UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

CLARISSA CASSALES MARQUEZAN

Integrated Cluster Environmnet (ICE) - Plataforma de Gerenciamento e de Acesso a Múltiplos Clusters

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Philippe O. A. Navaux
Orientador

Prof. Dr. Alexandre da Silva Carissimi
Co-orientador

Porto Alegre, agosto de 2006

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Marquezan, Clarissa Cassales

Integrated Cluster Environmnet (ICE) - Plataforma de Gerenciamento e de Acesso a Múltiplos Clusters / Clarissa Cassales Marquezan. – Porto Alegre: PPGC da UFRGS, 2006.

97 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2006. Orientador: Philippe O. A. Navaux; Co-orientador: Alexandre da Silva Carissimi.

1. Múltiplos clusters. 2. Gerenciamento. 3. Extensibilidade. 4. Transparência. 5. Interoperabilidade. 6. Web Services. I. Navaux, Philippe O. A.. II. Carissimi, Alexandre da Silva. III. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. José Carlos Ferraz Hennemann
Pró-Reitor de Coordenação Acadêmica: Prof. Pedro Cezar Dutra Fonseca
Pró-Reitora de Pós-Graduação: Prof^{a}. Valquíria Linck Bassani
Diretor do Instituto de Informática: Prof. Flávio Rech Wagner
Coordenador do PPGC: Prof. Carlos Alberto Heuser
Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

"Um amigo é alguém que sabe tudo a seu respeito e gosta de você assim mesmo." ELBERT HUBBARD

AGRADECIMENTOS

Em primeiro lugar quero agradecer à minha família pelo apoio irrestrito, incentivo e vibração com as conquistas que tive até hoje e tenho certeza que ela vai estar sempre comigo. Também agradeço pela paciência e compreensão que tiveram nos momentos em que não pude estar junto. Em especial agradeço às minhas irmãs, Luciana, Anna Lucia e Gabriela, ao meu pai e à minha mãe, vocês são realmente muito importantes pra mim.

Agradeço ao meu orientador, prof. Philippe Olivier Alexandre Navaux, com quem trabalho em pesquisa desde a iniciação científica. Quero agradecer pelo investimento que fez em mim, por ter sempre me incentivado, mostrado os melhores caminhos a seguir para atingi os objetivos. Agradeço todo o conhecimento que o prof. Navaux me ajudou a formar, não somente conhecimento acadêmico, mas também o conhecimento de vida, de como interagir com as pessoas, de como conduzir a pesquisa, de como motivar e incentivar as pessoas a evoluírem.

Também agradeço ao meu co-orientador, prof. Alexandre Carissimi da Silva, com quem comecei a trabalhar em pesquisa no mestrado. Um agradecimento por ter sempre me lembrado de manter o foco no meu trabalho de mestrado, de não dispersar minha atenção com trabalhos paralelos. Essas recomendações foram realmente muito importantes para mim. Agradeço também pelas discussões técnicas, pelas correções de rumo do trabalho e pelo incentivo.

Existem pessoas que participaram ativamente do meu trabalho de mestrado, as quais não posso deixar de agradecer: Lucas Schnorr, Rodrigo da Rosa Righi, Nicolas Maillard e Alexandre Ilha. As discussões que tivemos, as trocas de idéias foram muito importantes para o amadurecimento e para a conclusão deste trabalho. Ainda quando agiram como 'advogados do diabo' foram perfeitos, pois me fizeram correr atrás de respostas e de soluções.

Agradeço também a uma pessoa que não contribuiu diretamente no meu trabalho de mestrado, mas as oportunidades que ele me proporcionou contribuíram a aquisição e utilização de conhecimento nesta dissertaçaõ. Muito obrigada ao prof. Lisandro Zanbenedetti Granville.

Não posso deixar de agradecer a compreensão do meu chefe na Divisão de Redes e Suporte (DRS) do CPD da UFRGS, Leandro Rey, possibilitando-me uma grande flexibilidade de horário para que eu pudesse terminar a dissertação. Agradeço também, aos colegas de trabalho na DRS, pelo incentivo.

Aos amigos Rafael Ennes Silva e Priscilla Kurtz um agradecimento pelo companheirismo durante os vários finais de semana, noites e madrugadas que passamos na sala 209 terminando nossas dissertações e trabalho de conclusão, no caso da Priscilla. Foram dias e noites de grande trabalho, mas também de muita diversão.

Agradeço aos amigos 99, principalmente o Diego Contessa (Caçula) e o Daniel Gaspary (Chewie), por sempre me motivarem, seja ao vivo ou pelo ICQ ou MSN. Aos amigos que estão longe, muitos em outros continentes (Karina Roggia e Tiago Fiorezi), agradeço pelo carinho e também pelo incentivo remoto. Aos amigos das áureas épocas do Labcom pelas conversas, momentos divertidos e pela amizade.

Aos meus amigos, ou melhor, à minha família do Coral Procergs, pela paciência e compreensão nos momentos que tive de me afastar dos ensaios para terminar a dissertação. Quero que vocês saibam que participar deste grupo me dá motivação para a vida e para o trabalho. Vocês são muito importantes para mim.

Agradeço o incentivo, algumas vezes não ortodoxo, dos companheiros e amigos da sala 209, os churrascos, os momentos de descontração em bares conhecidos da cidade de Porto Alegre, e no famoso bar perto do Campus do Vale. E Hermann, eu terminei a dissertação antes que tu! (Existiam apostas de que eu não terminaria antes dele, hehehe.)

Também agradeço aos amigos de pedaladas por me proporcionarem passeios para distrair um pouco a mente dos percalços do trabalho.

Não poderia deixar de agradecer a compreensão, amizade e incentivo dos meus amigos de Rosário do Sul. Em especial às minhas amigas Ana Paula e Maria José que, por muitas vezes, não pude estar junto delas, mas que sabem que a nossa amizade está acima da presença física.

Enfim, existem muitas pessoas importantes na minha vida e que contribuíram para a minha formação e vida pessoal. A todas elas, não citadas nominalmente, agradeço com a mesma intensidade e vibração.

Muito obrigada!!!

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS 8

LISTA DE FIGURAS 10

LISTA DE TABELAS 12

RESUMO 13

ABSTRACT 14

1 INTRODUÇÃO 15
1.1 Objetivos 16
1.2 Organização 16

2 ESTADO DA ARTE EM GERENCIAMENTO DE SISTEMAS DE ALTO DESEMPENHO 17
2.1 Sistemas Distribuídos 17
2.2 Sistemas de Alto Desempenho 19
2.2.1 *Clusters* 21
2.2.2 *Grids* 24
2.2.3 Intersecção entre *clusters* e *grids*: sistemas Multicluster 26
2.3 Resumo 28

3 PROPOSTA DE AMBIENTE INTEGRADO PARA CLUSTERS: ICE 29
3.1 ICE como um Sistema Distribuído 31
3.2 Arquitetura ICE 32
3.2.1 *Middleware* de Serviços 34
3.2.2 Portal Web 35
3.2.3 Cenário de Aplicação da Arquitetura ICE 37
3.3 Resumo 38

4 DESENVOLVIMENTO DO PROTÓTIPO DO AMBIENTE ICE . . . 39
4.1 Decisões de Projeto 39
4.1.1 Definição do *Middleware* Empregado 39
4.1.2 Análise dos Padrões e Especificações 41
4.2 Descrição do Protótipo Implementado 43
4.2.1 Tecnologias Empregadas 44
4.2.2 Middleware de Serviço para Gerenciamento de Aplicações 45
4.2.3 Portal Web 60

4.3 Resumo 69

5 AVALIAÇÃO DO PROTÓTIPO ICE 71
5.1 Avaliação Qualitativa 71
5.1.1 Metodologia 71
5.1.2 Comparação entre HPC2N, M3C e ICE 72
5.2 Avaliação Quantitativa 74
5.2.1 Definição do escopo das medições 74
5.2.2 Metodologia 75
5.2.3 Plataforma 76
5.2.4 Experimentos com a operação de submissão de aplicações 77
5.2.5 Experimentos com a operação de visualização de *scripts* de submissão 80
5.2.6 Experimentos com a operação de finalização de aplicações 82
5.2.7 Experimentos com a operação de visualização de *status* 83
5.2.8 Experimentos com a operação de recuperação das saídas padrão . . . 83
5.3 Resumo 84

6 CONCLUSÃO 85
6.1 Trabalhos em Paralelo 87
6.2 Trabalhos Futuros 88

REFERÊNCIAS 89

LISTA DE ABREVIATURAS E SIGLAS

C^3	Cornell Checkpoint (pre)Compiler
C3	Cluster Command and Control
CORBA	Commom Object Request Broker Architecture
DCOM	Distributed Component Object Model
ER	Entidade Relacionamento
GGF	Global Grid Forum
GMA	Grid Monitoring Architecture
GPPD	Grupo de Processamento Paralelo e Distribuído
GRAM	Grid Resource Allocation and Management
GSI	Grid Security Infrastructure
GT	Globus Toolkit
HPC	High Performance Computing
HPC2N	High Performance Computing Center North
HTTP	Hypertext Transfer Protocol
ICE	Integrated Cluster Environment
IOR	Internet-wide Object Reference
JSP	Java Server Pages
M3C	Managing and Monitoring Multiple Clusters
MPP	Massive Parallel Processing
MTBF	MeanTime Between Failures
MVC	Model View Controller
OASIS	Organization for the Advancement of Structured Information Standards
OGSA	Open Grid Service Architecture
OGSI	Open Grid Service Infrastructure
OpenPBS	Open Portable Batch System
OpenSCE	Open Scalable Cluster Environment

ORB	Object Request Broker
OSCAR	Open Source Cluster Application Resource
P2P	Peer-to-Peer
PDA	Personal Digital Assistants
PUNCH	Purdue University Network Computing Hubs
QAME	QoS-Aware Management Environment
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SCMS	Smile Cluster Management System
SGBD	Sistema de Gerenciamento de Base de dados
SGE	Sun Grid Engine
SI	Service Implementation
SM	Service Module
SMM	System Management Module
SNMP	Simple Network Management Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
UDDI	Universal Description Discovery and Integration
URI	Identificador Universal de Recursos
USI	Unified Service Interface
VPN	Virtual Private Network
W3C	World Wide Web Consortium
WS	Web Services
WSDL	Web Services Description Language
WSGRAM	Web Services Grid Resource Allocation and Management

LISTA DE FIGURAS

Figura 2.1: Lista de aplicações existentes nas máquinas que fazem parte do Top 500 20

Figura 3.1: Visão geral do cenário de utilização do ICE 32
Figura 3.2: Arquitetura ICE 33
Figura 3.3: Instanciação da Arquitetura ICE 37

Figura 4.1: Arquitetura do Globus Toolkit v4 - Figura retirada do *draft* (GLOBUS Toolkit 4.0 RELEASE MANUALS, 2005) 42
Figura 4.2: Protótipo ICE Implementado 44
Figura 4.3: JMF - Job Management Framework 51
Figura 4.4: Diagrama de classes para o pacote de classes auxiliares 52
Figura 4.5: Diagrama de classes para as implementações do OpenPBS-SI e OAR-SI 54
Figura 4.6: Elementos XML da requisição das operações de submissão e visualização de *scripts* 55
Figura 4.7: Exemplo de retorno da operação de submissão. (a) Submissão com OpenPBS (b) Submissão com OAR 55
Figura 4.8: Exemplo de retorno da operação de submissão contendo erro. . . 56
Figura 4.9: Elementos XML de retorno da operação de visualização de *scripts*. 56
Figura 4.10: Exemplo de retorno da operação de visualização em um *cluster* com OpenPBS. 57
Figura 4.11: Exemplo de retorno da operação de visualização em um *cluster* com OAR. 57
Figura 4.12: Elementos XML de requisição e retorno da operação de finalização. 58
Figura 4.13: Exemplo de retorno da operação de verificação de estado das filas do gerenciador OAR. 59
Figura 4.14: Elementos XML da requisição e resposta da operação de recuperação de arquivos de saída. 60
Figura 4.15: Diagrama de classes do JM-SM 61
Figura 4.16: Diagrama Entidade Relacionamento (ER) do ambiente ICE . . . 62
Figura 4.17: *Snapshot* Portal Web - Configuração de operações 63
Figura 4.18: *Snapshot* Portal Web - Página principal da configuração de grupos 64
Figura 4.19: *Snapshot* Portal Web - Associação de *clusters* a grupos 64
Figura 4.20: *Snapshot* Portal Web - Associação de funcionalidades a grupos . . 65
Figura 4.21: *Snapshot* Portal Web - Associação de funcionalidades a *clusters* . 66
Figura 4.22: *Snapshot* Portal Web - Página principal do menu *Job Management* 67
Figura 4.23: *Snapshot* Portal Web - Página de submissão de aplicações 67

Figura 4.24: ER da funcionalidade de *Job Management* 68
Figura 4.25: Exemplo 1 de visualização de *scripts* de submissão 68
Figura 4.26: Exemplo 2 de visualização de *scripts* de submissão 69

Figura 5.1: Cenário de aquisição das medidas de *overhead* 75
Figura 5.2: Gráfico Operação de Submissão - OpenPBS 77
Figura 5.3: Gráfico Operação de Submissão - OAR 78
Figura 5.4: Gráfico Operação de Visualização de Scripts - OpenPBS 80
Figura 5.5: Gráfico Operação de Visualização de Scripts - OAR 81

LISTA DE TABELAS

Tabela 3.1: Descrição dos perfis atualmente definidos no ambiente ICE 36

Tabela 4.1: Comparação entre CORBA e Web Services 40
Tabela 4.2: Operações da JM-USI 46

Tabela 5.1: Comparação entre o ambiente ICE e algumas ferramentas relacionadas 73
Tabela 5.2: Descrição das variações de parâmetros para submissão de aplicações. 76
Tabela 5.3: Descrição do *cluster* utilizado para os experimentos 77
Tabela 5.4: Desvio padrão para operação de submissão 79
Tabela 5.5: Desvio padrão para operação de verificação de *scripts* 81
Tabela 5.6: Desempenho da operação de finalização de aplicações 82
Tabela 5.7: Desempenho da operação de verificação do *status* das aplicações . 83
Tabela 5.8: Desempenho da operação de recuperação das saídas padrão 84

RESUMO

Facilitar o gerenciamento e o acesso a sistemas de alto desempenho vem sendo uma área de pesquisa explorada nos últimos anos. Isto acontece uma vez que se verifica o aumento do número de usuários, muitos pertencentes a outras áreas, como por exemplo, biologia, geologia, hidrologia, etc e, desta forma, é preciso facilitar os meios de interação destes usuários com tais sistemas, assim como melhorar as técnicas de gerenciamento dos mesmos. Além do aumento do número e da multidisciplinaridade desses usuários, existe também o fato de que grande parte deles tem acesso a diferentes tipos de sistemas de alto desempenho. Em geral, estes sistemas possuem ferramentas não padronizadas, sendo que cada uma apresenta uma interface e um conjunto de ações e comandos a serem realizados para que possam dispor de suas funcionalidades. Este trabalho propõe a definição de uma arquitetura de gerenciamento e acesso a múltiplos *clusters*, que seja capaz de ser facilmente extensível, transparente, interoperável e de fácil utilização, configuração e manutenção. Como resultado da arquitetura proposta, foi desenvolvido um protótipo, denominado ICE - *Integrated Cluster Environment*. Os principais objetivos da arquitetura e do ambiente ICE são: (i) capacidade de uniformização do modo como as ferramentas de *clusters* são utilizadas e, também, uniformização na maneira como os *clusters* são acessados; (ii) transparência na forma de acesso e uso dos *clusters*; e (iii) capacidade de extensibilidade em dois níveis: o primeiro refere-se à extensibilidade do número de funcionalidades (serviços) providos pelo sistema e o segundo está relacionado à capacidade do sistema lidar com o uso de diferentes ferramentas que possuem a mesma funcionalidade. Além da descrição da arquitetura e do protótipo, neste trabalho, também é realizada uma avaliação do ambiente ICE. Essa avaliação foi dividida em dois momentos. O primeiro traz a comparação das características do ambiente proposto com algumas ferramentas relacionadas. No segundo momento são apresentados alguns experimentos que visam identificar o *overhead* inserido pelo ambiente ICE na execução das operações do módulo de gerenciamento de aplicações, desenvolvido e descrito neste trabalho.

Palavras-chave: Múltiplos clusters, gerenciamento, extensibilidade, transparência, interoperabilidade, Web Services.

Integrated Cluster Environmnet (ICE): Management and Access Platform for Multiple Clusters

ABSTRACT

Some researches have been done over the last years in order to improve the management and access of high performance systems. One of the motivations of these researches is the continuously increase in the number of users who, not rarely, belong to other areas, such as biology, geology, hydrology, etc; so it is necessary to provide some access and also management facilities in these kinds of systems. Furthermore, the users also have access to different kinds of high performance systems, and they have to deal with distinct tools of their underlying, which are not standardize. Therefore, the users need to learn the specificities of each tool in each high performace system that they have permission to access. Aiming to solve this problem, this work proposes an architecture to provide access and management of multiple clusters with extensibility, transparance, interoperability, user-friendly, manageability and maintainability. As a result of the proposed architecture, a prototype called ICE - Integrated Cluster Environment - was developed. The main goals of the architecture and the ICE environment are: (i) the capability of uniforming the manner the cluster tools are used and accessed; (ii) cluster access and transparence use; and (iii) two extensibility levels: the first one refers to capability of extending the number of functionalities/services provided by the environment and the second one concerns to the capability of extending the number of tools, with the same functionality, the system is able to manage. Beyond the architecture and prototype description, this research presents the evaluation of ICE environment, which is divided in two parts. The first one brings the comparison of the features between ICE and some related work. The second part shows some experiments that intend to verify the overhead inserted by ICE environment when performing the tasks of the job management module, also developed and explained in this work.

Keywords: multiple clusters, management, extensibility, transparence, interoperability, Web Services.

1 INTRODUÇÃO

Os sistemas distribuídos estão presentes em quase todas as ações realizadas nos tempos atuais. Desde a operação de verificar o saldo bancário, o pagamento de uma conta através do cartão de crédito, até a verificação da previsão do tempo (a qual é atingida através do uso de sistemas de alto desempenho). Sistemas distribuídos compreendem uma gama muito grande de cenários, como por exemplo: a Internet, intranets, redes locais de universidades e empresas, redes sem fio etc. No escopo deste trabalho serão considerados os sistemas distribuídos cujo foco é a obtenção de melhor desempenho das aplicações. Tais sistemas são chamados de Sistemas de Alto Desempenho (HPC - High Performance Computing). Em geral são formados por máquinas de redes locais, que podem ou não estar ligadas a outras redes.

Cada vez mais, o gerenciamento desse tipo de sistema torna-se uma tarefa complexa, que requer um número maior de variáveis e ferramentas. Pode-se dizer que dentro da área de HPC existem três grandes áreas de utilização: *clusters*, *grids* e múltiplos *clusters*. Para cada uma destas áreas existem diversas ferramentas para gerenciamento. Algumas delas são: OSCAR (LIGNERIS; et al., 2003; OSCAR - OPEN SOURCE CLUSTER APPLICATION RESOURCES, 2005), PUNCH (ADABALA; KAPADIA; FORTES, 2000; PARK; et al., 2000), ROCKS (PAPADOPOULOS; KATZ; BRUNO, 2001), OpenSCE (UTHAYOPAS; ANGSKUN; MANEESILP, 2001, 2002; OPENSCE - SCALABLE CLUSTER ENVIRONMENT, 2005), ferramenta de gerenciamento de *clusters* através de SNMP (ALVES et al., 2004; ALVES; MARQUEZAN; GRANVILLE, 2004; ALVES et al., 2005), entre outras no cenário de *clusters*; e Globus (FOSTER; KESSELMAN, 1997; GLOBUS - WELCOME TO THE GLOBUS TOOLKIT HOMEPAGE, 2005), Condor (GONZ; et al., 2002; THAIN; TANNENBAUM; LIVYN, 2003), GridRM (BAKER; SMITH, 2003), GridSphere (NOVOTNY; RUSSELL; WEHRENS, 2004), entre várias outras no escopo de gerenciamento de *grids*. No cenário de gerenciamento de múltiplos *clusters* não existe a mesma variedade de ferramentas como encontrado nos demais cenários. Exemplos de ferramentas voltadas para o gerenciamento de múltiplos *clusters* são: M3C (BRIM; et al., 2001) e HPC2N (ELMROTH; NYLÉN; OSCARSSON, 2005). Também identifica-se que as ferramentas que compõem este último cenário não apresentam características importantes de sistemas distribuídos, como por exemplo: transparência de acesso, extensibilidade, uniformização das operações, completeza, interoperabilidade, capacidade de incorporação dos sistemas de gerenciamento legados e construção de ferramentas amigáveis para os usuários.

Dadas as características de sistemas distribuídos ainda não disponibilizadas nas ferramentas de gerenciamento de múltiplos *clusters*, propõe-se o ambiente ICE - *Integrated Cluster Environment*.

1.1 Objetivos

Os objetivos deste trabalho são: (i) definição de uma arquitetura de gerenciamento e acesso a múltiplos *clusters* que seja capaz de ser facilmente extensível, transparente, interoperável e de fácil utilização, configuração e manutenção; (ii) implementação de um protótipo seguindo esta arquitetura; e (iii) validação das idéias contidas na arquitetura e no protótipo implementado. A principal questão do trabalho desenvolvido é permitir que os usuários lidem com as funcionalidades dos *clusters*, como por exemplo gerenciamento de aplicações e monitoramento de recursos, em alto nível, enquanto as especificidades de cada ferramenta serão abstraídas e providas pela camada definida no trabalho. Uma alternativa seria a utilização das ferramentas de *grid* para prover essas abstrações. Entretanto, essas ferramentas inserem um alto grau de complexidade, e dependendo do tipo de ferramenta de *grid* empregada a execução de aplicações verdadeiramente paralelas, e não do tipo *bag of tasks*, podem ser afetadas pela forma como elas são distribuídas nos recursos.

No contexto do trabalho que é apresentado nesta dissertação defende-se a idéia de trazer os conceitos da área de *grid*, como por exemplo arquiteturas orientadas a serviços e interoperabilidade, para a área de múltiplos *clusters*, mantendo a simplicidade do cenário e provendo as facilidades que as ferramentas de *grid* disponibilizam para os usuários e administradores.

1.2 Organização

Este trabalho está organizado como descrito a seguir.

O Capítulo 2 possui o levantamento do estado da arte do gerenciamento de sistemas distribuídos, mais especificamente focando em sistemas de alto desempenho (HPC - *High Performance Computing*). Neste escopo serão apresentados três grandes áreas: *clusters*, *grids* e múltiplos *clusters*. Para cada uma dessas áreas serão apresentados os cenários de utilização os problemas de gerenciamento e algumas ferramentas para lidar com esses problemas.

O Capítulo 3 apresenta a proposta de desenvolvimento do ambiente ICE. Um ambiente voltado para o gerenciamento e acesso de múltiplos *clusters*. Suas principais características são transparência, extensibilidade e interoperabilidade. Seu escopo de utilização e arquitetura são descritos nas subseções desse capítulo.

No Capítulo 4 é feita a descrição do protótipo implementado com base nas definições e na arquitetura do ambiente ICE, propostos no capítulo anterior. Aqui são detalhados os componentes do ambiente, seus relacionamentos, o modelo de informação e, como resultado final, são apresentadas telas do protótipo ICE.

O Capítulo 5 contém a avaliação do ambiente proposto e da implementação do protótipo. Este capítulo está dividido em dois momentos. Primeiramente é realizada uma comparação entre o ambiente ICE e algumas ferramentas relacionadas. Em um segundo momento é descrita a avaliação quantitativa do protótipo desenvolvido, onde foi medido o *overhead* inserido pelo protótipo no *front-end* do *cluster* utilizado para testes.

Finalmente no Capítulo 6 são apresentadas as considerações finais sobre este trabalho e são destacados alguns trabalhos relacionados ao ambiente ICE que vêm sendo desenvolvidos em paralelo ao andamento desta dissertação, assim como trabalhos futuros que podem ser desenvolvidos com base no ambiente ICE.

2 ESTADO DA ARTE EM GERENCIAMENTO DE SISTEMAS DE ALTO DESEMPENHO

Neste capítulo serão discutidas as idéias existentes na área de gerenciamento de sistemas de alto desempenho (HPC - *High Performance Computing*). Para que se possa abordar com maior riqueza de detalhes este tópico, será realizada uma breve apresentação de sistemas distribuídos, suas características e alguns *middlewares* comumente utilizados para construção de tais sistemas. Essa caracterização inicial será importante para que se possa compreender os problemas que existem na construção de ambientes de gerenciamento de sistemas de HPC. De certa forma, pode-se dizer que eles são uma especialização de sistemas distribuídos, visto que são formados por hardware e software interligados por uma rede de comunicação. A principal diferença está em sua especialização. Sistemas de alto desempenho tem o objetivo específico de solucionar problemas no menor tempo possível. Eficiência e eficácia são parâmetros considerados nesse tipo de contexto. Seu gerenciamento requer cuidados, como por exemplo: menor intrusão possível para que o desempenho do sistema não seja afetado; cuidados específicos na alocação de recursos para usuários; monitoramento dos recursos do sistema HPC assim como das aplicações que estão em execução; etc. Em sistemas distribuídos esses problemas de gerenciamento também aparecem, mas não são cruciais. Algumas técnicas e soluções de gerenciamento de sistemas distribuídos e HPC serão apresentadas a seguir.

2.1 Sistemas Distribuídos

Sistemas distribuídos estão presentes em praticamente todos os segmentos da sociedade. Estes tipos de sistemas são caracterizados por componentes de software e de hardware, localizados em computadores interligados por uma rede, que se comunicam e coordenam suas ações através de troca de mensagens (COULOURIS; DOLLIMORE; KINDBERG, 2005). Como exemplos de sistemas distribuídos tem-se: a Internet; as intranets de corporações, universidades, e de redes domésticas; a computação móvel (desde telefones celulares a PDAs, *pagers*, etc); e a computação obíquoa (COULOURIS; DOLLIMORE; KINDBERG, 2005).

De modo geral, sistemas distribuídos abrangem uma quantidade ampla de áreas de pesquisa, desenvolvimento e negócios dentro da computação. Cada qual com um determinado objetivo, como por exemplo, o comércio eletrônico (*e-business* ou *e-commerce*). Nesse tipo de negócio, o objetivo é manter os sistemas o mais tempo possível em funcionamento, além de tentar integrar os diferentes sistemas de fornecedores e compradores para obtenção de maiores lucros. Um sistema de *e-commerce*

não disponível implica em perdas financeiras para a empresa. Em contra-partida a esses tipos de sistemas voltados para empresas, existem sistemas distribuídos acadêmicos cuja finalidade é manter uma rede de dados e serviços para seus usuários (no caso professores, alunos e funcionários). Enfim, são inúmeros os tipos de sistemas distribuídos e sua aplicação cotidiana.

Entretanto, o desenvolvimento e projeto de sistemas distribuídos desperta alguns desafios. De acordo com Coulouris em (COULOURIS; DOLLIMORE; KINDBERG, 2005) os principais desafios a serem considerados são: heterogeneidade, capacidade de extensibilidade (*openness*), segurança, escalabilidade, tolerância a falhas, concorrência e transparência. É bastante complexo o desenvolvimento de um sistema distribuído que consiga satisfazer a contento todos esses desafios. Em geral, eles solucionam alguns deles e outros são preteridos muitas vezes em razão do objetivo final do sistema distribuído projetado. Por exemplo, em um sistema distribuído de tempo real as características mais relevantes são a transparência, concorrência e tolerância a falhas. Nesse caso, procurar soluções ótimas para extensibilidade não é tão relevante. A seguir são descritos com maiores detalhes cada um dos desafios apontados em (COULOURIS; DOLLIMORE; KINDBERG, 2005).

Heterogeneidade - Este conceito está ligado à variedade de redes de comunicação, hardware dos computadores, sistemas operacionais, linguagens de programação, implementações de diferentes desenvolvedores. Para tentar lidar com a heterogeneidade dos sistemas são desenvolvidos *middlewares*, os quais são camadas de software que proporcionam abstração e mascaramento da heterogeneidade. Além disso, *middlewares* também provêem um modelo de uniformização na forma como os sistemas distribuídos são utilizados.

Extensibilidade - A extensibilidade em um sistema define sua capacidade de agregar novas funcionalidades ou ferramentas, assim como sua capacidade de ser reimplementado. Esse conceito, no que se refere a sistemas distribuídos é basicamente determinado pela capacidade de inserção de novos serviços e de sua disponibilização para diferentes programas clientes. A extensibilidade pode ser atingida através da especificação de interfaces dos componentes do sistema e de sua publicação.

Segurança - Existem três componentes de segurança para manter as informações dos recursos em um sistema distribuído: (i) confidencialidade, proteção contra usuários não qualificados e não autorizados; (ii) integridade, proteção contra alteração ou corrupção dos dados; e (iii) disponibilidade, proteção contra as interferências do meio no acesso aos recursos.

Escalabilidade - Característica que um sistema possui de se manter eficiente mesmo com o aumento no número de usuários e de recursos. O desenvolvimento de sistemas distribuídos escaláveis apresenta os seguintes desafios: controle do custo dos recursos físicos; controle da perda de desempenho; prevenção da escassez de recursos de software e prevenção de gargalos.

Tratamento de falhas - Falhas em sistemas distribuídos são parciais, isto é, alguns componentes falham mas outros continuam funcionando. Existem algumas técnicas para o tratamento de falhas: detecção, mascaramento, tolerância, recuperação de falhas e redundância.

Concorrência - Em sistemas distribuídos tanto serviços quanto aplicações possuem recursos que podem ser compartilhados por seus clientes. Nesse caso, é preciso garantir que o sistema funcione corretamente em um ambiente concorrente. Para isto, mecanismos de sincronização que mantenham os dados consistentes devem ser adotados.

Transparência - Característica que faz com que usuários e aplicações de um sistema distribuído tenham uma visão única do sistema ao invés de uma idéia de um sistema formado por diferentes componentes. Existem oito formas de transparência, sendo elas transparência de: acesso, localização, concorrência, replicação, falha, mobilidade, desempenho e escalabilidade.

Para a construção de um sistema distribuído, que seja capaz de suportar total ou parcialmente as características detalhadas acima, é preciso a existência de um *middleware*. Esse componente é capaz de lidar com as especificidades do sistema como um todo e apresentar aos usuários (sejam pessoas ou outras aplicações) uma abstração em alto nível dos recursos do sistema distribuído (COULOURIS; DOLLIMORE; KINDBERG, 2005). *Middlewares* como CORBA (SIEGEL, 1996), RMI (RMI - JAVA REMOTE METHOD INVOCATION, 2005), DCOM (GRIMES, 1997) e recentemente Web Services (CERAMI, 2002) têm sido desenvolvidos no intuito de prover essas abstrações.

Considerando que sistemas de alto desempenho são uma especificação de sistemas distribuídos, a construção de ambientes de gerenciamento e de acesso aos recursos desse cenário também empregam *middlewares* que ofereçam as abstrações necessárias. Muitas vezes existem diferenciações bem claras dos tipos de ambientes de gerenciamento e acesso em HPC e em sistemas distribuídos. Mas de forma geral os objetivos da construção de ambientes de gerenciamento e de acesso a sistemas HPC são exatamente os mesmos existentes em sistemas distribuídos. A seguir, serão apresentados alguns tipos de sistemas HPC e algumas ferramentas de gerenciamento e acesso empregadas na área.

2.2 Sistemas de Alto Desempenho

Sistemas de alto desempenho são tipicamente empregados na solução de problemas que precisam de uma grande quantidade de cálculos ou então casos que existe uma grande quantidade de dados para serem processados. O objetivo é alcançar a resolução desses problemas em um tempo menor ou então com um grau de precisão ou riqueza de detalhes maior. A linha de evolução dos sistemas de alto desempenho foi marcada pela utilização de máquinas massivamente paralelas (MPP - *Massive Parallel Processing*), máquinas vetoriais, *clusters*, constelações e *grids* (BUYYA, 1999).

Alguns exemplos de campos de aplicação de sistemas HPC são:

- física e engenharias em geral: resolução de simulações de corpos celestes, simulações de equações e sistemas em geral;

- hidrologia: simulação de impactos ambientais nos aquíferos, simulação de construção de equipamentos para usinas hidrelétricas, outros tipos de simulações ligados aos recursos hídricos;

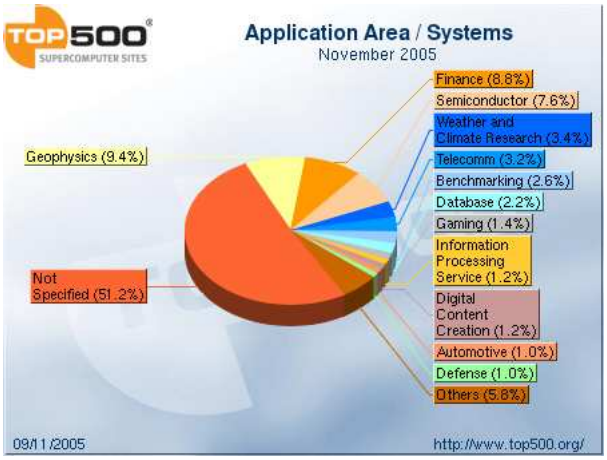

Figura 2.1: Lista de aplicações existentes nas máquinas que fazem parte do Top 500

- geoprocessamento: simulação da existência de petróleo em uma determinada região;
- meteorologia: previsão do tempo, simulação de mudanças climáticas;
- medicina e bio-informática: simulação de efeitos de remédios, simulação de composição de novos medicamentos;
- computação: simulação de circuitos eletrônicos, processamento de imagens e na renderização de imagens, simulação de arquiteturas de computadores, entre outras.

