 2.4, 3.0.2 e 3.2.1), para implementações do protocolo de execução de comando remotos *secure shell* (SSH) e para o *middleware* de grade Unicore. A adaptação dos *handlers* de Globus para a sua versão 4 está em fase de desenvolvimento.

Grid Application Toolkit (GAT) [10] fornece uma interface de programação unificada e de alto nível, abstraindo a complexidade da grade. Disponibiliza um mecanismo baseado no padrão *Adaptor* [66] que possibilita a extensão da API para diferentes sistemas de *middleware*. O projeto é descrito em detalhes na próxima seção.

Os dois projetos anteriores oferecem APIs de propósito geral. Reality Grid [138] é um projeto direcionado à realização de *Computational Steering* em aplicações científicas. No contexto de computação em grade, *Computational Steering* é a atividade de controle da evolução do processamento baseado na análise em tempo real e na visualização do estado da simulação. Este tipo de funcionalidade é de especial interesse de muitas comunidades científicas. O uso de *Computational Steering* por aplicações científicas tem apresentado uma necessidade de uma API de alto nível para simplificar o trabalho do cientista desenvolvedor. Reality Grid fornece tal API que, entre outras coisas, permite que o usuário interfira na seqüência do processamento de acordo com análises de resultados intermediários da execução.

O surgimento de diferentes APIs motivou a criação da *Simple API for Grid Applications* (SAGA) [79]. SAGA é uma iniciativa de padronização do Global Grid Forum (GGF) que tem o intuito de oferecer uma API unificada que integre as necessidades mais comuns de programação das aplicações para a grade. Para atingir seu objetivo, a concepção da SAGA foi feita de forma *top-down*, partindo das necessidades das aplicações. Um levantamento de casos de uso foi realizado junto à comunidade científica, incluindo diversos grupos do GGF tais como DRMAA-WG [46], GridRPC-WG [88], GridCPR-WG [83] e OGSA-WG [133]. Além disso, SAGA conta com a participação de diferentes projetos de pesquisa e iniciativas comerciais, incluindo os três projetos descritos anteriormente, cujos pesquisadores integram o grupo da SAGA fornecendo contribuições provenientes da experiência adquirida em seus projetos. A arquitetura da SAGA, por exemplo, é muito similar à do GAT.

SAGA tem o objetivo de ser uma API simples, portanto não tem a pretensão de abranger todas as potencialidades oferecidas pelo complexo ambiente da grade. Assim, o projeto da SAGA seguiu a regra do 80:20: "Projete uma API com 20% de esforço e que sirva para 80% dos casos de uso das aplicações". Então, SAGA não pretende atender todos os casos de uso, mas apenas aqueles que abrangem a maior parte das aplicações.

SAGA é complementar e não concorrente de outras iniciativas de padronização. Com relação à OGSA [55], as seguintes diferenças podem ser mencionadas: atuação em níveis diferentes, OGSA

define interfaces em nível de serviços e de *middleware*, enquanto SAGA define interfaces em nível de aplicação; OGSA é uma arquitetura e SAGA é uma API; as implementações de SAGA integram-se a serviços compatíveis com OGSA; OGSA é para desenvolvedores de *middleware*, SAGA é para desenvolvedores de aplicação.

A semântica da API SAGA está definida por uma especificação independente de linguagem [80], publicada em setembro de 2006. Esse documento serve como referência para a implementação da SAGA em diferentes linguagens. A implementação em C++ [104] encontra-se em fase inicial de desenvolvimento sendo conduzida pelas universidades de Vrije, na Holanda e Louisiana, nos Estados Unidos. O código fonte está disponível para *download* no website deste projeto[1]. A implementação em Java está em desenvolvimento sob a responsabilidade do Open Middleware Infraestructure Institute (OMII) [2], mas encontra-se ainda muito incipiente.

As funcionalidades inicialmente oferecidas pela SAGA incluem o gerenciamento de segurança, o gerenciamento de dados, permitindo o acesso remoto a arquivos, o gerenciamento de processos, incluindo submissão e controle e um limitado suporte à comunicação entre processos.

Apesar das implementações da SAGA ainda estarem em estágio embrionário, a comunidade científica tem sinalizado uma ampla adoção deste projeto. O projeto GRID Superscalar [148], por exemplo, já mostrou a viabilidade de utilização da API SAGA na construção de um modelo de programação para aplicações não cientes da grade. O estudo conclui que apesar do propósito de simplicidade do SAGA, este já oferece as funcionalidades necessárias para o desenvolvimento de um projeto nos moldes do seu. GRID Superscalar utilizou SAGA na sua arquitetura para obter independência de plataforma de grade.

SAGA tem potencial para ser a melhor API dentre as aqui apresentadas. Primeiro, por incorporar as melhores práticas dos principais projetos relacionados (descritos acima). Segundo, pela padronização, os grupos *Simple API for Grid Applications Core Working Group* (SAGA-CORE-WG)[3] e o *Simple API for Grid Apps Research Group* (SAGA-RG)[4] do GGF têm participação ativa de pesquisadores dos diferentes projetos acima mencionados, o que sinaliza uma boa adoção do padrão. Entretanto, no início do desenvolvimento deste projeto nem mesmo a especificação de SAGA es-

[1] http://saga.cct.lsu.edu/
[2] http://www.omii.ac.uk/news/newsdetail.jsp?id=49
[3] http://www.ogf.org/gf/group_info/view.php?group=saga-core-wg
[4] http://www.ogf.org/gf/group_info/view.php?group=saga-rg

tava disponível. Apesar disso, direcionamos a escolha da API a ser utilizada no nosso trabalho com a preocupação de obter um caminho curto de migração para o SAGA. Essa preocupação nos levou à escolha do GAT principalmente pela semelhança da arquitetura. Segundo informações dos pesquisadores participantes da implementação em Java do GAT, o projeto ainda vai ser mantido por pelo menos dois anos e será fornecida uma migração para o SAGA. A seguir apresentamos GAT com mais detalhes.

3.3 Grid Application Toolkit (GAT)

O Grid Application Toolkit (GAT) [10] foi a ferramenta escolhida para ser utilizada no nosso projeto. O GAT foi projetado e implementado pelo projeto europeu GridLab [9]. GAT é uma interface de programação unificada, simples e de alto nível para a infra-estrutura de grade. O projeto e a implementação dessa API foi guiado pelas necessidades de aplicações reais, incluindo um conjunto de requisitos comuns a uma gama de aplicações de diferentes domínios, abrangendo centenas de cenários de usuários. O principal objetivo do GAT é desacoplar a aplicação do *middleware* de grade e de seus serviços. Essa idéia é similar à de MPI [51], mas em um nível mais alto de abstração (grade).

A construção desta camada foi feita baseado-se nas experiências do *testbed* pan-Europeu [11]. Desta forma, o projeto é fortemente baseado nas necessidades das aplicações, identificadas através do envolvimento com grupos de aplicações reais. Isto qualificou o projeto para o desenvolvimento de uma camada que refletisse diretamente as necessidades reais dos usuários e desenvolvedores de aplicações para a grade.

GAT foi projetado para satisfazer os seguintes requisitos não-funcionais:

- facilidade de uso;
- suporte para diferentes linguagens de programação;
- suporte para diferentes sistemas de *middleware* de grade;
- mecanismos de portabilidade para aplicações escritas utilizando a API;
- orientado para aplicações cientes da grade dinâmicas e adaptativas.

3.3.1 Arquitetura

A arquitetura do GAT consiste de três partes (Figura 3.2): a API GAT, o *GAT engine* e os diferentes *GAT adaptors*. A API GAT define uma interface simples, estável e independente de *middleware* de grade. É a única parte do GAT exposta para as aplicações. O motor GAT (*GAT engine*) tem a responsabilidade de encaminhar para os adaptadores (*GAT adaptors*) todas as chamadas feitas à API, que então fornecem a implementação para a funcionalidade requerida.

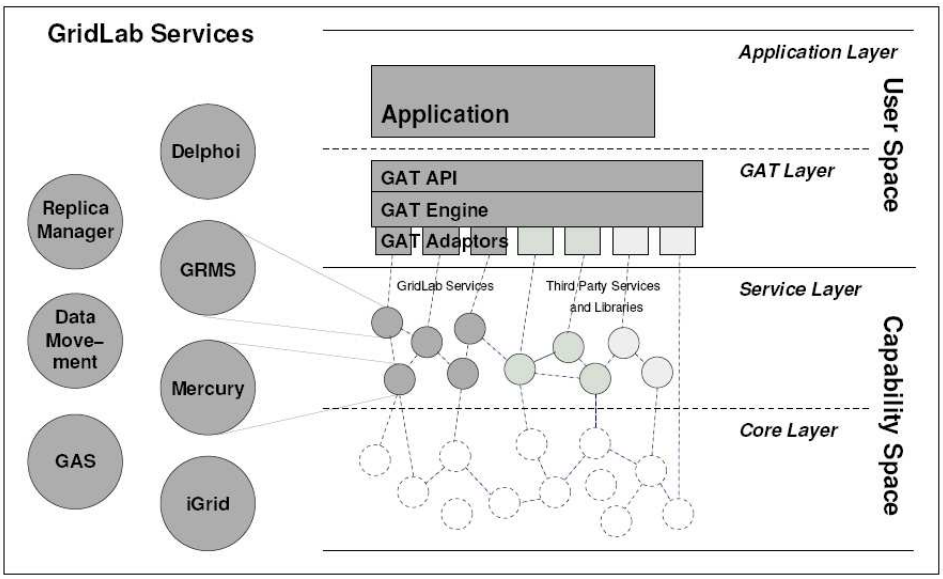

Figura 3.2: Arquitetura do GAT.

O motor GAT é uma biblioteca de tempo de execução que expõe uma API para as aplicações e outra API para a conexão com os adaptadores. Essa abordagem permite o desacoplamento da aplicação do *middleware* de grade e a conexão dessas a qualquer serviço de *middleware* que dê suporte à funcionalidade apropriada. O motor GAT foi projetado para ser uma camada extremamente fina (pequena quantidade da lógica implementada), fornecendo uma forma eficiente de alternar entre diferentes implementações de uma determinada funcionalidade da API GAT. Toda lógica das funcionalidades e toda interação com os serviços da grade são implementadas nos adaptadores.

Os adaptadores são elementos de software modulares que implementam funcionalidades específicas, dentre as definidas pela GAT API. Esses elementos seguem o padrão de projeto *Adaptor* [66], con-

vertendo a interface oferecida pelos sistemas de *middleware* à interface esperada pelo *engine*.

A interface entre o motor GAT e os adaptadores é chamada *Capability Provider Interface* (CPI). Essa interface espelha a própria GAT API. A partir das chamadas à GAT API, o motor GAT seleciona dinamicamente da lista de adaptadores o que fornece a funcionalidade pedida e encaminha a chamada ao adaptador escolhido. A técnica atual utilizada nessa seleção sob demanda baseia-se na seleção do primeiro adaptador que atende a funcionalidade pedida à API e que atende também uma série de preferências do usuário (`GATPreferences`) que define requisitos especificados no formato de pares atributo-valor. Usando essas preferências, o usuário pode especificar quais adaptadores específicos ele deseja usar. A utilização do mecanismo de seleção sob demanda do GAT para implementar técnicas de seleção dinâmica de adaptadores em tempo de execução é considerada pelos pesquisadores do grupo um tema interessante para trabalhos futuros. Essas técnicas podem considerar aspectos como custo do serviço, desempenho e confiabilidade para fazer a seleção do adaptador mais adequado à situação.

Conforme mostrado na Figura 3.2, uma aplicação escrita usando GAT consiste de quatro camadas de software descritas a seguir:

- *Application layer*: contém todo o código específico da aplicação. Esse código utiliza as funções da GAT API.
- *GAT layer*: representada pelo motor GAT, apresenta a GAT API para a camada de aplicação e traduz as chamadas feitas à API a chamadas ao adaptador adequado para a funcionalidade pedida. Os adaptadores conectam as chamadas feitas à GAT API à implementação da funcionalidade solicitada para serviço do *middleware* de grade específico.
- *Service layer*: representa as funcionalidades fornecidas pelo ambiente de grade que está sendo utilizado (GridLab, Globus, InteGrade ou outro *middleware*).
- *Core layer*: representa os recursos disponíveis na grade.

Tanto a camada de aplicação (*Application layer*) quanto a camada do GAT (*GAT layer*) executam no ambiente do usuário (*user space*). Os adaptadores formam a interface entre a ambiente do usuário e o ambiente de funcionalidades (*capability space*), formado pela camada de serviço (*Service layer*) e a camada de recursos (*Core layer*).

3.3.2 A GAT Application Programming Interface

A API GAT está dividida em vários subsistemas que atendem a diferentes aspectos da grade. Esses aspectos incluem gerenciamento e acesso a dados, gerenciamento de recursos, monitoramento e manipulação de eventos e gerenciamento de informações. Esses subsistemas são acompanhados de mais dois, um utilitário e outro base. Este último fornece a interface para o motor GAT.

A especificação da GAT API é orientada a objetos [43]. Apesar disso, sua primeira implementação foi feita em C, o que comprova a viabilidade do mapeamento da especificação para linguagens não orientadas a objetos, conforme inicialmente planejado. *Wrappers* C++ e Python para a implementação C estão disponíveis no website do projeto[5]. Uma implementação nativa em Java, iniciada posteriormente, já está disponível[6] e foi a utilizada no nosso trabalho.

Cada subsistema do GAT possui um conjunto de classes (definido na especificação) que são as abstrações a serem utilizadas pelas aplicações. O subsistema de gerenciamento e acesso a dados, por exemplo, apresenta a classe `GATFile`, que representa um arquivo físico independente de sua localização. A seguir listamos as principais classes do subsistema de gerenciamento de recursos, o mais extenso da API.

- `GATJob`: representa uma aplicação da grade.
- `GATResourceBroker`: responsável por toda interação com recursos da grade.
- `GATSoftwareDescription`: descreve um software a ser submetido para a grade.
- `GATHardwareResourceDescription`: descreve um recurso de hardware da grade utilizado como requisito para a submissão de uma aplicação.
- `GATJobDescription`: é a descrição de uma aplicação a ser submetida para a grade.

Um exemplo de utilização da GAT API em uma aplicação pode ser visto na listagem 3.1. Neste exemplo uma aplicação é submetida para execução em uma grade, utilizando algumas classes do subsistema de gerenciamento de recursos e uma do subsistema de gerenciamento e acesso a dados (`GATFile`).

[5] http://www.gridlab.org/WorkPackages/wp-1/gatreleases.html
[6] https://gforge.cs.vu.nl/projects/javagat

```
GATContext context = new GATContext();

SoftwareDescription sd = new SoftwareDescription();
sd.setLocation(``file://///bin/hostname'');
GATFile stdout = GAT.createFile(context, ``hostname.txt'');
sd.setStdout(stdout);

ResourceDescription rd = new HardwareResourceDescription();
rd.addResourceAttribute(``machine.type'', ``i686'');

JobDescription jd = new JobDescription(sd, rd);
ResourceBroker broker = GAT.createResourceBroker(context);
Job job = broker.submitJob(jd);

while (job.getState() != Job.STOPPED && job.getState() != Job.SUBMISSION_ERROR) {
        Thread.sleep(1000);
}
```

Listagem 3.1: Exemplo de uso da API GAT utilizando sua implementação Java

A API GAT vem sendo utilizada por vários projetos para o desenvolvimento tanto de ferramentas como Triana, Cactus [78], GridSphere e Ibis, quanto por aplicações cientes da grade [27, 129]. A utilização do GAT nesses importantes projetos comprova a viabilidade de sua utilização.

3.3.3 Estendendo GAT

O conjunto de adaptadores disponíveis para o GAT inclui os sistemas de *middleware* GridLab, Globus, Unicore [157], Sun Grid Engine (SGE) [151], e outros sistemas como GridFTP [85], implementações do SSH e ProActive (em desenvolvimento). Uma lista completa e dinâmica dos adaptadores disponíveis para o GAT pode ser encontrada em [67].

Para a integração de serviços de outros sistemas de *middleware* é necessário entender a integração entre os subsistemas da API expostos para as aplicações e os adaptadores que implementam toda a lógica de interação com a grade. Cada subsistema possui várias *Capability Provider Interfaces* (CPIs), que são compostas por classes e interfaces. Nem todos os objetos de um subsistema são CPIs. Uma CPI pode ter vários adaptadores que a implementam para diferentes sistemas de *middleware*.

A Figura 3.3 mostra a integração do subsistema de gerenciamento e acesso a dados com uma de

suas CPIs e os adaptadores que a implementam. Na figura, o objeto `GATFile` localizado no espaço de usuário (*user space*) está relacionado à sua CPI, chamada `FileCPI`, que possui dois adaptadores que a implementam. Assim, aplicações que utilizam `GATFile` podem executar uma operação de cópia de arquivo, por exemplo, usando o adaptador para o sistema Local ou o adaptador de uma implementação do SSH sem qualquer modificação no seu código.

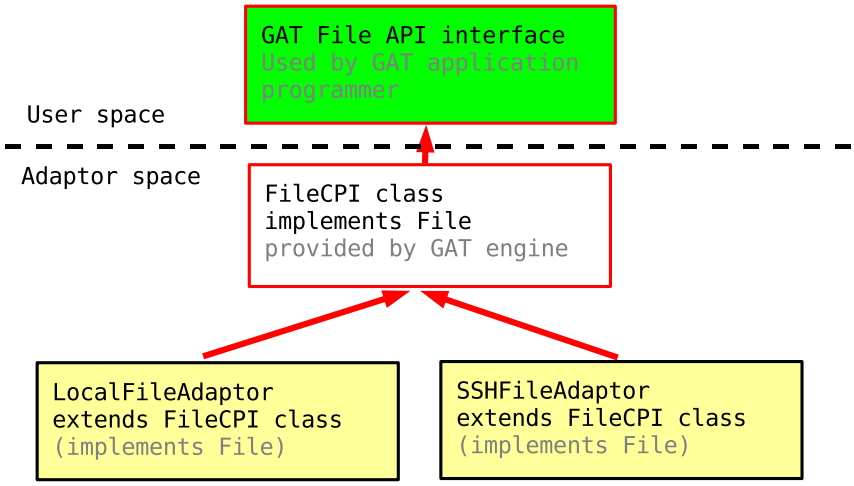

Figura 3.3: Integração do subsistema de gerenciamento e acesso a dados e seus adaptadores.

Conforme ilustrado no exemplo acima, a integração de um *middleware* ao GAT está relacionada à implementação de adaptadores para as CPIs dos subsistemas que esse *middleware* visa atender. Portanto, para fazer a integração desse *middleware* com GAT é necessário primeiro definir quais subsistemas do GAT vão ser atendidos. Depois, deverão ser desenvolvidos os adaptadores que implementam as CPIs desses subsistemas.

