Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Informática

Dissertação de Mestrado

AutoMan

Gerência Automática de Grades Computacionais Entre-Pares

Celso Augusto Raposo Lisboa Brennand

Campina Grande, Paraíba, Brasil

Fevereiro - 2008

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Informática

AutoMan

Gerência Automática de Grades Computacionais Entre-Pares

Celso Augusto Raposo Lisboa Brennand

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Redes de Computadores e Sistemas Distribuídos

Marco Aurélio Spohn
(Orientador)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

B838g
2008

Brennand, Celso Augusto Raposo Lisboa. Gerência Automática de Grades Computacionais Entre-Pares / Celso Augusto Raposo Lisboa Brennand. - Campina Grande, 2008.
77f. : il. Col.

Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.
Orientador: Dr. Marco Aurélio Spohn.
Referências.

1. Sistemas Distribuídos. 2. Sistemas Peer-to-Peer. 3. Gerenciamento Automático. 4. Monitoração de Sistemas Distribuídos I. Título.

CDU - 681.324(043)

"AutoMan: GERÊNCIA AUTOMÁTICA DE GRADES COMPUTACIONAIS ENTRE-PARES"

CELSO AUGUSTO RAPOSO LISBOA BRENNAND

DISSERTAÇÃO APROVADA COM DISTINÇÃO EM 29.02.2008

PROF. MARCO AURÉLIO SPOHN, Ph.D
Orientador

PROF. FRANCISCO VILAR BRASILEIRO, Ph.D
Examinador

RAQUEL VIGOLVINO LOPES , D.Sc
Examinadora

PROFa. LÍVIA MARIA RODRIGUES SAMPAIO, D.Sc
Examinadora

CAMPINA GRANDE – PB

"Todo dia é dia

Toda hora é hora

De saber que esse mundo é seu

Se você for amigo e companheiro

Com alegria e imaginação

Vivendo e sorrindo

Criando e rindo

Será muito feliz

E todos serão também ..."

Agradecimentos

Agradeço primeiramente a Deus e a todos que me ajudaram em todos os momentos difíceis da minha vida.

Agradeço aos meus pais Geraldo e Sônia, meu irmão Brenno e meus avós, Anacleto, Angélica, Ana, minha namorada Michelle. Agradeço ao meu orientador Marco Spohn por todos os ensinamentos, companheirismo e educativos "Pongs"durante todo o mestrado que me fez crescer enormemente sendo um verdadeiro pai.

Também gostaria de expressar minha honra em ser seu primeiro aluno de mestrado. Ao professor Francisco Brasileiro pela reuniões e grandes contribuições para o trabalho. Ao professor Dalton Dario Serey Guerrero com os seus grandes ensinamentos e a grande oportunidade que ele me deu de participar da maratona de computação.

Ao professor Walfredo Cirne que durante pouco tempo de convívio contribuiu bastante com o trabalho e como tirar fotos maravilhosas. Gostaria de agradecer a duas pessoas em especial que foram praticamente meus outros dois orientadores, a minha vó do mestrado Ayla Débora e meu irmão mais velho Álvaro Coêlho, também conhecido pelos doidos da cidade dele como Degas. Os dois sempre estiveram presentes em todos os momentos do mestrado, ajudado com todas as forças durante os meus momentos de desespero e também sorrindo e se divertindo com todo o seu entusiasmo nos momentos de descontração. Plagiando uma frase que acho que foi de Fubica: "Ayla e Degas eram meus guias com tochas guiando o meu caminho nesta incrível e desafiadora jornada. Ayla na frente iluminado o meu caminho e Degas atrás impedindo que eu recuasse".

A Milena Micheli a coaching do coração que sempre estava pronta para ajudar vinha sempre com frases clássicas "você preencheu o Xplaner? "ou "vai ter o acarajé la em casa". A Sidney Dória, meu grande amigo no mestrado que me colocou e me tirou de várias encrencas.

A Saulo de Tarso, meu companheiro do AP 101 do Residência Flamingo e companheiro para

todas as horas. Também gostaria de agradecer aos meus amigos e companheiros de trabalho da linha AutoMan David Candeia, Guilherme Germoglio, Flávio Santos, Gustavo Pereira, que ajudaram no desenvolvimento do AutoMan e sem eles não teria crescido tanto como pesquisador e pessoa.