A Figura 2.1 apresenta algumas aplicações que são executadas nas máquinas que fazem parte do Top500 (TOP500 SUPERCOMPUTER SITES, 2005). Essa é uma lista dos 500 supercomputadores mais poderosos do mundo.

Através dessa figura pode-se observar que existem aplicações bastante variadas nas máquinas pertencentes a lista do Top500. Esta variedade cobre aplicações que vão desde científicas, como por exemplo: geo-física (9,4%), *benckmarking* (2,6%), pesquisas de clima e tempo (3,4%), a aplicações mais comerciais, como por exemplo: financeiras (8,8%), telecomunicações (3,2%) e automotiva (1,0%).

Pode-se perceber que o escopo de utilização de sistema de HPC não está mais restrito ao campo da computação. Isto faz com que seja necessária a criação de ambientes e mecanismos que facilitem sua utilização. Outro fator é o tamanho que esse tipo de sistema tem assumido. Atualmente existem máquinas formadas por milhares de nós (elementos processadores). O gerenciamento manual desse tipo de sistemas torna-se inviável. Portanto, também existe uma necessidade de prover plataformas capazes de facilitar a utilização e manipulação desses sistemas.

Atualmente, boa parte dos trabalhos em HPC concentram-se em dois tipos de sistemas: *clusters* e *grids*. Segundo a definição apresentada em (BAKER, 2000), *clusters* são sistemas computacionais locais que encerram um conjunto de computadores independentes inter-conectados por uma rede. Um *cluster* é local no sentido de que todos os seus componentes e sub-sistemas estão localizados dentro de um único domínio administrativo, tipicamente localizados em uma sala e gerenciados como um simples computador. Em contrapartida, *grids* assumem um caráter de

sistemas de larga escala. *Grids* se caracterizam pelo foco na construção de sistemas de larga escala que sejam capazes de coordenar o compartilhamento de recursos e que tenham a capacidade de resolver problemas em organizações virtuais dinâmicas e multi-institucionais (FOSTER; KESSELMAN; TUECKE, 2001). Ainda segundo Ahmar Abbas (ABBAS, 2004) a computação em *grid* pode juntar em um único escopo todos os esforços realizados nas áreas de computação de alto desempenho, peer-to-peer (P2P) e Internet. A seguir esses dois tipos de sistemas são descritos, sendo que são apresentados os problemas de gerenciamento existentes e algumas soluções para tais problemas.

2.2.1 *Clusters*

Apesar do cenário de *clusters* ser uma área bem desenvolvida e consolidada, com um grande número de ferramentas e ambientes disponibilizados, não existe uma padronização na forma como eles foram desenvolvidos. Mais do que isto, não existe uma padronização do que é gerenciamento em *clusters*. Por exemplo, quando se fala em gerenciamento de redes, e mais especificamente em um modelo de gerenciamento de redes tem-se a ligação direta ao *framework* SNMP (CASE; et al., 1990). Em *clusters* não existe uma metodologia, taxonomia, ou padrão na área de gerenciamento.

Uma tentativa de delimitar essa área é apresentada em (STERLING, 2000). Segundo Thomas Sterling, os softwares que compõe um *cluster* podem ser divididos em: ambientes de programação de aplicações e softwares de gerenciamento de recursos. Este último componente pode ser dividido nas categorias de software listadas abaixo.

- Instalação e configuração - Esta categoria lida com os desafios de desenvolvimento de software que sejam capazes de implementar e manter imagens comuns entre os nós do *cluster* e que sejam de fácil utilização. Este é um problema a ser bastante considerado quando se trabalha com *clusters* formados dos centenas de nós. Exemplos de ferramentas que possuem essas características são: C3 (Cluster Command and Control) (FLANERY; et al., 2000), System Imager (SYSTEM IMAGER, 2005), SSI (VALLEE; et al., 2005), entre outros.

- Escalonamento e alocação - A colocação das aplicações nos recursos distribuídos dos *clusters* requer ferramentas que sejam capazes de alocar os componentes de software nos nós e que escalonem o tempo de sua execução. Segundo (STERLING, 2000), a alocação pode ser realizada em diferentes níveis de granularidade. Pode ser alocação: de trabalhos, transações, processos ou *threads*. Alguns exemplos de ferramentas desenvolvidas são: OpenPBS (PORTABLE BATCH SYSTEM, 2005), OAR (CAPIT; et al., 2003, 2005), CCS (KELLER; REINEFELD, 1998), Maui (MAUI SCHEDULER OPEN CLUSTER SOFTWARE, 2005), CADEO (CERA; ROSA RIGHI; PASIN, 2005), etc.

- Administração do sistema - Para a supervisão do funcionamento dos sistemas de *clusters* é necessário ferramentas que sejam capaz de prover: gerenciamento das contas de usuários, filas de trabalhos; segurança; *backups*; armazenamento de dados; *log* de informações; terminais para usuários; entre outras atividades simples de manutenção do sistema. Exemplos de ferramentas que tentam prover essas facilidades administrativas são: OSCAR (LIGNERIS; et al., 2003; OSCAR - OPEN SOURCE CLUSTER APPLICATION RESOURCES,

2005), PUNCH (ADABALA; KAPADIA; FORTES, 2000; PARK; et al., 2000), ROCKS (PAPADOPOULOS; KATZ; BRUNO, 2001), entre outros.

- Monitoramento e diagnósticos - Nessa categoria, considera-se ferramentas que possam ser usadas no monitoramento do estado e das operações dos elementos de sistema. Como exemplo, dessas ferramentas têm-se: Ganglia (SACERDOTI; et al., 2003), SCMS/RMS (UTHAYOPAS; PHATANAPHEROM, 2001), Supermom (SOTTILE; MINNICH, 2002), entre outros.

- Armazenamentos secundários distribuídos - Grande parte das computações necessitam acessar fontes de armazenamento secundário, isto inclui o acesso a discos locais e remotos através do sistema de arquivos. Para prover esses mecanismos de acesso, muitos sistemas foram desenvolvidos, como por exemplo: dNFSp (KASSICK et al., 2005; ÁVILA et al., 2004), PVFS (HADDAD, 2000), iPVFS (OU; HE, 2005), etc.

- Disponibilidade - Na medida que o *cluster* se torna mais escalável, o seu MTBF (*MeanTime Between Failures*) diminui. Isto faz com que sejam necessárias medidas para garantir a continuidade da operação do sistema e minimizar o tempo que os recursos estão indisponíveis. Alguns exemplos de ferramentas empregadas com o intuito de prover mecanismos de reabilitação do sistema e detecção e recuperação de falhas são: C^3 (Cornell Checkpoint (pre)Compiler) (SCHULZ; et al., 2004; BRONEVETSKY; et al., 2004), CONDOR (THAIN; LIVNY, 2003), entre outros.

Através da descrição dos campos que envolvem o conceito de gerenciamento em *clusters*, e da apresentação de várias ferramentas em cada uma de suas categorias, pode-se perceber que não existe uma padronização. O que existem são diversas ferramentas, desenvolvidas muitas vezes para suprir as necessidades de cada grupo de pesquisa. Isto leva a existência de uma heterogeneidade de ferramentas de gerenciamento de *clusters* muito grande. Além disso, muitas delas não foram projetadas levando em consideração questões de interoperabilidade e de facilidade de uso.

No intuito de facilitar o gerenciamento dos *clusters* e também a forma como os seus usuários utilizam seus recursos, foram desenvolvidos alguns ambiente e plataformas de gerenciamento de *clusters*. A seguir, serão descritas as ferramentas: OSCAR, OpenSCE (UTHAYOPAS; ANGSKUN; MANEESILP, 2001, 2002; OPENSCE - SCALABLE CLUSTER ENVIRONMENT, 2005), ROCKS e o trabalho que tenta integrar o gerenciamento de *clusters* com SNMP (ALVES et al., 2004; ALVES; MARQUEZAN; GRANVILLE, 2004; ALVES et al., 2005).

2.2.1.1 OSCAR - Open Source Cluster Application Resource

OSCAR é uma ferramenta que visa a construção de um *cluster* que possua suas ferramentas integradas em um único ambiente. Seu objetivo é permitir que usuários e administradores de *cluster* possam criar, manter e utilizar *clusters* baseados em plataforma Linux. O OSCAR é uma coleção de ferramentas comumente usadas em *clusters*, as quais são disponibilizadas em forma de arquivos `tar` que devem ser instalados no *front-end*. O OSCAR é composto pelos seguintes módulos: (i) núcleo da infra-estrutura e do gerenciamento, (ii) administração e configuração, (iii) serviços e ferramentas de HPC e (iv) segurança. Algumas ferramentas que fazem

parte do OSCAR são: C3, System Imager, Ganglia, OpenPBS, MPI (DONGARRA; et al., 1995), PVM (SUNDERAM, 1990), entre outras. Para utilização do OSCAR é preciso um nível aprofundado de conhecimento da arquitetura e dos sistemas de *clusters*.

2.2.1.2 OpenSCE - Open Scalable Cluster Environment

O projeto OpenSCE é caracterizado por utilizar ferramentas próprias, ou seja, que foram desenvolvidas pelo mesmo grupo que dá suporte ao projeto. O OpenSCE é um software constituído pelos seguintes tipos de ferramentas: (i) administrativas e de monitoramento (Beowulf Builder, Ksix, SQMS), (ii) um sistema de gerenciamento de filas (SCMS), um sistema Web de monitoramento (SCMSWeb), uma ferramenta de visualização e depuração de aplicações (MPView) e uma biblioteca de programação paralela que é baseada no MPI (MPITH). O ambiente OpenSCE pode ser uma boa alternativa para gerenciamento de *cluster* quando este está sendo montado desde o início. Entretanto, quando já existe uma infra-estrutura montada, com ferramentas diferentes das existentes no OpenSCE, pode-se tornar muito custosa a troca de todas as ferramentas. Mais do que isso, é preciso levar em consideração que os usuários deste *cluster* terão que se readaptar às novas ferramentas, e muitas vezes a curva de aprendizado não compensa o investimento.

2.2.1.3 NPACI Rocks Toolkit

Os trabalhos realizados no desenvolvimento do NPACI Rocks buscam encontrar soluções para as dificuldades que existem na construção de *clusters* que sejam gerenciáveis. De forma resumida, Rocks é uma distribuição Linux voltada para *clusters* baseada no Red Hat (RED HAT - THE OPEN SOURCE LEADER, 2005). Além disso, possui alguns pacotes adicionais e configurações programadas para automatizar a montagem de um *cluster* Linux de alto desempenho. O sistema é baseado em uma arquitetura tradicional de *clusters*: *front-end*, uma rede de interconexão Ethernet (adicionalmente é possível uma rede de alto desempenho - Myrinet, por exemplo) e os nós. Na distribuição NPACI Rocks foi desenvolvido uma infra-estrutura de configuração com propriedades bem definidas baseadas em padrões *de facto*, o que inclui: XML (com parsers padrão), RedHat Kickstart, HTTP, CGI, base de dados SQL e construtores de grafos para facilmente definir as ferramentas do *cluster* (PAPADOPOULOS; KATZ; BRUNO, 2001). A idéia principal do projeto é tornar o uso de *clusters* fácil (ROCKS CLUSTER DISTRIBUTION: USERS GUIDE, 2005). Como apresentado em (BRUNO; et al., 2004), o objetivo do desenvolvimento do Rocks é permitir que pessoas que não sejam experientes em *clusters* consigam facilmente montar e gerenciar seus *clusters*.

2.2.1.4 Clusters + SNMP

Esta ferramenta de gerenciamento de *clusters* baseado em SNMP foi desenvolvida no intuito de integrar o gerenciamento de *clusters* dentro de plataformas de gerenciamento de redes. Visto que padronização é um problema nesta área e que os *clusters* podem apresentar uma visão única de todo o sistema, pode-se gerenciar *clusters* como se eles fossem mais um dispositivo da rede. Desta forma, consegue-se inserir a tarefa de gerenciamento de *clusters* juntamente com o gerenciamento de redes. A vantagem desta abordagem é o aproveitamento de diversas ferramentas e

plataformas de gerenciamento já desenvolvidas no escopo de redes no contexto de *clusters*. Seguindo o *framework* SNMP, padrão *de facto* em gerenciamento de redes, foram desenvolvidos: MIBs de gerenciamento do *front-end* e dos nós, agentes que seguem esta MIB e um gerente. A ferramenta desenvolvida possui suporte: para monitoramento dos recursos dos *clusters* e das aplicações que estão executando; para escalonamento das aplicações; para configuração da política de alocação do *cluster*. Este ambiente foi integrado na plataforma de gerenciamento QAME (*QoS-Aware Management Environment*) (GRANVILLE; TAROUCO, 2001), mostrando que pode-se integrar o gerenciamento de *clusters* e redes.

2.2.1.5 Resumo sobre ambientes de gerenciamento de clusters

Existe um conjunto de ferramentas de gerenciamento de *clusters* bastante grande. Nesta seção tentou-se mostrar alguns ambientes que oferecem um conjunto maior de funcionalidades. Mesmo assim, pode-se perceber que o OSCAR e o Rocks Toolkit são ferramentas mais voltadas para a parte de configuração de *clusters*, direcionadas para o momento de estabelecimento e configuração dos *cluster* em si. Enquanto isto, as ferramentas OpenSCE e *Cluster* + SNMP são mais voltadas para o uso de *clusters* e monitoramento de seus recursos e aplicações.

2.2.2 *Grids*

Segundo Ahmar Abbas (ABBAS, 2004) existem os tipos de *grids* apresentados a seguir.

- *Grids* Departamentais (*Departamental Grids*)- São construídos para resolver problemas de um grupo particular dentro de uma empresa. Os recursos não são compartilhados por outros grupos da mesma instituição.

- *Grids* Empresariais (*Enterprise Grids*) - São formados por recursos espalhados pela empresa e que provêem serviços para todos os usuários dentro da empresa.

- *Grids* Extra-empresariais (*Extraprise Grids*) - Caracteriza-se por ser estabelecido entre companhias, seus parceiros e seus clientes. Os recursos são disponibilizados através de uma rede virtual privada (VPN - *Virtual Private Network*).

- *Grids* Globais (*Global Grids*) - São formados através da Internet. Podem ser formados no intuito de facilitar os negócios entre instituições.

- *Grids* Computacionais (*Compute Grids*) - São criados com o objetivo exclusivo de prover acesso a recursos computacionais. Eles são divididos de acordo com o tipo de hardware empregado, os quais podem ser: *desktops*, servidores e baseados em sistemas de alto desempenho.

- *Grids* de Dados (*Data Grids*) - Este tipo de *grid* é otimizado para trabalhar com operações orientadas a dados.

- *Utility Grids* - São definidos como sendo recursos computacionais que são mantidos e gerenciados por um provedor de serviço.

Como pode-se perceber existem muitas formas diferentes de encarar a utilização de *grids*. Isto faz com que apareçam diferentes visões das definições e padrões a

serem adotados na área de *grids* (BAKER; et al., 2005). No que diz respeito a parte de gerenciamento de *grids*, existem muitos aspectos que devem ser considerados. Alguns deles são descritos a seguir.

1. Segurança - É preciso garantir que somente usuários que façam parte do *grid* tenham acesso aos recursos, além disso é preciso restringir acesso de alguns usuários a determinados serviços existentes em *grids*. Neste escopo, o GSI (*Grid Security Infrastructure*), promovido pelo GGF (*Global Grid Forum*), é um padrão bem estabelecido e que trata dos problemas de segurança em *grid* (WELCH; et al., 2003).

2. Monitoramento - Dado o fato de *grids* serem estruturas extremamente distribuídas e complexas, é preciso que essas estruturas sejam monitoradas. Segundo Maozhen Li e Mark Baker (LI; BAKER, 2005), o objetivo do monitoramento de *grids* é a medição e a publicação do estado dos recursos em um certo período de tempo. O GMA (*Grid Monitoring Architecture*), proposto pelo GGF, é uma tentativa de criação de um mecanismo padronizado para obtenção e disponibilização das informações de monitoramento de *grids* (TIERNEY; et al., 2002; LI; BAKER, 2005). Alguns exemplos de ferramentas que implementam o GMA são: GridRM (BAKER; SMITH, 2003), HRIC (ZOU; et al., 2005) e R-GMA (COOKE; et al., 2003).

3. Escalonamento e gerenciamento de recursos - As aplicações dos usuários são disparadas em máquinas que estão localizadas em diferentes domínios administrativos. É preciso a existência de ambientes capazes de gerenciar essas aplicações e escaloná-las. Ferramentas como por exemplo: Globus (FOSTER; KESSELMAN, 1997; GLOBUS - WELCOME TO THE GLOBUS TOOLKIT HOMEPAGE, 2005), Condor (GONZ; et al., 2002; THAIN; TANNENBAUM; LIVYN, 2003), Legion (GRIMSHAW; NATRAJAN, 2005), OurGrid (ANDRADE; et al., 2003), UNICORE (BENEDYCZAK; et al., 2005), SGE (GENTZSCH, 2001), entre outros.

4. Facilidade de uso - Para que os desenvolvedores de aplicações de *grid* tenham facilidades de acesso e utilização de *grids* foi criado o conceito de Portais de *Grid*. Um portal de grid é um *gateway* baseado em tecnologia Web que provê facilidades de acesso a uma variedade de recursos. Segundo Li e Baker (LI; BAKER, 2005) existem duas gerações de portais. Na primeira geração os portais eram construídos utilizando uma arquitetura baseada em três camadas. Exemplos de portais desta geração são: GridPort (THOMAS; et al., 2001; DAHAN; et al., 2004), GridSpeed (SUZUMURA; et al., 2004), Genius (ANDRONICO; et al., 2003), Ninf (NAKADA; et al., 2004), entre outros. A segunda geração de portais de *grid* é caracterizada por ser construída a partir de *portlets*, os quais são componentes de software escritos em Java, gerenciados por *portlet containers*, e que são responsáveis por gerenciar as requisições dos usuários e pela geração dinâmica de conteúdo (LI; BAKER, 2005). Exemplos de portais da segunda geração são: PortalLab (LI; et al., 2003) e GridSphere (NOVOTNY; RUSSELL; WEHRENS, 2004),

Através da descrição apresentada acima sobre alguns aspectos que devem ser considerados no gerenciamento de *grids* e da apresentação de algumas das ferra-

mentas disponíveis, é possível perceber que o cenário de *grids* é bastante complexo e ainda em estágio de evolução.

A utilização de *grids* é um passo natural para os usuários de *clusters*. Atualmente, é cada vez mais necessária a utilização de uma quantidade maior de recursos computacionais para solucionar os problemas existentes. Desta forma, é bastante plausível que se integre recursos existentes em diferentes instituições, visando reduzir o custo de aquisição de toda uma infra-estrutura computacional.

No entanto, existe um custo de adaptação de ferramentas e de reeducação dos usuários de *clusters* que muitas vezes não é considerado. Nem todos os centros de pesquisa, corporações e empresas, estão prontas para migrarem suas aplicações e usuários de um mundo baseado em *clusters* para um baseado em *grids*. Considerando este cenário, existe um meio termo entre o contexto de *grids* e o de *clusters*. Este contexto é descrito a seguir.

2.2.3 Intersecção entre *clusters* e *grids*: sistemas Multicluster

Nem todos os centros de pesquisa e instituições em geral, que possuem múltiplos *clusters*, aderem ao cenário de *grids* para aumentar o seu potencial de processamento, armazenamento, etc. Não é raro que em uma mesma instituição existam vários *clusters* que possuem seu funcionamento completamente isolado um do outro. Essa situação geralmente acontece, porque na maioria das vezes esses *clusters* são adquiridos em diferentes pontos do tempo.

Existem trabalhos no sentido de integrar os diferentes *clusters* como se eles formassem uma única máquina, o que se chamou de sistemas *multiclusters* (BARRETO; ÁVILA; NAVAUX, 2000; AUMAGE, 2002). *Multiclusters* podem ser encarados sob a visão dos usuários que utilizam os recursos ou sob a visão do administrador desses *clusters*. No caso dos usuários a idéia de multicluster faz com que seja necessária a criação de ferramentas que dêem o suporte para a comunicação entre os diferentes *clusters* de forma amigável. No caso de gerenciamento de múltiplos *clusters*, não é preciso a existência dessa infra-estrutura de comunicação. Basta que se tenha acesso aos diferentes *clusters*. A seguir serão abordados os aspectos de gerenciamento e acesso a múltiplos *clusters*.

A heterogeneidade é a característica que permeia o ambiente de múltiplos *clusters*. Para gerenciar e acessar esses recursos é preciso, primeiramente, lidar com as diferentes ferramentas instaladas em cada um. Essa heterogeneidade passa a ser uma dificuldade enfrentada por seus usuários, pois eles têm que aprender a lidar com cada uma dessas ferramentas para desempenharem suas atividades.

Para usuários acostumados com a ciência da computação isto pode não representar um empecilho. No entanto, para usuários que pertencem a outras áreas de atuação a dificuldade de ter de se acostumar e aprender os detalhes e semântica do grande número de ferramentas passa a ser um problema considerável. No intuito de facilitar o gerenciamento e o acesso de múltiplos *clusters* foram desenvolvidas algumas ferramentas. Duas dessas ferramentas serão apresentadas a seguir.

2.2.3.1 M3C Managing and Monitoring Multiple Clusters

O M3C (BRIM; et al., 2001) proporciona uma interface gráfica Web para administração de *clusters* e também um *framework* para o desenvolvimento dos sistemas de gerenciamento das camadas inferiores. Ele foi projetado para ser extensível, uma vez que novas funcionalidades de *clusters* podem ser inseridas no ambiente através

da aplicação do *framework*. Embora apresente a capacidade de estender o conjunto de funcionalidades, o M3C não tem suporte para a extensão das ferramentas para cada funcionalidade. Ou seja, ele não é capaz de lidar com diferentes ferramentas para uma mesma funcionalidade. O M3C foi implementado utilizando Java Applets e Servlets e também utiliza arquivos que armazenam informações sobre os recursos dos *clusters*. Este ambiente apresenta suporte para: monitoramento dos nós, reserva de nós e particionamento de *clusters*. Questões de segurança são tratadas utilizando o protocolo HTTPS para encriptação dos dados transmitidos, e realizando verificações de quais *clusters* cada usuário pode utilizar. O M3C apresenta-se como um ambiente contendo basicamente funcionalidades de acesso e monitoramento de múltiplos *clusters*, além disto, por ser implementado em Java possui um certo grau de independência de plataforma.

2.2.3.2 HPC2N - High Performance Computing Center North

O HPC2N (ELMROTH; NYLÉN; OSCARSSON, 2005) foi desenvolvido para ser utilizado em sistemas de alto desempenho. Ele foi projetado para ser um ambiente com foco nos usuários, provendo funcionalidades para lidar com suas aplicações e suporte para monitoramento. O acesso ao HPC2N é proporcionado através da Internet utilizando a tecnologia CGI. As comunicações entre o servidor HPC2N e os navegadores dos usuários são realizadas utilizando o protocolo HTTPS e a autenticação dos usuários é feita através do protocolo Kerberos (COULOURIS; DOLLIMORE; KINDBERG, 2005). Diferentemente do M3C, o HPC2N não apresenta nenhum suporte para ser estendido. Ou seja, não é possível inserir novas funcionalidades e também nem inserir novas ferramentas nesse ambiente.

2.2.3.3 Problemas em Aberto no Cenário de Gerenciamento de Múltiplos Clusters

Por se tratar de um ambiente heterogêneo, alguns aspectos ligados a prover facilidades para os usuários devem ser considerados na criação de ambientes para gerenciamento e acesso de múltiplos *clusters*. Alguns aspectos importantes são:

- transparência de acesso;
- extensibilidade;
- uniformização das operações;
- completeza;
- interoperabilidade;
- capacidade de incorporação dos sistemas de gerenciamento legados;
- construção de ferramentas amigáveis para os usuários.

Analisando as ferramentas de gerenciamento e acesso de múltiplos *clusters* apresentadas acima percebe-se que elas não possuem suporte para todos os aspectos descritos acima. Transparência de acesso é provida tanto pelo ambiente M3C quanto pelo HPC2N. Extensibilidade é provida somente pelo M3C, e ainda de forma parcial. Uniformização das operações está relacionada com a característica de extensibilidade, pois implica na possibilidade de lidar com diferentes ferramentas mas com

a mesma funcionalidade. Nenhum dos ambientes proporcionam essa característica. Da mesma forma como não apresentam completeza (capacidade de lidar com diferentes funcionalidades de *clusters*), interoperabilidade, capacidade de lidar com sistemas legados. Os ambientes M3C e HPC2N apresentam interfaces gráficas que facilitam as operações dos usuários. Enfim, esses ambientes promovem facilidades no gerenciamento e acesso a múltiplos *clusters*, mas ainda não estão suficientemente maduros para prover características fundamentais como extensibilidade, uniformização, interoperabilidade e suporte a sistemas legados.

2.3 Resumo

Neste capítulo foram apresentadas as principais características de sistemas distribuídos e de sistemas de alto desempenho. Como visto, sistemas de alto desempenho podem ser considerados como uma especialização de sistemas distribuídos. Eles apresentam, em geral, os mesmos problemas de gerenciamento, facilidade de utilização, só que possuem um fim mais específico - resolver problemas de forma eficiente. Dentro de sistemas HPC, têm-se *clusters* e *grids* como os cenários mais utilizados pelos usuários. O gerenciamento desse tipo de sistema apresenta desafios. É preciso prover aos usuários e administradores ferramentas de fácil utilização e que consigam gerenciar as funcionalidades desses cenários. Na área de gerenciamento de *clusters* existem muitos trabalhos consolidados e bem estabelecidos, enquanto na área de *grids* ainda existem discussões e campo aberto para inovações e novas abordagens. Entretanto, também foi apresentado neste capítulo que existe uma intersecção entre *clusters* e *grids*, denominada de *multiclusters*. Essa intersecção apresenta algumas ferramentas de gerenciamento, mas elas ainda não provêem características importantes para um escopo como este, distribuído e heterogêneo.

3 PROPOSTA DE AMBIENTE INTEGRADO PARA CLUSTERS: ICE

O ambiente ICE (*Integrated Cluster Environment*) (MARQUEZAN et al., 2006) é proposto com o intuito de prover as características de gerenciamento de múltiplos *clusters* que ainda não foram contempladas pelas plataformas já existentes, como apresentado na Capítulo 2. Sendo assim, os objetivos do ambiente ICE são prover as características descritas abaixo.

- Extensibilidade - O ambiente tem que ser capaz de incorporar novas funcionalidades sem que isso impacte na sua execução em geral. Por funcionalidades entende-se os serviços tipicamente disponibilizados para os usuários dos *clusters* ou aqueles utilizados pelos próprios administradores. Exemplos de funcionalidades são: monitoramento dos recursos de um *cluster*, monitoramento das aplicações, gerenciamento das aplicações dos usuários, ações administrativas nos componentes dos *clusters* (nós, *front-end* e rede de intercomunicação), etc. Mas além de prover extensibilidade no nível das funcionalidades do sistema, o ambiente ICE também deve visar a extensibilidade de ferramentas. Isso significa que o ambiente deve suportar a utilização de qualquer ferramenta, e que não está restrito a um conjunto fixo de ferramentas.

- Transparência - A proposta do ambiente ICE é prover transparência de acesso às ferramentas e também transparência na forma de utilização dessas ferramentas. Dessa forma os usuários poderão acessar as ferramentas sem ter a noção de quais exatamente estão utilizando. A mesma idéia se aplica para execução das suas operações, isto é, os usuários não irão lidar diretamente com o formato dos parâmetros e nem com os comandos de cada uma delas.

- Uniformização das operações - Através de sua arquitetura o ambiente ICE se propõe a uniformizar a forma como as operações das ferramentas integradas são realizadas. O objetivo é fazer com que os usuários do sistema executem as mesmas ações independentemente do tipo de ferramenta que está instalada nos *clusters* aos quais eles têm acesso.

- Completeza - O ambiente proposto visa abranger as atividades tipicamente executadas em um *cluster*, como por exemplo: gerenciamento de aplicações (submissão, término, visualização) monitoramento de recursos e aplicações e intervenções administrativas. Pretende-se que o usuário acesse o ambiente ICE e através dele consiga fazer todas as tarefas que anteriormente ele fazia através de um terminal. Para isso a definição da arquitetura ICE procura ser

o mais flexível possível para permitir que diferentes tipos de atividades sejam incluídas na plataforma.

- Interoperabilidade - Para que o ambiente ICE seja capaz de gerenciar múltiplos *clusters*, cada um com diferentes ferramentas, é preciso que ele seja capaz de lidar com os sistemas legados desses *clusters*. Essa é uma característica fundamental e que diferencia o ambiente ICE dos demais trabalhos realizados no mesmo escopo. O ICE é capaz de ser instalado sem que isso interfira na infraestrutura existente no *cluster*. Além disso, o ambiente ICE é independente de plataforma de hardware e software. Ele é capaz de lidar com essas especificidades de maneira transparente. Portanto, dentro do ambiente pode-se gerenciar *clusters* baseados em diferentes arquiteturas de hardware e sistemas operacionais.

- Facilidade de uso - Uma característica também importante do ambiente ICE é prover facilidade de uso aos seus usuários. Esse ambiente permite que seus usuários trabalhem com as ferramentas existentes nos *clusters* sem ter que lidar com os parâmetros e comandos específicos das mesmas. Através da apresentação gráfica, eles podem realizar suas atividades sem ter que aprender a usar as ferramentas. No cenário de múltiplos *clusters* essa característica é importante. Ao invés de aprender a lidar com todas as ferramentas dos *clusters*, aos quais os usuários tem acesso, eles aprendem a lidar somente com o ambiente ICE. Esse, por sua vez, apresenta um ponto único de acesso aos usuários e trata toda a complexidade das ferramentas sem a intervenção de seus usuários.

O público alvo do ambiente ICE são usuários que não são nativos da área da ciência da computação e mais especificamente da área de alto desempenho, como por exemplo, pesquisadores nas áreas de física, bio-informática, geo-processamento, etc. Esse tipo de usuário utiliza os recursos que os sistemas HPC disponibilizam, mas não são seus objetivos saber exatamente como as coisas funcionam e nem lidar com as especificidades das ferramentas que executam as tarefas que eles precisam. Por exemplo, usuários da área de recursos hídricos necessitam de *clusters* para executarem suas simulações, mas não é seu interesse saber como funciona um gerenciador de aplicações (como o OpenPBS (PORTABLE BATCH SYSTEM, 2005)) ou uma ferramenta de monitoramento (como o Ganglia (SACERDOTI; et al., 2003)). Muitas vezes esse tipo de usuários tem acesso a diferentes sistemas de HPC. Para cada um deles seria necessário aprender o funcionamento das ferramentas. É para atender às necessidades desse tipo de usuários que o ambiente ICE torna-se mais indicado. Entretanto, isto não significa que usuários da área de computação de alto desempenho não possam tirar proveito do ambiente. Esse tipo de usuário, em geral com maior conhecimento das ferramentas empregadas, pode se beneficiar das facilidades de acesso e de visualização que o ambiente ICE proporciona.

Outra característica do ambiente ICE é a filosofia de incorporar e buscar adaptar padrões e especificações ao contexto de gerenciamento de múltiplos *clusters*. A idéia é evitar criar novas infra-estruturas, protocolos e especificações. O objetivo é fazer com que os sistemas de *clusters* que já existem sejam integrados em um único ambiente sem que isso exija a utilização de mecanismos e *middlewares* que restrinjam extensibilidade e interoperabilidade do ambiente ICE. Organizações, como por exemplo, W3C (W3C ARCHITECTURE DOMAIN - WEB SERVICES ACTIVITY, 2005) e OASIS (OASIS-WS - OASIS COMMITTEES BY CATEGORY:

WEB SERVICES, 2005), tem trabalhado na padronização de protocolos e em especificações para criação de sistemas inter-operáveis. Exemplo destes esforços são as especificações ligadas a *Web services*. Outras organizações como por exemplo GGF (GLOBAL GRID FORUM, 2005) estão trabalhando na definição de especificações e padronizações na área de *grids*. A filosofia que se pretende adotar no ambiente ICE é aproveitar estas iniciativas só que adaptando-as para o cenário de múltiplos *clusters*.

O contexto de *grid* e de múltiplos *clusters* é semelhante em alguns aspectos. Ambos lidam com recursos que estão dispersos fisicamente, o que exige a inserção de um certo grau de segurança no sistema. O acesso aos recursos tem que ser transparente e deve existir uma maneira de uniformizar a utilização das ferramentas dos diferentes recursos participantes. Entretanto, o cenário de *grids* requer mais mecanismos de controles. Em *grids* é preciso levar em consideração questões como: dinamicidade de recursos, manter canais de comunicação entre recursos alocados em domínios administrativos diferentes, manter informações de controle de onde estão sendo executadas as aplicações dentre os diversos domínios, entre outros (LI; BAKER, 2005). Todos esses controles são decorrentes do fato dos recursos que fazem parte dos *grids* serem alocados de forma transparente e independente. Isto é, o usuário solicita um determinado número de recursos independentemente de sua localização física. No contexto de múltiplos *clusters*, ao qual o ambiente ICE se destina, o usuário sabe onde estão os recursos que ele está solicitando. Dessa forma, não é preciso manter toda a infra-estrutura de controle de localização de recursos neste contexto. Baseado nas semelhanças e distinções entre estes dois cenários percebeu-se que alguns conceitos, especificações e padrões do mundo de *grid* podem ser mapeados e utilizados no gerenciamento de múltiplos *clusters*. Esses reaproveitamentos serão abordados com mais detalhes no Capítulo 4.