3.4 Resumo

Os sistemas aqui apresentados têm como objetivo comum a abstração da plataforma de *middleware*. GAT e SAGA destacam-se por estarem alinhados a iniciativas de padronização. Optamos pelo GAT para utilização em nosso projeto, por este ter a implementação mais madura no momento. A integração entre GAT e o *middleware* utilizado como estudo de caso no nosso projeto, o Inte-

Grade, será apresentada em detalhes no próximo capítulo. Além disso, o capítulo apresenta a nossa ferramenta, que utiliza o GAT como alternativa para independência de plataforma.

Capítulo 4

InGriDE: um IDE para grades baseado em GAT

Conforme visto nos capítulos anteriores, várias ferramentas estão disponíveis para os desenvolvedores de aplicações para a grade. Vimos também que, de acordo com o problema que se pretende solucionar, existem ferramentas adequadas para ajudar na resolução de aspectos do mesmo. Nesse cenário o desenvolvedor de aplicações se depara com um conjunto de diferentes ferramentas e ambientes com os quais tem que estar familiarizado para solucionar seus problemas. Essa situação é extremamente desencorajadora para os cientistas desenvolvedores utilizarem a grade em função da complexidade envolvida em lidar com diferentes ferramentas, além da complexidade inerente ao ambiente de grade.

Neste capítulo, apresentamos a ferramenta desenvolvida no contexto desse trabalho, que visa a minimizar a dificuldade de desenvolvimento de aplicações para a grade. O InGriDE (*Integrated Grid Development Environment*) é um ambiente integrado de desenvolvimento (IDE) para computação em grade que tem como objetivo oferecer, em um mesmo ambiente, as ferramentas necessárias no processo de *desenvolvimento de aplicações* para a grade. O InGriDE é uma ferramenta extensível, desenvolvida baseada na plataforma Eclipse (Seção 4.3.2) e que possibilita a interação com diferentes sistemas de *middleware* por meio da interface de programação Grid Application Toolkit (GAT) (Seção 3.3). Utilizamos, em nosso estudo de caso, o *middleware* InteGrade, desenvolvido no nosso grupo de pesquisa. Nas seções que se seguem são apresentados os requisitos funcionais e não funcionais considerados no projeto do InGriDE, as tecnologias utilizadas na sua construção, sua arquitetura e implementação. Em seguida, fazemos algumas considerações das lições aprendidas durante o desenvolvimento do projeto.

4.1 Requisitos Funcionais

O conjunto de requisitos funcionais para a construção de um IDE para computação em grade pode ser tão vasto quanto a quantidade de ferramentas disponíveis atualmente para a grade. Nosso levantamento de requisitos identificou funcionalidades para as diversas fases do ciclo de vida de uma aplicação de grade [6] incluindo desenvolvimento, implantação, testes, depuração e execução. Diante da grande quantidade de requisitos identificados, consideramos para nossa prova de conceito um subconjunto proveniente de um cenário típico de desenvolvimento de aplicação para a grade. Esse cenário, também presente no nosso grupo de pesquisa, está ilustrado na Figura 4.1 e é descrito a seguir:

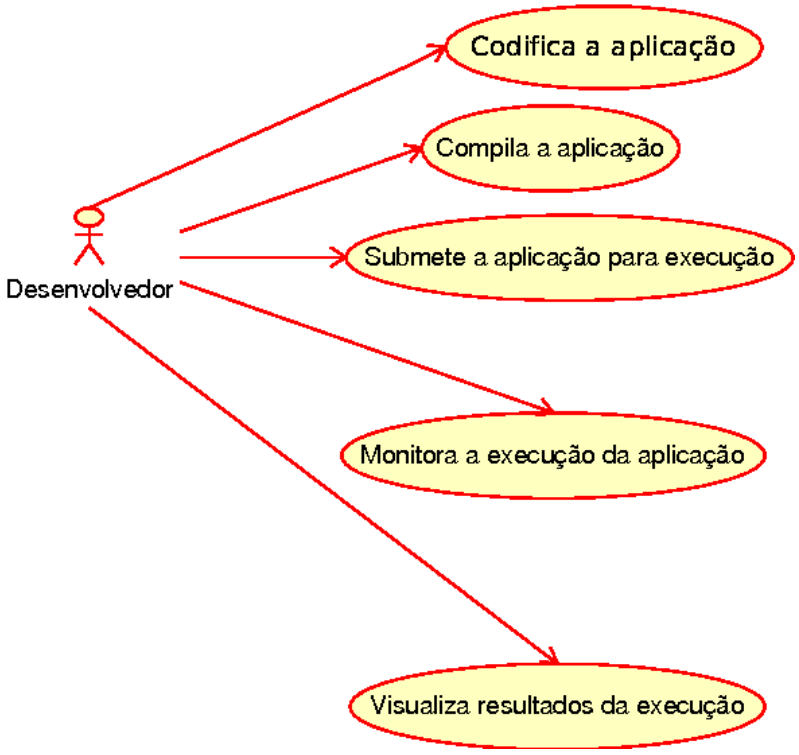

Figura 4.1: Diagrama de casos de uso dos requisitos considerados.

- O usuário escreve o código fonte da sua aplicação (utilizando uma biblioteca específica);
- O usuário compila sua aplicação;

- O usuário submete sua aplicação para execução na grade;
- O usuário monitora a execução da aplicação;
- O usuário obtém e visualiza os resultados da execução.

O atendimento destes requisitos possibilita a utilização do InGriDE por um grupo considerável de desenvolvedores que têm como necessidade comum a realização das atividades descritas acima.

4.2 Requisitos Não Funcionais

Os requisitos não funcionais considerados no projeto do InGriDE foram três: facilidade de uso, extensibilidade e independência de *middleware*.

A *facilidade de uso* considerada para o projeto do InGriDE se resume a dois aspectos. O primeiro aspecto está relacionado à disponibilização de diferentes ferramentas integradas e acessíveis em um mesmo ambiente. Conforme descrito no início desse capítulo, a diversidade de ferramentas não integradas é um fator bastante negativo para a disseminação da computação em grade junto aos usuários. Este problema já vem sendo resolvido com eficácia na Engenharia de Software desde a década de 80 por intermédio de ambientes integrados de desenvolvimento (IDE), indicando o caminho seguido pelo nosso projeto. O segundo aspecto é a interação homem-computador (IHC) que está relacionado à construção de interfaces com o usuário "amigáveis". Para esse aspecto, nosso trabalho baseou-se na usabilidade oferecida pelo arcabouço de criação de IDEs escolhido (Seção 4.3.2), excluindo do escopo deste trabalho uma avaliação da usabilidade do sistema.

A *extensibilidade* foi outro requisito considerado fundamental para a construção do nosso IDE. Seria impossível, mesmo para uma grande equipe, desenvolver e manter um ambiente que atendesse a todos os requisitos funcionais da comunidade de desenvolvedores da grade. Portanto, o desenvolvimento do InGriDE foi feito com essa preocupação, optando por tecnologias que possibilitassem a ampliação do conjunto de funcionalidades inicialmente fornecido.

O terceiro requisito considerado no projeto do InGriDE foi a *independência de middleware*. Para que o IDE tivesse um conjunto consistente de ferramentas, era importante que uma funcionalidade fosse disponibilizada através de uma única interface com o usuário. Entretanto, os desenvolvedores deveriam ter a liberdade de utilizar diferentes sistemas de *middleware*. Para que a interface com

o usuário se tornasse genérica, sem incluir características específicas de um *middleware*, concluímos que seria necessária uma camada de abstração para isso.

Os requisitos aqui apresentados implicaram em decisões de projeto que resultaram na escolha de tecnologias que os atendessem. Para facilitar o atendimento dos dois primeiros requisitos, *facilidade de uso* e *extensibilidade*, foi utilizada a plataforma Eclipse (Seção 4.3.2). Essa plataforma fornece um arcabouço para criação de IDEs que tem entre os seus objetivos principais questões de usabilidade e extensibilidade. A utilização do Eclipse como arcabouço permitiu que as funcionalidades básicas de uma IDE fossem reutilizadas, e que outras fossem estendidas para as necessidades específicas do InGriDE. O mecanismo de extensibilidade utilizado da plataforma Eclipse foi utilizado tanto para agregar as funcionalidades iniciais do nosso projeto, quanto para atender ao requisito de extensibidade do InGriDE para inclusão de novas funcionalidades.

A obtenção de *independência de middleware* foi alcançada através do desenho da arquitetura do InGriDE baseada no *Grid Application Toolkit* (GAT), já apresentada na Seção 3.3. A arquitetura utiliza GAT como uma interface de programação que abstrai a plataforma de *middleware* utilizada. Esse modelo faz com que InGriDE seja compatível com qualquer sistema (por exemplo, Globus, Unicore) que forneça os componentes adequados de integração com GAT. Para comprovar a eficácia do modelo adotado utilizamos como estudo de caso o *middleware* InteGrade (Seção 4.3.1), desenvolvido pelo Grupo de Sistemas Distribuídos do IME-USP [1], grupo de pesquisa no qual este trabalho está sendo desenvolvido. Para viabilizar essa solução desenvolvemos a integração do GAT com o InteGrade.

4.3 Tecnologias Utilizadas

Nesta seção apresentamos as tecnologias utilizadas na concepção do InGriDE.

4.3.1 InteGrade

O InteGrade [76] é uma infra-estrutura de *middleware* para grades onde um dos objetivos principais é a reutilização de recursos computacionais ociosos de computadores pessoais. O objetivo do InteGrade é permitir que o poder computacional ocioso de computadores comuns compartilhados seja utilizado em tarefas de computação de alto desempenho. A arquitetura permite que organizações

[1] http://gsd.ime.usp.br

possam se beneficiar do poder computacional do hardware já existente. Isso é obtido com a integração de estações de trabalho e recursos de laboratórios compartilhados, em uma grade computacional de intranet ou Internet.

Esse sistema de computação em grade vem sendo desenvolvido por cinco universidades brasileiras: IME-USP, DCT-UFMS, DI-PUC-Rio, UFMA e UFG. Diferentemente dos sistemas de computação em grade existentes, InteGrade está baseado em uma sofisticada tecnologia de interligação de objetos distribuídos (CORBA) [134], que é um padrão da indústria para sistemas de objetos distribuídos. O uso desta tecnologia traz ao InteGrade duas vantagens: a primeira é a interoperabilidade entre diferentes linguagens de programação, obtida através das interfaces IDL de CORBA; e a segunda é a reutilização de uma série de serviços como Serviço de Nomes, *Trading*, Transações e Persistência. A interoperabilidade permite que os componentes do InteGrade se comuniquem mesmo sendo escritos em diferentes linguagens de programação. O uso dos serviços de CORBA pelo InteGrade ajuda a diminuir a complexidade do sistema e o custo de manutenção.

Um importante requisito do InteGrade é que os usuários que decidam compartilhar suas máquinas na grade nunca devem perceber perda na qualidade de serviço oferecida por suas aplicações. Desta forma, o *middleware* deve garantir que as aplicações de terceiros, sendo executadas nos recursos cedidos, não atrapalhem o proprietário. Este requisito pode levar a um cenário no qual uma aplicação de terceiro que esteja rodando em uma máquina, tenha que parar sua execução para a disponibilização dos recursos para o uso de seu proprietário. Para tal cenário dinâmico, InteGrade provê um mecanismo de *checkpointing* [33] que garante o progresso da aplicação através do armazenamento de seus estados em dados momentos (*checkpoints*). Com isso, uma aplicação interrompida pode retomar a execução a partir de seu último *checkpoint*.

Em um ambiente dinâmico como esse, torna-se difícil o agendamento de execução das aplicações de forma a evitar situações como a descrita no parágrafo anterior. Para minimizar este problema, a arquitetura do InteGrade prevê um componente para coleta e análise de padrões de uso [26]. Esse componente prevê a coleta de longas séries de informações de uso dos recursos das máquinas da grade e, baseado nessas informações, são feitas análises para detecção de padrões de uso. Isso possibilita que se estime a probabilidade de um nó ocioso tornar-se ocupado. Essas informações sobre os recursos ajudam o escalonador de aplicações a melhorar a sua capacidade de decisão de onde alocar uma tarefa a ser executada. Entretanto, convém lembrar que esse mecanismo fornece apenas dicas, e de maneira alguma representa uma garantia de que o recurso estará ocupado ou ocioso.

Apesar de várias tipos de aplicações poderem se beneficiar de sistemas de computação em grade, são as aplicações paralelas que se beneficiam mais. Em virtude de sua natureza de utilizar vários recursos simultaneamente, esse tipo de aplicação pode tirar maior proveito do ambiente de grade. Normalmente essas aplicações são executadas em ambientes de recursos dedicados, mas o InteGrade possibilita que elas sejam executadas em vários recursos compartilhados. Conforme descrito anteriormente, a execução de aplicações nesse ambiente necessita de mecanismos de *checkpoint* para as situações nas quais os recursos tornam-se indisponíveis. Entretanto, no caso especial de aplicações paralelas, existe uma dificuldade maior em manter o estado das aplicações em virtude da possível troca de mensagens entre os vários nós da aplicação. Para solucionar este problema, o InteGrade usa o modelo de programação paralela BSP [158], que impõe sincronizações periódicas entre os nós da aplicação permitindo retomadas de estágios anteriores de execução em caso de falhas.

A arquitetura do InteGrade é estruturada em aglomerados organizados de uma forma hierárquica. Um aglomerado á definido como uma unidade autônoma dentro da grade, uma vez que o mesmo contém todos os componentes necessários para funcionar independentemente. A Figura 4.2 mostra os principais módulos de um aglomerado InteGrade. O Global Resource Manager (GRM) faz o gerenciamento de recursos no nível do aglomerado; o Local Resource Manager (LRM) executa em cada nó do aglomerado coletando informações sobre os recursos do nó; o Repositório de Aplicações (*Application Repository* - AR) armazena as aplicações prontas para serem executadas no InteGrade; a BSPlib é a biblioteca que implementa o modelo de programação BSP descrito no parágrafo anterior. Para formar uma grade composta por uma federação de aglomerados, GRMs de diferentes aglomerados comunicam-se para permitir o compartilhamento global de recursos locais. Um LRM se comunica apenas com módulos do seu próprio aglomerado.

Além dos componentes ilustrados na Figura 4.2, a arquitetura do InteGrade possui dois componentes ainda em desenvolvimento que têm como objetivo analisar o padrão de utilização das máquinas do aglomerado, chamados Local Usage Pattern Analyzer (LUPA) e Global Usage Pattern Analyzer (GUPA). O LUPA é executado juntamente com o LRM nos nós provedores de recursos, coletando dados da máquina e analisando seu padrão de uso utilizando técnicas de aprendizado de máquina [126] e clusterização [32]. Esta informação é disponibilizada ao GRM e, futuramente, permitirá um escalonamento mais eficiente das aplicações da grade.

Com relação à segurança, o InteGrade dispõe de um serviço capaz de transmitir todos os dados de forma criptografada e de autenticar usuários e aplicações [99].

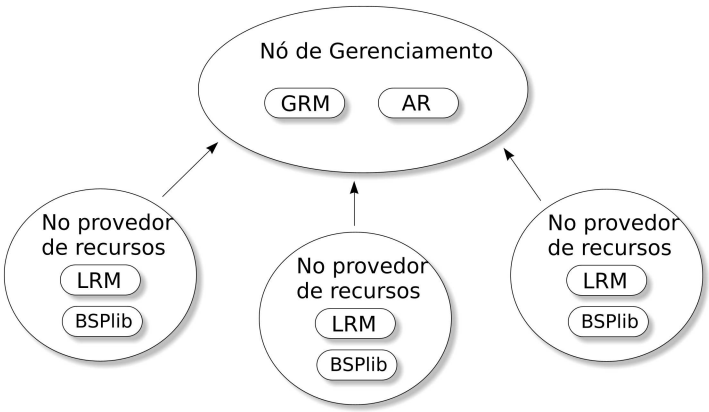

Figura 4.2: Arquitetura de um aglomerado InteGrade.

O InteGrade permite a execução de três tipos de aplicação:

- Aplicações Sequenciais: são aplicações convencionais, compostas por apenas um arquivo binário executável, e sua execução requer apenas uma máquina;
- Aplicações Paramétricas ou *Bag-of-Tasks*: aplicações onde um mesmo binário é executado várias vezes com diferentes conjuntos de parâmetros ou arquivos de entrada; normalmente, cada execução é realizada em um nó diferente e não há comunicação entre os nós;
- Aplicações BSP: aplicações paralelas SPMD (Single Program, Multiple Data), cujos nós comunicam-se. Tais aplicações fazem uso da biblioteca BSPlib do InteGrade.

InteGrade encontra-se em sua versão 0.3-RC1, seu código é baseado na licença LGPL e está disponível no website [2] do projeto.

Ferramentas Existentes

As ferramentas disponíveis no projeto InteGrade, dentro da taxonomia apresentada no Capítulo 2, estão presentes nas categorias *modelos de programação* com a BSPlib, e *ferramentas de interação com o usuário* com as ferramentas *Application Submission and Control Tool* (ASCTGui) e *Portal* para submissão de aplicações e o ClusterView para monitoramento. A hierarquia de virtualização (Figura

[2] http://www.integrade.org.br

3.1) utilizada por essas ferramentas é o acesso direto à API oferecida pelo *middleware* InteGrade, como na maioria das ferramentas disponíveis atualmente.

A utilização dessas ferramentas é feita de forma não integrada dificultando o trabalho do desenvolvedor. No cenário típico de desenvolvimento de uma aplicação para o InteGrade o desenvolvedor utiliza um editor comum para escrever suas aplicações (utilizando a BSPlib ou não), compila o código fonte utilizando ferramentas de linha de comando, submete o binário da aplicação para execução através do ASCTGui (ou do Portal) e obtém o resultado da execução nessa mesma ferramenta. O ClusterView pode ser utilizado para visualizar os recursos da grade e, caso seja necessário realizar tarefas nos recursos remotos, são utilizadas ferramentas de linha de comando como implementações do secure shell (SSH).

O ASCTGui (Figura 4.3) permite aos usuários do InteGrade a submissão de aplicações para a execução. O usuário pode especificar pré-requisitos como plataforma de hardware e software, requisitos de recursos como mínimo de memória, e preferências tais como a execução em uma CPU mais rápida. O usuário também pode monitorar o progresso da execução da aplicação. O ASCTGui apresenta ainda funcionalidades como o controle da execução das aplicações, representação do estado das aplicações com cores, visualização dos arquivos de saída, dentre outras. A utilização do ASCTGui é direcionada para a submissão de aplicações já desenvolvidas e registradas em um repositório de aplicações (Figura 4.3 - *Application Repository*). Portanto, nenhuma funcionalidade para o desenvolvimento da aplicações é fornecida pela ferramenta.

O Portal permite que usuários da grade realizem, utilizando um portal na Internet, as mesmas tarefas que podem ser realizadas com o ASCTGui. A vantagem do portal é que usuários podem acessar remotamente o InteGrade a partir de qualquer máquina conectada à Internet utilizando um navegador, sem a necessidade de instalação de um software cliente específico do InteGrade.