A Flavio Vinicius (Peruca) e Paulo Ditarso que dividiram a sala na segunda fase do mestrado e viraram grandes amigos de trabalho da vida e de cachaça. Como também outros grades amigos do LSD que sempre tomava uma cervejinha na sinuca no amarelinho em Danielys..., são eles: Marcus Carvalho (Marquinhos) o topa tudo, João Arthur o Sem sintoma, Leonardo de Assis (Conde do Amor) , Nazareno Andrade (Naza) o massada, Thiago Emmanuel (Manel) o tico meleca, Erick Moreno do Disco Voador, Cicero Alan(Toquinho) que durante a escrita não deixou de ir pelo menos 3 vezes ao dia perguntar se eu já tinha terminado. Ao grande casal Voorsluys, Bárbara com sua grande compreensão e alegria, e William sempre pronto a me aturar e ajudar. Não esquecendo suas festas do dia das bruxas.

Aos Alagoanos Felipe Pontes, Leandro Sales, Mario, companheiro de mestrado e de discussões em bares. Os amigos do OurBackup Marcelo Iury, Alexan-

dro(Gonzaguinha), Eduardo Colaço, Paolo Victor. A Rodrigo Vilar que deu grandes contribuições para o trabalho. As meninas super poderosas Lívia Maria Rodrigues Sampaio, Raquel Vigolvino Lopes que além de me ajudar durante todo mestrado participaram da minha banca examinadora (A terceira menina super poderosa é Ayla). Agradecer também ao pessoal do suporte Carla de Araujo (Carlinha), Moises Rodrigues, Roberto Wiest, Thiago Nóbrega e Zane Cirne o qual eu perturbei tanto durante o tempo que passei no LSD. As secretarias do curso Aninha e Vera por sempre estar nos atendendo com um alto astral.

. E aos meus amigos e companheiros de trabalho no LSD e fora dele, Abmar Grangeiro de Barros, Cleide, Eliane, Giovanni Farias, Jaindson Valentim, Zé Flávio, Lauro Beltrão, Lile Hattori, Matheus Gaudencio, Ricardo Araújo, Vinícius Ferraz, Pablo Tiburcio, Marcelo Meira, Daniel Fireman, Emiliano Rostand, Pryscilla Dóra (gaucha danada da peste). Aos grandes amigos da minha terra natal (Recife) Charles Sobral, Hugo Cabral, Augusto César e os malas Carlos Batista, Kosme Lustosa e Ygo Batista (Dota dota...ta já vou!). Este trabalho foi desenvolvido em colaboração com a HP Brasil P&D.


LSD
OurGrid
AutoMan

Conteúdo

Lista de Símbolos

CDDLM - *Configuration Description, Deployment and Lifecycle Management*

CPU - *Central Processing Unit*

GGF - *Global Grid Forum*

GMA - *Grid Monitoring Architecture*

GuM - *Grid Machine*

HP - *Hewlett-Packard*

JIC - *Java Internet Communication*

JMX - *Java Management Extensions*

LSD - *Laboratório de Sistemas Distribuídos*

NoF - *Network of Favors*

OGSA - *Open Grid Services Architecture*

OV - *Organização Virtual*

ReGS - *Reporting Grid Service*

RMI - *Remote Method Invocation*

SSH - *Secure Shell*

SOA - *Service Oriented Architecture*

SOAP - *Simple Object Access Protocol*

TIC - *Tecnologia de Informação e Comunicação*

WSDL - *Web Services Description Language*

WWW - *World Wide Web*

XML - *eXtensible Markup Language*

XMPP - *Extensible Messaging and Presence Protocol*

Lista de Figuras

Lista de Tabelas

Resumo

Grades computacionais vem sendo mais e mais usadas tanto para dar suporte às atividades de e-ciência, como pela indústria. Um grande desafio nesta área é a capacidade de gerência automática, que visa melhorar a disponibilidade da grade bem como simplificar o trabalho de administradores. Neste trabalho apresentamos o AutoMan, um sistema cujo principal objetivo é fornecer um certo nível de gerenciamento automático para uma grade computacional entre-pares e de livre acesso. Dessa forma, busca-se otimizar o uso de recursos perecíveis nessa grade, ao mesmo tempo que se simplifica as atividades de gerência do sistema. As principais contribuições deste trabalho são: a) a proposta para arquitetura e a implementação de Automan; e b) uma discussão sobre as lições aprendidas ao se aplicar gerenciamento automático para a grade computacional OurGrid. Os resultados apontam que, quando comparado com a abordagem de gerência manual, Automan promove uma melhoria substancial na disponibilidade da grade.

Abstract

Grid computing has been more and more adopted in eScience and in companies. One important challenge in grid systems is the ability for automatic management, improving grid availability and simplifying administrators work. In this work we present AutoMan, a system whose main objective is to provide some degree of automatic management to a free-to-join peer-to-peer grid system, optimizing the utilization of perishable resources in the grid and simplifying the management of the system. The main contributions of this work are: a) the architecture and implementation proposed for AutoMan; and b) a discussion on the lessons learned from applying automatic management to a OurGrid grid system. We show through an evaluation that, when compared to a manual management approach, AutoMan substantially improves the grid availability.