A seguir serão apresentados os cenários distribuídos nos quais o ambiente ICE pode ser adotado e o modelo da arquitetura projetada para o ambiente ICE.

3.1 ICE como um Sistema Distribuído

Antes de compreender a arquitetura projetada para a plataforma ICE é preciso entender o grau de distribuição inserido neste contexto. O primeiro ponto a ser considerado é o escopo em que o ambiente ICE está inserido. Ele foi definido para ser empregado em um cenário composto por múltiplos *clusters*, entretanto o acesso a estes diferentes *clusters* é realizado através de um único ponto, o qual esconde a complexidade de lidar com diferentes *clusters* distribuídos em diferentes domínios administrativos. A Figura 3.1 apresenta este cenário de utilização do ambiente ICE.

Os *clusters* gerenciados pelo ambiente ICE podem estar em quaisquer domínios administrativos, assim como os usuários que os acessam. Como a Figura 3.1 mostra, existem interações entre usuários e o ambiente ICE e entre esse e os *clusters* gerenciados. Esses dois tipos de interações possuem características diferentes. No primeiro caso trata-se de um sistema Web. Sendo assim, é preciso lidar com as questões relacionadas a aplicações Web, como por exemplo: separação entre a apresentação das informações da lógica do sistema (modelo de aplicação em 3 camadas), questões de segurança no servidor Web, entre outras. No segundo caso, trata-se de um *middleware* para gerenciamento das atividades em *clusters*. Questões de segurança, uniformização do acesso, interoperabilidade e flexibilidade devem ser consideradas

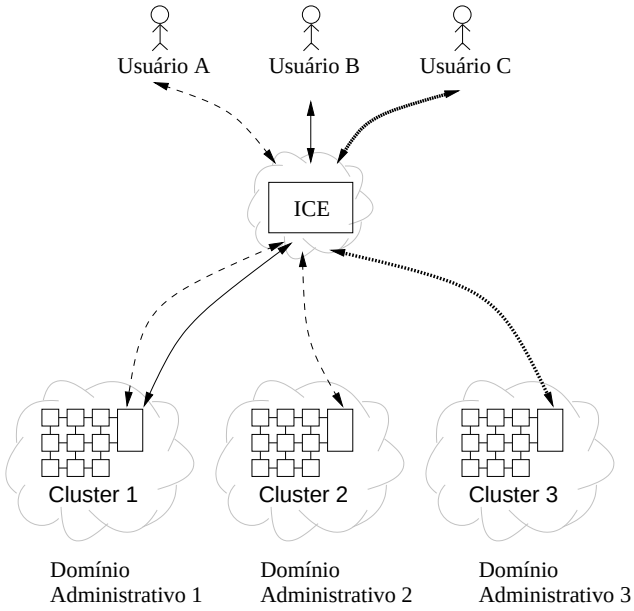

Figura 3.1: Visão geral do cenário de utilização do ICE

nas decisões de definição dessa parte do sistema.

O segundo ponto considerado relaciona-se com o fato do ambiente ICE não ser apenas um Portal Web ou um sistema rodando no *front-end* de um *cluster*. O ambiente ICE é a união entre o ponto único de acesso dos usuários - Portal Web - com o *middleware* definido para o gerenciamento e acesso das funcionalidades dos *clusters*.

3.2 Arquitetura ICE

No momento que se compreende o cenário no qual o ambiente ICE será utilizado pode-se perceber que a arquitetura desse sistema deve ser o mais flexível possível para poder lidar com a heterogeneidade de seu contexto.

Uma característica principal da arquitetura ICE é sua modularidade. O objetivo é que se possa inserir novas funcionalidades e novas ferramentas sem que isso altere o modelo do ambiente. A idéia por trás da arquitetura do ambiente ICE é fazer com que existam módulos básicos do sistema e uma infra-estrutura para prover serviços de *clusters*. Na visão do ambiente ICE, cada funcionalidade existente em um *cluster*, como por exemplo: monitoramento de recursos, gerenciamento de aplicações e recursos através do uso de escalonadores, é potencialmente um serviço a ser disponibilizado.

Como apresentado na Seção 3.1, esse ambiente é divido em dois contextos: ponto único de acesso (Portal Web) e o *middleware* para lidar com o gerenciamento e acesso aos *clusters*. De modo geral o mapeamento da arquitetura ICE para esses contextos pode ser feita da seguinte forma: os módulos básicos são mapeados para a infra-estrutura necessária para modelagem e construção do Portal Web; enquanto a infra-estrutura para prover os serviços é mapeada para o *middleware*.

Na concepção da arquitetura ICE, optou-se por utilizar o modelo de arquitetura orientada a serviços (SOA - *Service Oriented Architecture*) (MCGOVERN; et

al., 2003) para definição do *middleware* que viabiliza a infra-estrutura de serviços nos *clusters*. Essa decisão foi tomada dadas as características que este modelo de arquitetura proporciona às aplicações que a adotam (MCGOVERN; et al., 2003):

- os serviços são descobertos e dinamicamente localizados;
- os serviços são auto-contidos, modulares e apresentam um baixo acoplamento;
- os serviços possuem interoperabilidade;
- os serviços apresentam transparência de localização e podem ser compostos com outros serviços;
- a arquitetura orientada a serviços suporta a característica de auto-cura (*self-healing*), ou seja, é capaz de identificar problemas em sua estrutura e tomar medidas para tratá-las.

A maior vantagem em utilizar uma arquitetura orientada a serviços é poder separar a implementação de um serviço de sua interface. Essa característica é fundamental para a modelagem e o desenvolvimento do ambiente ICE.

A Figura 3.2 apresenta a arquitetura definida para o ambiente ICE, a qual está dividida em dois componentes principais: *Middleware* de Serviços e o Portal Web. O *Middleware* de Serviços é formado por três módulos: *Unified Service Interface* (USI), *Service Implementation* (SI) e *Service Module* (SM). Através da Figura 3.2 pode-se perceber que o SM também faz parte do Portal Web. Esse módulo será detalhado no decorrer desta seção e serão esclarecidas as razões pelas quais ele faz parte dos dois componentes. Ainda formam o Portal Web os módulos: *Security Module* (SecM) e *System Management Module* (SMM). A seguir cada componente e seus respectivos módulos serão descritos detalhadamente.

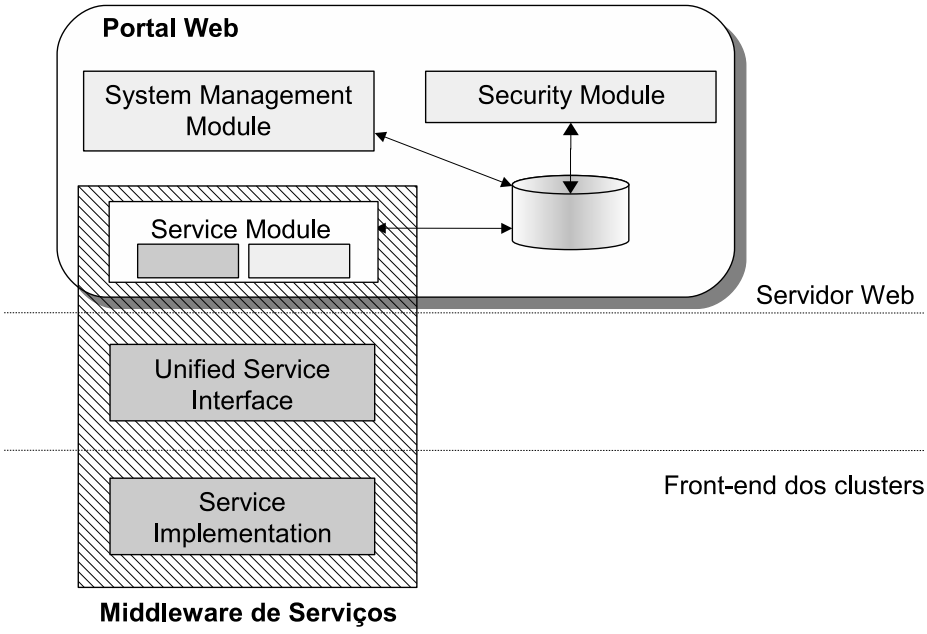

Figura 3.2: Arquitetura ICE

3.2.1 *Middleware* de Serviços

No *Midleware* de Serviços (os módulos cinza escuro dentro da área achurada na Figura 3.2) são considerados os aspectos ligados a arquitetura SOA utilizada no ambiente ICE. Esse modelo se caracteriza pela existência de um provedor, um consumidor e um registro de serviços. Mapeando essa estrutura para o contexto do ambiente ICE têm-se como provedor de serviços os módulos SI nos *front-ends* dos *clusters*, os consumidores são os módulos SM e o registro de serviços está no repositório do Portal Web. Segundo a definição da arquitetura SOA (MCGOVERN; et al., 2003; ALONSO; et al., 2004) é preciso que exista uma forma de descrever esses serviços. Essa descrição, no escopo do ambiente ICE é o módulo USI. Os módulos que compõem o *Middleware* de Serviços são estruturas conceituais. Através da instanciação das mesmas é que pode-se efetivamente definir e disponibilizar funcionalidades no ambiente ICE. Por questões de melhor compreensão seus módulos serão apresentados na seguinte ordem: USI, SI e SM.

Ao módulo **Unified Service Interface** (USI) é atribuída a função de servir como elemento de definição da interface do serviço a ser fornecido por um *cluster*. A definição de uma interface implica na especificação das operações desse serviço, de seus parâmetros e da semântica que lhe será atribuída. Através de uma USI pode-se uniformizar o acesso a diferentes ferramentas que possuam a mesma funcionalidade. O processo de criação de uma USI consiste na análise de ferramentas semelhantes, que são tipicamente utilizadas no cenário de *clusters*, e na descoberta de características em comum entre elas. Esse processo é necessário para que o ambiente ICE mantenha compatibilidade com os sistemas já empregados nos *clusters* e que não se criem operações e parâmetros que não poderão ser mapeados para as ferramentas existentes nos mesmos. A intersecção gerada por essa análise deve conter pelo menos um conjunto mínimo de operações e parâmetros utilizadas pelos usuários dessa funcionalidade e ao mesmo tempo deve conter o máximo de características em comum das ferramentas analisadas. Todo este processo de análise e definição pode se tornar bastante empírico dependendo do tipo de funcionalidade que se está tentando especificar. Como um dos objetivos do ambiente ICE é a utilização de padrões e especificações já existentes, recomenda-se que a definição de USIs seja baseada em alguma especificação em vigor.

O módulo **Service Implementation** (SI) é o elemento que provê os serviços dentro da arquitetura ICE. Para que se possa modelar um SI é preciso que exista uma USI definida. Para uma mesma USI pode-se definir diferentes SIs, isto é, para cada ferramenta com a mesma funcionalidade pode-se implementar um SI. É importante ressaltar que existem situações nas quais não é possível haver o mapeamento direto entre as operações de uma USI para os comandos e parâmetros da ferramenta que está sendo integrada no ambiente. Neste caso, cabe ao módulo SI tratar essas situações, fazendo as adaptações necessárias sem que elas interfiram no funcionamento da ferramenta em si. Não interferir no funcionamento é um requisito que deve ser seguido para que se possa atingir a capacidade de integração de sistemas legados que o ambiente visa. A combinação de USI mais SIs garante a capacidade de extensibilidade de ferramentas visada pelo ambiente ICE.

O *Middleware* de Serviços ainda é formado pelo **Service Module** (SM). Como apresentado anteriormente, o SM está presente tanto no Portal Web quanto nesse componente. Aqui, ele assume as responsabilidades de consumidor de serviços. Da mesma forma como a construção do SI depende de uma USI, o SM também deve

seguir a USI definida para a funcionalidade em questão. A caixa preenchida em cinza escuro no módulo *Service Module* da Figura 3.2 ilustra a diferença entre as funções que ele assume em cada componente. A porção consumidora de serviços do SM monta a requisição para uma determinada operação através das informações recebidas pela parte gráfica do SM, pertencente ao Portal Web. Ao receber a resposta do provedor de serviços, deve tratar as informações entregando-as para serem novamente apresentadas na interface Web. Para cada USI é preciso o desenvolvimento de somente um SM a ser integrado no Portal Web, pois a forma de acesso das operações deve ser uniforme e transparente.

3.2.2 Portal Web

Portal Web é o componente cujo foco principal é encapsular o acesso às funcionalidades dos *clusters* através de um único ponto. Por questões de flexibilidade e acessibilidade decidiu-se que esse componente será uma aplicação Web baseada no modelo de três camadas (ALONSO; et al., 2004) e no modelo MVC (*Model View-Controller*) (HANSEN; FOSSUM, 2005). Através desses modelos é possível, respectivamente, concentrar a lógica do sistema em um *middleware*, resguardando as informações e recursos do sistema; e também separar a lógica do sistema de sua apresentação. Como ilustrado na Figura 3.2, o Portal Web é composto por três módulos, os quais são descritos abaixo.

O **Security Module** (SecM) trata das questões de autenticação e de autorização dos usuários do ambiente ICE. Somente usuários autenticados podem utilizar esse ambiente. Além disso, esses usuários possuem restrições de acesso. Eles podem acessar somente áreas nas quais estão autorizados para tanto. O processo de autorização está ligado à idéia de perfis e grupos de usuários. A primeira idéia está relacionada a questões de autorização no Portal Web, enquanto a segunda está ligada aos *front-ends* dos *clusters* e conseqüentemente ao *middleware* de serviços. Maiores informações sobre o esquema de segurança definido para o ambiente ICE podem ser encontradas em (ILHA, 2005).

O **System Management Module** (SMM) foi definido no intuito de concentrar as atividades de gerenciamento e configuração do próprio ambiente ICE. Ele possui as seguintes atribuições:

- gerenciar os perfis dos usuários, isto inclui a definição de perfis e das permissões de acesso às áreas do ambiente ICE que cada tipo de perfil possuirá;
- coordenar o processo de autenticação e autorização dos usuários, para tanto serão utilizadas as estruturas de segurança definidas no módulo SecM;
- gerenciar as informações dos *clusters* que estão sob controle do ambiente ICE;
- manipular as funcionalidades oferecidas pelo ICE, isto implica a inserção, remoção e alteração das funcionalidades existentes no ambiente; o que impacta diretamente nos módulos de serviços (SM);
- gerenciar os relacionamentos entre usuários, *clusters* e funcionalidades desses *clusters*;
- tratar da apresentação das páginas web, uma vez que as páginas são geradas dinamicamente de acordo com (i) o perfil dos usuários, (ii) os *clusters* que eles

têm acesso e ainda (iii) quais funcionalidades disponibilizadas nesses *clusters* que eles podem acessar.

Como destacado acima, a idéia de perfil de usuários é um ponto relevante dentro do Portal Web. É através dos perfis que se definem as políticas de acesso do ambiente ICE. Essas políticas estão ligadas a permissões: de gerenciamento do sistema como um todo, de gerenciamento das informações dos *clusters* inseridos no ambiente, de autorização para utilização de *clusters* e de suas funcionalidades. A Tabela 3.1 apresenta os perfis atualmente definidos no ambiente ICE.

Tabela 3.1: Descrição dos perfis atualmente definidos no ambiente ICE

Perfil	Descrição das capacidades dentro do ICE
ICE_ROOT	Gerenciamento de usuários, *clusters* e funcionalidades (inserção, remoção e alterações) Especificação das funcionalidades dos *clusters* (SM)
CLUSTER_ROOT	Gerenciamento dos usuários dos *clusters* Associação de funcionalidades a *clusters* Associação de usuários a *clusters* Associação de usuários às funcionalidades dos *clusters* Manutenção dos relacionamentos entre usuários, *clusters* e funcionalidades
USER	Utilização das funcionalidades associadas ao usuário

No ambiente ICE os perfis são atribuídos para cada usuário em relação a cada *cluster* que ele tem permissão de utilização. A exceção é o perfil ICE_ROOT, pois ele não está ligado a nenhum *cluster* e pode ser associado a qualquer usuário que possuir as permissões de administrador do ambiente ICE. Através da abordagem de associação de perfis de acordo com os *clusters* acessíveis, é possível que um mesmo usuário tenha diferentes perfis entre os diferentes *clusters*. Também é possível que no mesmo *cluster* esse usuário apresente diferentes perfis. Por exemplo, considerando os usuários apresentados na Figura 3.1, é possível que o *Usuário A* possua o perfil CLUSTER_ROOT no *Cluster 2* e o perfil USER no *Cluster 1*; mas nada impede que ele também possua o perfil USER também no *Cluster 2*. Enfim, de forma resumida pode-se dizer que o perfil ICE_ROOT está relacionado com as ações de configuração do ambiente ICE, o perfil CLUSTER_ROOT está ligado às questões de preparação das informações dos *clusters* para serem integrados no ambiente ICE, enquanto o perfil USER está relacionado com as funcionalidades, ou seja SMs que os usuário terão acesso.

O **Service Module** (SM) é um módulo conceitual, cujo objetivo é servir de *framework* para a incorporação de funcionalidades no sistema. Como apresentado anteriormente, esse módulo está presente também no *Middleware* de Serviços. A caixa cinza mais clara no módulo *Service Module* da Figura 3.2 ilustra a porção pertencente ao Portal Web desse módulo. Nele, o SM assume as atribuições de prover as abstrações gráficas e facilidade de uso para os usuários da funcionalidade em questão. Esta porção gráfica é criada através dos parâmetros e operações existentes na USI definida para a funcionalidade.

3.2.3 Cenário de Aplicação da Arquitetura ICE

Para que se possa compreender melhor como cada módulo da arquitetura ICE é empregado e como ela pode ser estendida a Figura 3.3 ilustra um cenário de utilização do ambiente ICE.

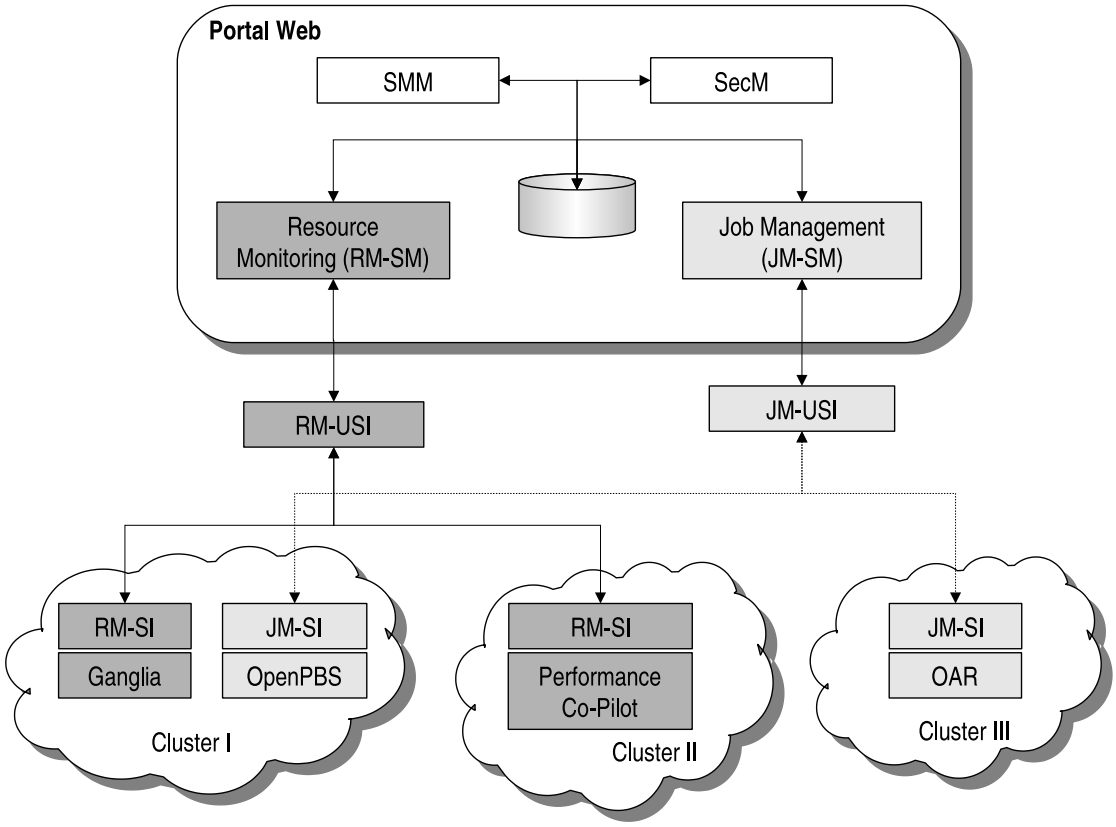

Figura 3.3: Instanciação da Arquitetura ICE

Na Figura 3.3 estão definidos dois módulos de serviço: *Resource Monitoring* (RM-SM) e *Job Management* (JM-SM). Esses módulos seguem, respectivamente, as interfaces de serviços definidas em RM-USI e JM-USI. No contexto de utilização apresentado na Figura 3.3 existem três *clusters*. O *Cluster I* possui suporte para as duas funcionalidades, enquanto o *Cluster II* possui suporte apenas para a funcionalidade de *Resource Monitoring* e o *Cluster III* apenas para a funcionalidade de *Job Management*.

Como em um cenário comum em HPC, cada *cluster* apresenta o seu conjunto de ferramentas. Para que se possa integrar esses *clusters* no ambiente de gerenciamento ICE é preciso preservar esta diversidade. A aplicação da arquitetura ICE nesse contexto garante essa característica. A Figura 3.3 mostra que para as mesmas funcionalidades é possível integrar ferramentas diferentes em *clusters* distintos. Cada *cluster* que provê uma funcionalidade (serviço) tem que ter um SI implementado de acordo com a USI da funcionalidade em questão. Desta forma, existe para a RM-USI o módulo RM-SI no *Cluster I* e no *Cluster II*, enquanto para a JM-USI existem dois SIs, localizados no *Cluster I* e *Cluster III*.

A motivação para apresentação da Figura 3.3 é ilustrar o potencial de extensibilidade, transparência e uniformização que a arquitetura do ambiente ICE proporciona

no gerenciamento e acesso de múltiplos *clusters*. O objetivo final do ambiente ICE é prover serviços de: monitoramento de recursos, monitoramento de aplicações, gerenciamento de aplicações, depuração de aplicações e gerenciamento de configuração de *clusters*. Sendo que essa não é uma lista fechada de funcionalidades que se pretende inserir nesse ambiente.

3.3 Resumo

O ambiente ICE é uma plataforma de gerenciamento e de acesso a múltiplos *clusters*. Os principais objetivos desse ambiente são prover: (i) extensibilidade das funcionalidades agregadas na plataforma, e extensibilidade das ferramentas que podem ser integradas; (ii) transparência de acesso e de uso das ferramentas existentes nos *clusters*; (iii) uniformização da forma como as operações das ferramentas dos *clusters* são executadas; (iv) completeza, ou seja, ser capaz de incorporar todos os tipos de serviços que um *cluster* pode disponibilizar; (v) interoperabilidade, garantindo que sistemas legados sejam incorporados ao ambiente; (vi) facilidade de uso, permitindo que os usuários possam interagir com uma interface gráfica amigável.

Para que se consiga atingir estes objetivos a arquitetura do ambiente ICE foi projetada para ser modular e baseada no modelo SOA (*Service Oriented Architecture*). Através desse modelo é possível garantir uma independência entre especificação dos serviços e a forma como eles são implementados. Esse tipo de característica é fundamental para que se consiga prover extensibilidade, transparência e uniformização.

A arquitetura ICE está dividida em dois componentes: *Middleware* de Serviços e Portal Web. O primeiro componente é formado pelos módulos de definição de um serviço (USI - *Unified Service Interface*), provedor de serviços (SI - *Service Implementation*) e consumidor de serviços (SM - *Service Module*). O segundo componente é formado pelos módulos de gerenciamento do próprio ambiente (SMM - *System Management Module*), de segurança do sistema (SecM - *Security Module*) e por um módulo de serviços (SM - *Service Module*). Esse último módulo possui funções nos dois componentes da arquitetura ICE, sendo que no Portal Web ele assume um caráter mais de apresentação gráfica das informações e no *Middleware* de Serviços possui o caráter de consumidor de serviços.

Através da arquitetura proposta é possível o desenvolvimento de um ambiente de gerenciamento e de acesso a múltiplos *clusters* que é capaz de incorporar os sistemas legados dos *clusters*, ser estendido para ser usado com diferentes funcionalidades e ferramentas, ser transparente e de fácil utilização. O próximo capítulo apresenta algumas decisões de implementações tomadas, além de descrever o protótipo implementado para validar a arquitetura proposta.

4 DESENVOLVIMENTO DO PROTÓTIPO DO AMBIENTE ICE

Este capítulo descreve o desenvolvimento do protótipo do ambiente ICE, o qual segue a arquitetura proposta no Capítulo 3. Serão apresentadas as decisões de projeto, os módulos que foram implementados e a forma como esse processo foi realizado.

4.1 Decisões de Projeto

A primeira decisão de projeto a ser considerada foi qual *middleware* deveria ser utilizado para o desenvolvimento da infra-estrutura de serviços. Dado o fato de que o *Middleware* de Serviço segue uma arquitetura SOA foi necessário analisar plataformas compatíveis com essa abordagem. Como apresentado no Capítulo 2, existem alguns *middlewares* para o desenvolvimento de sistemas distribuídos. Dentre estes, escolheu-se CORBA e Web Services para serem analisados visto que eles podem ser considerados os *middlewares* mais representativos (COULOURIS; DOLLIMORE; KINDBERG, 2005).

O segundo aspecto levado em consideração para o desenvolvimento do ambiente ICE está relacionado com a definição de que padrões e especificações poderiam ser incorporadas ao ambiente ICE. Foram analisados: o padrão *de facto* em infra-estrutura para *grid* - Globus Toolkit (GLOBUS - WELCOME TO THE GLOBUS TOOLKIT HOMEPAGE, 2005), e as especificações OGSA (*Open Grid Service Architecture*) e OGSI (*Open Grid Service Infrastructure*) (LI; BAKER, 2005).

A seguir são apresentadas as considerações sobre as análises destes pontos e as decisões tomadas para o desenvolvimento do protótipo do ambiente ICE.

4.1.1 Definição do *Middleware* Empregado

CORBA é um *middleware* amplamente usado na construção de sistemas distribuídos, sendo que inúmeras aplicações já foram implementadas sobre essa plataforma (ALONSO; et al., 2004). Apesar de não apresentar uma arquitetura SOA, CORBA possui algumas características da mesma, como por exemplo: prover interoperabilidade (mesmo que em contextos menores (COULOURIS; DOLLIMORE; KINDBERG, 2005)), sistema de descoberta de serviços e capacidade de isolar o funcionamento dos objetos de sua API. Essa última característica é fundamental em uma arquitetura SOA (como visto na Seção 3.2). Entretanto, CORBA não permite que os usuários acessem a lógica dos negócios facilmente através da Internet (MCGOVERN; et al., 2003). A tecnologia Web Services, por sua vez, é um *middleware*

baseado em uma arquitetura SOA. Ela não apresenta todas as características dessa arquitetura (MCGOVERN; et al., 2003), mas provê suporte para grande parte delas, ou pelo menos para as mais importantes.

A comparação rápida entre CORBA e Web Services apresentada em (COULOURIS; DOLLIMORE; KINDBERG, 2005) ilustra as principais diferenças entre esses *middleware*. A Tabela 4.1 mostra essas diferenças.

Tabela 4.1: Comparação entre CORBA e Web Services

Aspecto	CORBA	Web Services
Nomeação	Serviço de nomeação CORBA	DNS
Formato da Referência	IOR (*Internet-wide Object Reference*)	URL
Ativação e Localização	Integrados	Separados
Facilidade de Uso	Software complexo que requer instalação e suporte	Infra-estrutura já instalada (HTTP + XML)
Eficiência	Definido para ser eficiente (dados binários)	Menos eficiente (dados textuais)

O primeiro ponto que os distingue é quanto ao esquema de nomeação utilizado. CORBA baseia-se em uma estrutura de nomeação própria, enquanto Web Services utiliza DNS para localização dos seus serviços. Essa é uma diferença importante, pois a adoção de Web Services não requer a inserção de novas estruturas ou formatos para identificação de serviços. Além disso, por ser baseado em DNS, Web Services possuem toda uma infra-estrutura de nomeação legada para serem utilizados sem maiores problemas na Internet (visto que DNS é um serviço que funciona efetivamente nesse meio). O segundo aspecto está relacionado à forma como os serviços são referenciados. Através da Tabela 4.1 percebe-se que Web Services seguem o padrão utilizado na Internet, enquanto CORBA emprega um esquema próprio. Com relação a ativação e localização dos serviços, Web Services trata esses eventos de forma separada, enquanto em CORBA esse processo é realizado em um mesmo instante. O aspecto de facilidade de uso é um ponto que os diferencia significantemente. Para utilização de Web Services não é preciso a instalação e configuração de softwares complexos e grandes. Como ele é baseado em padrões da Internet, a sua infra-estrutura já está normalmente instalada na maioria (ou totalidade) das máquinas. Em contra-partida, para utilização de CORBA é preciso a instalação de softwares específicos e que não são disponibilizados de forma padrão nas máquinas. Quanto ao quesito eficiência, CORBA possui um desempenho melhor visto que utiliza formatos binários, enquanto Web Services empregam XML que é um formato textual. Atualmente existem iniciativas da criação de XML binário (W3C-XML BINARY - XML BINARY CHARACTERIZATION WORKING GROUP PUBLIC PAGE, 2005), o que solucionaria o défiti de eficiência que Web Services possui.

Considerando os aspectos ressaltados pela comparação direta entre estes *middlewares*, os objetivos que se pretende atingir com o ambiente ICE e a arquitetura desse ambiente, optou-se pela utilização de Web Services como infra-estrutura para o desenvolvimento do *Middleware* de Serviço. A tecnologia Web Service não é unitária, ela é formada por tecnologias como por exemplo, SOAP para a construção das men-

sagens a serem trocadas entre os provedores e os consumidores de serviços, WSDL para a descrição dos serviços e UDDI para a localização dos serviços. Neste trabalho utilizou-se apenas a tecnologia SOAP para a troca de informações entre as entidades envolvidas.

4.1.2 Análise dos Padrões e Especificações

Como apresentado no Capítulo 3, uma das filosofias do ambiente ICE é a utilização de padrões e especificações disponíveis. Na área de gerenciamento e de acesso a múltiplos *clusters* não existem trabalhos pontuais que busquem um certo grau de padronização de métodos, ferramentas, etc. Entretanto, na área de *grids* existem muito esforços neste sentido. Alguns exemplos de grupos que trabalham com este foco são: GGF (*Global Grid Force*) (GLOBAL GRID FORUM, 2005), Globus Aliance (THE GLOBUS ALLIANCE, 2005), OASIS (OASIS - ADVANCING E-BUSINESS STANDARDS SINCE 1993, 2005), W3C (W3C WORLD WIDE WEB - LEADING THE WEB TO ITS FULL POTENTIAL..., 2005), entre outros. Os dois últimos grupos apresentados não colaboram diretamente para área de *grids*, mas os padrões gerados por eles são muitas vezes empregados nesta área.

O gerenciamento e o acesso de múltiplos *clusters* possui características em comum com a área de *clusters*. Ambas têm que lidar com: múltiplos domínios administrativos, heterogeneidade de sistemas e ferramentas, localização dos recursos, segurança no acesso a esses recursos, entre outros. Entretanto, a forma como estes aspectos são tratados nestes escopos é diferente. No gerenciamento e acesso a múltiplos *clusters* os usuários sabem exatamente que recursos eles estão utilizando, podem não saber onde eles estão, mas sabem quem são estes recursos. Em *grids* os usuários não sabem quem são e nem onde estão os recursos que eles solicitaram. Essa pequena diferença faz com que, no cenário de *grids*, sejam necessários grandes mecanismos de controle para prover esse tipo de transparência aos seus usuários. Outra diferença que também implica em níveis de controle mais pesados, é a dinamicidade de um cenário de *grid*. Neste, os recursos são voláteis, ou seja, as máquinas que fazem no *grid* em um determinado momento podem não fazer mais parte no momento seguinte, enquanto que em um cenário de múltiplos *clusters* esses recursos são estáticos. Mesmo com essas diferenças estruturais ainda é possível adaptar as idéias da área de *grid* para o contexto do ambiente ICE. Além disso, pode-se tentar incorporar ferramentas já existentes no cenário de *grid* no ambiente ICE. Seguindo esta idéia analisou-se a possibilidade de utilização de módulos do Globus Toolkit dentro do ambiente ICE.

O Globus Toolkit (GT) tem sido desenvolvido desde o final dos anos 90 com o objetivo de dar suporte para o desenvolvimento de aplicações e infra-estruturas de sistemas distribuídos orientados a serviço. Alguns componentes básicos do GT estão relacionados com: segurança, gerenciamento e acesso de recursos, migração e gerenciamento de dados, descobrimento de recursos, entre outros. Atualmente o GT está em sua versão 4, a qual é toda baseada em Web services. A Figura 4.1 apresenta os módulos que compõem o GT4.