O ClusterView (Figura 4.4) fornece funcionalidades para o usuário fazer o monitoramento de um aglomerado InteGrade. É possível obter informações sobre a disponibilidade de recursos em qualquer nó do aglomerado.

A BSPlib [77] (Figura 4.2) é uma biblioteca que implementa o modelo de programação paralela BSP sobre o InteGrade. Essa biblioteca pode ser considerada do grupo de modelos de programação na taxonomia apresentada na Figura 2.2. Aplicações escritas baseadas na BSPlib têm a possibilidade de comunicação entre seus nós através de funções fornecidas pela biblioteca. Essas aplicações devem

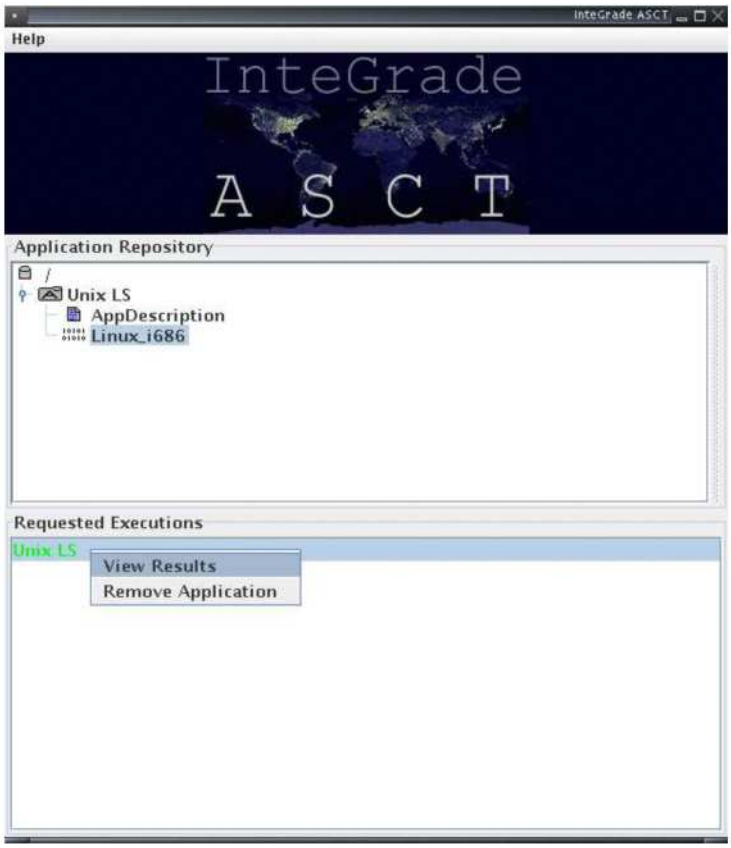

Figura 4.3: Ferramenta ASCT do projeto InteGrade.

ser compiladas e ligadas à biblioteca. Nenhuma ferramenta de interação com o usuário é fornecida para a construção de aplicações que usam a BSPlib.

4.3.2 Eclipse

O Eclipse [47, 65] é uma plataforma universal para integração de ferramentas, idealizada inicialmente como um arcabouço para a criação de ambientes integrados de desenvolvimento (IDEs). Esta plataforma fornece um mecanismo de extensibilidade que permite a integração de soluções de diferentes fornecedores no mesmo ambiente. O Eclipse é mantido por uma forte comunidade de 115 empresas que incluem nomes como Ericsson, HP, IBM e Intel.

A tela principal do Eclipse, chamada de *workbench*, é formada por diferentes componentes de

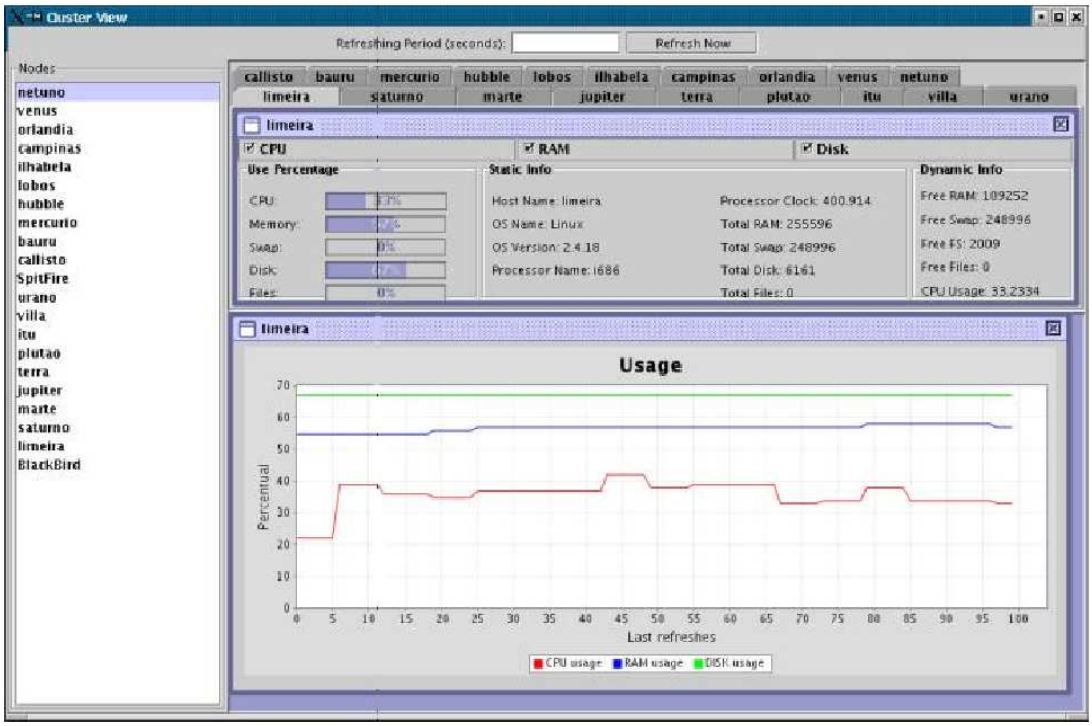

Figura 4.4: Ferramenta ClusterView do projeto InteGrade.

interação com usuário. Esses componentes incluem editores, visualizadores, perspectivas, barra de menu e barra de ferramentas. O conjunto de componentes visíveis é completamente configurável permitindo que o usuário personalize o seu espaço de trabalho. Um componente que ajuda nesse sentido é a perspectiva, que consiste de um conjunto de componentes visuais e uma forma de disponibilizá-los na *workbench*. As perspectivas fornecem uma interface limpa e simples de alternar entre diferentes atividades do processo de desenvolvimento. A maior parte da interação com o Eclipse é feita através dos visualizadores e editores. Um visualizador é um componente visual que pode ser utilizado para vários propósitos e são comumente usados para navegação e disponibilização de informações. Um editor é um tipo especial de visualizador usado para edição de recursos.

Para fornecer uma plataforma extensível, o Eclipse disponibiliza um arcabouço baseado em unidades plugáveis chamadas *plugins*, através da qual terceiros podem adicionar funcionalidades ao ambiente base. Este arcabouço permite a criação, integração e utilização de diferentes *plug-ins* facilitando a reutilização. O mecanismo de extensão funciona através de pontos de extensão

(*extension-points*) disponibilizados pelo arcabouço [65]. Os pontos de extensão fornecem uma extensibilidade controlada na medida em que definem em quais pontos e de que forma os *plugins* podem ser estendidos. O Eclipse é escrito na linguagem Java, é software livre, e traz ferramentas para a construção de *plugins*, além de vários exemplos. O Eclipse tornou-se uma ferramenta muito popular junto à comunidade de desenvolvedores e funciona em diversas plataformas tais como Linux, HP-UX, AIX, Solaris, QNX, Mac OS X e Windows.

O conjunto de pontos de extensão do Eclipse permite a extensão de funcionalidades para a interface com o usuário, manipulação de recursos, dentre outros. Alguns dos pontos de extensão mais relevantes são: editores, visualizadores, perspectivas, *wizards*, ações (botões), *natures* e *builders* (compilação) e *launchConfigurationTypes* (execução de aplicações).

4.4 Arquitetura

Nesta seção apresentamos a arquitetura do InGriDE iniciando com uma visão geral, seguindo com sua estrutura interna e finalizando com sua integração com o Grid Application Toolkit (GAT).

4.4.1 Visão Geral

A arquitetura completa da nossa solução é composta por diferentes componentes externos com os quais os módulos desenvolvidos interagem. A Figura 4.5 mostra essa arquitetura e ilustra a interação entre seus diferentes componentes. Podemos considerar a arquitetura dividida em duas partes: InGriDE, a parte relacionada ao IDE e os componentes externos relacionados (Eclipse e API GAT); e a integração com o *middleware* InteGrade, feita através dos adaptadores do GAT.

InGriDE é composto pelo conjunto de ferramentas desenvolvidas baseadas na plataforma Eclipse que acessam as funcionalidades da grade através da API GAT. Toda a interface com o usuário e alguns recursos próprios de IDEs foram desenvolvidos utilizando os pontos de extensão do Eclipse. As chamadas para funções da grade realizadas em InGriDE são feitas utilizando a API disponibilizada por GAT.

A parte que trata da integração com o *middleware* InteGrade foi feita baseada no mecanismo fornecido por GAT. Os adaptadores necessários para as funcionalidades oferecidas por InGriDE foram desenvolvidos de forma a adequar a interface oferecida pelo InteGrade à interface esperada por GAT. Para isso foi necessária a utilização de bibliotecas e serviços do InteGrade.

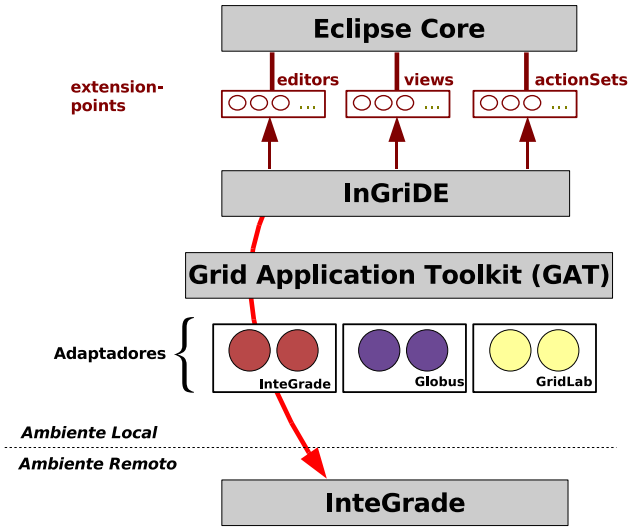

Figura 4.5: Visão geral da arquitetura.

A seguir apresentamos mais detalhes de cada uma das duas partes dessa arquitetura.

4.4.2 InGriDE

O InGriDE é responsável por fornecer a interface com o usuário construída utilizando os componentes da plataforma Eclipse, e também fazer a ligação entre essa interface e o GAT. Para fazer esse ligação, InGriDE precisa manter um modelo interno de abstração da grade para fornecer informações para a interface gráfica. Para atender essa necessidade, projetamos InGriDE baseado na arquitetura do projeto *Eclipse Parallel Tools Platform* [159], que oferece uma solução elegante para esse problema. Os principais componentes da arquitetura interna do InGriDE são mostrados na Figura 4.6.

Os quatro componente do InGriDE ilustrados na figura são descritos a seguir:

- *Grid Runtime Model*: mantém um modelo local que representa o estado atual dos componentes da grade tais como recursos e tarefas em execução;
- *Grid Runtime Controller*: controla a interação entre os componentes do InGriDE e o ambiente externo de execução através do GAT. Este é o único ponto de acesso do InGriDE ao ambiente externo;

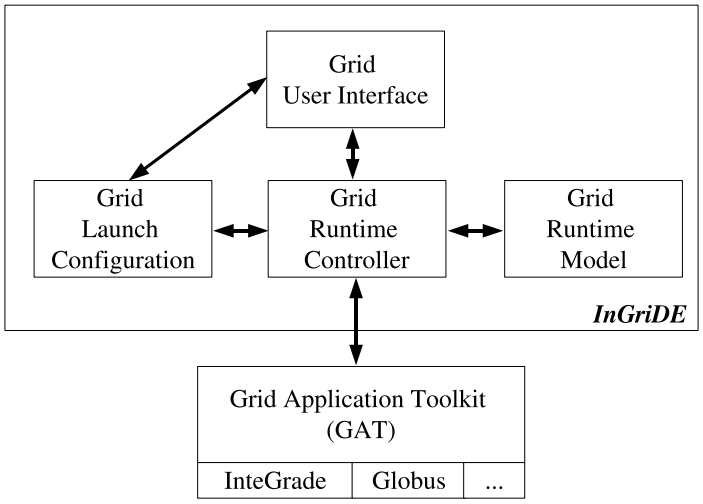

Figura 4.6: Arquitetura interna do InGriDE.

- *Grid User Interface*: contém os componentes da interface gráfica com o usuário, desenvolvidos a partir dos pontos de extensão da plataforma Eclipse;
- *Grid Launch Configuration*: usa o arcabouço de lançamento de aplicações do Eclipse para gerenciar a execução das aplicações na grade. É usado para a especificação dos requisitos de execução da aplicação. Este componente passa essas informações ao *Runtime Controller* que as utiliza na submissão feita através do GAT.

4.4.3 Integração GAT-InteGrade

Conforme visto no capítulo anterior, a integração de um *middleware* com GAT deve ser feita baseada em quais subsistemas do GAT este *middleware* visa a atender. No caso particular do InteGrade, a definição dos subsistemas necessários foi feita de acordo com os requisitos funcionais do InGriDE. Os subsistemas utilizados são: (1) gerenciamento e acesso a dados e (2) gerenciamento de recursos. O primeiro é utilizado para operações com arquivos das aplicações a serem submetidas para a grade. O segundo é responsável pela interação com os serviços da grade responsáveis pela submissão e monitoramento das aplicações.

A utilização desses dois subsistemas tem uma aplicação bem abrangente. Eles atendem não somente às necessidades específicas do InGriDE, mas também aplicações cientes da grade que utilizam

o modelo de programação *bag of tasks* como a apresentada por Blond [27]. Isso possibilita que essas aplicações possam ser executadas no InteGrade usando os adptadores desenvolvidos.

A Figura 4.7 apresenta os principais componentes envolvidos na integração do GAT com o InteGrade. No topo da figura, no ambiente local, InGriDE fornece a interface gráfica com o usuário através da qual serão disparadas as requisições à API GAT. A API GAT repassa as requisições para o GAT Engine que carrega os adaptadores adequados conforme descrito no Capítulo 3. Neste caso, os adaptadores pedidos serão um dos dois disponíveis para o InteGrade. Os adaptadores, por sua vez, interagem com os serviços do InteGrade por intermédio da biblioteca *IntegradeClientAPI* fornecida pelo próprio InteGrade. Essa biblioteca encapsula a lógica de comunicação CORBA feita a partir do ambiente local do sistema com os serviços do InteGrade. Essa biblioteca inclui os *stubs* do serviços do InteGrade, lógica para inicialização do ORB e para construção de objetos a serem passados por parâmetro nas chamadas remotas de método.

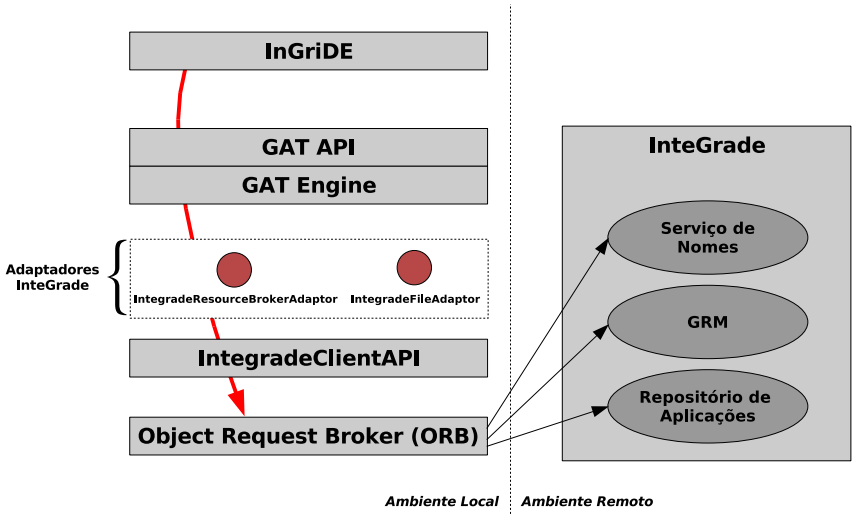

Figura 4.7: Integração GAT-InteGrade.

Os serviços utilizados do *middleware* InteGrade também são apresentados na Figura 4.7. O *Serviço de Nomes* é um serviço padrão oferecido por CORBA que funciona como uma espécie de catálogo de objetos. O serviço disponibiliza a associação de nomes abstratos aos objetos CORBA permitindo que clientes localizem esses objetos buscando-os pelos seus nomes abstratos. A localização de objetos remotos é um requisito comum em sistemas distribuídos e no caso do InteGrade isso não

é diferente. Ressaltamos que a utilização deste serviço é restrita à classe *IntegradeClientAPI*, não afetando os adaptadores.

O *Repositório de Aplicações* armazena as aplicações já desenvolvidas e prontas para serem executadas na grade. Estas aplicações são previamente registradas no repositório para posterior execução. Este serviço envolve principalmente operações relacionadas à listagem, criação e remoção de diretórios e transferência de arquivos. As operações deste serviço são utilizadas exclusivamente pelo *IntegradeFileAdaptor* através da *IntegradeClientAPI*.

O *Global Resource Manager* (GRM) é o componente do InteGrade que atua como um serviço de informações sobre os recursos da grade, além de atuar como o escalonador das requisições para execução de aplicações na grade. Seu principal cliente na arquitetura é o *IntegradeResourceBroker-Adaptor* que o utiliza para realizar as operações de submissão de execução das aplicações.

A comunicação realizada entre o ambiente local e o ambiente remoto é realizada baseada na tecnologia CORBA, modelo de comunicação utilizado no projeto InteGrade. A biblioteca *IntegradeClientAPI* encapsula os detalhes de utilização dos serviços do InteGrade e toda a interação com a camada de comunicação é feita através do *Object Request Broker* (ORB), módulo da arquitetura CORBA responsável pelo encaminhamento e recepção das requisições.

4.5 Implementação

Nesta seção apresentamos os detalhes da implementação dos componentes descritos na seção anterior. A apresentação está organizada em duas partes, acompanhando a estrutura da apresentação da arquitetura. Na Seção 4.5.1 detalhamos a implementação do InGriDE. Em seguida, na Seção 4.5.2, apresentamos a implementação dos adaptadores GAT desenvolvidos para o InteGrade.

4.5.1 Integrated Grid Development Environment (InGriDE)

A implementação do InGriDE é completamente baseada na plataforma Eclipse. Portanto, os componentes internos de sua arquitetura apresentados na Seção 4.4.2, estão organizados em três *plugins* da seguinte forma:

- `br.usp.ime.ingride.ui`: contém a implementação da *Grid User Interface*;
- `br.usp.ime.ingride.core`: contém a implementação dos componentes *Grid Launch Configu-*

ration, *Grid Runtime Controller* e *Grid Runtime Model*;

- `br.usp.ime.ingride.javagat`: é apenas um contenedor da biblioteca JavaGAT com todos os seus adaptadores.