Capítulo 1

Introdução

Nos últimos anos as Tecnologias de Informação e Comunicação (TICs) têm exercido uma grande influência na forma como a pesquisa científica é conduzida. Graças às facilidades de comunicação providas pelos avanços nas redes de computadores, a colaboração entre pesquisadores é hoje muito mais freqüente. Além disso, avanços na capacidade de processamento tornaram factíveis diversas técnicas simulação e análise de dados. Por conseguinte, passou a ser uma importante ferramenta para a expansão do conhecimento nas mais variadas áreas de pesquisa. O uso das TICs para dar suporte ao desenvolvimento científico tem sido chamado de e-ciência (ou *e-Science* em inglês). Como resultado dessa revolução, muitos laboratórios de pesquisa passaram a demandar serviços computacionais em uma quantidade e especificidade que não podem ser facilmente providos. Por outro lado, esses laboratórios também possuem muitos recursos ociosos (*e.g.*, máquinas ociosas em horário noturno e/ou finais de semana) que poderiam ser usados por outros laboratórios. Essa demanda de recursos ociosos em certos períodos, levou ao desenvolvimento de estratégias, tecnologias e técnicas de compartilhamento de recursos, entre as quais se destacam aquelas relacionadas com o conceito de grades computacionais.

Grades computacionais permitem que uma comunidade de usuários dispersos geograficamente (*e.g.*, laboratórios de pesquisa, unidades de empresas, parceiros comerciais) possa compartilhar seus recursos de forma coordenada e segura [22]. Os recursos de uma grade computacional são organizados ao redor de Organizações Virtuais (OVs) [22]. Organização Virtual é a agregação de organizações autônomas e independentes ligadas entre si através de uma rede de comunicação (*e.g.*, Internet). Os usuários podem fazer parte de uma ou mais OVs, tendo acesso aos recursos das mesmas em conformidade com as políticas de acesso adotadas por cada OV. Existem várias soluções estabelecidas que permitem a criação de grades computacionais e muitas destas já estão em

operação [33] [35] [37] [2] [24] [43] [44].

As grades computacionais possuem um grande potencial para alcançar níveis de paralelismo difíceis de serem atingidos por outras plataformas paralelas [39]. O alto grau de paralelismo provido por grades computacionais além de melhorar o desempenho das aplicações existentes, também possibilita a execução de novas aplicações com grandes requisitos de computação e armazenamento. Por outro lado, grades computacionais são mais suscetíveis a falhas [39]. Visto que, uma grade computacional pode potencialmente possuir milhares de máquinas, recursos, serviços e aplicações que interagem uns com os outros e que esses elementos são extremamente heterogêneos, as chances de falhas são grandes. Falhas não isoladas de cada elemento podem ocorrer, tais como as provenientes da interação entre eles (*e.g.*, versões incompatíveis do software utilizado para execução de alguma tarefa na grade). Além disso, partições na rede (reais/virtuais) e/ou suspensão da execução em máquinas remotas tornam o serviço indisponível caracterizando falha.

O paradigma entre-pares (*Peer-to-Peer*) tem se tornado uma técnica bastante utilizada no compartilhamento de recursos. Aplicações tais como compartilhamento de dados, armazenamento e serviços já são bastante difundidas [8]. As redes entre-pares proporcionam um alto grau de tolerância a falhas e escalabilidade em relação ao modelo cliente-servidor. Neste contexto, existem também as grades computacionais que se baseiam em redes entre-pares [8], em que cada *site* (*e.g.*, laboratório de pesquisa) é um par que pode compartilhar/receber serviços de outros pares. As grades entre-pares trazem algumas vantagens comparadas com as demais, como serem mais tolerantes a falhas, já que não possuem pontos únicos de falha e a escalabilidade, já que os serviços podem estar dispostos em pares diferentes. Este tipo de grade foi proposta como uma alternativa mais simples para implantação de grades computacionais de grande porte [14]. Grades entre-pares se caracterizam pela inexistência de negociação para a entrada de novos pares no sistema. No entanto, o desempenho de uma grade entre-pares pode ser degradado por pares *free riders* [30]. *Free riders* são pares que consomem e nunca doam recursos. Para estimular o compartilhamento é preciso prover algum mecanismo que incentive a doação de recursos para a grade [3].

Grades computacionais entre-pares ainda apresentam outras desvantagens, tais como o problema da descoberta dos pares dificultado devido ao ingresso e saída de pares a qualquer momento e também a natureza *best-effort* de sistemas entre-pares. Embora as características de grades entre-pares facilitem a implantação e aumentem a tolerância a falhas, elas ainda têm um grau ainda maior de heterogeneidade (*e.g.*, cada par possui um infra-estrutura diferente) e a sua administração em cada par é geralmente feita de forma autônoma, podendo assim alguns pares com uma má administração comprometer o desempenho da grade computacional.