Seguindo a idéia de aproveitar o que já existe na área de alto desempenho para ser integrado ao ambiente ICE, pensou-se na utilização do módulo de alocação e gerenciamento de recursos - GRAM (*Grid Resource Allocation and Management*). Esse módulo do GT4 é responsável pela submissão de processos no *grid*. Ele faz o interfaceamento com os escalonadores de aplicações instalados nos recursos com-

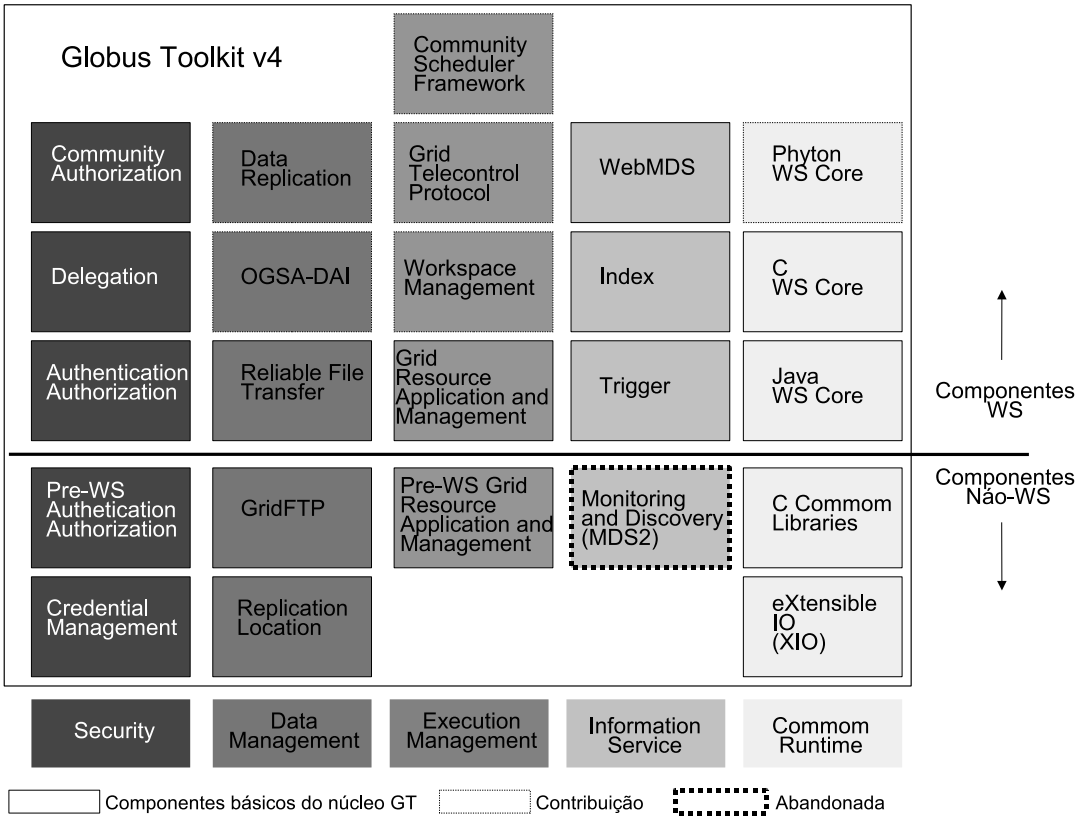

Figura 4.1: Arquitetura do Globus Toolkit v4 - Figura retirada do *draft* (GLOBUS Toolkit 4.0 RELEASE MANUALS, 2005)

putacionais. Entretanto analisando o toolkit, percebeu-se que não é possível isolar este módulo e integrá-lo no ambiente ICE. Isto não é possível pois para utilizar o GRAM é preciso ter todo o núcleo de componentes básicos do GT4. Eles servem, de forma geral, para prover a infra-estrutura de controle para gerenciamento das informações que mantêm o recurso participando do *grid*. Isto implica na inserção de comunicações, protocolos e outras abstrações que não são necessárias no contexto de *clusters*. Sendo assim, para evitar essa sobrecarga desnecessária decidiu-se não integrar o GRAM no ambiente ICE.

Da mesma forma como ocorre no caso do GRAM, os outros módulos do GT também só podem ser utilizados caso a infra-estrutura básica do GT esteja presente. Isto faz com que a utilização do GT para evitar novas implementações não seja apropriado no contexto do ambiente ICE. No entanto, as idéias por trás deste *middleware* para *grid* podem ser mapeadas para *clusters*, como por exemplo o esquema de segurança utilizado no GT, o qual é baseado no GSI (*Grid Security Infrastructure*) (WELCH; et al., 2003) - atualmente um padrão do GGF.

Outro exemplo de adaptação possível é utilizar os conceitos e especificações do OGSA (*Open Grid Service Architecture*) (FOSTER; et al., 2002; ABBAS, 2004) implementando-os para o cenário de múltiplos *clusters*. Através da utilização das especificações, como por exemplo GMA (*Grid Monitoring Architecture*), é possível desenvolver um sistema para múltiplos clusters que segue interfaces bem definidas. Se no futuro o ambiente ICE for migrado para um contexto de *grid*, a interface de seus serviços já estará de acordo com os padrões utilizados.

4.2 Descrição do Protótipo Implementado

A partir das decisões de projeto tomadas, principalmente da definição de que *middleware* seria usado para o desenvolvimento dos módulos de serviço, partiu-se para a definição de quais componentes fariam parte do protótipo.

O protótipo resultante deste trabalho possui a implementação de uma instância do *Middleware* de Serviços, ou seja, foi disponibilizada uma funcionalidade de *cluster*, e foi implementado o módulo de gerenciamento do sistema (SMM) no Portal Web. O módulo de segurança (SecM) projetado na arquitetura ICE foi implementado no trabalho de Conclusão de Curso do aluno Alexandre Ilha (ILHA, 2005).

Como descrito na Seção 3.2.3, estão previstas várias funcionalidades dentro do ambiente ICE. No contexto deste trabalho, foi implementada a funcionalidade de gerenciamento de aplicações (*Job Management*). No entanto, existem trabalhos de outros participantes do GPPD (Grupo de Processamento Paralelo e Distribuído) de desenvolvimento de outros módulos do *Middleware* de Serviços. Exemplos dessas iniciativas são os serviços para: monitoramento de aplicações e de recursos, e para configurações administrativas dos *clusters*. A Figura 4.2 apresenta a estrutura atual do ambiente ICE. Os módulos implementados no escopo deste trabalho são ilustrados pelas caixas mais escuras, enquanto os desenvolvidos por outras pessoas são ilustrados pelas caixas mais claras.

Além de apresentar os módulos do ambiente ICE implementados, a Figura 4.2 também indica os *clusters* que foram integrados dentro do ambiente ICE: `gppd` e o `frontal-minuano`. Como indicado na figura, cada *cluster* possui um tipo de ferramenta de gerenciamento. Essas ferramentas são, respectivamente: OpenPBS (*Open Portable Batch System*) (PORTABLE BATCH SYSTEM, 2005) e OAR (CAPIT; et

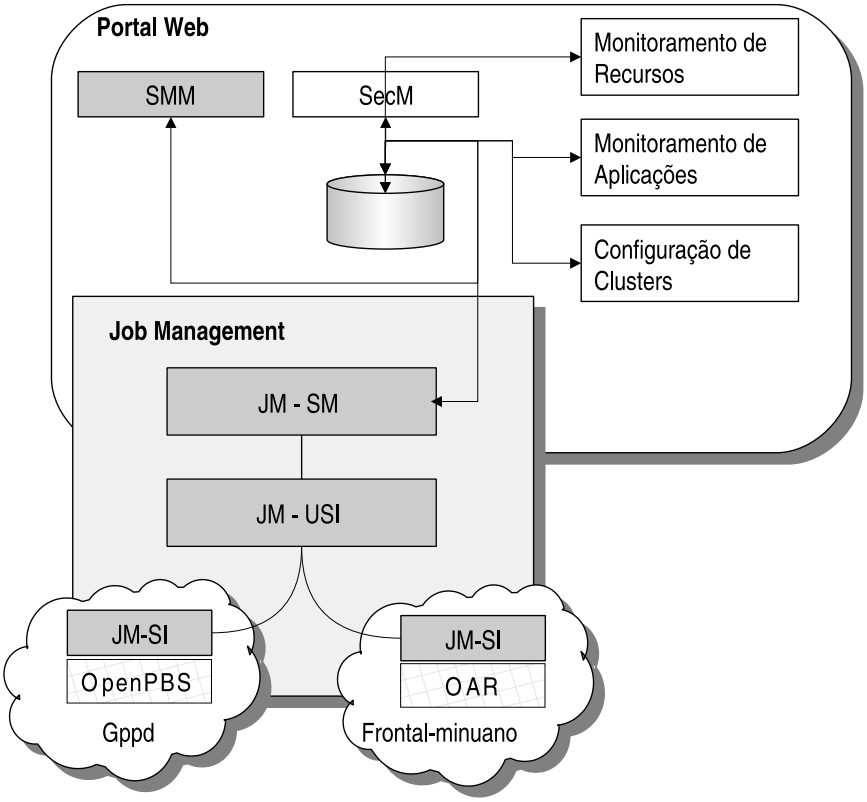

Figura 4.2: Protótipo ICE Implementado

al., 2005). Estes *clusters* pertencem ao GPPD.

4.2.1 Tecnologias Empregadas

Para implementação do protótipo foram adotadas as tecnologias descritas abaixo, em cada um dos componentes da arquitetura.

- *Middleware* de Serviço - Para o desenvolvimento dos Web services optou-se pela utilização da tecnologia Apache Axis (versão 2) (APACHE AXIS2, 2005). Essa escolha também levou à utilização do *container* Tomcat (versão 5.0.30) (APACHE TOMCAT - THE APACHE SOFTWARE FOUNDATION, 2006) e do Java (j2sdk-1.4.2) (JAVA PLATFORM, STANDARD EDITION , JAVA SE). Estas últimas tecnologias foram empregadas pois são pré-requisitos para utilização do Axis2. É importante ressaltar que o desenvolvimento do *Middleware* de Serviço poderia ter utilizado qualquer outra tecnologia, uma vez que Web services são independente de plataforma e linguagem.

- Portal Web - Esse componente possui três pontos distintos: apresentação das informações através de páginas Web, controle e separação da lógica do sistema e o papel de consumidor de serviços. A apresentação foi implementada através da tecnologia JSP (*Java Server Pages*) (JAVASERVER PAGES TECHNOLOGY, 2006). A lógica do sistema está separada das páginas Web devido a utilização de Java Servlets (JAVA SERVLET TECHNOLOGY, 2006). As páginas JSP não lidam diretamente com a inteligência do sistema, elas fazem requisições para Servlets, os quais interagem com os recursos e serviços ne-

cessários. E para finalizar, os consumidores dos módulos de serviço (SM) são implementados utilizando Apache Axis (versão 2) e também não são acessados diretamente pelas páginas JSP. Este acesso também é feito através de Servlets. Para o armazenamento das informações foi utilizado o SGBD (Sistema de Gerenciamento de Base de dados) PostgreSQL versão 8.1 (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2006).

A seguir, a implementação de cada componente da arquitetura ICE é descrita em detalhes. São apresentadas as características de cada módulo, suas funcionalidades e especificidades, assim como as interações entre esses módulos. Inicialmente será descrito o *Middleware* de Serviço implementado para funcionalidade de gerenciamento de aplicações *Job Management* e em seguida será detalhada a implementação do Portal Web.

4.2.2 Middleware de Serviço para Gerenciamento de Aplicações

Neste trabalho o *Middleware* de Serviços foi instanciado para que fosse possível a disponibilização da funcionalidade de gerenciamento de aplicações junto ao ambiente ICE. Primeiramente será descrita a USI definida para o gerenciamento de aplicações (*Job Management Unified Service Interface* - JM-USI), em seguida serão apresentados os provedores de serviços desenvolvidos para essa interface (*Job Management Service Implementation* - JM-SI). Finalmente serão descritas as funções referentes ao consumidor dos serviços desta funcionalidade (*Job Management Service Module* - JM-SM).

4.2.2.1 JM-USI - Job Management Unified Service Interface

A USI para o gerenciamento de aplicações foi definida considerando alguns parâmetros tipicamente utilizados pelos usuários de *clusters*. Para se chegar nestes parâmetros buscou-se suportar os parâmetros utilizados pelo GRAM do GT4 (GLOBUS Toolkit 4.0 RELEASE MANUALS, 2005; GT 4.0 WS_GRAM, 2005). De certa forma, pode-se dizer que a API disponibilizada pelo GRAM tenta prover as principais operações e parâmetros existentes nos escalonadores de aplicações. Alguns dos principais parâmetros utilizados no GRAM foram incorporados na funcionalidade de gerenciamento de aplicações do ambiente ICE.

Para finalizar a análise das possibilidades de operações e parâmetros que podem ser disponibilizados em uma interface uniformizada, foram estudadas as ferramentas de escalonamento: OpenPBS, SGE, CCS e OAR. Essas ferramentas foram escolhidas pois estão disponíveis nos *clusters* do GPPD e de instituições de pesquisas parceiras do grupo. Desta forma, essa infra-estrutura poderia ser utilizada no momento da avaliação do protótipo que foi desenvolvido. Através da análise dessas ferramentas, foi possível ter uma noção melhor das facilidades que elas ofereciam e poder ter um maior conhecimento acerca do que poderia ser disponibilizado aos usuários do ambiente ICE que utilizam a funcionalidade de gerenciamento de aplciações.

Sendo assim a definição da USI para esta funcionalidade foi baseada nas características disponibilizadas pelo *toolkit* de *grid* mais utilizado - o Globus Toolkit - e nas potencialidades oferecidas pelos escalonadores estudados. Como resultado têm-se a JM-USI denominada `JobManagementProviderInterface`. A Tabela 4.2 descreve as operações que fazem parte desta USI.

Uma questão que passa por todas as operações da JM-USI é a segurança. Não

Tabela 4.2: Operações da JM-USI

Operação	Descrição/Função
jobSubmissionOperation	Submissão de aplicações
jobFinalizationOperation	Finalização de aplicações
jobStatusVerificationOperation	Verificação do estado das aplicações, o que pode corresponder a: finalizada, em execução, em espera, ou então ainda na fila para ser submetida
jobOutputRetrievalOperation	Recuperação da saída de uma aplicação (padrão ou de erros)
submissionScriptVerificationOperation	Apresentação do script de submissão da aplicação de acordo com o escalonador do *cluster* em uso

é possível permitir que usuários não autenticados e não autorizados utilizem as operações dos serviços disponibilizados nos *front-ends* dos *clusters*. De forma resumida pode-se descrever como medidas de segurança do ambiente ICE a utilização de HTTPS para as comunicações; o emprego de mecanismo de mapeamento entre as identidades dos usuários, os quais podem assumir uma identidade no ambiente ICE e diferentes identidades nos *clusters* aos quais possuem acesso; e mecanismos de autenticação mútua entre o Portal ICE e o *front-end* que está sendo acessado. A construção do módulo de segurança segue a definição do *Middleware* de Serviços, isto é, foram definidos os elementos: USI, SI e SM.

Estas questões de segurança interferem na definição dos parâmetros das operações de qualquer serviço que for criado para ser integrado ao ambiente ICE. Desta forma, independentemente da funcionalidade que se está inserindo no ambiente, é preciso que todas as suas operações possuam a identificação do usuário que a está solicitando. Todo o processo de autenticação e autorização é realizado pelo serviço de segurança definido e desenvolvido para o ambiente ICE. Essa característica mostra a modularidade que este ambiente possui.

A seguir os parâmetros de cada uma das operações da JM-USI são descritos nas tabelas abaixo.

- **jobSubmissionOperation()**

Parâmetros de entrada

username:	Contém a identificação do usuário. Está relacionado com o sistema de segurança de cada *cluster*, mas na interface do serviço JM-USI este fator é transparente.
basedir:	Informa o diretório a partir de onde as aplicações e os arquivos do usuário serão encontrados durante o processo de submissão da aplicação.
param:	Possui os parâmetros que devem ser passados para a aplicação.
main_appname:	Representa o nome da aplicação que deve ser executada.
numnodes:	Indica o número total de nós que devem ser reservados.
numproc:	Possui o número de processos que devem ser lançados. Esse parâmetro deve ser utilizado no lançamento de aplicações paralelas.
queue:	Indica a fila onde a aplicação deve ser lançada. De acordo com o tipo de escalonador pode assumir outra semântica que não a de fila, mas continuará indicando o local a partir de onde a aplicação será lançada.
walltime:	Representa o tempo da reserva dos nós.
joinout:	Especifica se a saída padrão e a de erro devem ser colocadas em um único arquivo.
outfile:	Contém o nome do arquivo para onde será redirecionada a saída padrão.
errfile:	Possui o nome do arquivo para onde será redirecionada a saída de erro padrão.

jobtype:	Especifica que tipo de aplicação será submetida. Elas podem ser seqüenciais ou paralelas. No caso das aplicações paralelas elas ainda são identificadas de acordo com o tipo de biblioteca paralela utilizada.
numbatchiteration:	É possível configurar a execução da mesma aplicação um determinado número de vezes. Isto é, existe a possibilidade de disparar uma aplicação para ser executada como um *pool* de execuções. O número de vezes que a aplicação será executada é designado por este parâmetro.
multiplescripts:	Serve como complemento do parâmetro anterior. Indica se as várias execuções da mesma aplicação devem ser lançadas como *jobs* independentes para o escalonador ou não. Isto é, deve-se escolher entre gerar tantos scripts quantas foram as iterações, ou então se deve ser gerado apenas um script, e as repetições da execução da mesma aplicação devem ser gerenciadas dentro desse script único. É importante ressaltar que essa definição tem impacto direto sob o parâmetro relativo ao tempo da reserva que está sendo solicitada para a aplicação (*walltime*). Em caso de geração de apenas um único script, é preciso que o tempo da reserva leve em consideração o tempo necessário para o término de todas as iterações.

Parâmetros de saída

jobid:	Contém a identificação da aplicação que foi submetida.
errorcode:	Campo destinado a conter o valor de qualquer tipo de erro que ocorra durante o processamento e a montagem do script de submissão da aplicação, assim como qualquer tipo de erro ocorrido no momento da submissão da aplicação através do escalonador utilizado no *cluster*.

- **finalizeJobOperation()**

Parâmetros de entrada

username:	Idêntico ao parâmetro da operação de submissão.
jobid:	Identificação da aplicação que deve ser finalizada.

Parâmetros de saída

fin_output:	Possui a saída, em modo texto, da ordem de finalização da aplicação
errorcode:	Campo destinado a conter o valor de qualquer tipo de erro que ocorra durante a ação de finalização da aplicação através do escalonador utilizado no *cluster*.

- **verifyJobStatusOperation()**

Parâmetros de entrada

username:	Idêntico ao parâmetro da operação de submissão.

Parâmetros de saída

status:	Contém o estado em que as aplicações se encontram.
errorcode:	Campo destinado a conter o valor de qualquer tipo de erro que ocorra durante a ação de verificação do estado da aplicação.

- **jobOutputRetrievalOperation()**

Parâmetros de entrada

username:	Idêntico ao parâmetro da operação de submissão.
basedir:	Informa o diretório a partir de onde os arquivos referentes ao *job* solicitado estão armazenados.
file:	Indica no nome do arquivo que deve ser recuperado.

Parâmetros de saída

archive:	Contém os dados do arquivo solicitado.
errorcode:	Campo destinado a conter o valor de qualquer tipo de erro que ocorra durante a ação de recuperação dos dados do arquivo solicitado.

- **verifySubmissionScriptOperation()**

Parâmetros de entrada - Possui exatamente os mesmo parâmetros de entrada que a operação de submissão.

Parâmetros de saída

script:	Contém os dados que formam o script de submissão de uma aplicação, de acordo com o escalonador em uso no *cluster*.
errorcode:	Campo destinado a conter o valor de qualquer tipo de erro que ocorra durante a geração do script de submissão.

Analisando rapidamente a definição da JM-USI pode-se pensar em igualá-la ao WSDL do serviço de gerenciamento de aplicações. Entretanto esta equiparação não pode ser feita. Pela definição da arquitetura ICE, uma USI é a definição de uma funcionalidade, ou seja, de um serviço, a qual possui as operações e os parâmetros do mesmo. WSLD é a linguagem de descrição de um Web Service, a qual possui sua sintaxe baseada em XML e *tags* específicas que informam, por exemplo, a forma como será realizada a comunicação (*one-way* ou em estilo RPC) e ainda o endereço do provedor do serviço. Essas duas características do WSDL são os pontos que o diferem completamente de uma USI. Outro ponto que os distingue é o fato da JM-USI não estar atrelada à tecnologia de Web Services, ela pode ser mapeada para qualquer tecnologia que possua características de arquiteturas SOA.

Através das operações apresentadas acima e de seus parâmetros é possível realizar todas as ações básicas de gerenciamento de aplicações. A JM-USI definida disponibiliza aos seus usuários facilidades para lançamento de aplicações. Exemplos dessas facilidades são o lançamento de múltiplas execuções da mesma aplicação e o *upload* de arquivos.

É importante ressaltar que podem ocorrer casos em que as operações e os parâmetros definidos na JM-USI não possam ser diretamente mapeados para as características dos escalonadores e gerenciadores utilizados nos *clusters*. Esse tipo de situação deve ser tratada pelas implementações específicas da JM-USI, ou seja, devem ser tratadas em cada implementação do módulo JM-SI.

4.2.2.2 JM-SI - Job Management Service Implementation

Neste ponto serão detalhadas as implementações do serviço de gerenciamento de aplicações (JM-SI) para os escalonadores OpenPBS e OAR. Além disso será apresentado o *framework* definido para dar suporte no desenvolvimento dos diferentes JM-SIs. Esse *framework* garante a característica de extensibilidade, que poderá ser comprovada durante a descrição das implementações específicas.

O ***Job Management Framework*** **(JMF)**, nome atribuído ao *framework* definido, pode ser visualizado na Figura 4.3. Essa figura ilustra o diagrama de classes, baseado na modelagem UML (UML - UNIFIED MODELING LANGUAGE, 2006), que formam a estrutura básica da implementação do serviço de gerenciamento de aplicações.

Como pode-se perceber na Figura 4.3 existem classes e pacotes já definidos, que servem como base para as implementações futuras e existem as classes que devem ser implementadas. Para tanto é necessário que se utilize os mecanismos de herança presentes no modelo de programação Orientado a Objetos. A seguir cada pacote e classe do JMF serão descritas e contextualizadas.

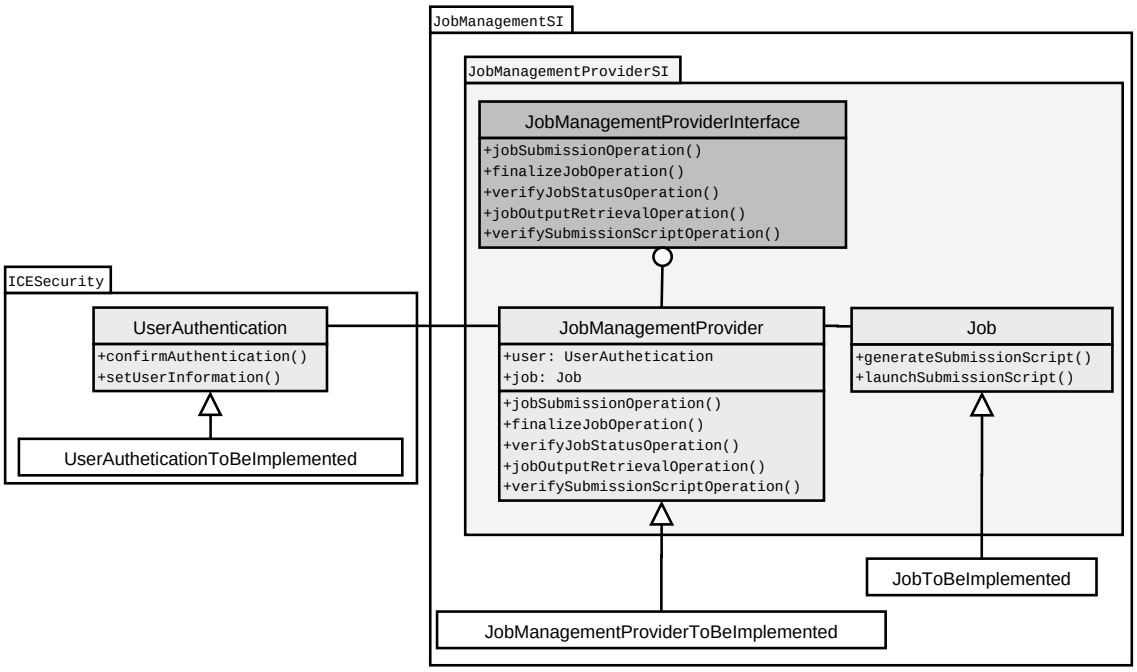

Figura 4.3: JMF - Job Management Framework

Antes de permitir que qualquer usuário tenha acesso para utilizar os recursos do serviço, é preciso garantir requisitos de segurança, como por exemplo, a autenticação do usuário que está solicitando o serviço. Este procedimento de verificação de identidade do usuário não está restrito apenas ao serviço de gerenciamento de aplicações. Na verdade ele deve ser acionado sempre que um serviço for solicitado. No intuito de garantir que somente usuários autenticados estarão utilizando os serviços do ambiente ICE foi definido o pacote `ICESecurity`. Esse pacote é formado pela classe abstrata `UserAuthentication`, que possui os métodos abstratos `setUserInformation()` e `confirmAuthentication()`. Optou-se por definir essa classe e seus métodos como abstratos para que fosse possível extendê-la e implementar seus métodos de acordo com o sistema de autenticação de cada *cluster*. Por exemplo, alguns *clusters* autenticam seus usuários através de mecanismos de *login* dos sistemas UNIX, enquanto isso, existem outros que utilizam LDAP para autenticar seus usuários.

A definição de classes extensíveis é fundamental, uma vez que o objetivo desse *framework* é prover uma estrutura organizacional mínima para que os desenvolvedores dos serviços sejam capaz de estendê-lo e adaptá-lo às suas necessidades. Sendo assim, a classe `UserAutheticationToBeImplemented`, apresentada na Figura 4.3, representa justamente a classe que conterá a implementação do sistema de autenticação de um *cluster* específico.

O pacote `JobManagementSI` (na Figura 4.3) tem a função de encapsular os pacotes e classes referentes à implementação do JM-SI. Dentro dele existe o pacote `JobManagementProviderSI`, o qual possui os elementos básicos de um JM-SI. Ele é formado pela interface `JobManagementProviderInterface`, e pelas classes abstratas `Job` e `JobManagementProvider`. A interface possui a assinatura das operações do serviço de gerenciamento de aplicações, detalhadas na seção 4.2.2.1.

A classe abstrata `Job` possui os atributos e os métodos que devem ser utilizados

sobre as informações que caracterizam uma aplicação. Essa classe será utilizada pelas operações de submissão de aplicações e de verificação do *script* de submissão gerado pelo JM-SI de um *cluster* específico. Por questões de simplificação os atributos que qualificam esta classe não são apresentados na Figura 4.3. Os métodos, também abstratos, `generateSubmissionScript()` e `launchSubmissionScript()` devem ser implementados de acordo com o tipo de ferramenta de escalonamento de aplicações instalada em cada *cluster*. A classe `JobToBeImlemented`, que não faz parte do pacote `JobManagementProviderSI`, deverá estender e implementar esses métodos. A classe abstrata `JobManagementProvider` está associada às classes `UserAuthentication` e `Job`, através dos respectivos atributos: `user` e `job`. Estas associações são importantes pois através delas é possível autenticar o usuário e acessar as informações de aplicação de forma transparente e extensível. Além destes atributos esta classe possui os métodos abstratos correspondentes a interface que ele está estendendo. A implementação das operações do serviço deve ser feita na classe que será capaz de lidar com especificidades de cada ferramenta de escalonamento de aplicações. Nesse caso a classe `JobManagementProviderToBeImplemented`, como indicado na Figura 4.3.

Visando o reuso de código definiu-se o pacote `JobManagementUtil`, ilustrado na Figura 4.4, o qual é formado pela classe `JobManagementAuxiliaryTools` e pelas interfaces `JobManagementErrorCode` e `JobManagementConstants`. A primeira possui os códigos de erros definidos para a funcionalidade de gerenciamento de aplicações do ambiente ICE. A segunda interface possui as constantes necessárias para o desenvolvimento dos provedores de Web Services. A classe `JobManagementAuxiliaryTools` possui métodos que foram desenvolvidos para ajudarem na construção das implementações específicas da JM-USI. Vale ressaltar que esse pacote não é utilizado apenas nos JM-SIs, mas também é empregado no desenvolvimento do JM-SM.

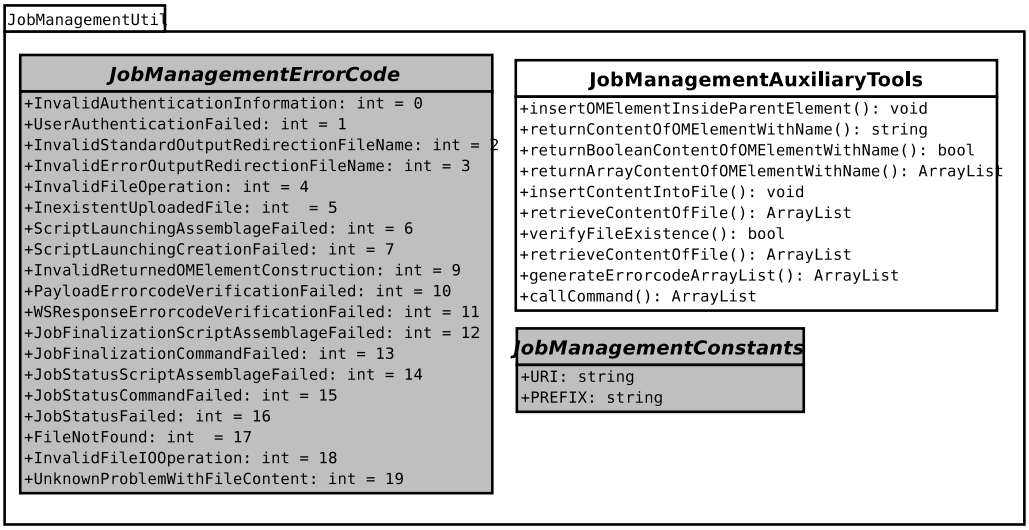

Figura 4.4: Diagrama de classes para o pacote de classes auxiliares

Tendo como base o JMF e o pacote com ferramentas auxiliares passou-se para o segundo passo do desenvolvimento do JM-SI: as implementações para cada ferramenta de gerenciamento de aplicações. A Figura 4.5 apresenta o diagrama de classes

das implementações do JM-SI para as ferramentas OpenPBS e OAR.

Por uma questão de organização decidiu-se criar um pacote para cada implementação dentro do pacote `JobManagementSI`. A Figura 4.5 apresenta os pacotes `OpenPBSProviderSI` e `OARProviderSI`. Dado o fato dos escalonadores OpenPBS e OAR possuirem fortes similaridades, as implementações de seus provedores de serviços tornaram-se bastante semelhantes. Em virtude disso, será descrita a implementação do OpenPBS JM-SI, a qual é equivalente a implementação do OAR JM-SI. Os pontos que as diferem serão ressaltados quando necessário.

Pode-se observar na Figura 4.5 que as implementações seguem o JMF definido, estendendo as classes necessárias e adicionando outras. As classes necessárias são: `OpenPBSJobManagementProvider` e `OpenPBSJob`. Na primeira classe são efetivamente implementados os provedores dos Web Services. Eles irão receber as mensagens através do *container* Tomcat processá-las, executando a operação solicitada (nesse caso executanto a chamada do método que implementa a operação) e construir a mensagem SOAP de resposta à solicitação. As comunicações entre os consumidores do JM-SM (localizados junto ao Portal Web) e os provedores dos serviços nos *front-ends* é realizada utilizando o estilo SOAP-RPC, o qual é caracterizado por empregar mensagens síncronas. Na segunda classe necessária são implementados os métodos utilizados para o processamento das operações de submissão de aplicações e visualização dos *scripts* de submissão. As demais operações definidas no JM-USI não demandaram a criação de classes extras, visto que elas apresentam um complexidade bem inferior as duas citadas anteriormente. Na Figura 4.5 também está indicado a utilização dos pacotes `ICESecurity` e `JobManagementUtil` que colaboram na modularização e simplicidade do ambiente ICE.

A seguir serão detalhadas as operações e serão descritas as estruturas de classes definidas para as implementações das operações mais complexas.

Operação de Submissão de Aplicações

Através da análise da Figura 4.5 percebe-se que as classes `OpenPBSJob` e `OARJob` possuem, além dos métodos herdados da classe `Job`, métodos que geram as informações comuns a todos os tipos de aplicações a serem lançadas. Por exemplo, o método `definePartialAppCall()` é utilizado em ambas implementações específicas para montar a chamada da aplicação com seus parâmetros (caso sejam especificados). Entretanto, existem alguns detalhes na montagem do *script* de submissão que devem considerar qual o tipo de aplicação que será lançada. Nesse caso foram definidas classes mais específicas para aplicações: seqüenciais e paralelas, sendo que está última dependerá do tipo de biblioteca ou linguagem paralela que será empregada. No ambiente ICE existe suporte para lançamento de aplicações paralelas baseadas em duas bibliotecas: MPI (DONGARRA; et al., 1995) (implementação MPICH) e DECK (BARRETO et al., 2000). Desta forma foram definidas as classes `OpenPBSSeqJob`, `OpenPBSMPICHParallelJob` e `OpenPBSDECKParallelJob`, as quais estendem a classe `OpenPBSJob`. Como apresentado na Figura 4.5 essas classes possuem os métodos necessários para geração das especificidades de lançamento de cada tipo de aplicação. Nessa figura também percebe-se que esta mesma estrutura de classes é aplicada no caso do gerenciador OAR.