A Figura 4.8 mostra o diagrama de classes que ilustra o relacionamento das principais classes dos diferentes componentes da arquitetura interna do InGriDE. As classes dos componentes estão agrupadas nos diferentes *plugins* descritos acima. A figura ilustra também um exemplo de utilização de classes do arcabouço Eclipse.

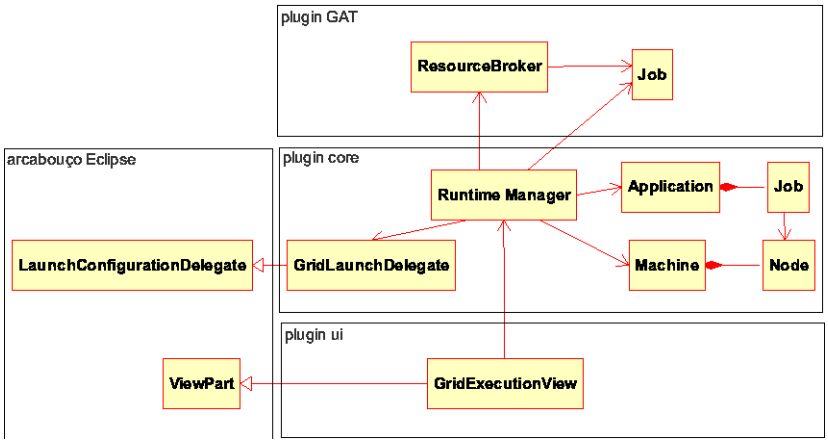

Figura 4.8: Diagrama de classes do InGriDE.

A seguir comentamos a atuação das classes de cada componente:

Grid Runtime Model

Esse componente, conforme mencionado na Seção 4.4.2, mantém um modelo de abstração da grade. As principais classes que representam esse modelo são `Application`, `Job`, `Node` e `Machine`. Essas classes representam aplicações (`Application`), compostas de vários processos (`Job`), que executam em nós (`Node`) de determinadas máquinas (`Machine`) da grade.

Grid Launch Configuration

Esse componente é representado na Figura 4.8 pela classe `GridLaunchDelegate` que fornece a implementação que determina como submeter uma aplicação à grade. Essa classe recebe os

parâmetros para execução e detalhes da submissão. A classe `GridLaunchDelegate` utiliza o arcabouço de lançamento de aplicações do Eclipse estendendo a classe `LaunchConfigurationDelegate`. Essa classe interage também com a classe `Runtime Manager` para ter acesso às informações da grade.

Grid Runtime Controller

Esse é o principal componente do InGriDE por interagir com todos os outros componentes da arquitetura. A Figura 4.8 mostra a classe principal do componente, a `Runtime Manager`, interagindo ao seu lado direito com as classes do *Grid Runtime Model*, à esquerda com as classes do *Grid Launch Configuration*, abaixo com as do *Grid User Interface* e acima com as do GAT.

Grid User Interface

A implementação da interface gráfica com o usuário do InGriDE foi complementamente desenvolvida baseada na plataforma Eclipse. Vários componentes foram utilizados, como por exemplo, perspectivas, visões e *wizards*. A Figura 4.8 mostra um exemplo de utilização onde a visão *GridExecutionView* estende a classe `ViewPart` do Eclipse para disponibilizar uma janela de monitoramento das aplicações submetidas para execução. A *GridExecutionView* utiliza os componentes do *plugin core* para obter as informações necessárias da grade a serem exibidas na janelas. A Figura 4.9 mostra a interface gráfica do InGriDE com o *GridExecutionView* (na parte inferior da figura).

4.5.2 Adaptadores GAT para o InteGrade

A implementação dos adaptadores GAT para o InteGrade foi feita de acordo com a definição do padrão de projeto *Adaptor* [66] utilizado na arquitetura do GAT. O objetivo desse padrão é adaptar uma interface para ser acessada por uma entidade que espera uma interface diferente. No nosso caso temos a interface oferecida pelo InteGrade a ser adaptada para a interface esperada pelo GAT Engine. A seguir mostramos a parte utilizada de cada uma dessas interfaces na nossa implementação, e em seguida detalhamos como foi feita a integração delas.

Interface esperada pelo GAT Engine

A interface esperada pelo GAT Engine é definida pelos *Capability Provider Interfaces* (CPI) dos dois subsistemas utilizados em nosso trabalho: acesso a dados e gerenciamento de recursos. Os elementos desses subsistemas são partes integrantes do GAT. Em função de utilizarmos a implementação

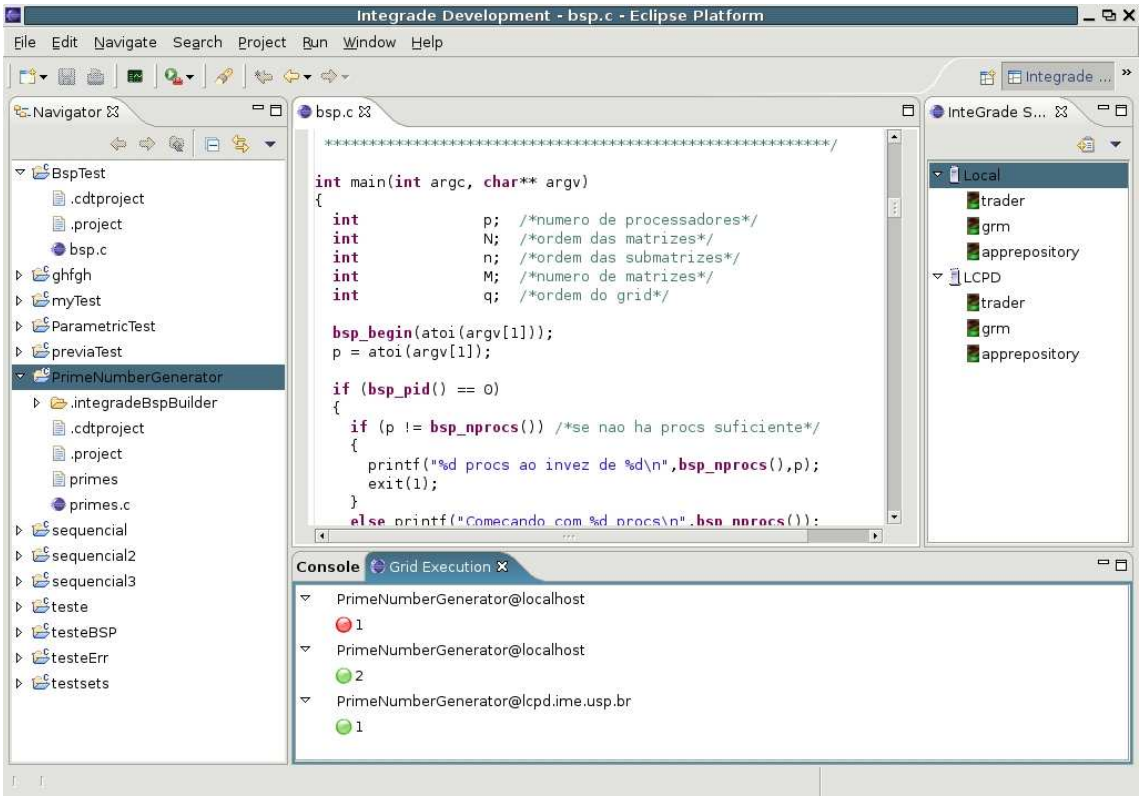

Figura 4.9: Interface gráfica do InGriDE.

em Java do GAT (JavaGAT, versão 1.6), esses elementos são representados por classes e interfaces Java.

A Figura 4.10 apresenta a interface e a classe responsáveis pela manipulação de arquivos do subsistema de acesso a dados. A classe `FileCpi` define a interface esperada para as operações com arquivos. Os métodos de `FileCpi` ilustrados na figura são implementações de métodos da interface `File` e são os utilizados na nossa implementação. Ressaltamos que vários outros métodos estão disponíveis, mas que a utilização dos mesmos não foi necessária para atender aos requisitos funcionais definidos para o InGriDE. Isso é uma característica interessante do GAT, sua arquitetura permite que novos adaptadores implementem apenas os métodos disponíveis no *middleware* em questão.

A Figura 4.11 apresenta as classes e interfaces utilizadas do subsistema de gerenciamento de recursos. A classe `ResourceBrokerCpi` representa o *broker* da grade para o qual serão feitas as

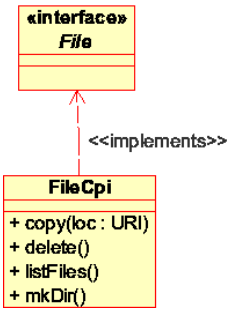

Figura 4.10: CPI do subsistema de acesso a dados.

submissões de execução das aplicações. O método `submitJob` mostrado na figura foi o único necessário na nossa implementação, apesar de outros estarem disponíveis. Esse método é a implementação do método de mesmo nome da interface `ResourceBroker` e devolve um objeto da classe `Job`, que representa uma aplicação submetida para execução. A classe `Job` também possui sua CPI que, junto com a `ResourceBrokerCpi`, constituem a interface esperada pelo GAT Engine para as operações de gerenciamento de recursos utilizadas na nossa implementação. Os métodos da classe `JobCpi` ilustrados na figura foram os utilizados na nossa implementação.

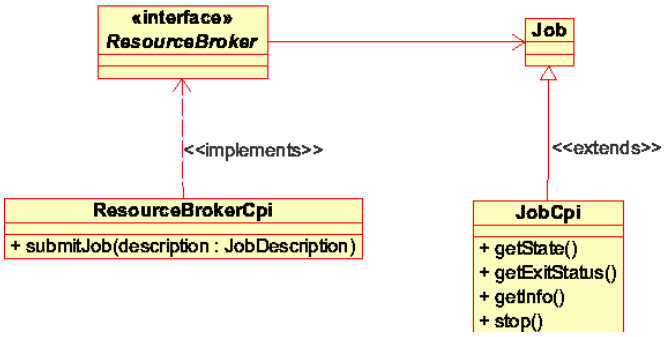

Figura 4.11: CPI do subsistema de gerenciamento de recursos.

É importante ressaltar que as classes CPI apresentadas para os dois subsistemas estão localizadas em pacotes específicos na organização de pacotes da JavaGAT. A classe `FileCpi`, por exemplo, faz parte do pacote `org.grdilab.io.cpi`, ao passo que a interface `File` do mesmo subsistema, por ser exposta para as aplicações, faz parte do pacote `org.grdilab.io`. A organização é análoga para o subsistema de gerenciamento de recursos.

Interface oferecida pelo InteGrade

A interface oferecida pelo InteGrade é representada pela sua biblioteca *IntegradeClientAPI* que disponibiliza um *Façade* [66] para acessar os serviços do *middleware*. A Figura 4.12 apresenta as principais classes que compõem a biblioteca que está disponível em um arquivo `jar`. A classe `ApplicationControlFacade` implementa o padrão de projeto mencionado e seus métodos caracterizam a interface oferecida pelo InteGrade.

Adaptadores para integrar as duas interfaces

A integração da interface esperada pelo GAT com a interface oferecida pelo InteGrade é feita através dos dois adaptadores desenvolvidos nesse trabalho. A implementação dos adaptadores foi feita baseada em uma análise das duas interfaces que identificou o mapeamento dos métodos do GAT para os do InteGrade que possuíam semântica compatíveis. A Tabela 4.1 mostra esse mapeamento, onde os métodos utilizados do GAT estão na primeira coluna e os métodos correspondentes do InteGrade na segunda. Alguns métodos do susbsitema de gerenciamento de recursos não têm um mapeamento direto pois as informações acessadas por eles são atualizadas através de *callbacks* feitos pelo InteGrade. Esses métodos retornam informações sobre o estado da execução dos `Jobs` que são atualizadas nas classes do GAT através de chamadas feitas a partir de classes da *IntegradeClientAPI* sempre que o estado de execução das aplicações muda. Portanto, quando os métodos `getState`, `getExitStatus` e `getInfo` da classe `JobCpi` são chamados, essa iformação já está disponível no GAT pois já foi atualizada previamente.

A ponte de ligação entre o GAT e o InteGrade formada pelos adaptadores está ilustrada na Figura 4.13. O adaptador do subsistema de acesso a dados é representado pela classe `IntegradeFileAdaptor` e o adaptador do subsistema de gerenciamento de recursos é composto pelas classes `IntegradeResourceBrokerAdaptor` e `IntegradeJob`.

O adaptador do subsistema de acesso a dados se concentra nas tarefas relacionadas à transferência e manipulação de arquivos realizadas com o *Repositório de Aplicações* do InteGrade. Seu funcionamento é bem simples resumindo-se em traduzir os pedidos feitos para os métodos definidos na classe `FileCpi` à assinatura dos métodos correspondentes da classe `ApplicationControleFacade` da *IntegradeClientAPI*.

O adaptador do subsistema de gerenciamento de recursos é o responsável pela submissão de

Interface esperada por GAT	Interface oferecida pelo InteGrade
Acesso a dados	
FileCpi.copy	uploadBinary
FileCpi.delete	deleteBinary
FileCpi.mkDir	registerApplication
FileCpi.listFiles	listDirectoryContents
Gerenciamento de recursos	
ResourceBrokerCpi.submitJob	executeBinary
JobCpi.getState	*callback*
JobCpi.getExitStatus	*callback*
JobCpi.getInfo	*callback*

Tabela 4.1: Mapeamento de interfaces entre GAT e InteGrade.

aplicações para execução na grade. A implementação do adaptador, representada pela classe `IntegradeResourceBrokerAdaptor`, se concentra em dois aspectos. O primeiro é a conversão da descrição das aplicações do formato GAT (`JobDesciption`) para a descrição utilizada pelo InteGrade. O segundo aspecto está relacionado às funcionalidades básicas de um *broker* como alocação de recursos, monitoramento e gerenciamento de dados (arquivos de entrada e saída). O monitoramento é realizado através da classe `IntegradeJob` que implementa a interface `IExecutionListener` da *IntegradeClientAPI* para receber *callbacks* disparados pelas mudanças de estado das execuções das aplicações. Uma das tarefas realizadas pela classe `IntegradeJob` é a conversão dos estados de uma aplicação previstos no InteGrade para os estados contemplados no GAT (`JobState`).

A contribuição feita pelo nosso trabalho à biblioteca JavaGAT está localizada no diretório específico de adaptadores, dentro do diretório base de JavaGAT (`GAT-BASE-PATH/adaptors`). O código fonte está organizado em dois pacotes: `org.gridlab.gat.io.cpi.integrade` e `org.gridlab.gat.resources.cpi.integrade`, contendo os adaptadores de acesso a dados e de gerenciamento de recursos, respectivamente.

Conforme mencionado nesse capítulo, todo o acesso dos adaptadores ao InteGrade é feito através da `integradeClientAPI.jar`. Portanto, a biblioteca foi adicionada ao diretório `GAT-BASE-PATH/adaptors/external` para ser utilizada no desenvolvimento dos adaptadores. O processo de implantação dos adaptadores fornecido por JavaGAT copia a biblioteca para o diretório `GAT-BASE-PATH/adaptors/lib`, onde ficam as bibliotecas utilizadas em tempo de execução.

Assim como a biblioteca do InteGrade, os adaptadores também são afetados pelo processo de implantação. Para incluir os novos adaptadores nesse processo acrescentamos ao arquivo `GAT-BASE-PATH/adaptors/build.xml` o código mostrado na listagem 4.1. Esse código adiciona, ao processo de implantação, a geração de dois arquivos `jar` referentes aos adaptadores do InteGrade. Os arquivos `IntegradeFileAdaptor.jar` e `IntegradeResourceBrokerAdaptor.jar` são gerados e copiados para o diretório `GAT-BASE-PATH/adaptors/lib`, de onde o *GAT Engine* os carregará para serem utilizados em tempo de execução.

```
<jar jarfile="${lib}/IntegradeFileAdaptor.jar" basedir="${tmp}"
   includes="**/IntegradeFileAdaptor.class">
 <manifest>
   <attribute name="FileCpi-class"
    value="org.gridlab.gat.io.cpi.integrade.IntegradeFileAdaptor"/>
 </manifest>
</jar>

<jar jarfile="${lib}/IntegradeResourceBrokerAdaptor.jar" basedir="${tmp}"
   includes="**/org/gridlab/gat/resources/cpi/integrade/*.class">
 <manifest>
   <attribute name="ResourceBrokerCpi-class"
    value="org.gridlab.gat.resources.cpi.integrade.IntegradeResourceBrokerAdaptor"/>
 </manifest>
</jar>
```

Listagem 4.1: Implantação dos adaptadores GAT para o InteGrade

4.6 Pontos Positivos, Negativos e Perspectivas de Utilização

Nesta última seção apresentamos considerações sobre as lições aprendidas durante o desenvolvimento desse trabalho nas duas frentes de desenvolvimento que o compõem: o IDE para computação em grade baseado na plataforma Eclipse e na API GAT e os adaptadores GAT para o InteGrade. Apresentamos também uma análise do potencial de utilização da solução e suas restrições.

A primeira frente de trabalho discutida aqui é o desenvolvimento do InGriDE, o IDE para computação em grade. A utilização da plataforma Eclipse como base na construção do IDE se mostrou muito útil nas questões de usabilidade, extensibilidade e reutilização de ferramentas típicas de IDEs. Os componentes fornecidos pelo arcabouço direcionaram a criação da interface gráfica com o usuário

de forma disciplinada quanto às questões de usabilidade. O mecanismo de extensibilidade teve contribuição fundamental tanto na integração da nossa IDE ao Eclipse, quanto na possibilidade de expansão do atual conjunto de funcionalidades. Ressaltamos que para a obtenção desses benefícios foi necessário superar uma considerável curva de aprendizado do arcabouço.

A integração entre InGriDE e o ambiente de grade é toda feita através da API GAT. A utilização da API GAT apresentou um ganho considerável em simplicidade de programação para a grade quando comparada ao acesso direto à API oferecida pelo InteGrade. Esse ganho de simplicidade ocorre principalmente em virtude dos detalhes da realização de uma determinada operação estarem escondidos na implementação fornecida pelos adaptadores. Além disso, a utilização dessa API unificada permitiu a construção de ferramentas no ambiente Eclipse independentes do *middleware* de grade. Essa flexibilidade é obtida através da carga dinâmica dos adaptadores dos sistemas de *middleware*, que permite a utilização de serviços de diferentes sistemas de *middleware* para a mesma função ajustando apenas algumas preferências (classe `Preferences`) que são passadas à API GAT. Essa propriedade permite que o usuário especifique o adaptador que pretende utilizar para a execução a ser feita.