Apesar da inserção de novos paradigmas que aumentam a tolerância a falhas das grades computacionais, estas ainda requerem uma equipe experiente para o suporte das TICs necessárias para a implantação da grade computacional. Além disso, o fato de existirem diferentes OVs em uma grade computacional implica em diferentes políticas de gestão, cada uma com suas prioridades e requisitos, além de diferentes quantidades e variedades de recursos disponibilizados na grade. Dessa forma, a quantidade e a qualidade técnica dos profissionais de suporte necessários têm uma relação direta com a quantidade e a qualidade dos recursos disponibilizados por uma Organização Virtual.

Existem vários sistemas de monitoramento e gerência para grades computacionais [47] [42] [6] [2] [50] [12] [35], no entanto eles foram criados para grades específicas ou para resolver apenas um certo problema não permitindo, ou tendo uma grande complexidade, para implementação de novas funcionalidades (vide Capítulo 4).

No intuito de reduzir a dependência de administradores e criar um sistema de monitoração e gerência automática de fácil atualização e implantação, este trabalho propõe um sistema autônomo de gerência para grades computacionais, denominado AutoMan. O AutoMan opera sobre as entidades da grade e otimiza a utilização de seus recursos disponíveis (*e.g.*, ciclos de CPU, memória, espaço em disco) tendo como maior contribuição a simplificação do uso e manutenção da grade computacional. AutoMan consiste de um conjunto de agentes que monitoram as entidades da grade. Esses monitores fornecem para outros agentes, denominados *Aggregators*, os valores coletados. A partir desses valores é possível avaliar o estado atual do sistema como um todo. Em situações anormais, um serviço chamado *Leukocyte* aplica as medidas necessárias, baseando-se nos valores das métricas capturadas, a fim de tentar corrigir os problemas e restabelecer o funcionamento normal da grade computacional.

Para realizar o estudo de caso do sistema desenvolvido foi escolhida a solução OurGrid [14] como a grade computacional hospedeira do AutoMan. O OurGrid é um *middleware* que dá suporte à criação e operação de grades computacionais cooperativas, abertas e de fácil implantação, na qual *sites* doam seus recursos computacionais ociosos em troca de acesso, quando necessário, a recursos computacionais ociosos de outros *sites* [14]. Apesar do OurGrid ter sido desenvolvido para ser um sistema de uso e administração simples, até então o sistema não fornecia nenhum mecanismo para facilitar sua gerência, exigindo pessoal especializado para manter o sistema funcionando corretamente.

1.1 Gerência de Grades Computacionais

Esta Seção apresenta as principais características e requisitos que um sistema de monitoramento para grades computacionais deve possuir. Também será apresentado o OurGrid e seu processo de gerência

manual.

1.1.1 Monitoramento de Grades Computacionais

Monitoramento pode ser definido como o processo de coleta dinâmica, interpretação e apresentação de informações relativas a recursos ou aplicações [32]. A monitoração é necessária para diversos fins, tais como a depuração, análise, visualização e estatística do programa. Ela também pode ser utilizada para atividades de gerência em geral como, gerenciamento de desempenho, de configuração, de segurança e de falhas.

O monitoramento de grades computacionais traz várias vantagens, tais como, detecção de falhas mais rapidamente, melhor distribuição dos recursos e escalonamento das tarefas. Porém, monitorar grades computacionais é um grande desafio devido às características peculiares como heterogeneidade e escalabilidade que estas possuem. Sistemas de monitoramento de grades computacionais têm que lidar com vários padrões de divulgação de dados (*e.g.*, modelo *pull* e *push*), e estes dados geralmente estão em grandes quantidades tendo sua origem tanto da coleta em tempo-real quanto de informações armazenadas anteriormente. Outro desafio é a constante evolução dos *middlewares* de grades computacionais e a falta de um consenso sobre um padrão para representação dos dados monitorados e protocolos de comunicação para a interoperabilidade entre sistemas de monitoramento.

O monitoramento de sistemas distribuídos, tais como grades computacionais, normalmente inclui quatro estágios [36]:

1. *Geração dos eventos*: são sensores implantados nas entidades para a captura dos eventos (*e.g.*, status do sistema, carga da CPU).

2. *Processamento*: processamento dos dados gerados pelos sensores. Alguns exemplos são agrupamento/filtragem de acordo com algum critério e armazenamento em um banco de dados.

3. *Distribuição*: refere-se ao envio dos eventos da sua origem para as partes interessadas que podem ser um administrador ou um sistema de gerência automática.