A Figura 4.6 ilustra o XML contendo a requisição que chega no provedor de

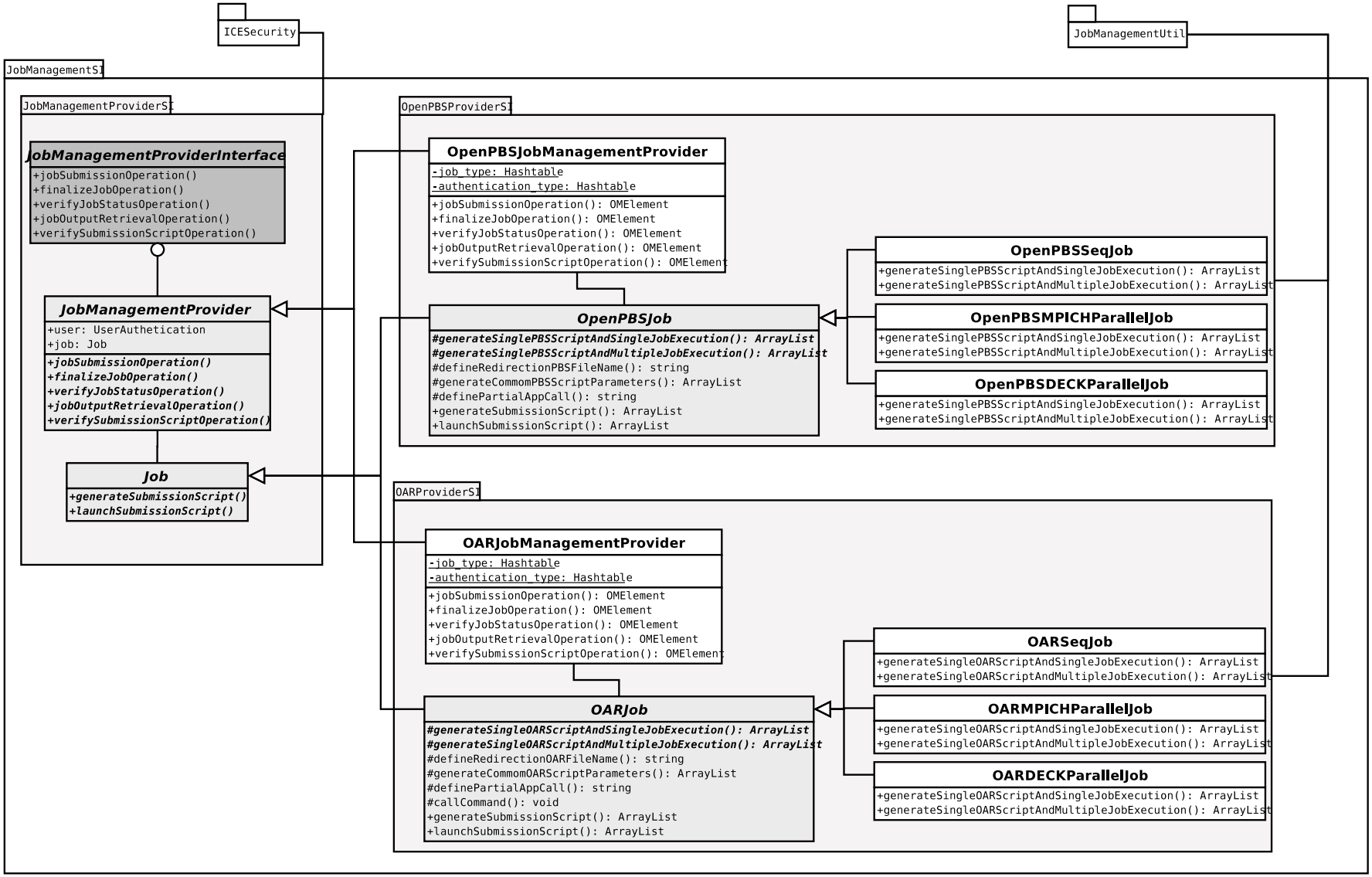

Figura 4.5: Diagrama de classes para as implementações do OpenPBS-SI e OAR-SI

serviços do JM-SI. As informações existentes nessa requisição são as mesmas tanto para a operação de submissão quanto para a de visualização dos *scripts*. A diferença é que no caso da primeira operação os dois métodos `generateSubmissionScript()` e `launchSubmissionScript()` são invocados, enquanto que na outra operação só o primeiro é chamado. A utilização desses métodos aciona os outros métodos apresentados na Figura 4.5.

```
<verifySubmissionScriptOperation>
   <username>clarissa</username>
   <basedir>/home/gppd/clarissa/</basedir>
   <main_appname>exemplo</main_appname>
   <param>param1 param2 inteiro1</param>
   <numnodes>10</numnodes>
   <numproc>10</numproc>
   <queue>default</queue>
   <walltime>00:20:00</walltime>
   <joinout>false</joinout>
   <outfile>saida</outfile>
   <errfile>erro</errfile>
   <jobtype>SEQ</jobtype>
   <numbatchiteration>1</numbatchiteration>
   <multiplescripts>false</multiplescripts>
</verifySubmissionScriptOperation>
```

Figura 4.6: Elementos XML da requisição das operações de submissão e visualização de *scripts*

Para que os consumidores do módulo JM-SM possam compreender o retorno dessas operações, foram definidos alguns elementos XML que padronizam a forma de lidar com as informações de retorno dos serviços. A seguir serão apresentados os elementos XML e exemplos de sua aplicação.

A Figura 4.7 possui o retorno da submissão de uma aplicação no escalonador OpenPBS, Figura 4.7 (a); e no OAR, Figura 4.7 (b). Nesse exemplo pode-se perceber que a aplicação conseguiu ser submetida aos gerenciadores com sucesso, visto que a informação retornada é a identificação do *job* submetido.

```
(a)
<jobSubmissionOperation>
   <command_output>41078.gppd.gppd</command_output>
</jobSubmissionOperation>

(b)
<jobSubmissionOperation>
   <command_output>10234</command_output>
</jobSubmissionOperation>
```

Figura 4.7: Exemplo de retorno da operação de submissão. (a) Submissão com OpenPBS (b) Submissão com OAR

Já a Figura 4.8 apresenta o retorno de um erro identificado antes do processo de submissão à ferramenta de gerenciamento de aplicações do *cluster*. Neste exemplo, o usuário não está devidamente autenticado para utilizar os recursos do *cluster* em questão.

```
<jobSubmissionOperation>
   <errorcode>UserAuthenticationFailed</errorcode>
</jobSubmissionOperation>
```

Figura 4.8: Exemplo de retorno da operação de submissão contendo erro.

Essa mesma estrutura de retorno de erro é empregada nas demais operações, sendo que o que pode variar é o tipo de erro que está sendo reportado.

Operação Visualização dos *Scripts* de Submissão

Como apresentado anteriormente, as diferenças em termos de diagrama de classes e o funcionamento das operações de submissão e visualização são pequenas. Mas existem diferenças em relação aos elementos XML de retorno. A seguir serão detalhados os elementos XML para a operação de visualização dos *scripts* de submissão.

Na Figura 4.9 são ilustrados os elementos básicos de retorno: `script`, `content`. Como a JM-USI permite que mais de um *script* seja montado para a mesma aplicação, a operação de visualização tem suporte para retornar as informações de mais de um *script*, facilitando a identificação dessa multiplicidade no JM-SM.

```
<verifySubmissionScriptOperation>
   <script>
      <name></name>
      <content> ... </content>
      ...
      <content> ... </content>
   </script>
   ...
   <script>
      ...
   </script>
</verifySubmissionScriptOperation>
```

Figura 4.9: Elementos XML de retorno da operação de visualização de *scripts*.

O elemento `script` delimita as informações dos eventuais diferentes componentes da mensagem de retorno, enquanto o elemento `name` indica o nome do *script* que deveria ser submetido e o `content` representa cada linha do *script*.

Abaixo são apresentados alguns exemplos de possíveis mensagens de retorno dessa operação. A Figura 4.10 é um exemplo da visualização de um *script* de submissão de uma aplicação MPI que deve ser lançada em 2 *scripts* diferentes. Nesse exemplo, a operação de visualização foi requisitada no *cluster* que possui como gerenciador de aplicações o OpenPBS.

A Figura 4.11 possui a visualização de um *script* gerado no *cluster* que opera com o gerenciador OAR. Nesse exemplo, simula-se o lançamento de uma aplicação DECK que deverá ser executada 5 vezes mas apenas um *script* deverá ser gerado.

Comparando os dois exemplos é possível verificar que os dois escalonadores apresentam características bastante semelhantes. Dessa forma, tanto na operação de submissão quanto na de visualização de *scripts* não foram necessárias grandes mudanças na estrutura de desenvolvimento. As diferenças se concentraram na forma

```
<verifySubmissionScriptOperation>
  <script>
    <name>pbs_launch_ex_1.sh</name>
    <content>#!/bin/sh</content>
    <content>#PBS -l nodes=1</content>
    <content>#PBS -l walltime=00:10:00</content>
    <content>#PBS -q labtec</content>
    <content>#PBS -o /home/gppd/clarissa/testes/saida_1_ex</content>
    <content>#PBS -e /home/gppd/clarissa/testes/erro_1_ex</content>
    <content>
      mpirun -machinefile $PBS_NODEFILE -np 1 /home/gppd/clarissa/testes/ex 2 3 4 5
    </content>
  </script>
  <script>
    <name>pbs_launch_ex_2.sh</name>
    <content>#!/bin/sh</content>
    <content>#PBS -l nodes=1</content>
    <content>#PBS -l walltime=00:10:00</content>
    <content>#PBS -q labtec</content>
    <content>#PBS -o /home/gppd/clarissa/testes/saida_2_ex</content>
    <content>#PBS -e /home/gppd/clarissa/testes/erro_2_ex</content>
    <content>
      mpirun -machinefile $PBS_NODEFILE -np 1 /home/gppd/clarissa/testes/ex 2 3 4 5
    </content>
  </script>
</verifySubmissionScriptOperation>
```

Figura 4.10: Exemplo de retorno da operação de visualização em um *cluster* com OpenPBS.

```
<verifySubmissionScriptOperation>
    <script>
        <name>oar_launch_ex.sh</name>
        <content>#!/bin/sh</content>
        <content>#OAR -l nodes=1</content>
        <content>#OAR -l walltime=00:10:00</content>
        <content>#OAR -q file</content>
        <content>for i in $(seq 1 5); do echo '--- Iteration '$; </content>
        <content>
          deckrun -f $OAR_NODEFILE -n 1 /home/users/clarissa/testes/ex 2 3 4 5;
        </content>
        <content>done;</content>
    </script>
</verifySubmissionScriptOperation>
```

Figura 4.11: Exemplo de retorno da operação de visualização em um *cluster* com OAR.

como as aplicações são lançadas (no caso da submissão) e nos parâmetros que formam o *script* de submissão, como mostraram os exemplos apresentados acima.

Analisando a Figura 4.5 é possível perceber que não existem classes e pacotes específicos para o desenvolvimento das operações de finalização de aplicações, verificação do estado e recuperação dos arquivos das saídas padrão. Essa característica ocorre visto que a aquisição e as interações com os gerenciadores de aplicações são simples. Para essas operações não é necessário a criação de *scripts*, basta receber processar o conteúdo dos parâmetros recebidos pelo provedor e gerar os comandos que interagem com as ferramentas de gerenciamento de aplicações. A seguir serão apresentados os elementos XML das mensagens de requisição e resposta de cada uma das operações ainda não descritas.

Operação de Finalização

Baseado nas definições da JM-USI, a Figura 4.12 (a) apresenta os elementos XML que formam a mensagem de requisição de término de uma aplicação submetida. Através da utilização do elemento `jobid` uma aplicação é finalizada.

```
(a)
<finalizeJobOperation>
    <username> ... </username>
    <jobid> ... </jobid>
</finalizeJobOperation>

(b)
<finalizeJobOperation>
    <status>OK</status>
</finalizeJobOperation>

(c)
<finalizeJobOperation>
    <content>qdel: Unknown Job Id 3345.gppd.gppd</content>
</finalizeJobOperation>
```

Figura 4.12: Elementos XML de requisição e retorno da operação de finalização.

Na implementação dessa operação, o comando de finalização do gerenciador do *cluster* é chamado e o elemento `jobid` é passado como parâmetro. Caso algum erro seja detectado antes do comando ser chamado então o provedor retornará a mensagem de erro já descrita anteriormente. Caso contrário ele informará o sucesso da finalização, como ilustrado na Figura 4.12 (b), ou poderá retornar a mensagem de erro passada pelo gerenciador do *cluster*, como apresentado na Figura 4.12 (c).

Operação de Verificação de Status

A mensagem de requisição dessa operação contém somente a identificação do usuário. Como apresentado na USI, outros parâmetros não são necessários. A Figura 4.13 apresenta um exemplo de uma mensagem de retorno contendo o estado de uma aplicação submetida em um *cluster* que utiliza OAR.

Esta informação, quando recebida pelo JM-SM, deverá ser tratada para ser apresentada aos usuários de uma forma mais fácil e clara.

Operação de Recuperação das Saídas

Esta operação é utilizada tanto para recuperar o conteúdo dos arquivos com a saída padrão como a saída padrão de erro. Através dos elementos `basedir` e `file`, ilustrados na Figura 4.14 (a), pode-se definir qual deverá ser o arquivo recuperado. Na Figura 4.14 (b) têm-se o retorno de um erro ocorrido no momento da recuperação do arquivo solicitado. A Figura 4.14 (c) apresenta um exemplo de retorno de mensagem com o conteúdo da saída padrão de uma aplicação.

Nesse caso de recuperação de arquivos sempre serão recuperadas informações em modo texto. Por isso não são necessários cuidados especiais com relação a formação

```
<verifyJobStatusOperation>
     <content>Job id           Name             User             Time Use S Queue</content>
     <content>---------------- ---------------- ---------------- -------- - -----</content>
     <content>41014.gppd       XX7_art0002.lgr  pilla            08:00:00 R corisco.special </content>
     <content>41015.gppd       XX7_bzip2002.i   pilla            07:59:58 R corisco.special </content>
     <content>41016.gppd       XX7_bzip2002.i   pilla            07:59:05 R corisco.special </content>
     <content>41017.gppd       XX7_bzip2002.i   pilla            07:59:02 R corisco.special </content>
     <content>41019.gppd       XX7_gcc0002.cpd  pilla            08:00:19 R corisco.special </content>
     <content>41020.gppd       XX7_gzip0002.gr  pilla            07:59:57 R corisco.special </content>
     <content>41022.gppd       XX7_gzip0002.pr  pilla            08:00:13 R corisco.special </content>
     <content>41024.gppd       XX7_gzip0002.so  pilla            08:00:27 R corisco.special </content>
     <content>41027.gppd       XX7_parser0002.  pilla            08:00:00 R corisco.special </content>
     <content>41029.gppd       XX7_vpr0002.pla  pilla            07:58:38 R corisco.special </content>
     <content>41036.gppd       YY7_gcc0002.cpd  pilla            07:38:04 R corisco.special </content>
     <content>41039.gppd       YY7_gzip0002.pr  pilla            04:59:41 R corisco.special </content>
     <content>41041.gppd       YY7_gzip0002.so  pilla            03:42:52 R corisco.special </content>
     <content>41042.gppd       YY7_mcf0002.lgr  pilla            03:01:30 R corisco.special </content>
     <content>41044.gppd       YY7_parser0002.  pilla            02:29:36 R corisco.special </content>
     <content>41045.gppd       YY7_vortex0002.  pilla            02:05:15 R corisco.special </content>
     <content>41046.gppd       YY7_vpr0002.pla  pilla            01:56:10 R corisco.special </content>
     <content>41047.gppd       YY7_vpr0002.rou  pilla            01:31:21 R corisco.special </content>
     <content>41070.gppd       l_5_2_lh         enmathias               0 Q labtec.wday  </content>
     <content>41075.gppd       reserva          mccera           00:00:00 R labtec.wday     </content>
     <content>41077.gppd       pbsc_2_2_lh      enmathias        00:00:00 R corisco.wday    </content>
     <content>41086.gppd       home/gppd/clari  clarissa                0 R labtec.wday   </content>
</verifyJobStatusOperation>
```

Figura 4.13: Exemplo de retorno da operação de verificação de estado das filas do gerenciador OAR.

da mensagem de retorno.

Resumindo a implementação dos JM-SIs pode-se dizer que os dois tipos de escalonadores são bastante semelhantes. Esta característica facilitou o processo de desenvolvimento dos JM-SIs uma vez que não foi necessário a criação de mecanismo complexos de mapeamento entre a JM-USI e os JM-SIs. Dentre as operações de gerenciamento de aplicações disponibilizadas no ambiente ICE, a submissão foi a que demandou mecanismos mais complexos e extensos de implementação.

4.2.2.3 Job Management Service Module

Como descrito no Capítulo 3 o componente *Service Module* possui uma parte de sua implementação no *Middleware* de Serviços e outra parte no Portal Web. Neste ponto serão apresentadas as características de implementação do componente JM-SM referente aos consumidores de Web Services.

Assim como os provedores de serviços, os consumidores foram desenvolvidos utilizando a tecnologia Apache Axis 2. Entretanto, também foi necessário utilizar Servlets java para repassar o resultado dos provedores de serviços para os módulos do JM-SM dentro do Portal Web. A Figura 4.15 ilustra o diagrama de classes definido para organizar as classes e métodos empregados no desenvolvimentos dos consumidores.

No intuito de organizar a implementação, foi definido o pacote `JobManagementSM`. Como pode ser visto na Figura 4.15, as classes dentro desse pacote utilizam os recursos já definidos no pacote `JobManagementUtil`. Dentro do pacote que organiza o JM-SM foi definido o pacote `JobManagementSMConsumer`. Ele foi criado com o intuito de separar toda e qualquer implementação relacionada a tecnologia Web *Service* do restante da lógica do JM-SM. Com essa filosofia pode-se simplesmente mudar o tipo de tecnologia empregado na comunicação e preservar o restante do funcionamento do JM-SM.

O pacote `JobManagementSMConsumer` é formado pelas classes que lidam com os

```
(a)
<jobOutputRetrievalOperation>
    <username> ... </username>
    <basedir>  ... </basedir>
    <file>     ... </file>
</jobOutputRetrievalOperation>

(b)
<jobOutputRetrievalOperation>
    <errorcode>
        FileNotFound - java.io.FileNotFoundException:
         testes/saida (Permission denied)
    </errorcode>
</jobOutputRetrievalOperation>

(c)
<jobOutputRetrievalOperation>
    <content>--- Iteration 1</content>
    <content>hello world!!!</content>
    <content>--- Iteration 2</content>
    <content>hello world!!!</content>
</jobOutputRetrievalOperation>
```

Figura 4.14: Elementos XML da requisição e resposta da operação de recuperação de arquivos de saída.

provedores de serviços existentes no diferentes *clusters* integrados no sistema. Para cada operação disponibilizada na JM-USI foram definidas classes que preparam as mensagens a serem enviadas aos provedores (`buildxxxxxxOperationPayload()`) e que efetivamente acessam os serviços (`callxxxxxOperation()`). Visando uma melhor modelagem do sistema foi definida uma classe que trata das especificidades da submissão de aplicações (`JobSubmission`). As classes que tratam da chamada para a submissão de aplicações (`JobSubmissionOperation()`) e a de verificação dos *scripts* de submissão (`VerifySubmissionScriptOperation()`) herdam as características da classe `JobSubmission`.

Ainda no pacote `JobManagementSM` existe a classe `WSConnections`. É através dessa classe que o JM-SM se comunica com o Portal Web. Esta classe é instanciada dentro do Portal Web a partir das páginas JSP's e é responsável por isolar as chamadas e primitivas empregadas em Web Services existentes nas classes do pacote `JobManagementSMConsumer`. Entretanto, para cada operação existe um método correspondente na classe `WSConnections`. Nesses métodos é que as informações retornadas serão manipuladas e por fim armazenadas no banco de dados.

Como Web *Service* é uma tecnologia *stateless* foi preciso a definição de algumas tabelas para o armazenamento das informações retornadas, para que futuramente se pudesse acessá-las. Esta estrutura de tabelas será apresentada juntamente com o diagrama entidade relacionamento (ER) do Portal Web a seguir.

4.2.3 Portal Web

Para iniciar a descrição da implementação dos módulos que compõe o Portal Web será apresentado o modelo de dados adotado nesse ambiente. Em seguida serão descritas as implementações dos módulos *System Management Module* (SMM) e a parte referente ao *Service Module* (SM) do Portal.

O modelo de informação do ambiente ICE é ilustrado na Figura 4.16. Ele está baseado no modelo Entidade Relacionamento (ER). As entidades e suas características relevantes serão descritas a seguir.

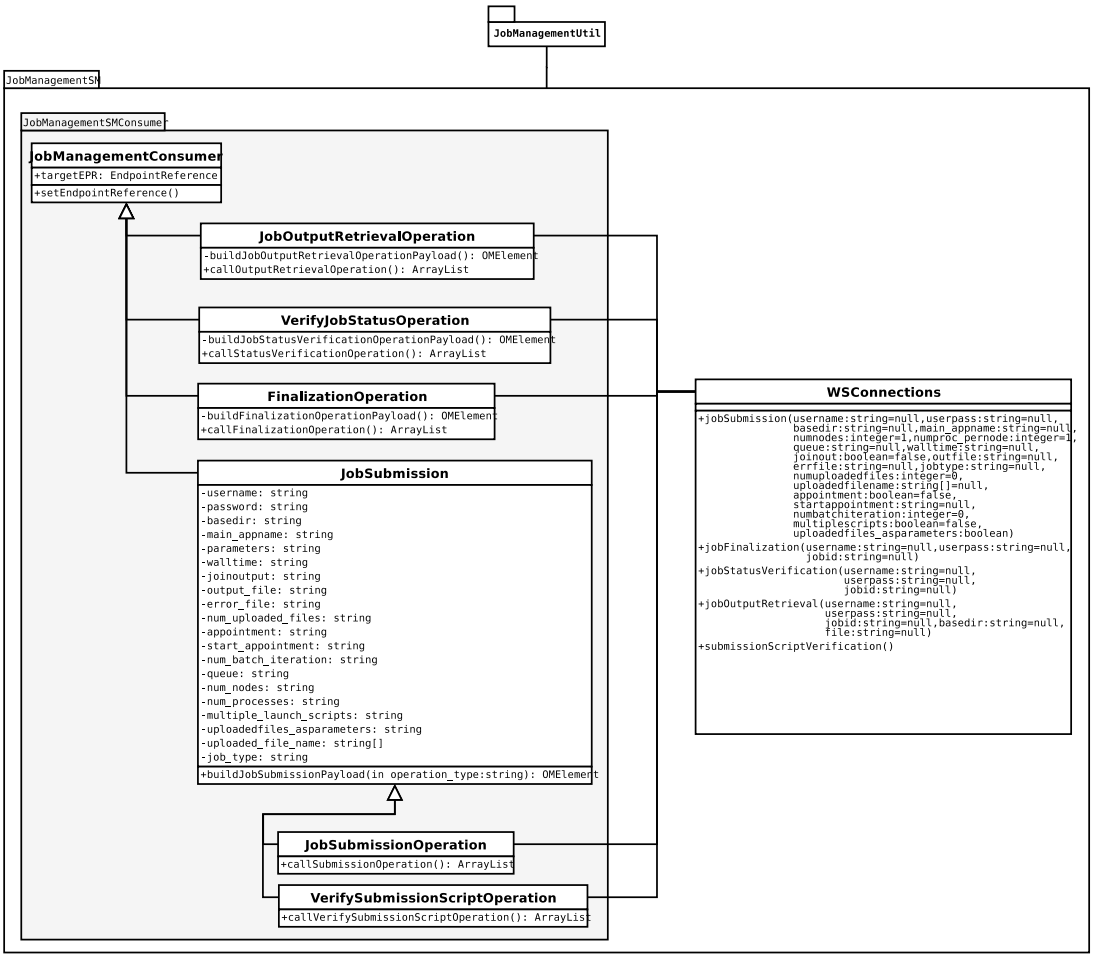

Figura 4.15: Diagrama de classes do JM-SM

Para compreender a forma como o Portal Web foi implementado é preciso entender alguns elementos básicos do ICE e os seus relacionamentos. Cada usuário cadastrado possui um grupo com o nome exatamente igual ao seu usuário. Esta estrutura se reflete nas entidades `User`, `Group` e `UsersPerGroup`. É possível que um usuário faça parte de mais de um grupo, mas o grupo que possui o nome exatamente igual ao usuário deve ser unitário. Esse tipo de associação é similar ao que ocorre nos sistemas do tipo Unix e Linux. A partir dessa associação a manipulação dos relacionamentos dos usuários com as demais entidades será realizada através do(s) grupo(s) ao(s) qual(is) ele está associado.

Além do usuário estar associado a um grupo, é preciso que esse grupo esteja associado a um perfil. As entidades `Profile` e `ProfileAssociatedToGroup` ilustram esse requisito. Através do perfil de um usuário é que se poderá definir quais são os direitos que este usuário terá dentro do ambiente ICE. A princípio já existem três perfis básicos: ICE_ROOT, CLUSTER_ROOT e USER, descritos na Seção 3.2.2, mas o modelo de dados e o ambiente foram projetados para suportarem definições de novos perfis. Entretanto, como o ambiente ICE é voltado para o gerenciamento e suporte de utilização de múltiplos *clusters* é possível que o usuário apresente diferentes perfis de acordo com o sistema de alto desempenho que está utilizando. Para garantir essa característica têm-se as entidades `Cluster` e `ClusterAccessAssociation`. Se-

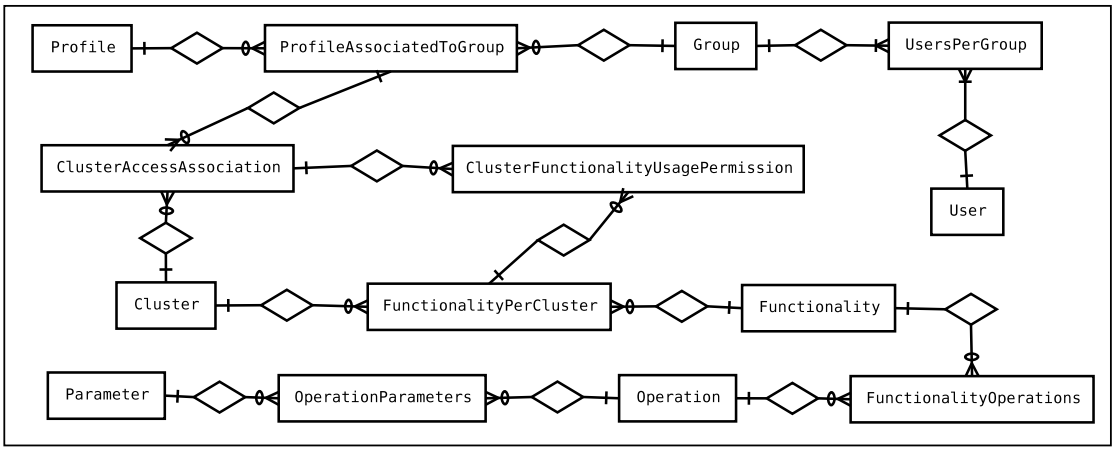

Figura 4.16: Diagrama Entidade Relacionamento (ER) do ambiente ICE

guindo a linha de relacionamentos apresentados na Figura 4.16, para um usuário, associado a um grupo, que está associado a um perfil, que está associado a um *cluster*, podem existir diferentes funcionalidades.

O ambiente ICE permite que sejam definidas diversos tipos de funcionalidades de *clusters*. Neste trabalho está sendo disponibilizada a funcionalidade de gerenciamento de aplicações, mas outras como monitoramento de recursos, de aplicações entre outras que podem ser incorporadas. Uma funcionalidade é formada por operações que são formadas por parâmetros. As entidades listadas a seguir representam estas associações: `Parameter`, `Operation`, `Funcionality`, `OperationParameters`, `FunctionalityOperations`. Uma ou mais funcionalidades podem estar associadas a um ou mais *clusters*. A entidade `FunctiolalityPerCluster` ilustra esta situação na Figura 4.16.

Com as associações apresentadas até o momento é possível definir as permissões de acesso dos usuários às funcionalidades de cada *cluster*. Dentro do ambiente ICE para que um usuário esteja devidamente autorizado a utilizar alguma funcionalidade é preciso que ele possua um grupo, um perfil, um *cluster* e uma funcionalidade associados. Isto é representado na entidade `ClusterFunctionalityUsagePermission`.

O gerenciamento dos dados do modelo de informação do ICE é realizado pelo SMM. No momento em que usuário acessa o Portal Web, identificando-se através de um *login* e senha, o SMM verifica quais os *clusters*, perfis e funcionalidades que esse usuário possui permissão de utilização. A partir dessa verificação é montada a página Web que será enviada ao *browser* do usuário. A Figura 4.17 ilustra um usuário que tem perfil de administrador do ICE (ICE_ROOT) e do *cluster gppd* (CLUSTER_ROOT) (*cluster* pertencente ao Grupo de Processamento Paralelo e Distribuído do Instituto de Informática da UFRGS), além disso ele também possui o perfil de usuário comum (USER).

Na Figura 4.17 tem-se a sobreposição de duas telas. A que está em segundo plano é a página principal do menu de operações. A partir dela é possível inserir novas operações, remover e visualizar as existentes, assim como alterá-las. A tela em primeiro plano ilustra exatamente a página de alteração das informações de uma operação, no caso a *Job Submission*, pertencente à funcionalidade de gerenciamento de aplicações (*Job Management*).

À esquerda das telas existe o menu principal. Esse é o menu montado a partir

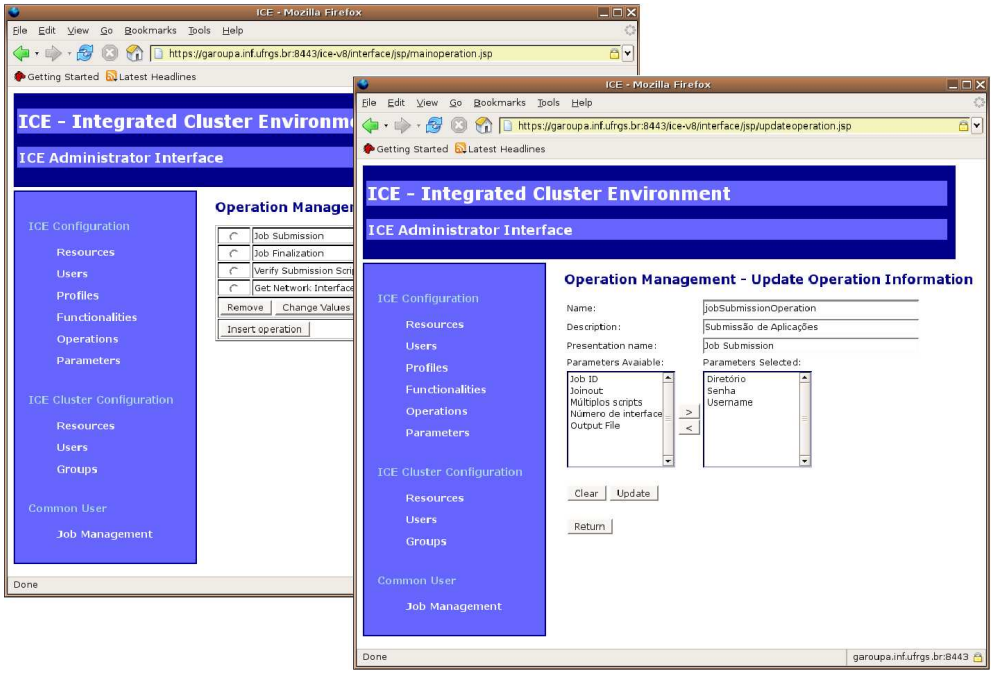

Figura 4.17: *Snapshot* Portal Web - Configuração de operações

do tipo de perfil que o usuário possui. O menu *ICE Configuration* representa as opções que existem para os usuário que possuem o perfil ICE_ROOT. O menu *ICE Cluster Configuration* representa as possibilidades de configuração disponibilizadas pelo perfil CLUSTER_ROOT. E, finalmente, o menu *Common User* representa o perfil de usuário comuns (USER). A estrutura de telas apresentada na Figura 4.17 e suas opções é idêntica para os demais itens do menu *ICE Configuration*.

Foi definido que usuários com direito de configuração de ambiente ICE só podem inserir usuários administrativos, isto é, administradores do ICE e de *clusters*. Usuários comuns só poderão ser cadastrados pelos administradores dos *clusters*. Foram criadas restrições para fazer com que os administradores de *clusters* só tivessem acesso para configurar os requisitos dos *clusters* os quais são responsáveis. Dessa forma se um usuário tem acesso a mais de um *cluster* que é gerenciado por diferentes administradores é preciso que cada administrador o cadastre como usuário de seu *cluster*.

O gerenciamento de grupos é feito exclusivamente pelos administradores dos *clusters*. A Figura 4.18 apresenta a tela principal da configuração de grupos.

É neste momento que os administradores de *clusters* podem associar usuários (*Users*), perfis (*Profile*), *clusters* (*Resource*) e funcionalidades (*Functionality*) aos grupos. Através do preenchimento dos formulários disponibilizados nessa etapa de configuração do ambiente ICE é que as tabelas de dados, geradas a partir do modelo ER apresentado na Figura 4.16, serão atualizados. Os processos associações de recursos e funcionalidades a grupos são os mais complexos.