Apesar da independência de *middleware* das ferramentas do IDE, algumas funcionalidades necessitam de parâmetros específicos do *middleware* a ser utilizado. Para esses casos, foram utilizados os pontos de extensão do Eclipse para os quais cada *middleware* fornece os seus parâmetros específicos. Isso permite que o IDE continue sendo flexível sem perder a transparência da grade. Ressaltamos entretanto, que essa flexibilidade nem sempre é possível de ser alcançada. Para algumas funcionalidades muito específicas de cada *middleware*, como as relacionadas às bibliotecas dos modelos de programação, as ferramentas do IDE precisam ser específicas. Um cenário no qual essa situação não aconteceria incluiria duas mudanças na situação atual: (1) inclusão no GAT de bibliotecas para modelos de programação padronizados, (2) os sistemas de *middleware* utilizassem modelos de programação padronizados. Entretanto, achamos que esse seria um cenário difícil de ser alcançado brevemente.

A segunda frente do nosso trabalho foi o desenvolvimento dos adaptadores GAT para o InteGrade. Ao contrário do esperado, a implementação dos adaptadores mostrou-se bastante simples. A organização da estrutura da biblioteca JavaGAT e os exemplos fornecidos pelos outros adaptadores já desenvolvidos facilitaram bastante o nosso trabalho. Conforme mostrado na Seção 4.5.2, a maior dificuldade encontrada na integração do InteGrade ao GAT foi a identificação do que era preciso estender, ou seja, quais adaptadores deveriam ser desenvolvidos. Uma vez identificados os adaptadores necessários, a implementação não apresentou maiores problemas.

Apesar da simplicidade da implementação, alguns problemas de incompatibilidade foram encontrados entre os modelos de programação do GAT e do InteGrade. No InteGrade a submissão para execução é feita considerando uma aplicação que pode ser sequencial, *Bag-of-Tasks* ou BSP. Os dois últimos tipos são aplicações normalmente compostas por vários processos. O processo de submissão no InteGrade recebe a descrição completa da aplicação e, no caso da aplicação ser composta por vários processos, as descrições de todos esses processos devem ser fornecidas. Essas descrições contém informações de arquivos de entrada, arquivos de saída, argumentos, etc. A partir da descrição completa da aplicação, o *broker* do InteGrade se responsabiliza por lançar todos os processos da aplicação nos recursos disponíveis na grade.

No GAT a submissão é feita de forma diferente. A unidade considerada na submissão é um `Job` que corresponde a um processo. Portanto, a submissão de uma aplicação com vários processos só pode ser feita submetendo cada processo separadamente. Em função disso, a aplicação precisa fornecer um módulo de software responsável pela gerência das diferentes submissões. No InGriDE implementamos uma funcionalidade para fazer várias submissões da mesma aplicação, uma para cada processo, cada uma com os parâmetros de cada processo informados pelo usuário através da interface gráfica. Os dois modelos de programação são mostrados na Figura 4.14. Essa limitação poderia ser resolvida com a inclusão na API GAT de uma entidade que representasse uma aplicação composta de um ou vários `Jobs`. Além disso, deveria ser adicionado um método `submitJob` à classe `ResourceBroker` que recebesse como parâmetro uma aplicação ao invés de um `Job`.

Ressaltamos que essa limitação não se aplica às aplicações *Bag-of-Task* (BOT) ou BSP nas quais existe apenas um arquivo de entrada e argumentos para todos os nós. Essas aplicações são *Single Program Multiple Data* (SPMD) e possuem um único arquivo de entrada e argumentos, portanto, a própria aplicação realiza a divisão da entrada e define as partes que cada nó vai processar. Essas aplicações podem ser submetidas como *Jobs* passando como parâmetro o tipo de `Job` (`jobType`) que está sendo submetido (BOT ou BSP). A comunicação entre os nós nas aplicações BSP dá-se através da utilização da biblioteca BSPlib disponível nos recursos da grade através da infra-estrutura do InteGrade.

Outro problema encontrado no GAT foi a ausência de um mecanismo de alto desempenho para comunicação entre processos. O mecanismo de comunicação disponível visa apenas o monitoramento e controle de aplicações. Comunicação de alto desempenho entre processos paralelos está fora do escopo do GAT. Isso caracteriza GAT como uma biblioteca a ser utilizada somente do lado "cliente"

da grade, e a comunicação entre processos paralelos deve ser feita de outra forma.

Uma alternativa para contornar essa limitação do GAT é a utilização de bibliotecas dos próprios sistemas de *middleware* que forneçam comunicação entre processos. Neste caso, as aplicações são escritas baseadas nessas bibliotecas, para as quais os sistemas de *middleware* oferecem suporte nos seus ambientes de execução. Outra alternativa que oferece mais flexibilidade é a utilizada por Nieuwpoort [129]. Nessa, a submissão da aplicação é feita junto com a biblioteca que fornece comunicação entre os nós, não necessitando que esta esteja disponibilizada pela infra-estrutura da grade.

Finalmente, é preciso considerar o potencial de utilização da solução desenvolvida no contexto desse trabalho. As funcionalidades atuais disponíveis no InGriDE atendem aos requisitos funcionais do cenário descrito na Seção 4.1. Apesar de ser um conjunto pequeno, é abrangente o suficiente para atender as necessidades de várias comunidades de desenvolvedores.

As aplicações para a grade para as quais InGriDE está inicialmente preparado são as compatíveis com o GAT e as do *middleware* InteGrade. As primeiras são aplicações baseadas no modelo de programação do GAT descrito acima, podendo ser aplicações já desenvolvidas por terceiros ou aplicações novas. Para serem compatíveis com InGriDE, podem somente depender dos subsistemas atendidos pelos adaptadores do InteGrade. O outro tipo de aplicação contemplado são as do *middleware* InteGrade, que inclui aplicações do tipo sequencial, BOT e BSP. Lembrando que para as BOT e BSP existe a restrição, descrita anteriormente nessa seção, de que tenham apenas um arquivo de entrada e um conjunto de argumentos.

Atualmente InGriDE é compatível com o *middleware* InteGrade. Sua utilização com outros sistemas de *middleware* de grade (por exemplo, Globus) depende de que esses sistemas forneçam adaptadores para os dois subsistemas dos quais InGriDE depende. Outra questão crítica para sua portabilidade é a segurança, um importante requisito não funcional não considerado nesse trabalho. Todas as funcionalidades apresentadas até aqui não realizam nenhum tipo de operação de autenticação e autorização junto à grade. Portanto, a utilização do InGriDE no seu estado atual com um *middleware* que tenha como requisito a segurança não está contemplada.

O próximo capítulo apresenta as principais ferramentas que fornecem ambientes integrados de desenvolvimento (IDE) para computação em grade e que se relacionam com o nosso trabalho.

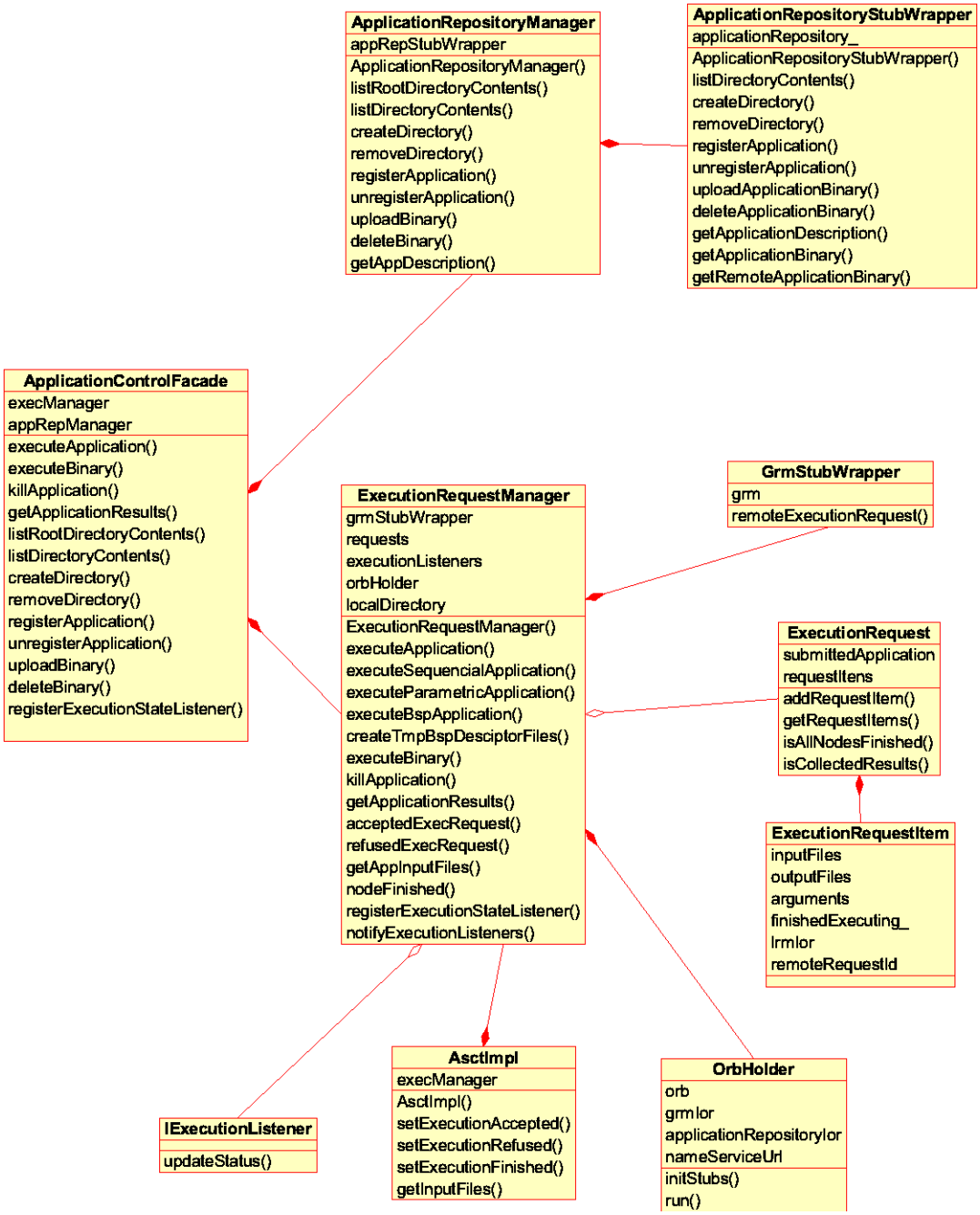

Figura 4.12: Biblioteca de acesso ao InteGrade.

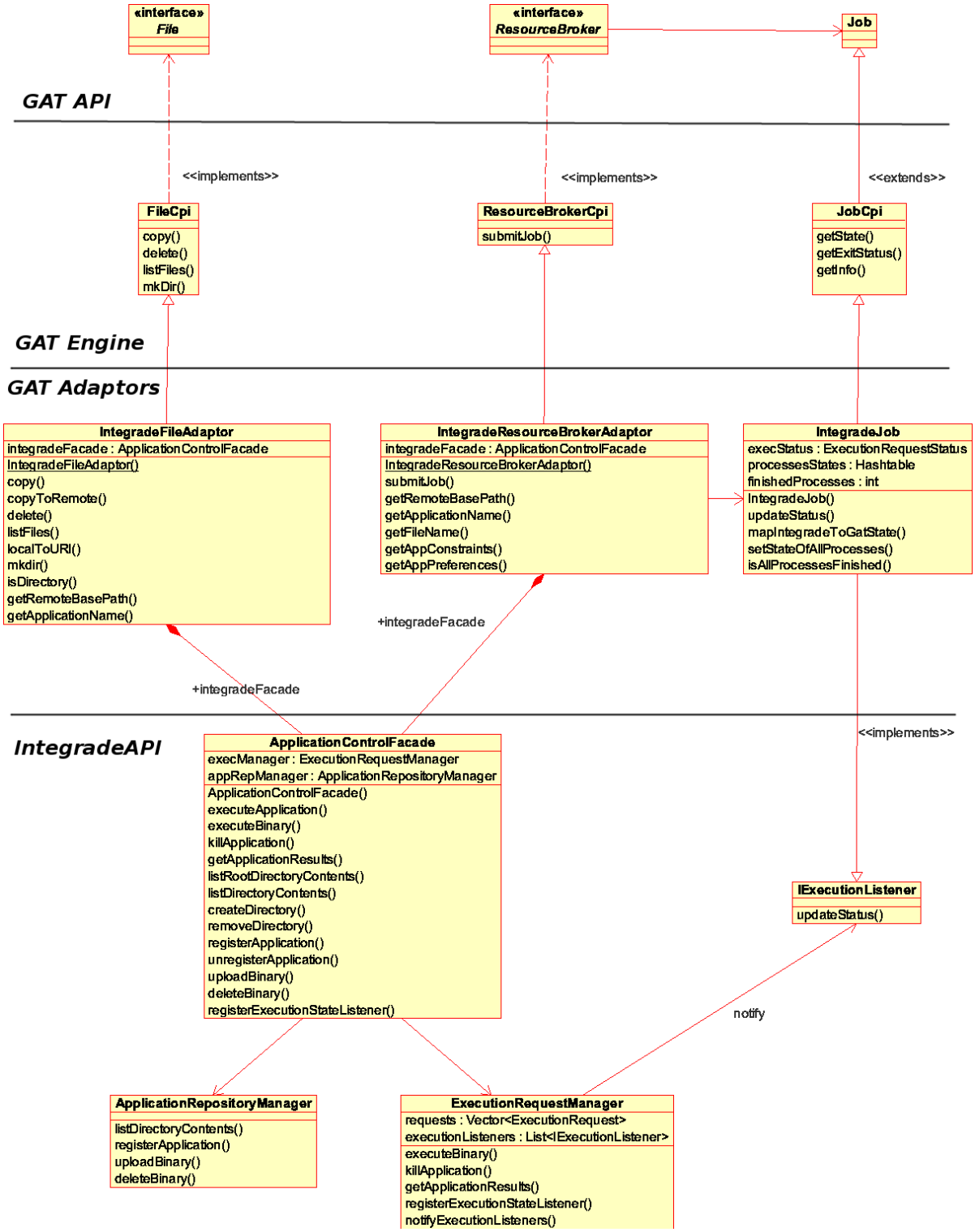

Figura 4.13: Adaptadores GAT desenvolvidos para o InteGrade.

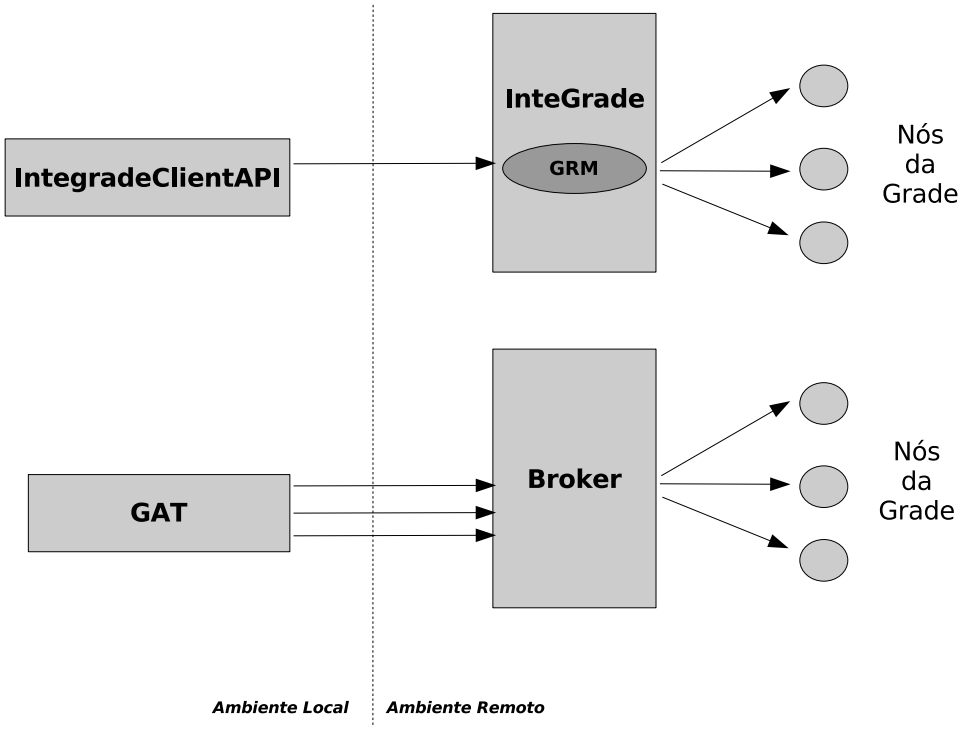

Figura 4.14: Modelos de programação do GAT e do InteGrade.

Capítulo 5

Trabalhos Relacionados

No Capítulo 2 vimos as principais ferramentas que oferecem funcionalidades para desenvolvimento de aplicações e interação com os serviços da grade. Neste capítulo apresentamos as principais ferramentas que fornecem ambientes integrados de desenvolvimento (IDE) para computação em grade. Diferente das apresentadas no Capítulo 2, todas as ferramentas aqui apresentadas utilizam algum dos modernos arcabouços para construção de IDEs disponíveis (por exemplo, Eclipse e NetBeans [127]) na atualidade.

As iniciativas nessa categoria estão dividas em duas linhas de acordo com o objetivo funcional da ferramenta. A primeira se concentra no desenvolvimento e implantação de serviços da grade. A segunda, que é a mais próxima do nosso trabalho, se concentra no desenvolvimento, implantação e monitoramento de aplicações para a grade. A seguir resumimos os principais projetos de cada uma dessas duas linhas relacionando-os com o nosso trabalho.

5.1 IDEs para Serviços

GT4IDE [91] é uma ferramenta já descontinuada, que foi desenvolvida como um *plugin* da plataforma Eclipse visando facilitar o desenvolvimento de Web Services Java compatíveis com WSRF[1]. A idéia era fornecer aos desenvolvedores de serviços para o *middleware* Globus Toolkit 4 um ambiente que integrasse todos os passos desde a codificação à implantação destes Web Services. As funcionalidades fornecidas por GT4IDE incluíam:

[1] Web Services Resource Framework

- Geração de arquivos referentes aos serviços com base em parâmetros fornecidos pelos usuários;
- Geração de código Java;
- Sincronização entre código Java e o arquivo WSDL;
- Geração do arquivo de implantação GAR (*Grid Archive*), que contém todos os arquivos e informações do serviço para a grade.

Após o projeto GT4IDE ser descontinuado, outros projetos emergiram com o mesmo propósito na incubadora de projetos da Globus Alliance. O GDTE (*Grid Development Tools for Eclipse*) [68] é o principal deles e oferece um ambiente de desenvolvimento e composição de serviços de grade baseado em modelos. Os desenvolvedores dos serviços podem escolher entre diferentes modelos incluindo classes Java com anotações ou modelos UML2 [156]. GDTE possui um mecanismo baseado em um meta-modelo que possibilita a especificação dos serviços independente da plataforma de grade utilizada. A partir do meta-modelo existe um processo de transformação para código dependente do *middleware*. Esse processo é definido pelo *Target Platform Mapping Model* (TPMM), cuja implementação atual é compatível com os sistemas Globus e MAGE [124]. Existe a possibilidade de extensão do TPMM para atender a diferentes sistemas de *middleware*.