4. *Apresentação da informação*: apresentação para usuário/sistema das informações capturadas e processadas. Geralmente estas informações são apresentadas com certo grau de abstração para facilitar o entendimento sobre os dados monitorados.

Requisitos

A seguir serão descritos alguns requisitos que devem ser considerados ao construir um sistema de monitoramento e de gerência para grades computacionais [51].

Escalabilidade e baixo impacto sobre os recursos monitorados: Uma grade computacional tipicamente possui milhares de serviços, recursos e aplicações que podem ser monitorados. Estas também podem falhar implicando em várias ações corretivas a serem tomadas para restabelecimento do sistema. Sistemas de monitoramento de grades têm que manter a eficiência mesmo com o seu crescimento ou seja, ser escalável. O sistema tem que apresentar baixa latência na transmissão das informações e introduzir o mínimo de *overhead* no sistema monitorado [5]. Se isto não for levado em consideração, o número de informações geradas pelo sistema de monitoramento pode ser tão significativo que acarretaria em atraso nas entregas dos dados monitorados. Este atraso, poderia comprometer o sistema de monitoramento e o usuário final dessas informações geradas (e.g., administrador, sistema de gerência automática). Já que estas informações são geralmente dinâmicas e podem ter um tempo de utilidade, se estas demorarem a chegar ao seu destino, podem não ser mais úteis.

Extensibilidade: Um sistema de monitoramento deve ser extensível a novos recursos da grade que eventualmente poderão vir a ser monitorados. Com a grande dinamicidade e heterogeneidade das grades computacionais, a inserção/remoção de serviços e recursos pode ocorrer. Com isso, é desejável que o sistema de monitoramento possa monitorar esses novos requisitos sem a necessidade de grandes mudanças.

Modelos de entrega dos dados: A informação monitorada de diferentes recursos e serviços pode necessitar de diferentes maneiras de entrega dos dados e periodicidade. A manipulação de dados com informação dos eventos ocorridos na grade como, por exemplo, término de uma tarefa, que são entregues sob demanda, tem de ser tratado diferentemente dos dados de monitoramento sobre carga de CPU, que são entregues continuamente.

Portabilidade: A portabilidade dos sistemas de monitoramento, principalmente de seus sensores, é altamente desejável. Sem esta característica alguns serviços podem não ser monitorados como, por exemplo, serviços que estejam rodando em sistemas operacionais diferentes (*e.g.*, Windows, Linux).

Segurança: Alguns cenários podem requerer que os serviços de monitoramento tenham mecanismos de segurança. As informações geradas por estes sistemas podem não ser de interesse de todos e só apenas aos integrantes de seus domínios administrativos ou de seu nível de prioridade de acesso. Controle de acesso, autenticação, certificados e transporte das informações geradas de maneira segura são algumas das características que podem ser implantadas para o provimento de segurança nos sistemas de monitoramento.

Adaptabilidade: Sistemas de monitoramento freqüentemente são utilizados para verificar se os serviços e recursos estão funcionando corretamente (*e.g.*, detecção de falhas) e/ou determinar a carga da grade para alocação e escalonamento dos recursos. Para a coleta destas informações, o sistema de monitoramento não deve ser comprometido ou comprometer o desempenho da grade devido à variação da carga na grade. O ideal seria que o sistema de monitoramento utilizasse as informações capturadas para calibrar a freqüência com que estas são capturadas. Idealmente, estes sistemas devem ser o mais adaptáveis possível, aplicando técnicas de auto-ajuste [12].

1.1.2 OurGrid

O OurGrid é uma grade computacional entre-pares, aberta e cooperativa [14]. A grade executa aplicações *Bag-of-Tasks* (BoT), que são aplicações paralelas compostas por um conjunto de tarefas cujas execuções independem umas das outras. Os interessados se juntam à comunidade podendo assim executar suas aplicações paralelas. O poder computacional do OurGrid é obtido através dos recursos ociosos dos participantes da comunidade.

Esta grade foi desenvolvida para ser simples, rápida, escalável e segura [9]. Ela usa replicação de tarefas [45] [17], para obter um bom tempo de resposta mesmo na ausência de informações sobre a aplicação e os recursos. A implantação do OurGrid é simples e escalável pois não há necessidade de negociação para entrada de novos pares.

OurGrid explora a idéia de que uma grade computacional é composta de vários *sites* que têm o interesse em trocar favores computacionais (*e.g.*, ciclos de CPU ociosos) entre si. Com isso, é formado uma rede entre-pares de troca de favores que permite que os recursos ociosos de um site sejam fornecidos para outro quando solicitado. Em uma grade em que a entrada de novos pares é aberta, a possibilidade da entrada de pares maliciosos como *free riders* é alta. Para tentar evitar estes pares e manter o equilíbrio do sistema, o OurGrid possui um mecanismo denominado *Network of Favors* (NoF) [3] ou Rede de Favores em português. A NoF contabiliza cada favor (*e.g.*, tempo de CPU) de cada par da grade, acrescentando créditos quando o par doa um favor e diminuindo quando recebe, assim pares que doaram mais recursos (quando estes estavam ociosos) deverão ter prioridade junto à comunidade quando solicitar recursos.