Para estabelecer o relacionamento entre um grupo e um *cluster* são preciso dois passos. A Figura 4.19 mostra ao fundo o primeiro passo e mais na frente o segundo passo.

A associação de um recurso a um usuário implica que esse usuário esteja associado a um grupo, o qual deve estar associado a um perfil. No primeiro passo

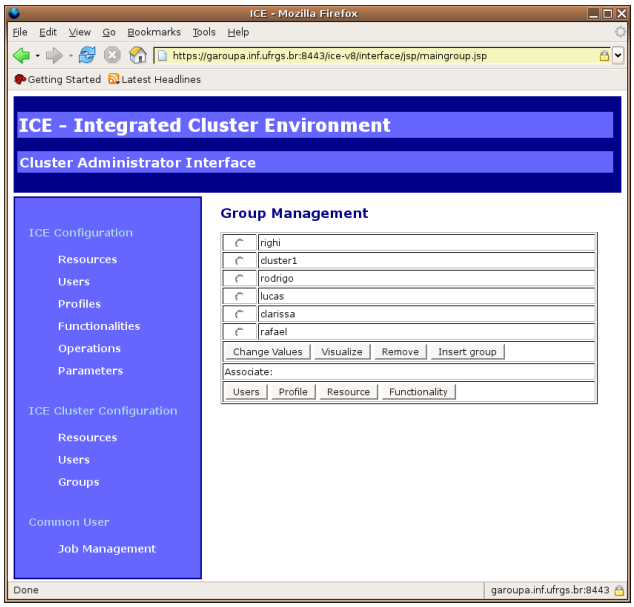

Figura 4.18: *Snapshot* Portal Web - Página principal da configuração de grupos

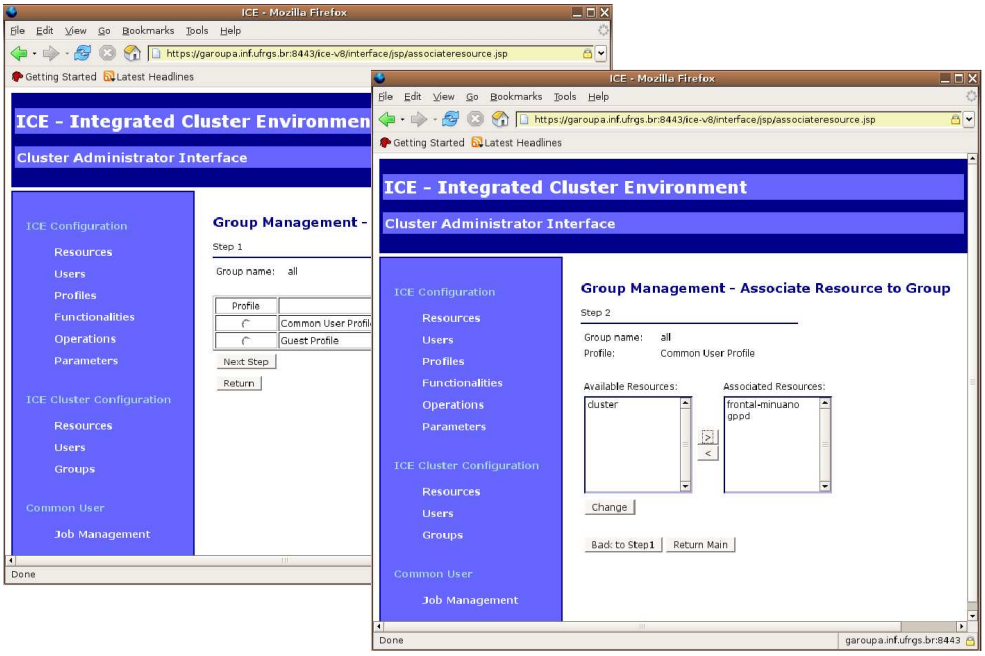

Figura 4.19: *Snapshot* Portal Web - Associação de *clusters* a grupos

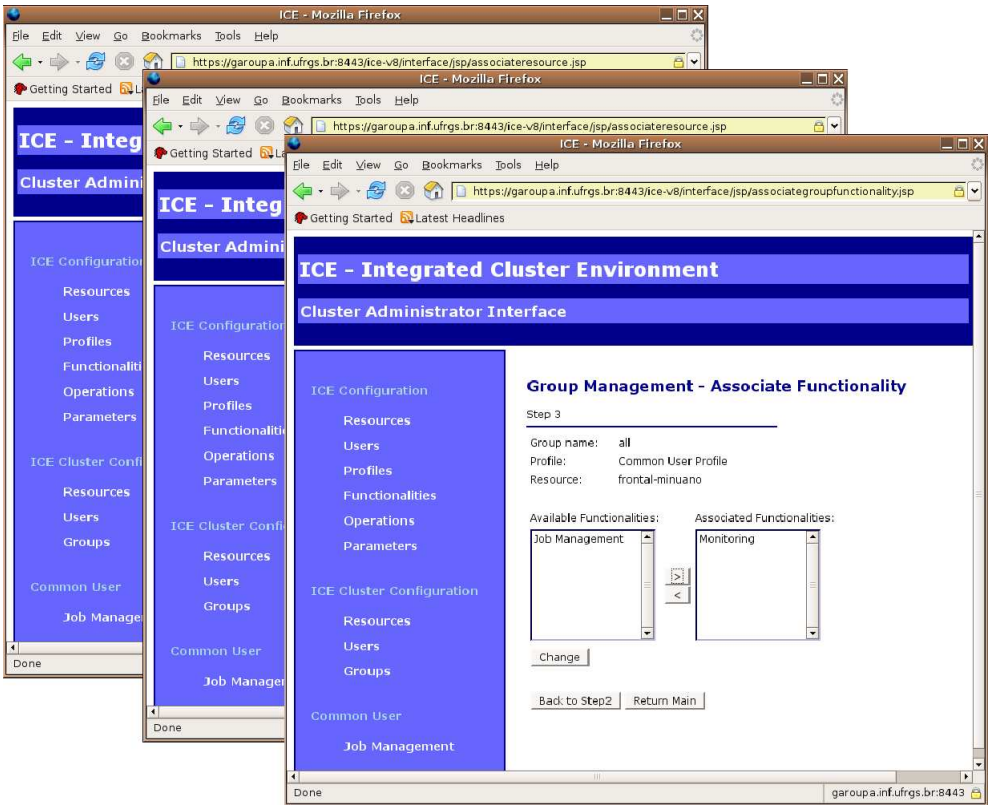

Figura 4.20: *Snapshot* Portal Web - Associação de funcionalidades a grupos

o administrador pode escolher qual perfil do grupo vai associar a um determinado *cluster*. No segundo passo são apresentados os *clusters* que esse grupo tem acesso e a quais *clusters* esse grupo já está associado.

Na associação de funcionalidades a grupos são necessários os mesmos passos anteriores e mais um terceiro. A Figura 4.20 ilustra esse último passo.

Depois de definido qual o perfil e qual o recurso, o administrador deverá indicar qual a funcionalidade que será associada. No passo três, apresentado na Figura 4.20, são disponibilizadas somente as funcionalidades associadas com o *cluster* escolhido. A associação entre *cluster* e funcionalidade é estabelecida no menu *Resource* do administrador do ambiente ICE, ela é ilustrada na Figura 4.21.

Nessa figura são disponibilizadas aos administradores as funcionalidades providas pelo ambiente. O administrador deve selecionar as que deseja prover em cada *cluster* e deve indicar o endereço do provedor dos serviços. Nesse ponto acontece a ligação entre o Portal Web e o módulos JM-SIs existentes nos *front-ends* dos *clusters*.

Os *snapshots* apresentados neste trabalho resumem as telas mais importantes do ambiente ICE. Resumidamente pode-se dizer que os itens nos menus referentes ao ICE_ROOT e ao CLUSTER_ROOT tem a funcionalidade de configurar o ambiente ICE para que ele possa ser utilizado pelos usuários dos *clusters* gerenciados neste ambiente. Como as informações disponibilizadas aos usuários são todas dinâmicas tornam-se necessárias essas ações de configuração do ambiente realizadas dentro do SMM.

Além das operações realizadas pelo SMM ainda existem aquelas executadas pelos

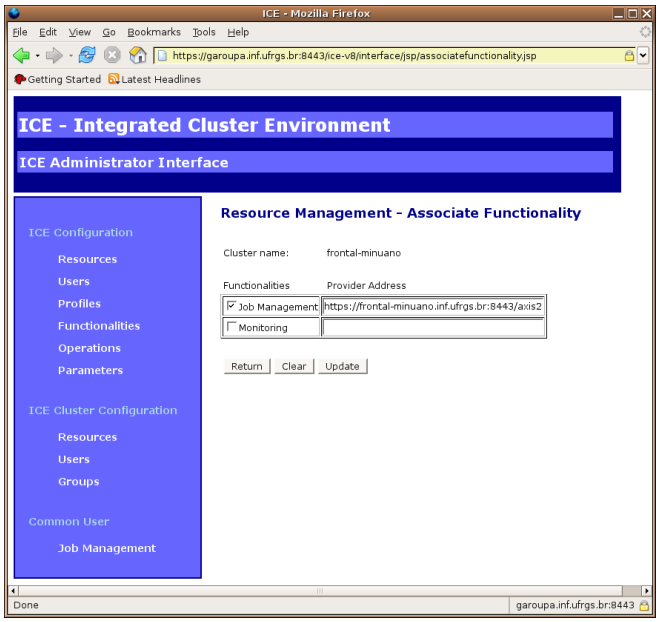

Figura 4.21: *Snapshot* Portal Web - Associação de funcionalidades a *clusters*

SMs. No caso deste trabalho tem-se as ações relacionadas ao SM-JM. Analisando os *snapshots* apresentados acima pode-se perceber que no menu principal do ambiente ICE também existe o menu *Common User*, relacionado com o perfil USER. Isto significa que o usuário com perfil USER no *cluster gppd* possui a permissão de execução da funcionalidade de gerenciamento de aplicações (*Job Management*). A Figura 4.22 ilustra a página principal do menu de gerenciamento de aplicações.

Através dessa página é possível submeter aplicações, gerar um script de visualização de teste, finalizar uma aplicação ou então recuperar os arquivos de saída padrão e saída de erro padrão de uma aplicação que esteja rodando ou que tenha sido finalizada. A tabela da Figura 4.22 apresenta as aplicações que estavam associadas ao usuário naquele momento. É possível perceber que ele possuía uma aplicação que estava sendo executada (12444.gppd.gppd) e outra que estava ainda na fila do gerenciador do *cluster gppd* (12456.gppd.gppd). Nesse caso o gerenciador instalado nesse *cluster* é a ferramenta OpenPBS, mas como se pode perceber esta informação não é necessária para o usuário do ambiente ICE.

As operações disponibilizadas na Figura 4.22 são exatamente as mesmas definidas na JM-USI. Entretanto, algumas delas não aparecem explicitamente. A operação de verificação de estado das aplicações não é disponibilizada através de um botão, como por exemplo, a operação de submissão. Entretanto, a tabela ilustrada na página Web dessa figura é formada através da consulta a esta operação. As operações de finalização e recuperação das saídas padrão também são disponibilizadas através dessa tabela. A operação de submissão de uma aplicação é ilustrada na Figura 4.23.

Os parâmetros informados serão processados pelo SM-JM do Portal Web, sendo que o consumidor de serviços do *Middleware* de Serviços será contactado e dessa forma a requisição ao Web *service* do *cluster* em questão será acionado. A mensagem de retorno do provedor de serviços será tratada e os dados devidamente apresentados ao usuário. Caso algum erro ocorra o usuário receberá uma página especificando esse erro. Caso contrário, as informações da aplicação serão armazenadas em uma

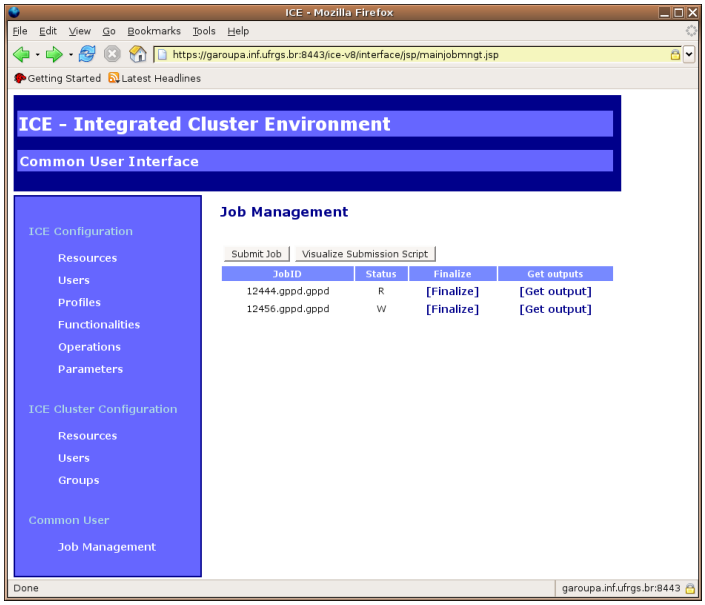

Figura 4.22: *Snapshot* Portal Web - Página principal do menu *Job Management*

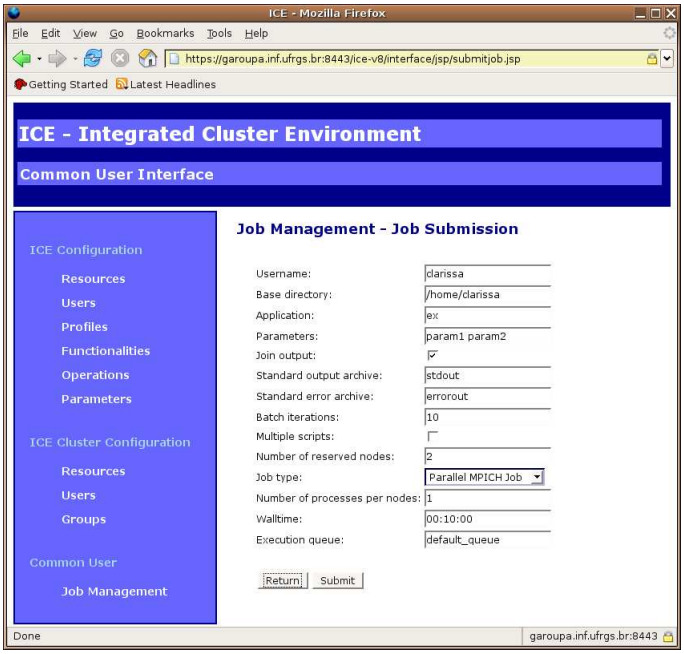

Figura 4.23: *Snapshot* Portal Web - Página de submissão de aplicações

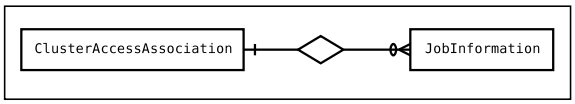

Figura 4.24: ER da funcionalidade de *Job Management*

tabela que foi acrescentada no modelo de informação do ambiente. A Figura 4.24 apresenta as entidades e relacionamentos definidos para armazenar as informações relacionadas com a funcionalidade de submissão de aplicações.

Dado o fato dos Web *Services* não manterem o estado de suas operações foi necessária a definição da entidade *JobInformation*. Ela está relacionada com a entidade *ClusterAccessAssociation* pois uma aplicação para ser submetida deve estar associada a um usuário com o perfil que permita esse tipo de operação em um determinado *cluster*.

Para visualizar como funcionam as opções de submissão de aplicações e ter certeza de que o *script* que deverá ser submetido condiz com a semântica que o usuário deseja foi definida a operação de visualização de *scripts*. Para utilizar essa operação o usuário deve preencher o um formulário idêntico ao formulário da operação de submissão. Entretanto, ao invés de submeter uma aplicação ele receberá o conteúdo do *script* que deverá ser submetido ao gerenciador do *cluster*. As Figuras 4.25 e 4.26 ilustram exemplos de visualização de *scripts*, apresentando algumas das opções de submissão disponibilizadas no ambiente.

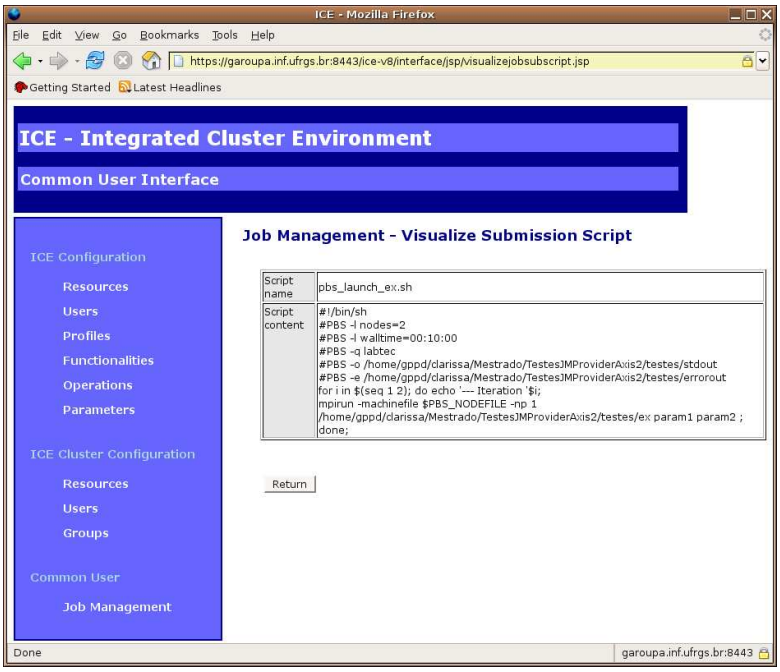

Figura 4.25: Exemplo 1 de visualização de *scripts* de submissão

Nesse primeiro exemplo foi solicitada a visualização de *script* para o lançamento de uma aplicação paralela utilizando a biblioteca MPICH, onde deveriam ser alocados 2 nós mas deve ser disparado apenas 1 processo. Através da interface Web especificou-se que essa aplicação deveria ser executada 2 vezes (referente ao campo

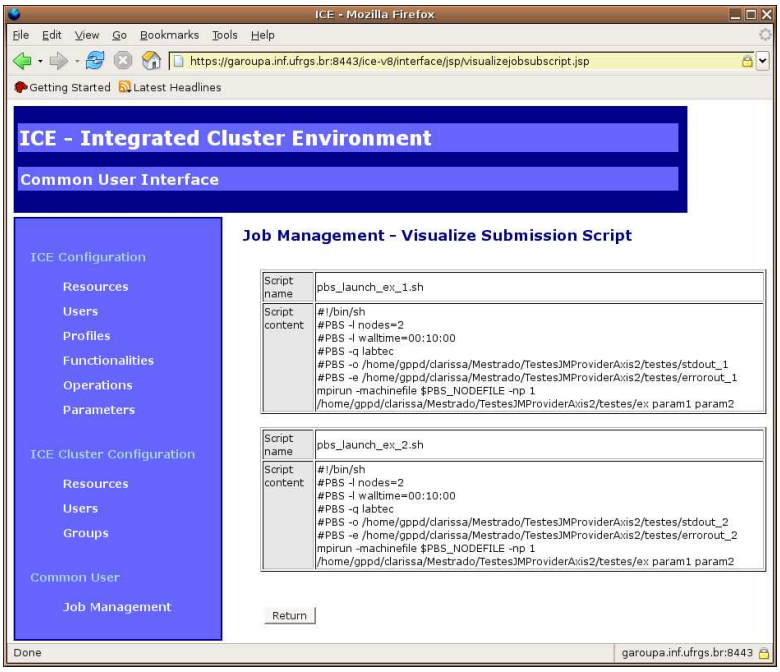

Figura 4.26: Exemplo 2 de visualização de *scripts* de submissão

Batch iterations na Figura 4.23) e que não ela não deveria ser disparada em múltiplos arquivos (referente ao campo *Multiple scrips* na Figura 4.23). A diferença entre os exemplos da Figura 4.25 e 4.26 está no fato de que a segunda habilita a criação de diferentes scripts para cada uma das iterações especificadas.

Comparando as informações apresentadas na interface Web com as informações originais recebidas no consumidor (comparando as figuras da Seção 4.2.2.2) percebe-se que em alguns casos o XML recebido no consumidor é apresentado praticamente sem alterações na página Web, como no caso da visualização de *scripts*. Em compensação, existem casos em que o conteúdo do XML é completamente adaptado para facilitar sua apresentação aos usuários, como no caso da verificação do estado das aplicações submetidas.

De acordo com o projeto do ambiente ICE é possível a adição de outras operações na funcionalidade de gerenciamento de aplicações sem que isto altere o que já está disponibilizado na ferramenta.

4.3 Resumo

Neste capítulo foram apresentadas as decisões de projeto e a descrição da implementação do protótipo desenvolvido. No que diz respeito às decisões de projeto discutiu-se qual seria o *middleware* adotado e quais deveriam ser os padrões atuais que poderiam ser empregados no ambiente ICE. Através de uma comparação entre dois bem documentados e estabelecidos *middlewares* (Corba e Web *Services*) optou-se por utilizar Web *Services* na implementação do *Middlware* de Serviços do ambiente ICE. Com relação aos padrões que poderiam ser utilizados no ambiente optou-se por não utilizar o Globus e seu módulo de gerenciamento de recursos (WS-GRAM). Essa decisão foi tomada pois verificou-se que não possível utilizar apenas o módulo WSGRAM. Para utilizá-lo é preciso instalar também os módulos princi-

pais do Globus, o que corresponde a praticamente todo o Globus, como mostrado na Seção 4.1.2. Depois desta análise inicial passou-se para a descrição em si da implementação do protótipo. Foram apresentados cada um dos componentes do ambiente e seus módulos. Para tanto foram empregados exemplos de estruturas XML que descrevem o funcionamento do *Middleware* de Serviços assim como foram apresentados o modelo de informação e alguns *snapshots* do Portal Web. Através da leitura deste capítulo pode-se perceber que o ambiente ICE possui uma complexidade em seus relacionamentos e interligações de módulos. Entretanto, acredita-se que essa característica não compromete a capacidade de extensibilidade do ambiente e transparência de uso propostos no ambiente. Ao contrário, a modularidade de seus componentes e a arquitetura definidas facilitam o oferecimento dessas características.

5 AVALIAÇÃO DO PROTÓTIPO ICE

Este capítulo está dividido em duas partes. A primeira trata de uma comparação entre o ambiente ICE, proposto neste trabalho, e algumas ferramentas relacionadas. A segunda possui uma análise quantitativa do ambiente desenvolvido. Serão apresentados gráficos avaliando o *overhead* inserido pelo ambiente ICE.

5.1 Avaliação Qualitativa

A avaliação qualitativa foi realizada através da comparação das características de ferramentas de gerenciamento de alto desempenho com as características do ambiente ICE. Esta comparação não foi realizada entre todos os ambientes porque, como apresentado no Capítulo 2, existem escopos diferentes de emprego das ferramentas (*clusters*, *grids* e múltiplos *clusters*). Sendo assim, as ferramentas de *clusters* e *grids* não podem ser diretamente comparadas ao ambiente ICE, o qual é voltado para múltiplos *clusters*. Isto faz com que seja realizada uma análise qualitativa entre o ambiente ICE e as ferramentas HPC2N e M3C, apresentadas no Capítulo 2.

O objetivo desta análise qualitativa é verificar qual a capacidade que esses ambientes apresentam para prover facilidades aos usuários e desenvolvedores dos sistemas, considerando os desafios existentes na construção de sistemas distribuídos.

5.1.1 Metodologia

A metodologia utilizada, para realizar a comparação entre o ambiente ICE e os trabalhos relacionados na área de gerenciamento de múltipos *clusters*, está baseada nos desafios apresentados por Coulouris em (COULOURIS; DOLLIMORE; KINDBERG, 2005) sobre a construção de sistemas distribuídos. A lista dos critérios de avaliação é apresentada abaixo.

- Heterogeneidade - A comparação entre os ambientes, considerando este quesito, foi realizada seguindo as características listadas abaixo.

    1. Independência de plataforma: considera se o ambiente é capaz de operar sobre diferentes plataformas de hardware e de sistemas operacionais.
    2. Independência de linguagem: avalia se os módulos que compõem o ambiente podem ser desenvolvidos em qualquer linguagem de programação.
    3. Interoperabilidade: compara a capacidade que o *middleware*, empregado no desenvolvimento do ambiente, possui para integrar sistemas legados e para lidar com as diferenças que existem entre os módulos que compõem o sistema.

- *Openness* - Este critério está relacionado com a extensibilidade do sistema. Neste trabalho foram considerados dois tipos de extensibilidade, descritas abaixo.

  1. Extensibilidade de funcionalidades: está relacionada com a capacidade que o ambiente apresenta para prover mecanismos de inserção de novas funcionalidades.
  2. Extensibilidade na integração de ferramentas: considera o suporte disponível para prover abstrações para inserção de novas ferramentas que apresentam a mesma funcionalidade. Por exemplo, se já existe uma ferramenta integrada no sistema para o gerenciamento de aplicações, este critério mede quais os recursos que os ambientes apresentam para inserir uma nova ferramenta de gerenciamento, sem que isso altere o funcionamento global do ambiente.

- Transparência - São considerados os seguintes tipos de transparência: acesso, localização, concorrência, replicação, falha, mobilidade, desempenho e escalabilidade.

- Tratamento de falhas - Considera a capacidade que o ambiente apresenta para lidar com as falhas de seus componentes.

- Escalabilidade - Avalia a capacidade que o ambiente tem de lidar com o aumento do número de usuários e recursos, sem que isso afete o seu desempenho.

- Concorrência - Considera a modelagem de segurança no acesso aos recursos, quando o ambiente é concorrente e existe a possibilidade de ocorrer condições de corrida, *deadlocks*, etc.

- Segurança - A análise deste critério foi baseada na existência de canais de comunicação com encriptação e na existência de autenticação de usuários.

5.1.2 Comparação entre HPC2N, M3C e ICE

A tabela 5.1 apresenta a existência ou não dos critérios avaliados em cada uma das ferramentas: HPC2N, M3C e ICE.

Os primeiros cinco critérios da Tabela 5.1 são relacionados com as características de heterogeneidade e *openness* dos sistemas distribuídos. O ambiente ICE provê estes cinco critérios devido a decisão de utilizar como *middleware* Web *services*. O ambiente M3C pode apresentar independência de plataforma, uma vez que foi desenvolvido utilizando Applets e Servlets Java. Este ambiente também pode prover extensibilidade de funcionalidades, visto que ele pode permitir que os seus usuários insiram novas funcionalidades no sistema. Entretanto o M3C não proporciona um *framework* para lidar com as diferentes ferramentas que possuem a mesma funcionalidade e nem a capacidade de lidar com os sistemas legados dos *clusters*. Ao contrário dos dois primeiros ambientes discutidos, o HPC2N não provê características de heterogeneidade e nem de *openness* porque é restrito a um conjunto fechado de ferramentas. Além disso, ele foi desenvolvido utilizando tecnologias que não proporcionam a capacidade de serem interoperáveis e, também, não disponibilizam um *framework* para que ele se torne extensível.

Tabela 5.1: Comparação entre o ambiente ICE e algumas ferramentas relacionadas

	Parâmetros de Comparação	HPC2N	M3C	ICE
1	Independência de plataforma		x	x
2	Independência de linguagem			x
3	Interoperabilidade			x
4	Extensibilidade de funcionalidades		x	x
5	Extensibilidade de integração de ferramentas			x
6	Transparência de acesso	x	x	x
7	Transparência de localização			
8	Transparência de concorrência	x	x	x
9	Transparência de replicação			
10	Transparência de falhas			
11	Transparência de mobilidade	x	x	x
12	Transparência de desempenho			
13	Transparência de escalabilidade		x	x
14	Tratamento de falhas			
15	Escalabilidade	-	-	-
16	Concorrência	x	x	x
17	Segurança	x	x	x

Todos os ambientes comparados apresentam transparência de acesso, concorrência e mobilidade. Seus usuários podem acessá-los sem levar em consideração quais são os passos e operações realizados para que se alcance, efetivamente, o dispositivo. Vários usuários podem acessar estes ambientes de forma concorrente através de seus *browsers*. Eles também podem mudar sua localização física sem afetar os sistemas aqui comparados. Entretanto, nenhum dos ambientes apresentam transparência de localização, visto que eles sabem quais *clusters* estão utilizando. Os critérios de transparência de replicação e falhas, da mesma forma como o tratamento de falhas, não são providos por estes ambientes, pois seus módulos não foram desenvolvidos para identificarem e lidarem com as eventuais falhas que possam ocorrer nos componentes que os formam.

Com relação ao critério de escalabilidade, não se pode fazer nenhuma afirmação, já que não foram realizados estudos e experimentos capazes de estabelecer a escalabilidade destes ambientes. Levando em consideração o fato dos três ambientes serem baseados em plataformas Web e que são executados sob servidores Web, tais como Apache e Tomcat, pode-se especular que suas escalabilidades podem ser limitadas pela configuração desses servidores. Da mesma forma como a escalabilidade, o critério de concorrência está relacionado com a infra-estrutura sobre a qual os ambientes ICE, M3C e HPC2N são executados. Cada acesso a estes ambientes é, primeiramente, tratado pelos servidores Web, sendo que a concorrência é tratada nestes servidores. Considerando o aspecto de segurança, têm-se os três ambientes utilizando HTTPS para garantir a segurança do canal de comunicação. Entretanto, cada um apresenta um conjunto de técnicas distintas para garantir as características de autenticação e autorização de seus usuários.

Finalmente, analisando a Tabela 5.1, pode-se perceber que o ambiente ICE pos-

sui suporte para quase todos os critérios de sistemas distribuídos, considerados nesta comparação. Comparado com os outros ambientes de gerenciamento de múltiplos *clusters*, apresentados neste trabalho, pode-se dizer que o ambiente ICE é o que possui mais recursos. As características que mais o distinguem das demais ferramentas são sua capacidade de extensibilidade e *openness*. Em um cenário onde é preciso prover facilidade de uso para os usuários de múltiplos *clusters*, cujo custo de implantação da solução seja o menos oneroso, tem-se o ambiente ICE como uma boa opção.

5.2 Avaliação Quantitativa

O objetivo dos testes quantitativos com o ambiente ICE foi verificar qual o *overhead* inserido pela infra-estrutura deste ambiente nos processos que eram, tipicamente, executados através de linha de comando pelos usuários.

Foram realizados experimentos com as cinco operações providas pela JM-USI:

- submissão de aplicações;
- verificação do *script* de submissão gerado pelo JM-SI da respectiva ferramenta;
- finalização de aplicações;
- verificação do estado das filas de execução das ferramentas;
- recuperação dos arquivos de saída (padrão ou de erro).

A seguir são apresentadas algumas informações que caracterizam como foi conduzida a avaliação quantitativa do ambiente ICE. Em primeiro lugar, será apresentado exatamente o que foi medido, em seguida qual a metodologia de medição e por fim os resultados encontrados para cada uma das operações suportadas no módulo de gerenciamento de aplicações do ambiente ICE.

5.2.1 Definição do escopo das medições

O *overhead* observado leva em consideração apenas o tempo de processamento dentro do *front-end* do *cluster* empregado nos testes. A Figura 5.1 ilustra os elementos que fizeram parte da aquisição das medidas apresentadas nesta seção.

As as três entidades que foram empregadas na realização desses testes são caracterizadas a seguir.

- Ferramenta de Gerenciamento: corresponde à ferramenta de gerenciamento de aplicações, instalada no *cluster* onde as medidas estão sendo realizadas.
- Provedor JM-SI: representa as implementações do JM-SI. No caso dos experimentos aqui apresentados existe o OpenPBS JM-SI e o OAR JM-SI.
- Consumidor *standalone*: consiste em uma aplicação que consulta o provedor de serviços da JM-SI. Esta aplicação não está integrada ao ambiente ICE, por isso foi considerada *standalone*. Ela é utilizada para acessar tanto o OpenPBS JM-SI quanto o OAR JM-SI. Entretanto, para evitar a interferência da rede de comunicação, ela foi instalada no *front-end* utilizado para os testes.

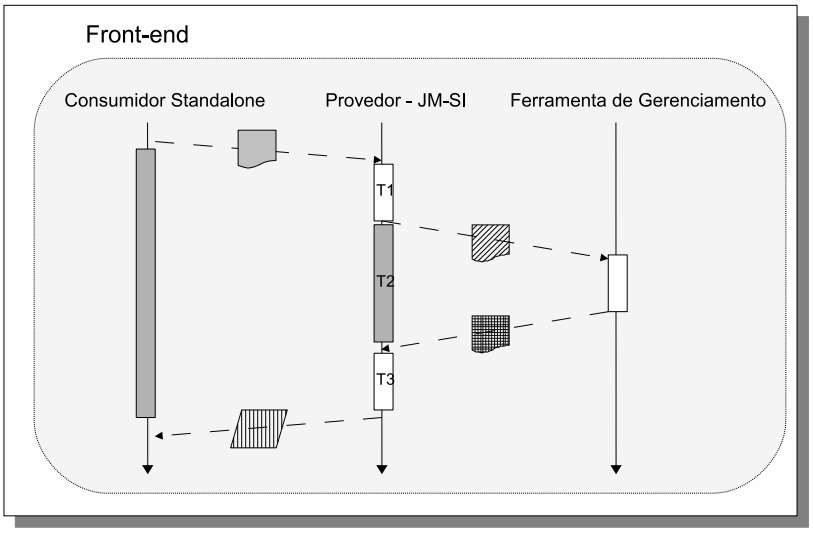

Figura 5.1: Cenário de aquisição das medidas de *overhead*

O consumidor *standalone* faz as requisições das operações do serviço de gerenciamento de aplicações ao provedor da respectiva JM-SI. Ao receber estas requisições o JM-SI faz o tratamento das informações e realiza os passos necessários, interagindo com a ferramenta de gerenciamento. Após esta interação, retorna para o consumidor os dados resultantes da execução da operação.