Os trabalhos dessa categoria se relacionam com o nosso em dois aspectos. O primeiro é a utilização do mesmo arcabouço para criação de IDEs, adotando o Eclipse. O segundo é a preocupação com independência de *middleware*. No GT4IDE a interação era feita diretamente através da API oferecida pelo sistema Globus, não se preocupando em atender outros sistemas de *middleware*. Já o GDTE, utiliza um meta-modelo para abstrair o *middleware* para o qual está sendo desenvolvido o serviço. Seu mecanismo de geração de código prevê a extensão para outros sistemas.

5.2 IDEs para Aplicações

Neste seção apresentamos as ferramentas que se concentram em funcionalidades para o desenvolvimento, implantação e monitoramento de aplicações para a grade.

5.2.1 JOpera

JOpera [98] é uma ferramenta para o desenvolvimento de aplicações de fluxo de trabalho baseadas em serviços, podendo ser desde Web Services até Grid Services. JOpera oferece para o desenvolvedor

um ambiente visual de fluxo de trabalho (Figura 5.1) similar aos apresentados na Seção 2.2.1. O ambiente é baseado na plataforma Eclipse e oferece funcionalidades para o desenvolvimento, execução e monitoramento dos fluxos de trabalho. JOpera possui uma plataforma para execução dos fluxos de trabalho que apresenta uma arquitetura flexível que possibilita a interação com serviços de diferentes sistemas. Essa arquitetura apresenta uma estrutura baseada em adaptadores muito parecida com a de GAT (Seção 3.3), utilizada em nosso trabalho. Atualmente estão disponíveis adaptadores para os sistemas Globus, Condor e implementações do SSH.

O JOpera se relaciona com o nosso trabalho tanto nas questões de extensibilidade oferecida pela plataforma Eclipse, quanto na consideração do requisito de independência de *middleware*. Entretanto, ao contrário da nossa abordagem, o mecanismo utilizado para abstrair a grade é próprio, não considerando padrões emergentes como SAGA.

5.2.2 GriDE

GriDE [84, 145] é um ambiente integrado de desenvolvimento (IDE) para computação em grade que visa fornecer em uma única ferramenta recursos para o desenvolvimento, depuração, execução e monitoramento de aplicações para a grade. Esta ferramenta, desenvolvida no *Asia Pacific Science and Technology Center* da Sun Microsystem, foi projetada como um módulo da plataforma NetBeans.

As funcionalidades disponíveis no GriDE incluem a submissão de tarefas na grade, busca de recursos na grade e monitoramento das tarefas submetidas com recuperação de resultados. Além disso, existem algumas funções preliminares de um editor de fluxo de trabalho para a grade. Todas essas funcionalidades são realizadas atualmente baseadas no Globus. A plataforma de fluxo de trabalho também é compatível com o *middleware* de grade *Sun Grid Engine* (SGE)[2].

GriDE apresenta um conjunto de funcionalidades bem interessante para o desenvolvedor de aplicações. Entretanto, não existe uma preocupação com relação à independência de *middleware*. Apesar do editor de fluxo de trabalho ser compatível com o SGE, todas as outras ferramentas de GriDE são fortemente acopladas ao Globus.

[2] http://gridengine.sunsource.net/

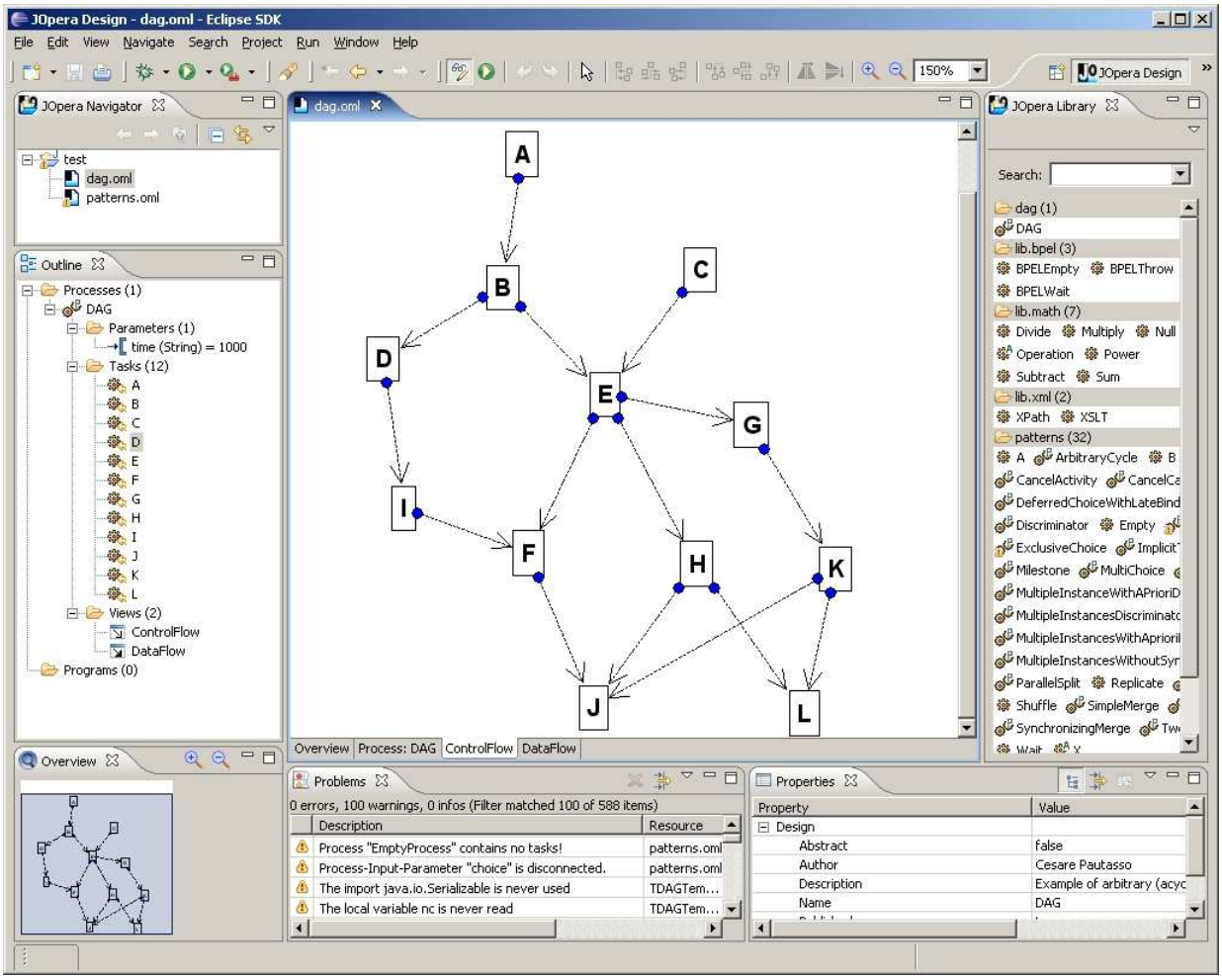

Figura 5.1: Ambiente do JOpera.

5.2.3 ProActive ICD2

O IC2D (*Interactive Control and Debugging of Distribution*) [24, 94] é um ambiente gráfico para monitoramento e controle de aplicações construídas usando o arcabouço para programação paralela e distribuída ProActive [20, 139]. O ambiente fornece ao programador instrumentos para visualizar as ocorrências internas de uma *aplicação ProActive* em tempo de execução e ainda permite que ele controle interativamente o mapeamento de tarefas a máquinas, tanto na criação das tarefas quanto depois, realizando migrações. O objetivo da ferramenta é ajudar os usuários a implantar, monitorar

e controlar a execução de aplicações ProActive em plataformas distribuídas, incluindo as grades.

As funcionalidades de monitoramento e controle de IC2D são oferecidas às aplicações desenvolvidas usando a biblioteca ProActive sem a necessidade de qualquer modificação. A comunicação das ferramentas de monitoramento e a aplicação em execução é feita através de um dos protocolos disponíveis para ProActive podendo ser RMI, HTTP, Ibis/RMI, RMI com tunelamento SSH ou Jini. Vale ressaltar que na fase anterior ao monitoramento, essas aplicações são implantadas no sistema distribuído utilizado. Em ambientes de grade essa implantação é feita através dos protocolos disponíveis de cada *middleware* (por exemplo, Globus, Unicore ou GLite).

IC2D fornece três tipos de ferramentas (Figura 5.2): visualização gráfica, visualização textual, e controle e monitoramento. A primeira inclui a visualização gráfica de máquinas, máquinas virtuais Java, topologia de *Active Objects* [3], estado dos objetos (executando, esperando) e a migração deles. A segunda exibe a lista de eventos e de mensagens trocadas pelos objetos. O terceiro permite o controle interativo do mapeamento objeto-máquina na criação, execução passo a passo e a migração de *Active Objects* em execução através de movimentos do mouse.

Recentemente o grupo desenvolvedor do ProActive, o instituto francês INRIA de Sophia Antipolis[4], migrou o IC2D para a plataforma Eclipse. A ferramenta está disponível em dois formatos: uma aplicação *standalone* baseada na *Eclipse Rich Client Platform*, ou *plugins* para o Eclipse IDE. Ambos formatos apresentam o mesmo conjunto de funcionalidades descritos acima, sendo que o segundo possui funcionalidades adicionais (por exemplo, editores e *wizards*) para o desenvolvimento de aplicações.

A Figura 5.2 mostra a versão de IC2D para IDE, onde três máquinas estão sendo monitoradas e duas aplicações estão em execução. As aplicações em execução são: um renderizador 3D colaborativo distribuído (C3D) [140], e uma aplicação para o clássico problema de programação concorrente conhecido como *Dinning Philosophers* [141]. Na figura é possível visualizar o rico conjunto de ferramentas gráficas para monitoramento fornecido pelo IC2D.

IC2D tem um relacionamento especial com o nosso trabalho. No capítulo anterior mencionamos a possibilidade de extensão de InGriDE através da integração de diferentes modelos de programação. Levando em conta que o ProActive caracteriza-se como tal, podemos considerar IC2D como um

[3] Active objects são unidades básicas de atividade e distribuição usadas para construir aplicações concorrentes usando ProActive. A arquitetura utilizada é baseada no padrão de projeto *Active Object* [113].

[4] http://www-sop.inria.fr/

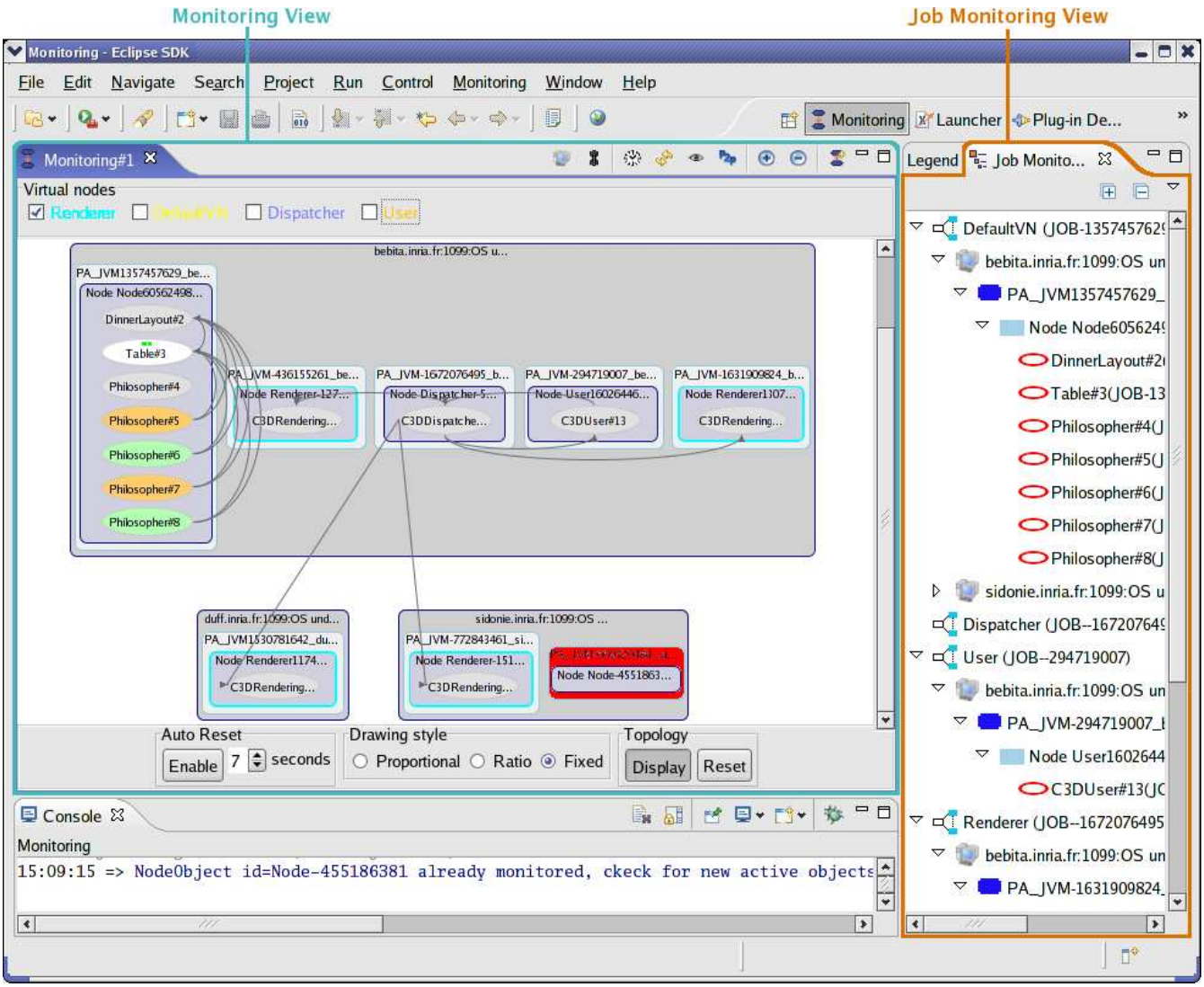

Figura 5.2: Visão geral de duas aplicações em execução monitoradas pelo IC2D.

candidato para essa integração. Uma diferença que precisaria ser tratada nessa integração seria o processo de implantação, que em IC2D é feito baseado no mecanismo de ProActive e em InGriDE é feito através do GAT. A princípio sugerimos a adoção do mecanismo utilizado em InGriDE por questões de conformidade com padrões já mencionados nesse capítulo.

5.2.4 g-Eclipse

g-Eclipse [69] é um projeto patrocinado pela União Européia que tem como membros seis instituições européias incluindo universidades, institutos de pesquisa e empresas. Esse projeto tem como principal objetivo a construção de uma plataforma integrada baseada na plataforma Eclipse que simplifique o acesso dos usuários aos recursos da grade. g-Eclipse tem uma importância especial com relação aos outros trabalhos aqui apresentados por ter se tornado um projeto integrante da plataforma Eclipse [70].

g-Eclipse tem como objetivo integrar em seu ambiente diferentes ferramentas, incluindo as já existentes. Por isso, existe uma preocupação em oferecer mecanismos de extensibilidade para contribuição de terceiros. Outro objetivo do projeto é a compatibilidade com diferentes sistemas de *middleware*. Por isso, apesar de estar previsto no primeiro ano do projeto (até agosto de 2007) compatibilidade apenas com o *middleware* gLite [73], no segundo será feito o desenvolvimento para outros sistemas de *middleware*.

Os requisitos funcionais [62] do g-Eclipse foram agrupados baseados na identificação de três atores principais da grade: usuários, operadores e desenvolvedores de aplicações. Os usuários tem como característica a utilização da grade para a submissão, monitoramento e obtenção de resultados de aplicações já disponíveis. Os operadores caracterizam-se por executar operações de administração da grade como configurações, monitoramento de recursos e gerenciamento de organizações virtuais (*Virtual Organizations*). Os desenvolvedores de aplicações são os responsáveis pela construção das aplicações.

g-Eclipse planeja fornecer interfaces com o usuário para atender as necessidades dos três atores descritos acima. Essas interfaces estão em desenvolvimento e são todas baseadas nos componentes do Eclipse incluindo três perspectivas, uma para cada ator que agrupam diferentes editores e visualizadores. Também estarão disponíveis *wizards* e páginas de preferências. Exemplos dessas interfaces com o usuário incluem os *wizards* para definição de processos a serem submetidos e para autenticação, o editor para o arquivo de submissão `jsdl`, menus de contexto para submissão de aplicações e um visualizador de resultados de execuções, inicialmente integrado com a ferramenta GVid [92], que oferece visualização remota de resultados em vídeo para grades.

A arquitetura do g-Eclipse [63] foi desenhada com a preocupação em atender vários sistemas de *middleware*. Para tanto, foi criado um mecanismo de abstração da grade baseado na arquitetura

de *plugins* do Eclipse. Esse mecanismo fundamenta-se em um conjunto de abstrações (classes e interfaces) para os principais componentes da grade que formam a base da arquitetura. Essas abstrações são especializadas para o *middleware* em questão através de *plugins*. Esses *plugins* oferecem a implementação da abstração utilizando os serviços específicos do *middleware*.

A primeira versão da arquitetura foi lançada em outubro de 2006 e aborda aspectos considerados principais da grade como segurança e recursos. São considerados recursos os seguintes elementos: arquivos, processos, recursos computacionais, recursos de armazenamento, redes, aplicações, aglomerados, canais de comunicação e sistemas de arquivos. Existe também uma entidade chamada *ResourceManager* responsável pelo gerenciamento de um conjunto de recursos.

O desenvolvimento do projeto g-Eclipse foi planejado para uma duração de dois anos. Teve seu início em julho de 2006, durante o desenvolvimento do nosso projeto, e disponibilizou o projeto da arquitetura inicial em outubro do mesmo ano. Em janeiro de 2007 foi lançada internamente a primeira versão. Essa versão inclui funcionalidades para a configuração de uma aplicação para submissão e obtenção de resultados utilizando o *middleware* gLite. A versão também fornece ferramentas para interação com mecanismos de segurança do gLite. O cronograma do projeto prevê para junho de 2007 uma versão aperfeiçoada da atual, outra para dezembro de 2007 com suporte a outro *middleware* (baseado em OGSA, ainda indefinido) e a última para junho de 2008 com o suporte aperfeiçoado para o outro *middleware*.

g-Eclipse é o projeto mais próximo ao nosso trabalho. Os objetivos do g-Eclipse com relação à integração de ferramentas e extensibilidade da IDE, e a preocupação com a independência de *middleware* são iguais aos nossos. A principal diferença entre g-Eclipse e nosso trabalho é o mecanismo de abstração utilizado. Em g-Eclipse, conforme descrito, foi construído seu próprio mecanismo baseado na arquitetura extensível da plataforma Eclipse, em InGriDE utilizamos o GAT. Se por um lado a abordagem do g-Eclipse oferece um poderoso mecanismo de extensibilidade, por outro, sua utilização é restrita à tecnologia Eclipse e não se alinha com padrões emergentes como SAGA. Apesar disso, um contato com o coordenador do projeto [108] nos revelou que em algum ponto o g-Eclipse se preocupará em utilizar ou se adequar à SAGA. Por fim, vale enfatizar a magnitude do projeto e a tendência em se tornar vastamente utilizado em virtude da sua inclusão como projeto oficial da plataforma Eclipse.