O OurGrid possui quatro entidades principais: MyGrid, DiscoveryService (ou CorePeer), Peer e Worker (ou UserAgent/GridMachine). O primeiro é a interface de acesso (*i.e.*, *broker*) à comunidade do OurGrid que permite a submissão de tarefas à grade. Ele também é o responsável por fazer o escalonamento dessas tarefas para as máquinas da grade (*i.e.*, *grid machines (GuM)*); O Peer provê a comunicação entre os *sites* (*e.g.*, requisitar *grid machines* para executar uma tarefa). O Worker

é o serviço responsável pela execução das tarefas enviadas pelos usuários, e o DiscoveryService é uma entidade auxiliar que implementa um serviço de *rendezvous* possibilitando a formação da grade computacional; ele monitora Peers ativos e distribui uma lista com informações dos mesmos. A grade pode ter vários *sites*, cada um gerenciado por um Peer. O estado atual da comunidade OurGrid pode ser visto em: *http://status.ourgrid.org/*. A Figura 1.1 apresenta um exemplo de comunidade OurGrid.

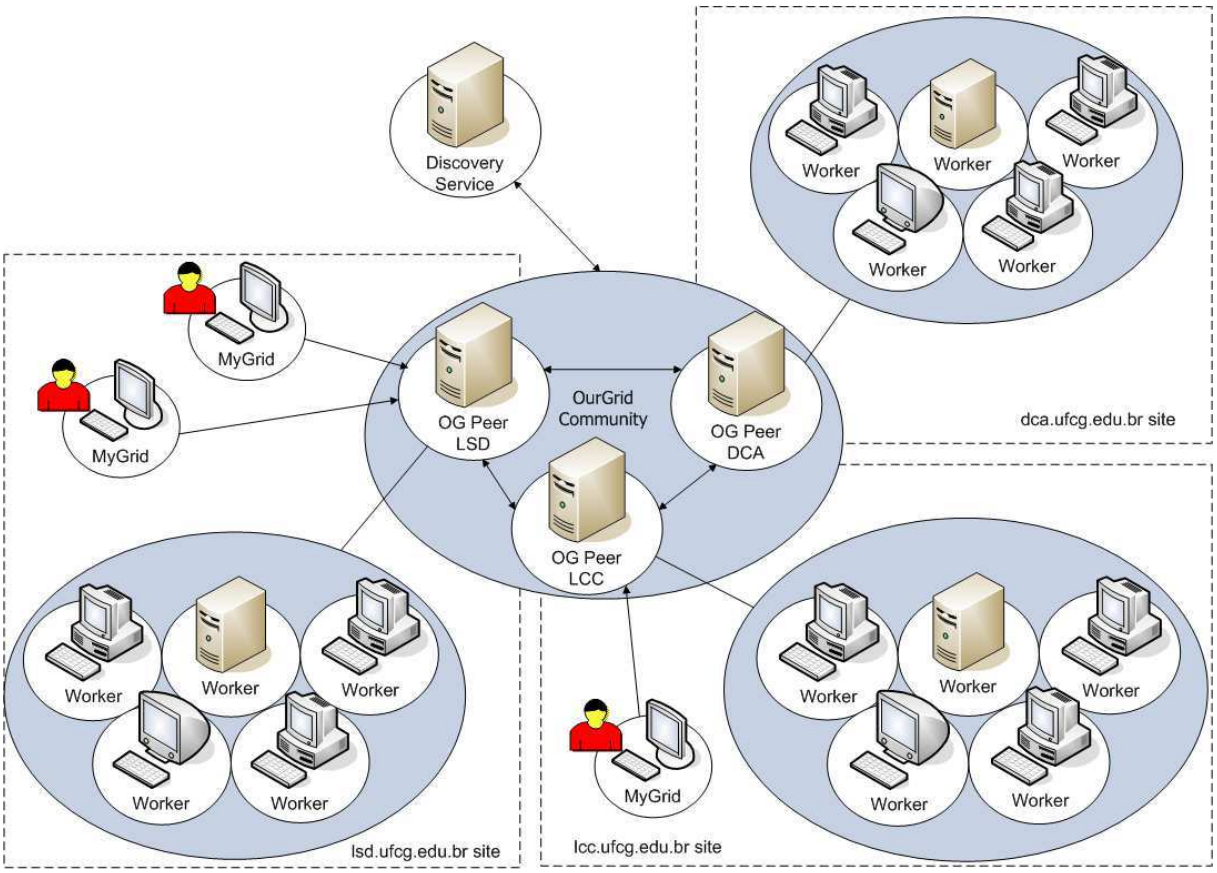

Figura 1.1: Comunidade OurGrid

Versões do OurGrid

Neste trabalho foram utilizados duas versões do OurGrid, o OurGrid 3.3 e o OurGrid 4.0.