Nos testes que foram realizados, o que se mediu foi a soma dos tempos T1, T2 e T3 indicados na Figura 5.1. T1 representa o tempo despendido pelo provedor para reconhecer os parâmetros da operação e traduzi-los para serem solicitados à ferramenta de gerenciamento. T2 consiste no tempo que o provedor tem que esperar para receber os dados da operação executada. T3 é, por fim, o tempo que o provedor gasta para processar os dados e montar a mensagem de retorno ao consumidor. Nestes experimentos, não foi considerado o tempo gasto pelo *container* Tomcat para receber a requisição e encaminhá-la ao devido provedor de serviços, assim como, também, não foi considerado o tempo que este mesmo elemento leva para encaminhar a resposta do serviço solicitado.

Para que se pudesse identificar qual o *overhead* inserido pelo ambiente ICE foi realizado o seguinte cálculo:

$$Overhead = T_{ice} - T_{ferramenta}$$

onde `Tice` representa o tempo total (T1+T2+T3) de execução de uma operação do provedor JM-SI, e `Tferramenta` representa o tempo de total execução de uma ação da ferramenta sem intervenção do ambiente ICE. Desta forma, durante as análises o *overhead* é apresentado em termos absoluto, ou seja, quantos milisegundos de *overhead* foi observado.

5.2.2 Metodologia

Foram realizados experimentos e análises para todas as operações definidas na JM-USI e implementadas na JM-SI. Para cada uma destas análises foram executadas 300 iterações. Definiu-se este tamanho de amostra depois de sucessivas rodadas de execução com diferentes tamanhos de amostras. Através da análise dos desvios

padrões encontrados percebeu-se que amostras a partir de 300 iterações mantinham o desvio padrão constante. Os valores apresentados nos gráficos correspondem às médias encontradas de acordo com esta amostra. Além dos gráficos com as médias, também serão apresentadas algumas tabelas com o desvio padrão e a representação percentual desse desvio em relação à média encontrada. Ainda foram realizados testes utilizando, diretamente, os comandos dos gerenciadores de aplicações. Estes testes também seguiram a metodologia de aquisição de valores relatada acima.

A aquisição do tempo, dentro da implementação dos provedores JM-SI, foi realizada através da chamada à primitiva `System.currentTimeMillis()` logo no início de cada operação e no final das mesmas, exatamente antes de retornar a resposta aos consumidores. Esta primitiva faz parte da linguagem Java, e retorna o tempo em milisegundos no momento de sua chamada. Para que se possa ter o tempo de execução da operação, faz-se a subtração do tempo final menos o inicial. Para aquisição dos tempos, no caso da execução direta das ferramentas, foi utilizado o comando `time`, que retorna o tempo de cpu em segundos utilizado por uma aplicação. Este tempo em segundos foi convertido para milisegundos no intuito de uniformizar as medições realizadas neste trabalho.

Nas análises feitas, com as operações de submissão de aplicações e de visualização de *scripts* de submissão, foram realizadas variações na forma como os parâmetros das aplicações podem ser utilizados. Estas variações consideram os parâmetros de: múltiplas execuções de mesma aplicação e no caso de serem solicitadas múltiplas execuções, é preciso que sejam criados diferentes *scripts*. Além disso também foram testadas aplicações seqüenciais e paralelas com as bibliotecas MPICH e DECK. A nomenclatura utilizada nas legendas dos gráficos dos experimentos destas operações é apresenta na Tabela 5.2.

Tabela 5.2: Descrição das variações de parâmetros para submissão de aplicações.

Opções	Descrição
DECK	Aplicação paralela baseada na biblioteca DECK
MPICH	Aplicação paralela baseada na biblioteca MPICH
Seq	Aplicação seqüencial
Multiple	Indica se a aplicação foi executada mais de uma vez. No caso dos testes realizados, quando habilitado, significa que a aplicação foi executada cinco vezes.
Single	Indica que a aplicação foi executada uma única vez.
Arc	Indica que somente um *script* foi gerado, independentemente do número de vezes que a aplicação será executada
Arcs	Indica que foram gerados tantos *scripts* quantas foram as vezes que a aplicação deverá ser executada. Neste caso, quando habilitado, significa que foram gerados cinco *scripts* de submissão da mesma aplicação.

5.2.3 Plataforma

Os experimentos foram realizados no *cluster* `frontal-minuano` pertencente ao GPPD, o qual tem suas principais características apresentadas na Tabela 5.3. Neste

cluster foram instaladas as duas ferramentas de gerenciamento de aplicações: OpenPBS e OAR.

Tabela 5.3: Descrição do *cluster* utilizado para os experimentos

Característica	Frontal-minuano
Sistema Operacional	Linux, kernel 2.6.12
Distribuição	Debian Sarge
Processador	Pentium III 598.998 MHz
Cache	512 KB
Memória RAM	256 MB
Swap	494 MB

Decidiu-se realizar os experimentos em um único *cluster* para que se pudesse comparar, diretamente, a intrusão do ambiente ICE em cada uma das ferramentas de gerenciamento de aplicações suportadas.

5.2.4 Experimentos com a operação de submissão de aplicações

O primeiro experimento realizado considerou a operação de submissão de aplicações. Os gráficos da Figura 5.2 ilustram os tempos de submissão encontrados com a implementação OpenPBS JM-SI.

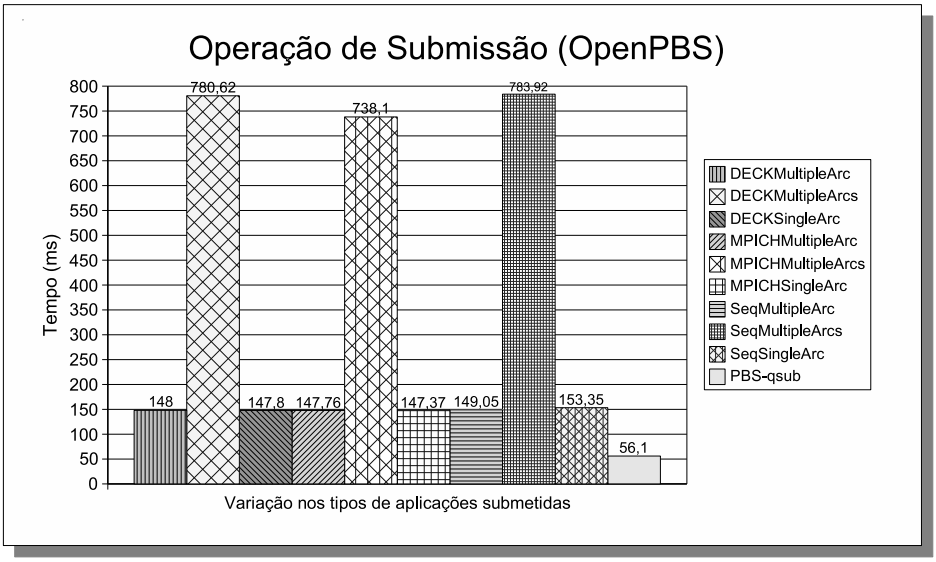

Figura 5.2: Gráfico Operação de Submissão - OpenPBS

No caso dos testes apresentados na Figura 5.2, o tempo medido para geração e submissão de um único *script* manteve-se em média entre 147 e 153 milisegundos, enquanto a geração e submissão de múltiplos *scripts* manteve-se em torno de 730 e 740 milisegundos. O tempo médio de submissão de uma aplicação, através dos comandos do gerenciador OpenPBS, manteve-se na casa de 56,1 milisegundos. Observando os gráficos dessa figura pode-se perceber que a submissão de um único *script* apresentou um comportamento bastante homogeneo, independentemente do tipo de aplicação que estava sendo lançada. No caso do disparo de múltiplos *scripts*

de execução da mesma aplicação, houve uma pequena diminuição no tempo da geração para aplicações MPICH. Este comportamento pode ter sido causado devido às condições do *front-end*, onde, possivelmente, devem ter ocorrido variações na sua carga de trabalho que acabou se refletindo nos testes realizados. Esta hipótese é reforçada quando se analisa o desvio padrão deste experimento, que pode ser encontrado na Tabela 5.4. Pode-se perceber que o desvio padrão, nos casos de submissão de múltiplos *scripts* para a biblioteca DECK e para aplicações seqüenciais, manteve-se na casa de 18 milisegundos, enquanto para a biblioteca MPICH o desvio padrão encontrado foi de aproximadamente 19 milisegundos. Com estes valores, pode-se considerar que o tempo de execução em média foi menor do que nos demais, entretanto apresentou uma variação maior. Além das informações de desvio padrão, a Tabela 5.4 também apresenta o percentual de quanto cada desvio padrão representa em relação à respectiva média do experimento.

Comparando os valores das médias encontradas, pode-se dizer que o tempo de processamento de uma submissão de aplicações no ambiente ICE é, em média, três vezes o tempo necessário para o disparo direto. Sendo assim, o ambiente ICE insere um *overhead* de duas vezes o valor alcançado na submissão direta. Por exemplo, na submissão de uma aplicação a ser lançada em único *script* (em geral o tempo gasto, independentemente se é uma aplicação seqüencial ou paralela, com ou sem múltiplas execuções dentro do mesmo *script*) o *overhead* do ambiente ICE é de aproximadamente 90,9 milisegundos.

Considerar este valor de *overhead*, sem levar em consideração a facilidade de uso que o ambiente ICE traz, não é muito encorajador. Entretanto, apesar de possuir um tempo de processamento maior que o tempo de submissão direta, via OpenPBS, ainda pode ser mais vantajoso para os usuários perderem alguns milisegundos na submissão e contarem com a transparência e facilidade de uso que o ambiente ICE oferece.

Os gráficos da Figura 5.3 ilustram os tempos de submissão encontrados com a implementação OAR JM-SI.

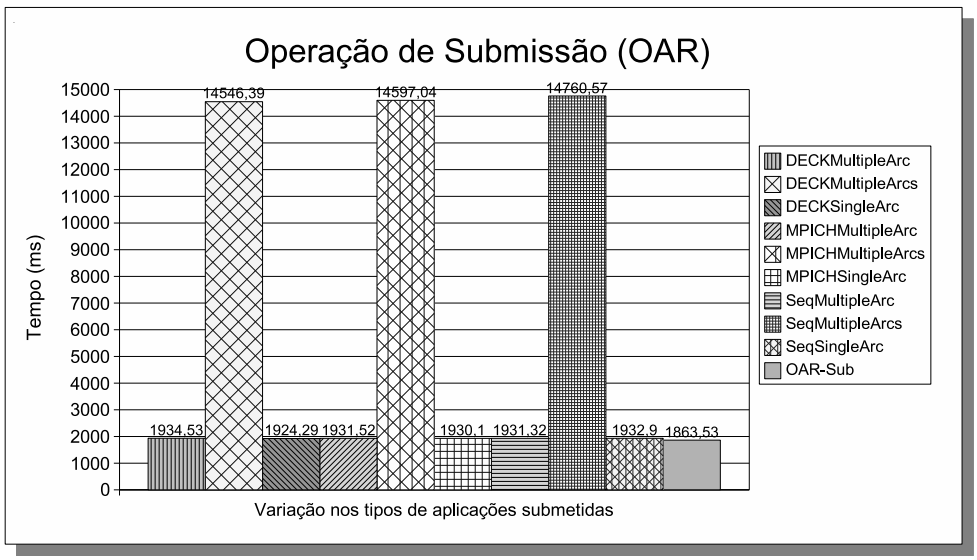

Figura 5.3: Gráfico Operação de Submissão - OAR

Analisando os gráficos da Figura 5.3, percebe-se que, de maneira geral, o comportamento do ambiente ICE, com a implementação OAR JM-SI, mostrou-se similar ao OpenPBS JM-SI. A diferença está no fato de que os tempos medidos na primeira implementação foram, consideravelmente, maiores do que na segunda. Esta diferença existe dadas as características das ferramentas de gerenciamento sobre as quais foram implementadas as JM-SIs. Muitas das informações que o OAR possui estão armazenadas em um banco de dados MySQL, enquanto isso o OpenPBS utiliza arquivos. Além disso, as características de implementação de cada uma são bastante diferentes. Tudo isso faz com que elas possuam tempos de execução bem distintos. Esses tempos não são maiores somente na comparação entre as implementações do JM-SI, mas também na comparação direta entre as ferramentas. A ferramenta OpenPBS apresenta o tempo de submissão na casa de 56,1 milisegundos, enquanto a OAR possui tempo de execução 1863,53 milisegundos.

Observando os tempos encontrados com o OAR JM-SI, que ficaram em torno de 1930 milisegundos no caso da geração e submissão de um único *script*, tem-se um desempenho bem próximo ao da própria ferramenta. Isto faz com que os usuários disponham da facilidade de utilização da ferramenta de gerenciamento de aplicações que o ambiente ICE disponibiliza não sendo penalizado com um *overhead* significativo. Por exemplo, para aplicações seqüenciais lançadas em um único arquivo tem-se um *overhead* de aproximadamente 66,47 milisegundos.

Na Tabela 5.4 também são apresentados os valores de desvio padrão e o percentual de desvio padrão encontrado para os testes de submissão da implementação OAR JM-USI. Pode-se perceber que os valores dos desvios padrão mantiveram-se estáveis quando comparados entre os tipos de experimentos realizados. Por exemplo, no caso de múltiplos *scripts*, os desvios padrão ficaram perto de 2 milisegundos, enquanto que para um único *script* ele se manteve em 4 milisegundos.

Tabela 5.4: Desvio padrão para operação de submissão

	OpenPBS		OAR	
Variações	Desvio Padrão	Percentual	Desvio Padrão	Percentual
Ferramenta	2,89	5,11	12,95	0,69
DECKMultipleArc	7,32	4,95	104,2	5,39
DECKMultipleArcs	18,92	2,42	1543,61	10,61
DeckSingleArc	6,35	4,3	149,19	7,75
MPIMultipleArc	7,33	4,96	108,13	5,6
MPIMultipleArcs	19,18	2,6	1508,67	10,34
MPISingleArc	6,7	4,55	117,65	6,1
SeqMultipleArc	6,17	4,14	105,28	5,45
SeqMultipleArcs	18,16	2,32	1722,46	11,67
SeqSingleArc	5,69	3,71	123,69	6,4

Através das análises realizadas acima, sobre os resultados encontrados nos experimentos, verificou-se que o ambiente ICE insere um *overhead* no processo de submissão de aplicações. Este *overhead* foi mais acentuado no caso da implementação OpenPBS JM-SI (90,9 milisegundos, que um tempo representa aproximadamente 2 vezes mais tempo do que a execução isolada da ferramenta) e foi bastante suavizado

na implementação OAR JM-SI (66,47 milisegundos, que representa 3,6% do tempo de execução da ferramenta isolada). Estes experimentos não podem ser analisados de forma pontual, ou seja, sem levar em consideração as características do ambiente ICE. Eles servem como um indicativo do custo a ser pago pelas facilidades de uso que estão sendo inseridas no uso do *cluster*.

5.2.5 Experimentos com a operação de visualização de *scripts* de submissão

A operação de visualização de *scripts* de submissão não possui um comando correspondente nas ferramentas de gerenciamento utilizadas neste trabalho. Isto também se deve ao fato de que são os usuários que montam os *scripts* de submissão, e sendo assim, eles já conhecem o seu conteúdo. No ambiente ICE os usuários não lidam com os detalhes de mais baixo nível, mas podem necessitar a verificação da semântica e a correção dos *scripts* que lançarão suas aplicações, como já argumentado na Seção 4.2.2.1. As Figuras 5.4 e 5.5 apresentam os tempos médios de processamento da operação de visualização de *scripts*, respectivamente, para as implementações OpenPBS JM-SI e OAR JM-SI.

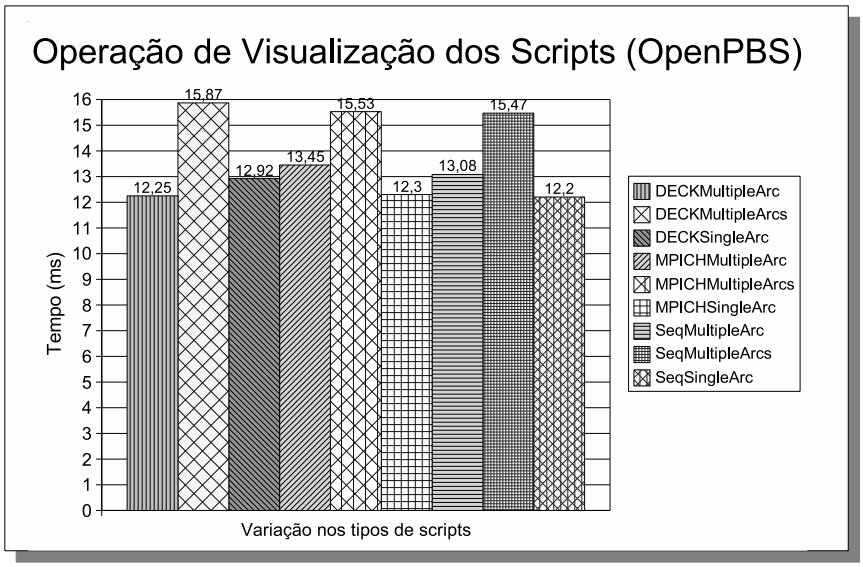

Figura 5.4: Gráfico Operação de Visualização de Scripts - OpenPBS

Analisando os gráficos das Figuras 5.4 e 5.5, é possível perceber que os tempos médios de processamento da operação de verificação de *scripts*, nas duas implementações, aproximaram-se bem mais do que no caso da operação de submissão. Os tempos médios para a geração de *scripts* únicos ficou em torno de 12 milisegundos para a implementação OpenPBS JM-SI e em torno de 11 segundos para a OAR JM-SI. No caso da verificação dos múltiplos *scripts*, o tempo médio de execução ficou na casa de 15 e 14 milisegundos respectivamente.

Comparando a execução das duas implementações, pode-se perceber que, em geral, existe uma diferença de 1 milisegundo. Esta diferença se deve às diferenças que existem no processo de tradução dos parâmetros para as estruturas de cada ferramenta.

Nos gráficos da Figura 5.4, ainda pode-se perceber uma pequena variação dos

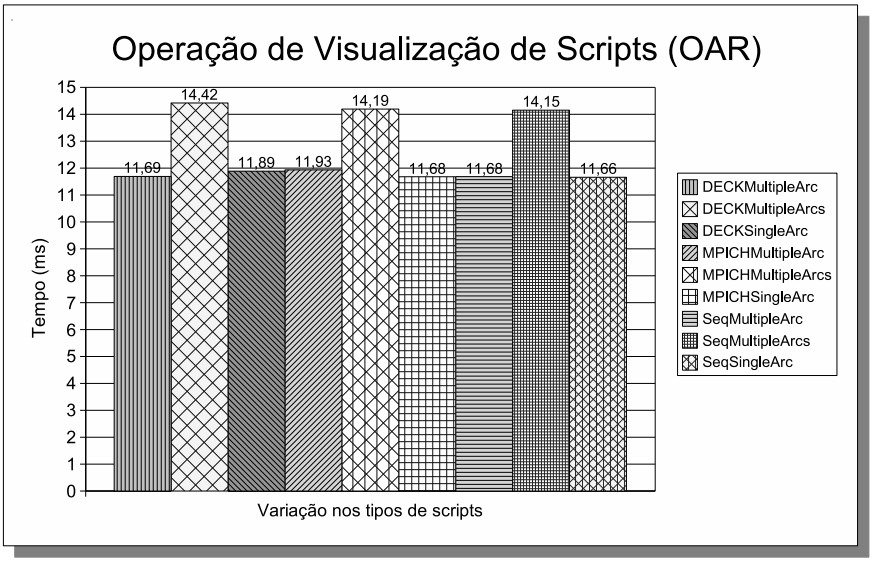

Figura 5.5: Gráfico Operação de Visualização de Scripts - OAR

tempos médios no caso da verificação de um único *script*. Mais uma vez essa alteração pode ter sido causada pela carga do *cluster* `frontal-minuano`.

A Tabela 5.5 apresenta os valores de desvio padrão e percentual de desvio padrão verificados nos experimentos com esta operação para as duas implementações. Analisando os valores encontrados, pode-se perceber que não houve nenhuma variação muito acentuada dos tempos de execução desta operação.

Tabela 5.5: Desvio padrão para operação de verificação de *scripts*

	OpenPBS		OAR	
Variações	Desvio Padrão	Percentual	Desvio Padrão	Percentual
DECKMultipleArc	1,09	8,94	0,89	7,57
DECKMultipleArcs	2,18	13,75	1,26	8,76
DeckSingleArc	1,49	11,54	0,8	6,74
MPIMultipleArc	1,35	10,01	0,88	7,39
MPIMultipleArcs	1,95	12,57	1,21	8,55
MPISingleArc	1,32	10,75	0,9	7,69
SeqMultipleArc	1,29	9,85	0,79	6,73
SeqMultipleArcs	1,86	12,01	1,18	8,36
SeqSingleArc	1,5	12,31	0,79	6,75

De maneira geral, pode-se dizer que os tempos encontrados na execução desta operação são satisfatórios. A vantagem de poder verificar o conteúdo e a semântica do *script* de submissão, e esta operação ser executada em um intervalo de tempo pequeno, só traz ganhos para o ambiente ICE.

5.2.6 Experimentos com a operação de finalização de aplicações

A Tabela 5.6 apresenta os tempos médios, desvio padrão e o percentual do desvio padrão em relação à média. Foram realizados experimentos com a submissão direta via OpenPBS, via OAR e através das implementações JM-SI sobre cada uma das ferramentas.

Tabela 5.6: Desempenho da operação de finalização de aplicações

	Média	Desvio Padrão	Percentual
OpenPBS JM-SI	138,96	3,61	2,59
OpenPBS	50,5	2,54	5,03
OAR JM-SI	2070,76	408,85	19,74
OAR	1989,19	405,42	20,38

A primeira constatação que se pode fazer, analisando esta tabela, é a grande diferença de tempos entre as ferramentas. Ao utilizar o OpenPBS diretamente o tempo médio de finalização é 50,5 milisegundos, enquanto com o OAR o tempo médio fica em 1989,19 milisegundos. Isto ocorre em vista das diferenças das ferramentas (ver Seção 5.2.4).

Comparando os resultados de cada ferramenta com a respectiva infra-estrutura no ambiente ICE, tem-se as situações apresentadas a seguir. No caso da comparação entre o gerenciador OpenPBS e o OpenPBS JM-SI, pode-se visualizar a diferença que existe nos tempos médios de finalização. Enquanto o gerenciador apresenta o tempo médio de 50,5 milisegundos, a infra-estrutura ICE possui o tempo médio de 138,96 milisegundos. Isto mostra que o tempo total de finalização no ambiente ICE, considerando a infra-estrutura para a ferramenta OpenPBS, é em geral três vezes maior que o tempo de execução da ferramenta de forma isolada. O *overhead* verificado foi de 88,46 milisegundos. Apesar deste número parecer um pouco elevado, é preciso lembrar que se tratam de milisegundos e que o maior objetivo do ambiente ICE é prover facilidades de uso aos seus usuários. Portanto, mais uma vez se torna interessante gastar 138,96 milisegundos para finalizar uma aplicação, havendo a possibilidade de não ter que lidar com os comandos específicos do gerenciador de aplicação.

Considerando o gerenciador OAR e o OAR JM-SI, pode-se perceber, através dos dados da Tabela 5.6, que a infra-estrutura ICE apresenta um tempo médio de execução bem próximo ao do gerenciador. A finalização direta, via OAR, levou em média 1989,76 milisegundos, enquanto a finalização através do ambiente ICE, via o OAR JM-SI, levou em média 2070,76 milisegundos. Esta diferença implica em *overhead* de 81 milisegundos. Este valor mostra que a maior parte do tempo de processamento da infra-estrutura ICE, nesta implementação, é gasto pela chamada ao comando do gerenciador OAR.

Analisando o desvio padrão encontrado nos experimentos, é possível notar, que no caso da ferramenta OAR, existe uma variação considerável que fica na casa de 20% em relação à média observada, tanto na execução direta da ferramenta quanto na implementação OAr JM-SI. No caso do OpenPBS, obteve-se uma estabilidade maior, ou seja, os tempos de execução da amostra não tiveram uma variação muito grande. Ela ficou na casa de 3,61 milisegundos para a finalização através da im-

plementação OpenPBS JM-SI e 2,54 milisegundos através da operação direta da ferramenta OpenPBS. Em termos percentuais, estes valores representam uma variação nos tempos de execução das amostras de 2,59% e 5,03% em relação à média, respectivamente, aos tipos de experimentos.

Da mesma forma como constatado nas demais operações, percebe-se que o *overhead*, imposto pelo ambiente ICE, pode ser tolerado visto a facilidade que ele insere.

5.2.7 Experimentos com a operação de visualização de *status*

A visualização do *status* das aplicações consiste na listagem de todas as aplicações que estão presentes nas filas dos gerenciadores. A Tabela 5.7 possui os dados obtidos com os experimentos de visualização dos *status* das aplicações para ambas ferramentas e implementações das JM-SIs.

Tabela 5.7: Desempenho da operação de verificação do *status* das aplicações

	Média	Desvio Padrão	Percentual
OpenPBS JM-SI	123,22	3,9	3,17
OpenPBS	36,34	0,99	2,73
OAR JM-SI	857,67	6,77	0,79
OAR	765,92	12,95	1,69

Através da análise dos tempos das ferramentas de gerenciamento percebe-se que o gerenciador OpenPBS apresenta um desempenho superior ao verificado com o OAR. Eles possuem, respectivamente, os tempos médios de execução de 36,34 e 765,92 milisegundos.

Assim como em experimentos anteriores, a análise dos experimentos sobre a plataforma OpenPBS mostrou, mais uma vez, que a implementação OpenPBS JM-SI apresentou um tempo de execução médio três vezes maior do que o da execução direta. Os valores mostrados na Tabela 5.7 mostram que a implementação OpenPBS JM-SI apresentou um tempo médio de 123,22 milisegundos, enquanto a execução direta utilizou 36,34 milisegundos. O *overhead* medido ficou em 86,88 milisegundos.

No caso dos experimentos com a ferramenta OAR, verificou-se que o tempo de execução dessa operação, diretamente pelo OAR, foi de 765,92 milisegundos, enquanto o tempo de execução, via implementação OAR JM-SI, foi igual a 857,67 milisegundos. O *overhead* inserido foi de 91,75 milisegundos.

Comparando os tempos das execuções diretas das ferramentas com as implementações das JM-SI, pode-se observar que se obteve o mesmo comportamento da operação de finalização. Isto é, a implementação OpenPBS JM-SI apresentou uma diferença de tempo de execução maior, se comparada com a verificação do estado, diretamente via ferramenta, do que na implementação OAR JM-SI. E da mesma forma como as demais operações comparadas com as execuções diretas das ferramentas, pode-se considerar que os tempos despendidos pelas implementações JM-SI, para operação de verificação dos estado das filas, é compensador.

5.2.8 Experimentos com a operação de recuperação das saídas padrão

A operação de recuperação de informação das saídas padrão não possui correspondente nos gerenciadores, visto que o arquivo pode ser visualizado pelo usuário

em sua própria conta. Sendo assim, a Tabela 5.8 apresenta os resultados alcançados com a avaliação da execução das implementações JM-SI do ICE para cada uma das ferramentas.

Tabela 5.8: Desempenho da operação de recuperação das saídas padrão

	Média	Desvio Padrão	Percentual
OpenPBS JM-SI	11,28	2,54	22,5
OAR JM-SI	10,74	0,74	6,93

Analisando os tempos médios apresentados na Tabela 5.8, verifica-se que os dois JM-SI apresentam um resultado muito semelhante. Como neste caso a infra-estrutura ICE não precisa interagir com os gerenciadores, a diferença de tempo existente entre eles é decorrente, unicamente, das diferenças entre as implementações das JM-SIs e também da carga do *front-end* no momento dos experimentos. No caso do OpenPBS JM-SI são gastos em média 11,28 milisegundos, enquanto no OAR JM-SI são necessários 10,74 milisegundos para recuperar a informação dos arquivos.

Para evitar que o tamanho do arquivo recuperado interferisse nas medições realizadas, optou-se por utilizar um arquivo vazio. Assim o tempo gasto para ler as linhas dos arquivos não foi computada neste experimento. Os resultados apresentados na Tabela 5.8 representam exatamente o *overhead* que a infra-estrutura ICE necessita para recuperar arquivos.

5.3 Resumo

Este capítulo foi dividido em dois momentos. O primeiro contém uma análise comparativa entre o ambiente ICE, proposto neste trabalho, e algumas ferramentas relacionadas que possuem o mesmo escopo de aplicação. Para tanto foram escolhidas duas ferramentas, que se enquadravam nestas características, para serem comparadas com o ICE, as quais são: HPC2N e M3C. A comparação teve como base os critérios estabelecidos por Coulouris em (COULOURIS; DOLLIMORE; KINDBERG, 2005) como sendo os desafios da construção de um sistema distribuído. Alguns dos critérios adotados foram: transparência, extensibilidade, portabilidade, segurança entre outros. No segundo momento foi apresentada uma análise quantitativa do ambiente ICE, onde foram verificados os tempos de execução dos provedores do serviço de gerenciamento de aplicações, implementados e disponibilizados no ambiente. Conforme esperado, o ambiente ICE inseriu um certo *overhead* na execução das operações, se comparado com a execução das operações diretamente através da ferramenta. Em alguns casos, o aumento foi significativo, principalmente nos experimentos realizados sobre a ferramenta OpenPBS. Entretanto, esse aumento no tempo de processamento não chega ser um fator limitante para a utilização do ambiente ICE, visto que os objetivos principais do ambiente ICE são prover: transparência das operações de mais baixo nível a seus usuários, extensibilidade e flexibilidade. Portanto, o *overhead* das operações, disponibilizadas por ele, não é um critério decisivo para definir a adoção ou não do ambiente ICE. Ele serve como um indicativo do custo/benefício que se deseja oferecer aos usuários do ambiente de alto desempenho.

6 CONCLUSÃO

O ambiente ICE - *Integrated Cluster Environment* - foi proposto com base em aspectos que ainda não tinham sido considerados pelas ferramentas atuais de gerenciamento de múltiplos *clusters*, como por exemplo: interoperabilidade, extensibilidade, integração de sistemas legados, transparência, entre outros. O foco deste ambiente é prover gerenciamento e acesso a múltiplos *clusters* de forma transparente, extensível e interoperável. Como apresentado no Capítulo 3, o ambiente ICE tem uma série de objetivos, mas se pode considerar que aqueles que o distinguem dos demais ambientes são: (i) capacidade de uniformização do modo como as ferramentas de *clusters* são utilizadas e, também, a uniformização na maneira como os *clusters* são acessados; (ii) transparência de acesso e uso dos *clusters*; e (iii) capacidade de extensibilidade em dois níveis: o primeiro refere-se à extensibilidade do número de funcionalidades (serviços) providas pelo sistema e o segundo está relacionado com a capacidade do sistema lidar com o uso de diferentes ferramentas que possuem a mesma funcionalidade. Estas principais capacidades podem ser providas pelo ambiente ICE devido ao tipo de arquitetura e ao tipo de *middleware* adotados neste ambiente. Optou-se por utilizar uma arquitetura orientada a serviços (SOA) e como *middleware* utilizou-se Web *Services*. As principais razões para estas escolhas são: o fato da arquitetura SOA poder separar a implementação de um serviço de sua interface; e Web *Services* ser uma tecnologia baseada em arquitetura SOA e em padrões Web, os quais já estão estabelecidos e amplamente utilizados. Estas características são fundamentais para a modelagem e o desenvolvimento do ambiente ICE. É importante ressaltar que o ambiente ICE não é apenas um portal ou então um sistema rodando no *front-end* de um *cluster*. O ambiente ICE é a união entre o ponto único de acesso dos usuários com o *middlware* definido para o gerenciamento e acesso das funcionalidades dos *clusters*.

Uma característica principal da arquitetura ICE é sua modularidade. O objetivo é que se possam inserir novas funcionalidades e novas ferramentas sem que isso altere o modelo do ambiente ICE. A idéia, por trás da arquitetura do ambiente ICE, é fazer com que existam módulos básicos do sistema e uma infra-estrutura capaz de prover serviços de *clusters*. Com base nestas características, definiu-se a arquitetura do ambiente ICE, a qual é formada por dois componentes: *Middleware* de Serviços e Portal Web. No *Middleware* de Serviços são considerados os aspectos ligados à arquitetura SOA empregada no ambiente ICE. Este componente é formado pelos seguintes módulos: *Unified Service Interface* (USI), *Service Implementation* (SI), e *Service Module* (SM). Uma USI é um módulo conceitual, onde o serviço é especificado em alto nível. O SI é um módulo que deve ser desenvolvido de acordo com uma USI. Ele é a implementação do provedor de um serviço. O SM caracteriza-se

por ser a implementação do consumidor de um serviço, também especificado por uma USI e que fará as requisições ao respectivo SI. O Portal Web é formado por três módulos: *Service Management Module* (SMM), *Security Module* (SecM) e *Service Module* (SM). O SMM é responsável pela lógica de controle do sistema Web. No SecM são definidas as ações para o controle de autenticação e autorização do ambiente ICE. O SM é responsável pela apresentação, no Portal Web, dos dados provenientes da porção SM pertencente ao *Middleware* de Serviço. Para cada funcionalidade de *cluster*, que se deseja inserir no ambiente ICE, é preciso a criação da infra-estrutura de serviços e o registro das informações desta nova funcionalidade no Portal Web.