5.3 Resumo

As iniciativas de desenvolvimento de ferramentas apresentadas no Capítulo 2 e de IDEs apresentados nesse capítulo mostram o grande esforço que tem sido feito nos últimos tempos para facilitar o uso da grade. Essa tendência reforça a importância dessa questão para a disseminação dessa tecnologia junto aos usuários e desenvolvedores.

Capítulo 6

Conclusões

A complexidade dos ambientes de grade e a atual diversidade de ferramentas fazem da utilização da grade pelos desenvolvedores um trabalho muito complexo. Nesse trabalho apresentamos uma proposta para minimizar a dificuldade de desenvolvimento de aplicações para a grade através de um ambiente integrado de desenvolvimento (IDE) para computação em grade extensível e independente de plataforma de *middleware*.

A independência de plataforma de *middleware* foi obtida através da interface de programação GAT. Sua utilização pelo IDE apresentou vários benefícios como simplicidade de programação, flexibilidade e transparência da grade. A unificação semântica proporcionada pelo GAT permitiu a criação de interfaces genéricas com o usuário. Esses benefícios proporcionaram uma proteção às ferramentas do IDE das freqüentes mudanças nos sistemas de *middleware*.

Entretanto, alguns problemas também foram identificados na utilização do GAT. O primeiro foi uma incompatibilidade entre os modelos de programação do GAT e do InteGrade. Problema para o qual apresentamos uma solução simples, baseada na extensão da semântica da API. O segundo foi a ausência de um mecanismo para a comunicação entre os processos das aplicações. Essa característica limita o uso do GAT ao lado "cliente" da grade, deixando às aplicações paralelas que necessitam de comunicação entre os nós uma única alternativa, a de utilizar bibliotecas específicas dos sistemas de *middleware* ou dos modelos de programação para grade.

O desenvolvimento dos adaptadores do GAT para o InteGrade mostrou-se simples, a partir do momento que a semântica do GAT foi compreendida e mapeada para os serviços do InteGrade.

A conclusão geral da utilização do GAT é positiva. Existem pontos negativos a considerar, mas

nenhum deles aparenta ser insolúvel. Na nossa opinião, a adequação completa do GAT para ser utilizado por IDEs é apenas uma questão de utilização em diferentes ambientes reais de desenvolvimento para obtenção de *feedback* e realização dos ajustes semânticos necessários na API.

Uma questão que fica em aberto é sobre o caminho que os IDEs para grades vão seguir quanto ao mecanismo de independência do *middleware*. Nesse trabalho apresentamos a viabilidade de utilização de uma camada de abstração como GAT, que têm como sucessora SAGA, um padrão emergente do GGF. Fica a dúvida se abordagens mais flexíveis, dependentes de determinadas tecnologias, como a utilizada no projeto g-Eclipse (Seção 5.2.4) não vão se estabelecer como padrão de fato.

O resultado obtido com a utilização do arcabouço Eclipse quanto à facilidade de uso e extensibilidade atendeu completamente nossas expectativas. O comprovado crescimento do uso de arcabouços de IDEs por várias ferramentas para a grade (Capítulo 5) nos faz acreditar que está havendo um movimento de utilização de IDEs em grades que se alinha com a vasta disseminação dessas IDEs em ambientes de desenvolvimento tradicionais. Acreditamos que a extensibilidade fornecida por arcabouços como Eclipse e NetBeans se adequam perfeitamente à necessidade de utilização de várias ferramentas no processo de desenvolvimento de aplicações para a grade. Acreditamos também que as atuais ferramentas deverão ser incorporadas nesses ambientes e que as novas já deverão ser desenvolvidas para estes.

6.1 Contribuições do trabalho

A principal contribuição científica desse trabalho é:

- Avaliação da utilização de uma camada de abstração de *middleware* (GAT) em um IDE para computação em grade.

As principais contribuições técnicas desse trabalho são:

- Integração do *middleware* InteGrade ao GAT e avaliação do mecanismo de extensão fornecido;
- Dois adaptadores JavaGAT para o *middleware* InteGrade. Esses adaptadores podem ser utilizados pelo IDE desenvolvido e por aplicações GAT ou pelo InteGrade. Está prevista a inclusão do código produzido na distribuição oficial da biblioteca;

- IDE para computação em grade baseado na plataforma Eclipse, que fornece funcionalidades para desenvolvimento, submissão, monitoramento e obtenção dos resultados da execução. O conjunto inicial de funcionalidades é pequeno mas abrangente, pois atende necessidades comuns a diversas comunidades de desenvolvedores de aplicações.

6.2 Trabalhos Futuros

Na área de computação em grade vários componentes do ecosistema ainda se encontram em fase de desenvolvimento. Apesar dos esforços do GGF quanto à padronização, várias questões permanecem em aberto (por exemplo, modelo de programação). Portanto, diferentes abordagens para o mesmo problema conviverão por algum tempo. Enquanto isso, usuários e desenvolvedores de aplicações terão de lidar com essa diversidade de tecnologias. Nosso trabalho tem como objetivo tornar menos traumática a experiência de desenvolvimento de aplicações para a grade nesse universo de diferentes tecnologias. Entretanto, muito trabalho ainda precisa ser feito. A seguir apresentamos algumas sugestões para trabalhos futuros identificadas durante o desenvolvimento do nosso projeto.

- Para avaliar a independência de plataforma do InGriDE é importante a sua utilização com outro *middleware* (por exemplo, Globus) que ofereça os mesmos adaptadores utilizados. Dependendo do *middleware* escolhido, pode ser necessário estender InGriDE para atender ao requisito de segurança;

- Os adaptadores desenvolvidos expandiram o horizonte de aplicações compatíveis com o InteGrade, para aplicações compatíveis com GAT. Portanto, é interessante que aplicações baseadas em GAT, originalmente desenvolvidas para executar em outro *middleware* (por exemplo, Globus), sejam executadas no InteGrade de forma a comprovar a transparência da grade oferecida por GAT. Estamos estudando uma aplicação que implementa um codificador MPEG paralelo [27] baseado em GAT originalmente desenvolvida para ser executada no Globus. O modelo de programação da aplicação é compatível para ser executada no InteGrade utilizando os adaptadores desenvolvidos. O código fonte da aplicação já nos foi fornecido pelos autores, que mostraram bastante interesse na experiência;

- Os adaptadores GAT do InteGrade também possibilitam o desenvolvimento de novas aplicações baseadas em GAT para o InteGrade. O uso do GAT nessas aplicações torna mais simples a

programação, além de possibilitar o uso do modelo de programação *master-worker* fornecido pela biblioteca. Além disso, essas novas aplicações poderão ser executadas em outros sistemas de *middleware* compatíveis com GAT. Uma aplicação que está sendo desenvolvida por Rodrigo Assirati Dias no seu projeto de mestrado é uma candidata para utilizar GAT. A aplicação consiste de um *middleware* de nível de usuário para processamento de dados estatísticos e biológicos utilizando o R, um ambiente para análise estatística;

- InGriDE foi desenvolvido com a preocupação de poder ser estendido. Portanto, a incorporação de modelos de programação já disponíveis é bastante interessante para o enriquecimento e avaliação da extensibilidade da ferramenta. Sugerimos como candidato, a ferramenta IC2D do projeto ProActive apresentada Capítulo 5. Além de utilizar a mesma plataforma, Eclipse, o IC2D possui uma boa documentação [95] sobre a sua estrutura interna;

- Conforme já mencionado, InGriDE foi desenvolvido para atender um conjunto reduzido de requisitos funcionais. Portanto, várias funcionalidades interessante para a nossa ferramenta, como a das ferramentas apresentadas nos Capítulos 2 e 5, ficaram ausentes. A inclusão de novas funcionalidades no InGriDE é importante para atender uma comunidade cada vez maior de usuários. Um exemplo de uma funcionalidade interessante seria a construção de aplicações utilizando um editor de fluxos de trabalho. Já existe um editor de fluxo de trabalho no Eclipse baseado na linguagem BPEL [30] que poderia ser aproveitado e adaptado para atender às necessidades específicas do projeto. O desenvolvimento de novas funcionalidades, em alguns casos, pode gerar a necessidade de desenvolvimento de novos adaptadores GAT. Um exemplo para esse caso seria a funcionalidade de monitoramento de recursos físicos da grade;

- Outro trabalho interessante é a implementação da nossa proposta de extensão da semântica do GAT. Sugerimos uma avaliação de sua utilização com o InteGrade e um estudo da aplicação do modelo para outros sistemas de *middleware*.

Referências Bibliográficas

[1] *Astrophysics Simulation Collaboratory*, http://wugrav.wustl.edu/ASC/, Último acesso em janeiro/2007.

[2] *Mercury*, http://www.gridlab.org/WorkPackages/wp-11/, Último acesso em janeiro/2007.

[3] *P-GRADE*, http://www.lpds.sztaki.hu/pgrade/main.php?m=1, Último acesso em janeiro/2007.

[4] *P-GRADE Portal*, http://www.lpds.sztaki.hu/pgportal/, Último acesso em janeiro/2007.

[5] *What Is a Portlet*, http://www.onjava.com/, Último acesso em janeiro/2007, ONJava web site.

[6] D. Abramson, *Applications Development for the Computational Grid*, Frontiers of WWW Research and Development - APWeb 2006, 8th Asia-Pacific Web Conference, 2006, pp. 1–12.

[7] D. Abramson, J. Giddy, and L. Kotler, *High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?*, IPDPS '00: Proceedings of the 14th International Symposium on Parallel and Distributed Processing, 2000, pp. 520–528.

[8] G. Allen, W. Benger, T. Dramlitsch, T. Goodale, H. Hege, G. Lanfermann, A. Merzky, T. Radke, E. Seidel, and J. Shalf, *Cactus Tools for Grid Applications*, Cluster Computing **4** (2001), no. 3, 179–188.

[9] G. Allen, K. Davis, K. Dolkas, N. Doulamis, T. Goodale, T. Kielmann, A. Merzky, J. Nabrzyski, J. Pukacki, T. Radke, M. Russell, E. Seidel, J. Shalf, and I. Taylor, *Enabling Applications on the Grid: A GridLab Overview*, International Journal of High Performance Computing

Applications: Special Issue on Grid Computing: Infrastructure and Applications **17** (2003), no. 4, 449–466.

[10] G. Allen, K. Davis, T. Goodale, A. Hutanu, H. Kaiser, T. Kielmann, A. Merzky, R. Nieuwpoort, A. Reinefeld, F. Schintke, T. Schütt, E. Seidel, and B. Ullmer, *The Grid Application Toolkit: Towards Generic and Easy Application Programming Interfaces for the Grid*, Proceedings of the IEEE, vol. 93, 2005, pp. 534–550.

[11] G. Allen, T. Dramlitsch, T. Goodale, G. Lanfermann, T. Radke, E. Seidel, T. Kielmann, K. Verstoep, Z. Balaton, P. Kacsuk, F. Szalai, J. Gehring, A. Keller, A. Streit, L. Matyska, M. Ruda, A. Krenek, H. Knipp, A. Merzky, A. Reinefeld, F. Schintke, B. Ludwiczak, J. Nabrzyski, J. Pukacki, H. Kersken, G. Aloisio, M. Cafaro, W. Ziegler, and M. Russell, *Early Experiences with the EGrid Testbed*, CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid (Washington, USA), IEEE Computer Society, 2001, p. 130.

[12] M. Alt, J. Dünnweber, J. Müller, and S. Gorlatch, *Component Models and Systems for Grid Applications*, ch. HOCS: Higher-Order Components for Grids, pp. 157–166, Springer US, 2005.

[13] K. Amin, G. Laszewski, M. Hategan, R. Al-Ali, O. Rana, and D. Walker, *An abstraction model for a Grid execution framework*, J. Syst. Archit. **52** (2006), no. 2, 73–87.

[14] D. Anderson, *BOINC: A System for Public-Resource Computing and Storage*, Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing, 2004, pp. 4–10.

[15] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, *SETI@home: an experiment in public-resource computing*, Communications of the ACM **45** (2002), no. 11, 56–61.

[16] N. Andrade, W. Cirne, F. Brasileiro, and P. Roisenberg, *Ourgrid: An approach to easily assemble grids with equitable resource sharing*, Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing, 2003.

[17] T. Andrews, *Business Process Execution Language for Web Services*, Tech. report, IBM, 2003, versão: 1.1.

[18] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski, *Toward a Common Component Architecture for High-Performance Scientific Computing*,

HPDC '99: Proceedings of the The Eighth IEEE International Symposium on High Performance Distributed Computing (Washington, USA), IEEE Computer Society, 1999, p. 13.

[19] R. Badia, J. Labarta, R. Sirvent, J. Perez, J. Cela, and R. Grima, *Programming Grid Applications with GRID Superscalar*, Journal of Grid Computing **1** (2003), no. 2, 151–170.

[20] L. Baduel, F. Baude, D. Caromel, A. Contes, F. Huet, M. Morel, and R. Quilici, *Grid Computing: Software Environments and Tools*, ch. Programming, Deploying, Composing, for the Grid, Springer-Verlag, January 2006.

[21] H. Bal, H. Casanova, J. Dongarra, and S. Matsuoka, *The Grid 2: Blueprint for a New Computing Infrastructure*, ch. 24, pp. 463–489, Morgan Kaufmann Publishers, 2004.

[22] Z. Balaton, P. Kacsuk, N. Podhorszki, and F. Vajda, *From Cluster Monitoring to Grid Monitoring Based on GRM*, Euro-Par '01: Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing (London, UK), Springer-Verlag, 2001, pp. 874–881.

[23] M. Balle and T. Hood, *GGF UPDT User Development Tools Survey*, Tech. report, March 2004.

[24] F. Baude, A. Bergel, D. Caromel, F. Huet, O. Nano, and J. Vayssière, *IC2D: Interactive Control and Debugging of Distribution*, LSSC '01: Proceedings of the Third International Conference on Large-Scale Scientific Computing-Revised Papers (London, UK), Springer-Verlag, 2001, pp. 193–200.

[25] Fran Berman, Geoffrey Fox, and Anthony J. G. Hey, *Grid computing: Making the global infrastructure a reality*, John Wiley & Sons, Inc., New York, NY, USA, 2003.

[26] G. Bezerra, *Análise de Conglomerados Aplicada ao Reconhecimento de Padrões de Uso de Recursos Computacionais*, Master's thesis, Instituto de Matemática de Estatística - Universidade de São Paulo, 2006.

[27] S. Blond, A. Oprescu, and C. Zhang, *Early Application Experience with the Grid Application Toolkit (GAT)*, Workshop on Grid Applications held in conjunction with the Fourteenth Global Grid Forum (GGF'14), June 2005.

[28] *BOINC: Berkeley Open Infrastructure for Network Computing*, http://boinc.berkeley.edu/, Último acesso em dezembro/2006.

[29] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard, *Web Services Architecture*, World Wide Web Consortium, Fevereiro de 2004.

[30] *BPEL Project*, http://www.eclipse.org/bpel, Último acesso em janeiro/2007.

[31] Braun, Broberg, Cassidy, Freedman, Jones, Schaeck, and Tayar, *Web Services for Remote Portlets Specification*, Tech. report, OASIS Organization, 2003.

[32] MORVEN LEESE Morven Morven Leese Brian Everitt, SABINE LANDAU, *Cluster analysis*, 4th ed., Hodder Arnold, 2001.

[33] R. Camargo, A. Goldchleger, F. Kon, and A. Goldman, *Checkpointing-based Rollback Recovery for Parallel Applications on the InteGrade Grid Middleware*, ACM/IFIP/USENIX 2nd International Workshop on Middleware for Grid Computing (2004).

[34] M. Cannataro, C. Comito, F. Schiavo, and P. Veltri, *PROTEUS: a Grid Based Problem Solving Environment for Bionformatics*, Workshop on DataMining Ontology for Grid Programming (KGGI 03), 2003.

[35] M. Cannataro, A. Congiusta, D. Talia, and P. Trunfio, *A Data Mining Toolset for Distributed High-Performance Platforms*, Proceedings of Data Mining 2002 (Bologna, Italy), Wessex Institute Press, 2002.

[36] H. Casanova and F. Berman, *Parameter Sweeps on the Grid with APST*, Grid Computing: Making the Global Infrastructure a Reality (T. Hey F. Berman, G. Fox, ed.), Wiley Publishers, 2003.

[37] M. Chetty and R. Buyya, *Weaving Computational Grids: How Analogous Are They with Electrical Grids?*, Computing in Science and Engg. **4** (2002), no. 4, 61–71.

[38] W. Cirne, F. Brasileiro, N. Andrade, L. Costa, A. Andrade, R. Novaes, and M. Mowbray, *Labs of the World, Unite!!!*, Journal of Grid Computing **4** (2006), no. 3, 225–246.

[39] W. Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauve, F. Silva, C. Barros, and C. Silveira, *Running Bag-of-Tasks Applications on Computational Grids: The MyGrid Approach*, Proceedings of International Conference on Parallel Processing, 2003, p. 407.

[40] *Condor: High Throughput Computing*, http://www.cs.wisc.edu/condor/, Último acesso em novembro/2006.

[41] *Condor User Manual: Parallel Applications*, http://www.cs.wisc.edu/condor/manual/v7.0/2_9Parallel Último acesso em abril/2007.

[42] *DAGman*, http://www.cs.wisc.edu/condor/dagman, Último acesso em outubro/2006.

[43] K. Davis, T. Goodale, and A. Merzky, *GAT API specification: Object based*, [online], EU Project GridLab, http://www.gridlab.org/WorkPackages/wp-1/, 2003, Deliverable D1.5.

[44] M. Department, *From Legion to Legion-G to OGSI.NET: Object-based Computing for Grids*, Proceedings of the IPDPS NSF Next Generation Software Workshop (Nice, France), April 2003.

[45] *D-Grid Initiative*, http://www.d-grid.de/, Último acesso em outubro/2006.

[46] *Distributed Resource Management Application API Working Group*, http://forge.gridforum.org/projects/drmaa-wg/, Último acesso em janeiro/2007.

[47] *Eclipse Project web site*, http://www.eclipse.org/, Último acesso em novembro/2006.

[48] D. Epema, M. Livny, R. Dantzig, X. Evers, and J. Pruyne, *A worldwide flock of Condors: load sharing among workstation clusters*, Future Generation Computer Systems **12** (1996), no. 1, 53–65.

[49] *UK E-Science Program*, http://www.rcuk.ac.uk/escience/, Último acesso em outubro/2006.

[50] Geist et al, *PVM: Parallel Virtual Machine, A User?s Guide and Tutorial for Networked Parallel Computing*, MIT Press, 1994.