A arquitetura e interação entre os componentes das duas versões permaneceram as mesmas, porém o OurGrid 4.0 teve todo o seu código refatorado e o seu paradigma de comunicação mudado. O paradigma de comunicação do OurGrid 3.3 era todo baseado em RMI [16]. No OurGrid 4.0, foi adotado o paradigma baseado no padrão XMPP [23]. Para isto foi adicionado um modelo de programação para aplicações amplamente distribuídas denominada Java Internet Communication (JIC) [34]. Com o JIC, o OurGrid ficou mais flexível, viabilizando a implementação plena dos conceitos de Service

Oriented Architecture (SOA) [48]; o JIC também fornece um suporte natural para comunicação entre os componentes do OurGrid.

Gerência Manual do OurGrid

Gerenciar *sites* OurGrid requer que o administrador verifique o estado dos serviços Worker e Peer, e os restabeleça caso seja observada alguma falha. Caso o administrador também seja responsável pela comunidade, cabe-lhe a gerência do serviço Discovery Service (ou CorePeer). Por exemplo, no Laboratório de Sistemas Distribuídos (LSD), o administrador tem de gerenciar vários *sites*. Sem uma ferramenta de gerência automática, a tarefa de gerenciamento demanda muito tempo, e serviços podem ficar indisponíveis por muito tempo, carecendo de intervenção do administrador.

1.2 Objetivos

O principal objetivo deste trabalho consiste em desenvolver uma arquitetura e implementar um sistema de gerência e monitoração para grades computacionais entre-pares. A arquitetura foi desenvolvida de maneira a prover segurança nas trocas de informações entre os agentes e as entidades da grade. Também foi levado em consideração o acréscimo (*i.e.*, *overhead*) em processamento e tráfego na rede. A arquitetura foi projetada de modo que para a sua implementação em qualquer outra grade computacional entre-pares sejam necessárias poucas mudanças.

A arquitetura do AutoMan atende aos seguintes requisitos:

- Segurança no sistema de gerência e monitoração;
- Introdução de *overhead* minimo à grade hospedeira;
- Monitoramento dos serviços e informações;
- Escalabilidade;
- Adaptabilidade;
- Restabelecimento os serviços monitorados quando necessário.

Para avaliar o trabalho, foi realizado um estudo comparativo de desempenho e disponibilidade do OurGrid com e sem o AutoMan.

1.3 Relevância do Trabalho

As principais contribuições desse trabalho são a arquitetura e a implementação provida pelo AutoMan aumentando a disponibilidade da grade computacional utilizando-se de menos recursos humanos (*i.e.*, administradores da grade). Desse modo, um considerável aumento da qualidade do serviço prestado pela grade é obtido com redução do custo de manutenção e gerência.

Outra contribuição são as lições aprendidas durante o desenvolvimento do AutoMan e a comparação entre as diferenças do gerenciamento automático e manual em um sistema de grade computacional.

1.4 Estrutura do Trabalho

A Dissertação está organizada da seguinte forma.

O Capítulo 2 apresenta o AutoMan, discutindo sua arquitetura e detalhes de implementação. Também é discutida algumas lições aprendidas durante o seu desenvolvimento e avaliação.

O Capítulo 3 apresenta os resultados obtidos na comparação do OurGrid com e sem o AutoMan.

O Capítulo 4 discute trabalhos relacionados a grades computacionais e serviços de gerência automática.

O Capítulo 5 conclui o trabalho indicando direções para investigações futuras.

Capítulo 2

AutoMan

Este Capítulo apresenta a solução desenvolvida para a gerência e monitoração em grades computacionais entre-pares tendo como estudo de caso a grade OurGrid. Descreve-se a arquitetura, os componentes e detalhes técnicos das duas versões implementadas do AutoMan e, por fim, apresentam-se as lições resultantes do desenvolvimento do AutoMan.