O suporte para extensibilidade, transparência e capacidade de integração com sistemas legados, projetado na arquitetura ICE, pôde ser comprovado na implementação de um protótipo do ambiente ICE (Capítulo 4). Neste protótipo foram desenvolvidas as estruturas básicas do Portal Web e foi provido o suporte para a funcionalidade de gerenciamento de aplicações (*Job Management* - JM). Foi definida uma USI para esta funcionalidade (JM-USI) e foram desenvolvidos dois provedores deste serviço (JM-SIs) e um consumidor (JM-SM). Os JM-SIs desenvolvidos levaram em consideração a integração dos *clusters* existentes no Grupo de Processamento Paralelo e Distribuído (GPPD) do Instituto de Informática da UFRGS. O *cluster* denominado `gppd` possui o OpenPBS como ferramenta de gerenciamento de aplicações, enquanto o *cluster* denominado `frontal-minuano` possui a ferramenta OAR para tal finalidade.

No Capítulo 4 foram explorados os detalhes da definição da USI desta funcionalidade; foi especificado um *framework* para dar suporte à extensão da funcionalidade de gerenciamento de aplicações para diferentes ferramentas, o qual foi denominado *Job Management Framework* (JMF); foram detalhadas as implementações dos provedores deste serviço, denominadas OpenPBS JM-SI e OAR JM-SI, e do consumidor, denominado (JM-SM). Além da descrição completa dos componentes do *Middleware* de Serviços, neste capítulo, também foram apresentadas as estruturas que formam o Portal Web; o modelo de informação definido para armazenar os dados e os relacionamentos dos módulos que compõem o ambiente ICE; e foram ilustradas algumas telas do protótipo desenvolvido com alguns dos processos de configuração e utilização, atualmente disponibilizados neste ambiente.

No intuito de comparar o ambiente ICE com alguns ambientes voltados para o mesmo fim (gerenciamento de múltiplos *clusters*), foram realizadas comparações entre estes ambientes. Estas comparações levaram em consideração os desafios para a construção de sistemas distribuídos apresentados por Coulouris em (COULOURIS; DOLLIMORE; KINDBERG, 2005). A partir desta comparação direta, pôde-se verificar que o ambiente ICE, dentre os ambientes comparados (M3C e HPC2N), possui o maior grau de extensibilidade, interoperabilidade e capacidade de lidar com sistemas legados. Além da comparação das características do ambiente também verificou-se qual o *overhead* inserido pela infra-estrutura do ambiente ICE nos processos de gerenciamento de aplicações. A comparação foi realizada entre a execução direta dos comandos das ferramentas OpenPBS e OAR, instalados no *cluster* `frontal-minuano`, e a execução das operações das respectivas implementações dos módulos JM-SIs, instalados no mesmo *cluster*. Apesar de existir a possibilidade de utilização dos dois *clusters* do GPPD, optou-se por instalar a ferramenta OpenPBS também no *cluster* `frontal-minuano` para que se pudesse realizar

a comparação direta entre as ferramentas e as implementações JM-SI.

Conforme esperado, verificou-se a existência de um *overhead*, que em alguns casos mostrou-se significativo. Analisando os resultados, pode-se dizer, de maneira geral, que a implementação OpenPBS JM-SI apresenta um tempo de execução três vezes maior, se comparado com a execução via ferramenta OpenPBS diretamente. Considerando os experimentos que utilizaram a ferramenta OAR percebeu-se um aumento no tempo de execução. Entretanto, este aumento não ocorreu somente no tempo de execução da implementação OAR JM-SI. A execução direta das operações do OAR possuem um tempo maior comparando-se com o OpenPBS. Isto se deve às diferenças de construção e concepção dessas ferramentas. Através dos experimentos realizados, percebeu-se que o tempo de execução das implementações JM-SI, assim como o *overhead* inserido pelo ambiente ICE, dependem muito da ferramenta de gerenciamento de aplicações sobre a qual o módulo é construído. Outra conclusão está na relação custo/benefício que o ambiente ICE proporciona aos seus usuários. Acredita-se que, ainda, é mais vantajoso para os usuários gastarem um pouco mais de tempo utilizando o ambiente ICE do que realizando as operações diretamente através das ferramentas, onde, apesar de economizarem alguns segundos, teriam que lidar com as especificidades de cada ferramenta.

Enfim, através da definição do ambiente ICE, de sua arquitetura bastante modular e extensível e do desenvolvimento de um protótipo, pôde-se constatar que é viável a construção de um sistema de gerenciamento e acesso a múltiplos *clusters*, que seja capaz de lidar com as especificidades de diferentes *clusters* e de suas diferentes ferramentas de maneira transparente e interoperável. Além disso, também verificou-se que não houve a necessidade de mudar a infra-estrutura já estabelecida no *cluster* do GPPD, onde o ambiente ICE está em operação. Esta característica é bastante relevante, pois isso, mais uma vez, mostra a flexibilidade da arquitetura proposta e da implementação realizada.

A partir da infra-estrutura básica, definida e implementada nesta dissertação, verificou-se a existência de espaço para estender o ambiente, inserindo novas funcionalidades e suportando diferentes ferramentas. Sendo assim, foram desenvolvidos alguns trabalhos em paralelo com base no trabalho descrito nesta dissertação, e também foram identificados outros trabalhos para serem futuramente realizados. A seguir esses trabalhos são apresentados.

6.1 Trabalhos em Paralelo

Dentre os trabalhos em paralelo, que estão sendo realizados, citam-se os seguintes:

- incorporação dos mecanismos de segurança desenvolvidos no trabalho realizado por Alexandre Ilha em (ILHA, 2005), onde um *framework* de segurança para o ambiente ICE foi definido com base na arquitetura proposta neste trabalho;

- incorporação da funcionalidade de monitoramento de aplicações e recursos de *clusters*;

- sistema de *upload* de arquivos para os *front-end* dos *clusters*.

- implementação do ambiente ICE, levando em consideração o modelo MVC (*Model View Controller*);

- porte da implementação atual para uma versão mais robusta utilizando tecnologias J2EE;

As quatro últimas atividades listadas, assim como o desenvolvimento do módulo de gerenciamento de aplicações desta dissertação, foram realizadas no âmbito do projeto Java-WSPAD (Java - Web *Services* para Processamento de Alto Desempenho) (PORTAL DO PROJETO JAVA-WSPAD, 2006), cujo obejtivo é a construção de uma plataforma para a computação distribuída de alto desempenho, baseada em Web *Services* e Peer-to-Peer.

Através do desenvolvimento destes trabalhos, os integrantes do grupo GPPD conseguiram inserir, na plataforma básica do ambiente ICE, outras funcionalidades de maneira bastante modular. Estas novas funcionalidades contribuem para que se consiga ter o ambiente completo de gerenciamento e acesso de múltiplos *clusters* almejado na proposta desta dissertação.

6.2 Trabalhos Futuros

Como trabalhos futuros, alguns podem ser citados:

- prover suporte para um número maior de ferramentas de gerenciamento de *clusters*;

- prover suporte para lançamento de aplicações paralelas, baseadas em outras bibliotecas e ambientes paralelos, como por exemplo: LAN-MPI, PVM, cJAVA (SILVA; LOBOSCO; AMORIM, 2003; LOBOSCO; LOQUES; AMORIM, 2005), etc;

- definição de serviços de gerenciamento administrativo de *clusters*, isto é, permitir que os administradores dos *clusters* possam realizar intervenções de configuração nos *clusters* por eles gerenciados a partir do ambiente ICE;

Ainda como um trabalho futuro, mas não ligado diretamente à implementação do ambiente, pretende-se colocar este ambiente em um centro de pesquisa formado, basicamente, por usuários que não são nativos da área de computação. O objetivo desta avaliação é perceber quais as dificuldades que esses usuários possuem ao interagir com o ambiente ICE e, através do *feedback* dos mesmos, melhorá-lo. Enfim, de maneira geral, pode-se dizer que o ambiente ICE possui uma base sólida para ser estendido e melhorado a fim de se tornar um ambiente amplamente utilizado no cenário de múltiplos *cluster*.

REFERÊNCIAS

ABBAS, A. (Ed.). **Grid Computing**: a practical guide to technology and applications. [S.l.]: Charles River Media, 2004. 408p.

ADABALA, S.; KAPADIA, N. H.; FORTES, J. A. B. Performance and interoperability issues in incorporating cluster management systems within a wide-area network-computing environment. In: ACM/IEEE CONFERENCE ON SUPERCOMPUTING, 2000, Dallas, Texas, USA. **Proceedings...** Washington: IEEE Computer Society, 2000. 1 CD-ROM.

ALONSO, G. et al. **Web Services**: conepts, architectures and applications. [S.l.]: Springer, 2004. 354p.

ALVES, R. S.; MARQUEZAN, C. C.; GRANVILLE, L. Z. Experiences in the Implementation of an SNMP-Based High Performance Cluster Management System. In: IEEE SYMPOSIUM ON COMPUTERS AND COMMUNICATIONS, ISCC 9., 2004, Alexandria, Egito. **Proceedings...** [S.l.]: IEEE Computer Society, 2004.

ALVES, R. S.; MARQUEZAN, C. C.; GRANVILLE, L. Z.; NAVAUX, P. O. A. High Performance Cluster Management Based on SNMP: experiences on integration between network patterns and cluster management concepts. In: INTERNATIONAL CONFERENCE ON TELECOMMUNICATIONS, ICT, 2004, Fortaleza. **Telecommunications and Networking - ICT, 2004:** Proceedings. Berlin: Springer, 2004. p.782–791. (Lecture Notes in Computer Science, v.3124).

ALVES, R. S.; MARQUEZAN, C. C.; GRANVILLE, L. Z.; NAVAUX, P. O. A. Integração do Gerenciamento de Clusters de Alto Desempenho com o Gerenciamento de Redes através de Solução baseada em SNMP. In: SIMPóSIO BRASILEIRO DE REDES DE COMPUTADORES, SBRC, 23., 2005, Fortaleza, Brasil. **Anais...** Fortaleza: Sociedade Brasileira de Computação, 2005. p.3–16.

ANDRADE, N. et al. OurGrid: an approach to easily assemble grids with equitable resource sharing. In: JOB SCHEDULING STRATEGIES FOR PARALLEL PROCESSING, 2003. **Proceedings...** Berlin: Springer, 2003. p.61–86. (Lecture Notes in Computer Science, v.2862).

ANDRONICO, G. et al. The GENIUS web portal: grid computing made easy. In: INTERNATIONAL SYMPOSIUM ON INFORMATION TECHNOLOGY. ITCC, 2003. **Proceedings...** [S.l.]: IEEE Computer Society, 2003. p.425–431.

APACHE AXIS2, A. W. to. Disponível em: ⟨http://ws.apache.org/axis2/⟩. Acesso em: nov. 2005.

APACHE TOMCAT - The Apache Software Foundation. Disponível em: ⟨http://tomcat.apache.org/⟩. Acesso em: maio 2006.

AUMAGE, O. Heterogeneous Multi-Cluster Networking with the Madeleine III Communication Library. In: INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, IPDPS, 2002. **Proceedings...** [S.l.]:IEEE Computer Society, 2002.

BAKER, M. **Cluster Computing White Paper**. Disponível em: ⟨http://dsg.port.ac.uk/~mab/Links/tfcc/WhitePaper/final-paper.pdf⟩. Acesso em: dez. 2005.

BAKER, M. et al. Emerging Grid Standards. **IEEE Computer**, [S.l.], v.38, n.4, p.43–50, Apr. 2005.

BAKER, M.; SMITH, G. GridRM: an extensible resource monitoring system. In: IEEE INTERNATIONAL CONFERENCE ON CLUSTER COMPUTING, 2003, Tsim Sha Tsui, Kowloon, Hong Kong. **Proceedings...** [S.l.: s.n.], 2003. p.207– 214. Disponível em: ⟨http://ieeexplore.ieee.org/iel5/8878/28041/01253317.pdf?tp=&arnumber=1253317&isnumber=28041⟩. Acesso em: dez. 2005.

BARRETO, M.; ÁVILA, R.; NAVAUX, P. The MultiCluster Model to the Integrated Use of Multiple Workstation Clusters. In: WORKSHOP ON PERSONAL COMPUTER BASED NETWORKS OF WORKSTATIONS, 3., 2000, Cancun. **Proceedings...** Berlin: Springer-Verlag, 2000. p.71–80. (Lecture Notes in Computer Science, v.1800).

BARRETO, M.; ÁVILA, R.; OLIVEIRA, F. de; CASSALI, R.; NAVAUX, P. DECK: an enviroment for parallel programming on clusters of multiprocessors. In: SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH-PERFORMANCE COMPUTING - SBAC-PAD, 12., 2000, São Pedro, SP. **Proceedings...** São Carlos: UFSCAR, 2000. p.321–329.

BENEDYCZAK, K. et al. UNICORE as Uniform Grid Environment for Life Sciences. In: EUROPEAN GRID CONFERENCE, ADVANCES IN GRID COMPUTING - EGC, 2005, Amsterdam, Netherlands. **Proceedings...** [S.l.]: Springer, 2005. p.364–373. (Lecture Notes in Computer Science, v.3470).

BRIM, M. et al. M3C: managing and monitoring multiple clusters. In: INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, CCGRID, 2001, Brisbane, Australia. **Proceedings...** [S.l.]: IEEE Computer Society, 2001. p.386–393.

BRONEVETSKY, G. et al. Application-level checkpointing for shared memory programs. In: INTERNATIONAL CONFERENCE ON ARCHITECTURAL SUPPORT FOR PROGRAMMING LANGUAGES AND OPERATING SYSTEMS, 11., 2004, Boston, MA, USA. **Proceedings...** New York: ACM Press, 2004. p.235–247.

BRUNO, G. et al. Rolls: modifying a standard system installer to support user-customizable cluster frontend appliances. In: IEEE INTERNATIONAL CONFERENCE ON CLUSTER COMPUTING, 2004, San Diego, California, USA. **Proceedings...** [S.l.: s.n.], 2004. p.421–430. Disponível em: ⟨http://www.rocksclusters.org/rocks-documentation/4.1/papers/cluster2004-roll.pdf⟩. Acesso em: dez. 2005.

BUYYA, R. (Ed.). **High Performance Cluster Computing**: architectures and systems. Upper Saddle River: Prentice Hall PTR, 1999. 849p.

CAPIT, N. et al. **Expèriences autour d'une nouvelle approche de conception d'un gestionnaire de travaux pour grappe**. [S.l.]: HAL - CCSd - CNRS, 2003.

CAPIT, N. et al. A batch scheduler with high level components. In: INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND GRID, CCGRID, 5., 2005. **Proceedings...** [S.l.: s.n.], 2005.

CASE, J. et al. **A Simple Network Management Protocol (SNMP)**: IETF RFC 1098. Disponível em: ⟨http://www.ietf.org/rfc/rfc1098.txt?number=1098⟩. Acesso em: dez. 2005.

CERA, M. C.; ROSA RIGHI, R. da; PASIN, M. Alocação Dinâmica e Transparente de Computadores Ociosos em Java. In: WORKSHOP EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO, WSCAD, 6., 2005, Rio de Janeiro, RJ. **Anais...** Rio de Janeiro: Sociedade Brasileira de Computação, 2005.

CERAMI, E. **Web Services Essentials**. [S.l.]: O'Reilly & Associates, 2002.

COOKE, A. et al. R-GMA An Information Integration System for Grid Monitoring. In: INTERNATIONAL CONFERENCE ON COOPERATIVE INFORMATION SYSTEMS, COOPIS, 11., 2003, Catania, Sicily, Italy. **Proceedings...** [S.l.: s.n.], 2003. Disponível em: ⟨http://scholar.google.com/url?sa=U&q=http://www.macs.hw.ac.uk/dis4g/publications/coopis.pdf⟩. Acesso em: dez. 2005.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. (Ed.). **Distributed Systems - Concepts and Design. Fourth Edition**. [S.l.]: Addison Wesley, 2005. 832p.

DAHAN, M. et al. Grid Portal Toolkit 3.0 (GridPort). In: INTERNATIONAL SYMPOSIUM ON HIGH-PERFORMANCE DISTRIBUTED COMPUTING, 13., HPDC, 2004, Honolulu, Hawaii, USA. **Proceedings...** [S.l.]: IEEE Computer Society, 2004. p.272–273.

DONGARRA, J. et al. **An Introduction to the MPI Standard**. Knoxville, USA: University of Tennessee, 1995. (Technical report, CS-95-274).

ELMROTH, E.; NYLÉN, M.; OSCARSSON, R. A User-Centric Cluster and Grid Computing Portal. In: ICPP WORKSHOPS, 2005. **Proceedings...** [S.l.]:IEEE Computer Society, 2005. p.103–110.

FLANERY, R. et al. Cluster command and control (CS) tools suite. In: KACSUK, P.; KOTSIS, G. (Ed.). **Distributed and parallel systems**: from instruction parallelism to cluster computing. Norwell, MA, USA: [S.l.]: Springer, 2000. p.205–214.

FOSTER, I. et al. **The Physiology of the Grid**: an open grid services architecture for distributed systems integration. [S.l.:s.n.], 2002.

FOSTER, I.; KESSELMAN, C. Globus: a metacomputing infrastructure toolkit. **The International Journal of Supercomputer Applications and High Performance Computing**, [S.l.], v.11, n.2, p.115–128, Summer 1997.

FOSTER, I.; KESSELMAN, C.; TUECKE, S. The Anatomy of the Grid: enabling scalable virtual organizations. **International Journal of High Performance Computing Applications**, Thousand Oaks, CA, USA, v.15, n.3, p.200–222, 2001.

GENTZSCH, W. Sun Grid Engine: towards creating a compute power grid. In: IEEE/ACM INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, 1., 2001, Brisbane, Australia. **Proceedings...** [S.l.]: IEEE Computer Society, 2001. p.35–36.

GLOBAL Grid Forum. Disponível em: ⟨`http://www.gridforum.org/`⟩. Acesso em: nov. 2005.

GLOBUS - Welcome to the Globus Toolkit Homepage. Disponível em: ⟨`http://www.globus.org/toolkit/`⟩. Acesso em: nov. 2005.

GLOBUS Toolkit 4.0 Release Manuals. Disponível em: ⟨`http://www.globus.org/toolkit/docs/4.0/key/`⟩. Acesso em: nov. 2005.

GONZ, F. J. et al. Condor grid computing from mobile handheld devices. **SIGMOBILE Mob. Comput. Commun. Rev.**, New York, NY, USA, v.6, n.2, p.18–27, 2002.

GRANVILLE, L.; TAROUCO, L. QAME - QoS-aware management environment. In: COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE, COMPSAC, 25., 2001, Chicago, Illinois, USA. **Proceedings...** [S.l.: s.n.], 2001. p.269–274. Disponível em: ⟨`http://ieeexplore.ieee.org/iel5/7609/20754/00960627.pdf?tp=&arnumber=960627&isnumber=20754`⟩. Acesso em: dez. 2005.

GRIMES, R. **Professional DCOM Programming**. [S.l.]: Birmingham, UK: Wrox, 1997.

GRIMSHAW, A. S.; NATRAJAN, A. Legion: lessons learned building a grid operating system. **Proceedings of the IEEE**, [S.l.], v.93, n.3, p.589–603, 2005.

GT 4.0 WS_GRAM. Disponível em: ⟨`http://www-unix.globus.org/toolkit/docs/4.0/execution/wsgram/`⟩. Acesso em: dez. 2005.

HADDAD, I. F. PVFS: a parallel virtual file system for linux clusters. **Linux Journal**, Seattle, WA, USA, v.2000, n.80es, p.5, 2000.

HANSEN, S.; FOSSUM, T. V. Refactoring model-view-controller. **Journal of Computing Sciences in Colleges**, USA, v.21, n.1, p.120–129, 2005.

ILHA, A. S. **Uso de Web Services no Controle de Acesso a Clusters**. 2005. Trabalho de Conclusão (Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

JAVA Platform, Standard Edition (Java SE). Disponível em: ⟨http://java.sun.com/javase/index.jsp⟩. Acesso em: maio 2006.

JAVA SERVLET Technology. Disponível em: ⟨http://java.sun.com/products/servlet/⟩. Acesso em: maio 2006.

JAVASERVER Pages Technology. Disponível em: ⟨http://java.sun.com/products/jsp/⟩. Acesso em: maio 2006.

KASSICK, R.; MACHADO, C.; HERMANN, E.; ÁVILA, R.; NAVAUX, P.; DENNEULIN., Y. Evaluating the performance of the dNFSP file system. In: IEEE/ACM INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, CCGRID, 5., 2005, Cardiff, UK. **Proceedings...** [S.l.: s.n.], 2005.

KELLER, A.; REINEFELD, A. CCS resource management in networked HPC systems. In: HETEROGENEOUS COMPUTING WORKSHOP, HCW, 7., 1998, Orlando, Florida. **Proceedings...** [S.l.: s.n.], 1998. p.44–56.

LI, M.; BAKER, M. **The Grid Core Technologies**. [S.l.]: John Wiley & Sons, 2005. 423p.

LI, M. et al. PortalLab: a web services toolkit for building semantic grid portals. In: INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, 3., CCGRID, 2003, Tokyo, Japan. **Proceedings...** [S.l.]: IEEE Computer Society, 2003. p.190–197.

LIGNERIS, B. des et al. **Open Source Cluster Application Resources (OSCAR)**: design, implementation and interest for the Computer scientific community. White paper. Disponível em: ⟨http://www.ncassr.org/projects/cluster-sec/papers/oscar03.pdf⟩. Acesso em: dez. 2005.

LOBOSCO, M.; LOQUES, O.; AMORIM, C. L. de. Reducing Memory Sharing Overheads in Distributed JVMs. In: INTERNATIONAL CONFERENCE HIGH PERFORMANCE COMPUTING AND COMMUNICATIONS, HPCC, 1., 2005, Sorrento, Italy. **Proceedings...** [S.l.]: Springer, 2005. p.629–639. (Lecture Notes in Computer Science, v.3726).

MARQUEZAN, C. C.; ROSA RIGHI, R. da; SCHNORR, L.; CARISSIMI, A.; MAILLARD, N.; NAVAUX, P. O. A. ICE: a service oriented approach to uniform the access and management of cluster environments. In: IEEE INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, CCGRID, 6., 2006, Singapore. **Proceedings...** [S.l.]:IEEE Computer Society, 2006. 1 CD-ROM.

MAUI Scheduler Open Cluster Software. Disponível em: ⟨http://mauischeduler.sourceforge.net/⟩. Acesso em: dez. 2005.

MCGOVERN, J. et al. **Java Web Services Architecture**. [S.l.]: Morgan Kaufmann, 2003. 833p.

NAKADA, H. et al. The design and implementation of a fault-tolerant RPC system: ninf-c. In: INTERNATIONAL CONFERENCE ON HIGH PERFORMANCE COMPUTING AND GRID IN ASIA PACIFIC REGION, 7., 2004,

Omiya Sonic City, Tokyo Area, Japan. **Proceedings...** [S.l.: s.n.], 2004. p.9–18. Disponível em: ⟨http://ieeexplore.ieee.org/iel5/9244/29307/01324011.pdf?tp=&arnumber=1324011&isnumber=29307⟩. Acesso em: dez. 2005.

NOVOTNY, J.; RUSSELL, M.; WEHRENS, O. GridSphere: an advanced portal framework. In: EUROMICRO CONFERENCE, 30., 2004, Rennes, France. **Proceedings...** [S.l.]: IEEE Computer Society, 2004. p.412–419.

OASIS - Advancing E-Business Standards Since 1993. Disponível em: ⟨http://www.oasis-open.org/home/index.php⟩. Acesso em: nov. 2005.

OASIS-WS - OASIS Committees by Category: web services. Disponível em: ⟨http://www.oasis-open.org/committees/tc_cat.php?cat=ws⟩. Acesso em: nov. 2005.

OPENSCE - Scalable Cluster Environment. Disponível em: ⟨http://opensce.org⟩. Acesso em: dez. 2005.

OSCAR - Open Source Cluster Application Resources. Disponível em: ⟨http://oscar.openclustergroup.org/⟩. Acesso em: dez. 2005.

OU, L.; HE, X. Design and Evaluation of a High Performance Parallel File System. In: ANNUAL IEEE CONFERENCE ON LOCAL COMPUTER NETWORKS, 30., LCN, 2005, Sydney, Australia. **Proceedings...** [S.l.]: IEEE Computer Society, 2005. p.100–105.

PAPADOPOULOS, P. M.; KATZ, M. J.; BRUNO, G. NPACI Rocks Clusters: tools for easily deploying and maintaining manageable high-performance linux clusters. In: RECENT ADVANCES IN PARALLEL VIRTUAL MACHINE AND MESSAGE PASSING INTERFACE, PVM/MPI, 8., 2001, Santorini/Thera, Greece. **Proceedings...** [S.l.]: Springer, 2001. p.10–11. (Lecture Notes in Computer Science, v.2131).

PARK, I. et al. Towards an integrated, web-executable parallel programming tool environment. In: ACM/IEEE CONFERENCE ON SUPERCOMPUTING, 2000, Dallas, Texas, USA. **Proceedings...** Washington: IEEE Computer Society, 2000. 1 CD-ROM.

PORTABLE Batch System. Disponível em: ⟨http://www.openpbs.org/⟩. Acesso em: nov. 2005.

PORTAL do Projeto Java-WSPad. Disponível em: ⟨http://www.lcp.coppe.ufrj.br:9673/JavaWSPad⟩. Acesso em: jul. 2006.

POSTGRESQL Global Development Group. Disponível em: ⟨http://www.postgresql.org/⟩. Acesso em: maio 2006.

RED HAT - The Open Source Leader. Disponível em: ⟨http://www.redhat.com/⟩. Acesso em: dez. 2005.

RMI - Java Remote Method Invocation. Disponível em: ⟨http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html⟩. Acesso em: dez. 2005.

ROCKS Cluster Distribution: users guide. Disponível em: ⟨http://www.rocksclusters.org/rocks-documentation/4.1/introduction.html⟩. Acesso em: dez. 2005.

SACERDOTI, F. D. et al. Wide Area Cluster Monitoring with Ganglia. In: IEEE INTERNATIONAL CONFERENCE ON CLUSTER COMPUTING, CLUSTER, 2003, Hong Kong. **Proceedings...** [S.l.: s.n.], 2003. p.289–298.

SCHULZ, M. et al. Implementation and Evaluation of a Scalable Application-Level Checkpoint-Recovery Scheme for MPI Programs. In: ACM/IEEE CONFERENCE ON SUPERCOMPUTING, 2004, Washington, DC, USA. **Proceedings...** [S.l.]: IEEE Computer Society, 2004. 1 CD-ROM.

SIEGEL, J. **CORBA Fundamentals and Programming**. [S.l.]: John Wiley & Sons, 1996.

SILVA, A. F. da; LOBOSCO, M.; AMORIM, C. L. de. An Evaluation of cJava System Architecture. In: SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING, SBAC-PAD, 15., 2003, São Paulo, SP, BR. **Proceedings...** [S.l.]: IEEE Computer Society, 2003. p.91–99.

SOTTILE, M. J.; MINNICH, R. G. Supermon: a high-speed cluster monitoring system. In: IEEE INTERNATIONAL CONFERENCE ON CLUSTER COMPUTING, 2002, Chicago, Illinois. **Proceedings...** [S.l.]: IEEE Computer Society, 2002. p.39–46.

STERLING, T. **An Introduction to PC Clusters for High Performance Computing**. Capítulo 1 do white-paper entitulado Cluster Computing White Paper, editado por Mark Baker. Disponível em: ⟨http://dsg.port.ac.uk/~mab/Links/tfcc/WhitePaper/final-paper.pdf⟩. Acesso em: dez. 2005.

SUNDERAM, V. PVM: a framework for parallel distributed computing. **Concurrency: Practice and Experience**, Chichester, UK, v.2, n.4, p.315–339, 1990.

SUZUMURA, T. et al. GridSpeed: a web-based grid portal generation server. In: INTERNATIONAL CONFERENCE ON HIGH PERFORMANCE COMPUTING AND GRID IN ASIA PACIFIC REGION, 7., 2004, Omiya Sonic City, Tokyo Area, Japan. **Proceedings...** [S.l.: s.n.], 2004. p.26–33. Disponível em: ⟨http://ieeexplore.ieee.org/iel5/9244/29307/01324013.pdf?tp=&arnumber=1324013&isnumber=29307⟩. Acesso em: dez. 2005.

SYSTEM Imager. Disponível em: ⟨http://www.systemimager.org/⟩. Acesso em: dez. 2005.

THAIN, D.; LIVNY, M. Building Reliable Clients and Servers. In: FOSTER, I.; KESSELMAN, C. (Ed.). **The Grid**: blueprint for a new computing infrastructure. [S.l.]: Morgan Kaufmann, 2003.

THAIN, D.; TANNENBAUM, T.; LIVYN, L. **Condor and the Grid**. Capítulo 11 do livro entitulado Grid Computing - Making the Global Infrastructure a Reality.: Editado por F. Berman, A. Hey e G. Fox. Publicado por John Wiley & Sons. Disponível em: ⟨http://scholar.google.com/url?sa=U&q=http://searchoracle.

techtarget.com/searchOracle/downloads/Grid_Computing_Chap11.pdf⟩. Acesso em: dez. 2005.

THE GLOBUS Alliance. Disponível em: ⟨http://www.globus.org/alliance/⟩. Acesso em: nov. 2005.

THOMAS, M. et al. The GridPort Toolkit: a system for building grid portals. In: INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING, HPDC, 10., 2001, San Francisco, CA, USA. **Proceedings...** [S.l.]: IEEE Computer Society, 2001. p.216–227.

TIERNEY, B. et al. **A Grid Monitoring Architecture**. White paper. Disponível em: ⟨http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-3.pdf⟩. Acesso em: dez. 2005.

TOP500 Supercomputer Sites. Disponível em: ⟨http://www.top500.org/⟩. Acesso em: dez. 2005.

UML - Unified Modeling Language. Disponível em: ⟨http://www.uml.org/⟩. Acesso em: maio 2006.

UTHAYOPAS, P.; ANGSKUN, T.; MANEESILP, J. SCE: a fully integrated software tool for beowulf cluster system. In: LINUX CLUSTER: THE HPC REVOLUTION, A CONFERENCE FOR HIGH-PERFORMANCE LINUX CLUSTER USERS AND SYSTEM ADMINISTRATORS, 2001, Ubana,Illinois, USA. **Proceedings...** [S.l.: s.n.], 2001.

UTHAYOPAS, P.; ANGSKUN, T.; MANEESILP, J. On the Building of the Next Generation Integrated Environment for Beowulf Clusters. In: INTERNATIONAL SYMPOSIUM ON PARALLEL ARCHITECTURES, ALGORITHMS AND NETWORKS, I-SPAN, 2002, Makati City, Metro Manila, Philippines. **Proceedings...** [S.l.: s.n.], 2002. p.159–164.

UTHAYOPAS, P.; PHATANAPHEROM, S. Fast and Scalable Real-Time Monitoring System for Beowulf Clusters. In: EUROPEAN PVM/MPI USER'S GROUP MEETING ON RECENT ADVANCES IN PARALLEL VIRTUAL MACHINE AND MESSAGE PASSING INTERFACE, PVM/MPI, 8., 2001. **Proceedings...** London: Springer-Verlag, 2001. p.201–208. (Lecture Notes in Computer Science, v.2131).

VALLEE, G. et al. SSI-OSCAR: a cluster distribution for high performance computing using a single system image. In: INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE COMPUTING SYSTEMS AND APPLICATIONS, HPCS, 19., 2005, Guelph,Ontario, Canadá. **Proceedings...** [S.l.: s.n.], 2005. p.319–325.

W3C Architecture Domain - Web Services Activity. Disponível em: ⟨http://www.w3.org/2002/ws/⟩. Acesso em: nov. 2005.

W3C World Wide Web - Leading the Web to Its Full Potential... Disponível em: ⟨http://www.w3.org/⟩. Acesso em: nov. 2005.

W3C-XML BINARY - XML Binary Characterization Working Group Public Page. Disponível em: ⟨http://www.w3.org/XML/Binary/⟩. Acesso em: dez. 2005.

WELCH, V. et al. Security for Grid Services. In: IEEE INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING, 12., 2003, Seatle, Washington. **Proceedings...** [S.l.: s.n.], 2003. p.48–57.

ZOU, H. et al. HRIC: hybrid resource information service architecture based on gma. In: IEEE INTERNATIONAL CONFERENCE ON E-BUSINESS ENGINEERING, ICEBE, 2005, Beijing, China. **Proceedings...** [S.l.: s.n.], 2005. p.541–544.

ÁVILA, R.; NAVAUX, P.; LOMBARD, P.; LEBRE, A.; DENNEULIN, Y. Performance evaluation of a prototype distributed NFS server. In: SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING, SBAC-PAD, 16., 2004, Foz do Iguaçu, PR, BR. **Proceedings...** [S.l.]: IEEE Computer Society, 2004. p.100–105.