[51] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard*, Tech. report, Knoxville, USA, 1994.

[52] I. Foster, *A Globus Toolkit Primer: Describing Globus Toolkit Version 4*, www.globus.org/toolkit/docs/4.0/key/GT4_Primer_0.6.pdf, 2005.

[53] I. Foster and C. Kesselman, *Globus: A metacomputing infrastructure toolkit*, The International Journal of Supercomputer Applications and High Performance Computing **11** (1997), no. 2, 115–128.

[54] I. Foster and C. Kesselman (eds.), *The Grid 2: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, 2004.

[55] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, *The physiology of the grid: An open grid services architecture for distributed systems integration*, www.globus.org/research/papers/ogsa.pdf, 2002.

[56] I. Foster, C. Kesselman, and S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, International Journal of High Performance Computing Applications **15** (2001), no. 3, 200–222.

[57] G. Fox, *Portals for Web Based Education and Computational Science*, http://citeseer.ist.psu.edu/fox00portals.html, 2000.

[58] ———, *Grid Computing Environments*, Computing in Science and Engg. **5** (2003), no. 2, 68–72.

[59] G. Fox and W. Furmanski, *High Performance Commodity Computing*, The Grid: Blueprint for a new computing infrastructure (I. Foster and C. Kesselman, eds.), Morgam Kaufman, 1999.

[60] G. Fox, M. Pierce, D. Gannon, and M. Thomas, *Overview of Grid Computing Environments*, Tech. report, Global Grid Forum, Fevereiro 2003.

[61] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, *Condor-G: A Computation Management Agent for Multi-Institutional Grids*, Cluster Computing **5** (2002), no. 3, 237–246.

[62] g Eclipse project, *g-Eclipse: An integrated, Grid enabled workbench tool for Grid application users, Grid developers and Grid operators based on the Eclipse platform*, Tech. report, 2006, http://www.geclipse.eu/.

[63] ———, *g-Eclipse Architecture I*, Tech. report, 2006.

[64] E. Gallopoulos, E. Houstis, and J. Rice, *Computer as Thinker/Doer: Problem-Solving Environments for Computational Science*, IEEE Comput. Sci. Eng. **1** (1994), no. 2, 11–23.

[65] E. Gamma and K. Beck, *Contributing to Eclipse - Principles, Patterns and Plug-Ins*, Addison-Wesley, 2004.

[66] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[67] *GAT Adaptors list*, http://www.cs.vu.nl/ibis/javagat.html, Último acesso em janeiro/2007.

[68] *Grid Development Tools for Eclipse*, http://dev.globus.org/wiki/Incubator/GDTE, Último acesso em janeiro/2007.

[69] *g-Eclipse project*, http://www.geclipse.eu, Último acesso em janeiro/2007.

[70] *g-Eclipse Eclipse project*, http://www.eclipse.org/geclipse, Último acesso em janeiro/2007.

[71] *Workflow Management Research Group - Global Grid Forum*, http://www.isi.edu/~deelman/wfm-rg/, Último acesso em dezembro/2006.

[72] *Global Grid Forum*, http://www.ggf.org, Último acesso em outubro/2006.

[73] *gLite: Ligthweight Middleware for Grid Computing*, http://glite.web.cern.ch/glite/, Último acesso em janeiro/2007.

[74] *Globus Toolkit*, http://www.globus.org, Último acesso em setembro/2006.

[75] A. Goldchleger, *InteGrade: Um Sistema de Middleware para Computação em Grade Oportunista*, Master's thesis, Instituto de Matemática de Estatística - Universidade de São Paulo, 2004.

[76] A. Goldchleger, F. Kon, A. Goldman, M. Finger, and G. Bezerra, *InteGrade: object-oriented Grid middleware leberaging idle computing power of desktop machines*, Concurrency and Computation: Practice & Experience **16** (2004), 449–459.

[77] A. Goldchleger, C. Queiroz, F. Kon, and A. Goldman, *Running Highly-Coupled Parallel Applications in a Computational Grid* , Proceedings of the 20th Brazilian Symposium on Computer Networks, 2004.

[78] T. Goodale, G. Allen, G. Lanfermann, J. Massó, T. Radke, E. Seidel, and J. Shalf, *High performance computing for computational science - vecpar 2002: 5th international conference*, ch. The Cactus Framework and Toolkit: Design and Applications, pp. 197–227, Springer Berlin / Heidelberg, 2003.

[79] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, G. Laszewski, C. Lee, A. Merzky, H. Rajic, and J. Shalf, *SAGA: A Simple API for Grid Applications, High-Level Application Programming on the Grid*, Computational Methods in Science and Technology (2006).

[80] T. Goodale, S. Jha, T. Kielmann, A. Merzky, J. Shalf, and C. Smith, *A Simple API for Grid Applications (SAGA)*, Tech. report, Open Grid Forum, September 2006, versão: 1.0 RC 1.

[81] *Grid Portal Development Kit (GPDK)*, http://doesciencegrid.org/projects/GPDK/, Último acesso em dezembro/2006.

[82] *Grid Computing Info Centre*, http://www.gridcomputing.com/, Último acesso em janeiro/2007.

[83] *Grid Checkpoint Recovery WG*, http://forge.gridforum.org/projects/gridcpr-wg, Último acesso em janeiro/2007.

[84] *Gride web site*, http://apstc.sun.com.sg/content.php?l1=research, Último acesso em outubro/2006.

[85] *GridFTP*, http://www.globus.org/toolkit/data/gridftp/, Último acesso em janeiro/2007.

[86] *GridLab: A Grid Application Toolkit and Testbed*, http://www.gridlab.org/, Último acesso em janeiro/2007.

[87] *GridPort*, https://gridport.npaci.edu/, Último acesso em dezembro/2006.

[88] *Grid RPC Working Group*, http://forge.gridforum.org/projects/gridrpc-wg/, Último acesso em janeiro/2007.

[89] *GridSphere portal framework*, http://www.gridsphere.org, Último acesso em janeiro/2007.

[90] A. Grimshaw and W. Wulf, *Legion: The next logical step toward the world-wide virtual computer*, Communications of the ACM **40** (1997), no. 1, 39–45.

[91] *GT4IDE web site*, http://gsbt.sourceforge.net/content/view/12/29/, Último acesso em dezembro/2006.

[92] *GVid - Grid Visualization*, http://www.gup.jku.at/gvid/, Último acesso em janeiro/2007.

[93] I. Foster, *Globus Toolkit Version 4: Software for Service-Oriented Systems*, vol. 3779, IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS, 2006, pp. 2–13.

[94] *IC2D: Interactive Control and Debugging of Distribution*, http://www-sop.inria.fr/oasis/ProActive/IC2D/index.html, Último acesso em janeiro/2007.

[95] *ProActive manual: Chapter 46. Adding Grahical User Interfaces and Eclipse Plug-ins*, http://www-sop.inria.fr/oasis/ProActive/doc/release-doc/html/newIC2D.html, Último acesso em janeiro/2007.

[96] *ILab: Parameter Study Creation and Submission on the Infomation Power Grid*, http://www.nas.nasa.gov/sc2000/ARC/ilab.html, Último acesso em novembro/2006.

[97] *Java Community Process*, http://www.jcp.org/, Último acesso em novembro/2006.

[98] *JOpera for Eclipse*, http://www.jopera.org, Último acesso em janeiro/2007.

[99] Jose Ribamar Braga Pinheiro Jr., *Xenia: um sistema de segurança para grades computacionais baseado em cadeias de confiança*, Ph.D. thesis, Departamento de Ciência da Computação do IME/USP.

[100] *JSR 168: Portlet Specification*, http://www.jcp.org/en/jsr/detail?id=168, Último acesso em outubro/2006.

[101] P. Kacsuk, G. Dozsa, and Tibor Fadgyas, *Designing parallel programs by the graphical language GRAPNEL*, Microprocess. Microprogram. **41** (1996), no. 8-9, 625–643.

[102] P. Kacsuk, G. Dózsa, J. Kovács, R. Lovas, N. Podhorszki, Z. Balaton, and G. Gombás, *P-GRADE: A Grid Programming Environment*, Journal of Grid Computing **1** (2003), no. 2, 171–197.

[103] P. Kacsuk, Z. Farkas, G. Sipos, A. Toth, and G. Hermann, *Workflow-level Parameter Study Management in multi-Grid environments by the P-GRADE Grid portal*, Proceedings of the second International Workshop on grid Computing Environments, held in conjunction with SuperComputing (Tampa, Florida, USA), 2006.

[104] H. Kaiser, A. Merzky, S. Hirmer, and G. Allen, *The SAGA C++ Reference Implementation - Lessons Learnt from Juggling with Seemingly Contradictory Goals*, Proceedings of the Second International Workshop on Library-Centric Software Design (LCSD '06), held in conjunction with OOPSLA'06, October 2006.

[105] N. Karonis, B. Toonen, and I. Foster, *MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface*, Journal of Parallel and Distributed Computing **63** (2003), no. 5, 551–563.

[106] T. Kielmann, *Programming Models for Grid Applications and Systems: Requirements and Approaches*, IEEE John Vincent Atanasoff International Symposium on Modern Computing (JVA 2006) **0** (2006), 27–32.

[107] T. Kielmann, A. Merzky, H. Bal, F. Baude, D. Caromel, and F. Huet, *Future Generation Grids*, ch. Grid Application Programming Environments, pp. 283–306, Springer US, 2006.

[108] Harald Kornmayer, *Uso da API SAGA no projeto g-Eclipse*, comunicação privada, 2006.

[109] S. Krishnan and D. Gannon, *XCAT3: A Framework for CCA Components as OGSA Services*, Ninth International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS'04), 2004, pp. 90–97.

[110] G. Laszewski, I. Foster, J. Gawor, and P. Lane, *A Java commodity grid kit*, Concurrency and Computation: Practice and Experience **13** (2001), no. 645-662.

[111] G. Laszewski, J. Gawor, P. Lane, N. Rehn, M. Russell, and K. Jackson, *Features of the Java Commodity Grid Kit*, Concurrency and Computation: Practice and Experience **14** (2002), 1045–1055.

[112] G. Laszewski, G. Pieper, and P. Wagstrom, *Gestalt of the Grid*, Tools and Environments for Parallel and Distributed Computing (eu, ed.), Wiley, 2004.

[113] R. Lavender and D. Schmidt, *Active Object: an Object Behavioral Pattern for Concurrent Programming*, Proceedings of the Pattern Languages of Programs, (1995).

[114] *Legion: A Worldwide Virtual Computer*, http://www.cs.virginia.edu/~legion, Último acesso em novembro/2006.

[115] F. Leyman, *Web Services Flow Language (WSFL) 1.1*, Tech. report, IBM Software Group, 2001, Technical report.

[116] M. Li and M. Baker, *A Review of Grid Portal Technology*, Grid Computing: Software Environments and Tools. (J. Cunha and O. Rana, eds.), Springer Verlag, 2004.

[117] M. Li, P. Santen, D. Walker, O. Rana, and M. Baker, *PortalLab: A Web Services Toolkit for Building Semantic Grid Portals*, CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid (Washington, DC, USA), IEEE Computer Society, 2003, p. 190.

[118] N. Linde, U. Kuester, M. Resch, and B. Risio, *Science Experimental Grid Laboratory (SEGL) Dynamical Parameter Study in Distributed Systems*, Proceedings of the 2005 International Conference on Parallel Computing (ParCo 2005) (Malaga, Spain), 2005, pp. 49–56.

[119] M. Litzkow, *Remote Unix - Turning Idle Workstations into Cycle Servers*, Proceedings of the Usenix Summer Conference, 1987, pp. 381–384.

[120] M. Litzkow, T. Tannenbaum, J. Basney, and M. Livny, *Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System*, Tech. Report UW-CS-TR-1346, University of Wisconsin - Madison Computer Sciences Department, 1997.

[121] M. Livny, J. Basney, R. Raman, and T. Tannenbaum, *Mechanisms for High Throughput Computing*, Speedup Journal **11(1)** (1997).

[122] M. Li and M. Baker, *Grid Portals*, The Grid: Core Technologies, Paperback, 2005.

[123] _____, *Workflow Management for the Grid*, The Grid: Core Technologies, Paperback, 2005.

[124] *The Marburg Ad-hoc Grid Environment*, http://mage.uni-marburg.de/, Último acesso em janeiro/2007.

[125] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard*, Tech. Report UT-CS-94-230, 1994.

[126] T. M. Mitchell, *Machine learning*, McGraw-Hill, 1997.

[127] *Netbeans Project web site*, http://www.netbeans.org/, Último acesso em novembro/2006.

[128] R. Nieuwpoort, T. Kielmann, and H. Bal, *Efficient load balancing for wide-area divide-and-conquer applications*, PPoPP '01: Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming (New York, NY, USA), ACM Press, 2001, pp. 34–43.

[129] R. Nieuwpoort, J. Maassen, A. Agapi, A. Oprescu, and T. Kielmann, *Experiences Deploying Parallel Applications on a Large-scale Grid*, EXPGRID - Experimental Grid testbeds for the assessment of large-scale distributed applications and tools, workshop in conjunction with the 15th International Symposium on High Performance Distributed Computing (HPDC-15), June 2006.

[130] R. Nieuwpoort, J. Maassen, G. Wrzesiska, R. Hofman, C. Jacobs, T. Kielmann, and H. Bal, *Ibis: a flexible and efficient Java-based Grid programming environment: Research Articles*, Concurrency and Computation: Practice and Experience **17** (2005), no. 7-8, 1079–1107.

[131] J. Novotny, M. Russell, and O. Wehrens, *GridSphere: a portal framework for building collaborations*, Concurrency and Computation: Practice and Experience **16** (2004), no. 5, 503–513.

[132] *Organization for the Advancement of Structured Information Standards(OASIS)*, http://www.oasis-open.org/, Último acesso em janeiro/2007.

[133] *OGSA Working Group*, http://forge.gridforum.org/projects/ogsa-wg, Último acesso em janeiro/2007.

[134] OMG, *CORBA v3.0 Specification*, Tech. report, Object Management Group, 2002, OMG Document 02-06-33.

[135] *MyGrid/OurGrid*, http://www.ourgrid.org/, Último acesso em novembro/2006.

[136] M. Parashar and C. Lee (eds.), *Proceedings of the IEEE*, vol. 93, Março 2005, Special Issue on Grid Computing.

[137] *Perl*, http://www.perl.com/pub/a/2006/01/12/what_is_perl_6.html, Último acesso em outubro/2006.

[138] S. Pickles, R. Haines, R. Pinning, and et al., *A practical toolkit for computational steering*, Royal Society of London Philosophical Transactions Series A **363** (2005), 1843–1853.

[139] *ProActive: Programming, Composing and Deploying on the Grid*, http://proactive.inria.fr/, Último acesso em outubro/2006.

[140] *A distributed and collaborative 3D renderer*, Último acesso em janeiro/2007, http://www-sop.inria.fr/oasis/ProActive/apps/c3d.html.

[141] *The dinning philosophers*, http://www-sop.inria.fr/oasis/ProActive/apps/phil.html, Último acesso em janeiro/2007.

[142] R. Raman, M. Livny, and M. Solomon, *Matchmaking: Distributed resource management for high throughput computing*, HPDC '98: Proceedings of the The Seventh IEEE International Symposium on High Performance Distributed Computing (Washington, DC, USA), IEEE Computer Society, 1998, p. 140.

[143] M. Russell, J. Novotny, and O. Wehrens, *The Grid Portlets Web Application: A Grid Portal Framework*, Tech. report, GridSphere project, 2005.

[144] A. Schreiber, *The Integrated Simulation Environment TENT*, Concurrency and Computation: Practice and Experience **14** (2002), no. 13-15, 1553 – 1568.

[145] S. See, J. Song, L. Peng, A. Stoelwinder, and H. Neo, *GriDE: A Grid Enabled Development Environment*, Second International Workshop on Grid and Cooperative Computing (Shanghai, China), 2003, pp. 7–12.

[146] *SETI@home*, http://setiathome.berkeley.edu/, Último acesso em dezembro/2006.

[147] K. Seymour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee, and H. Casanova, *Overview of GridRPC: A Remote Procedure Call API for Grid Computing*, GRID '02: Proceedings of the Third International Workshop on Grid Computing (London, UK), Springer-Verlag, 2002, pp. 274–278.

[148] R. Sirventa, A. Merzky, R. Badia, , and T. Kielmann, *GRID superscalar and SAGA: forming a high-level and platform-independent Grid programming environment*, CoreGRID integration workshop. Integrated Research in Grid Computing, November 2005.

[149] A. Slominski and G. Laszewski, *Scientific Workflows Survey*, http://www.extreme.indiana.edu/swf-survey/, Último acesso em dezembro/2006.

[150] L. Smarr, *Infrastructures for science portals*, IEEE Internet Computing **4** (2000), no. 1, 71–73.

[151] *Sun Grid Engine*, http://gridengine.sunsource.net/, Último acesso em janeiro/2007.

[152] I. Taylor, M. Shields, I. Wang, and O. Rana, *Triana Applications within Grid Computing and Peer to Peer Environments*, Journal of Grid Computing **1** (2003), no. 2, 199–217.

[153] Triana Team, *Triana User Guide*, Último acesso em dezembro/2006, https://forge.nesc.ac.uk/docman/view.php/33/104/UserGuide.pdf.

[154] D. Thain, T. Tannenbaum, and M. Livny, *Distributed computing in practice: The condor experience*, Concurrency and Computation: Practice and Experience (2004).

[155] *Graphical User Interfaces with Tk*, http://docs.python.org/lib/tkinter.html, Último acesso em outubro/2006.

[156] *UML 2.1.1, The Current Official Version*, http://www.uml.org/, Último acesso em janeiro/2007.

[157] *Uniform Interface to Computing Resources*, http://www.unicore.eu, Último acesso em janeiro/2007.

[158] L. Valiant, *A bridging model for parallel computation*, Commununication of the ACM **33** (1990), no. 8, 103–111.

[159] G. Watson, *Eclipse parallel tools platform design document*, Tech. report, Eclipse comunity, abril 2005, versão: 0.2.0.

[160] B. White, M. Walker, M. Humphrey, and A. Grimshaw, *LegionFS: a secure and scalable file system supporting cross-domain high-performance applications*, Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM) (New York, USA), ACM Press, 2001, pp. 59–59.

[161] D. Wright, *Cheap Cycles from the Desktop to the Dedicated Cluster: Combining Opportunistic and Dedicated Scheduling with Condor*, Proceedings of Linux Clusters: The HPC Revolution (Champaign-Urbana, USA), 2001.

[162] *The Web Service Resource Framework*, http://www.globus.org/wsrf/, Último acesso em dezembro/2006.

[163] M. Yarrow, K. McCann, R. Biswas, and R. Wijngaart, *An Advanced User Interface Approach for Complex Parameter Study Process Specification on the Information Power Grid*, GRID '00: Proceedings of the First IEEE/ACM International Workshop on Grid Computing (London, UK), Springer-Verlag, 2000, pp. 146–157.