O AutoMan é um sistema de monitoramento e gerenciamento automático para grades computacionais entre-pares. Os seus objetivos principais são: a diminuição da complexidade de administração da grade computacional; o armazenamento dos históricos da grade computacional; o aumento da disponibilidade; e o aumento dos ganhos financeiros (*profit*) para instituições. Desenvolvido inicialmente para o OurGrid, o sistema faz a monitoração de uma série de eventos e coleta de dados que ocorrem nesta grade, tais como: submissão de *jobs*, saída de erros das tarefas, entrada de novas máquinas com algum serviço OurGrid, falhas das entidades do OurGrid, como também das máquinas onde seus serviços executam; carga de CPU, utilização do disco rígido e da memória. Esses dados são armazenados para a consulta futura e também para construção do *workload* do OurGrid. A gerência automática é feita utilizando os dados capturados na monitoração. Com base nos dados monitorados, o AutoMan toma algumas ações de gerência automática como, por exemplo, recuperação das entidades falhas do OurGrid e a limpeza do espaço em disco. Embora o AutoMan tenha sido desenvolvido para ser o sistema de gerência e monitoração do OurGrid, pode ser aplicado em outros sistemas distribuídos que necessitem de um sistema de monitoração e gerência automática.

2.1 Evolução do AutoMan

Duas versões do AutoMan foram implementadas, o AutoMan V1 para o OurGrid 3.3, e o AutoMan V2 para o OurGrid 4.

O AutoMan V1 foi desenvolvido para a versão 3.3 do *midlleware* OurGrid. Essa versão conseguiu melhorar significativamente a disponibilidade da grade computacional OurGrid (ver Capítulo 3), porém sua implantação e configuração mostraram-se custosas e com grande probabilidade de inserir erros em seus arquivos de configuração. Durante o ciclo de vida do AutoMan V1 (i.e., definição, implementação, testes, implantação) foram feitas várias melhorias na legibilidade e nas otimizações de seu código utilizando-se alguns padrões de projeto. Além disso, também foram feitas correções de *bugs*. Foram também observadas possíveis mudanças que melhorariam o AutoMan aumentando a sua eficácia e tolerância a falhas. Porém estas não foram implementadas na primeira versão, já que uma nova versão do OurGrid (i.e., Versão 4) estava em desenvolvimento e a substituiria em pouco tempo. Assim, foi preferível implementar essas melhorias na nova versão do AutoMan para o OurGrid que viria a ser lançado em breve.

Com o AutoMan V2 desenvolvido para a Versão 4 do *midlleware* OurGrid, conseguiu-se diminuir bastante o grau de complexidade da implantação e configuração do sistema como também alcançou-se uma melhoria considerável na tolerância a falhas do sistema AutoMan. Essas melhorias foram alcançadas devido a um novo arranjo da arquitetura, ao acréscimo de novas funcionalidades aos componentes do AutoMan e ao uso mais intensivo de alguns padrões de projeto. Como na versão anterior, esta versão também conseguiu melhorar significativamente a disponibilidade do serviço hospedeiro (ver Capítulo 3).

Embora tenham sido desenvolvidas duas versões do AutoMan, os componentes básicos continuam sendo os mesmos. As principais mudanças foram: o arranjo da sua arquitetura melhorando a eficiência e tolerância a falhas do sistema; a inserção de padrões de projeto melhorando o desempenho e facilitando o entendimento do código para futuras atualizações; e a interação entre alguns componentes do AutoMan devido a mudanças na sua arquitetura.

A arquitetura e implementação das duas versões serão descritas nas Seções 2.2 e 2.3. A Seção 2.2 descreve todos os componentes, implementação e arquitetura, e a Seção 2.3 apresenta as alterações e suas motivações, já que o funcionamento e implementação das duas versões do AutoMan são semelhantes.

2.2 AutoMan V1

Esta Seção descreve a arquitetura e implementação do AutoMan V1 que foi desenvolvido para o OurGrid Versão 3.3. A Subseção 2.2.1 apresenta a arquitetura do AutoMan com uma explicação de seus componentes e a interação entre eles. A Subseção 2.2.2 descreve a implementação do AutoMan, a forma como as métricas são obtidas, como o sistema de armazenamento e publicação das métricas funciona e como são feitos os tratamentos de falhas das entidades do OurGrid.

2.2.1 Arquitetura

Para incorporar o serviço de gerência automática ao OurGrid, desenvolveu-se uma infra-estrutura de monitoração. A fim de monitorar o OurGrid, foi adicionado às entidades desse sistema o código necessário para monitoração. O sistema de monitoração captura informações sobre desempenho (*e.g.*, carga da CPU, uso da memória), disponibilidade (e.g., espaço em disco, estado das máquinas e dos serviços) e sobre eventos ocorridos (*e.g.*, submissão/termino de um *job*, novo Peer ativo, recebimento de Workers) e baseado nessas informações, constrói-se as métricas utilizadas pelo AutoMan. Dependendo dos valores das métricas, o AutoMan toma ações de tratamento (*i.e.*, *healing actions*) restabelecendo o funcionamento normal da grade. O principal objetivo desse trabalho consiste em propor uma arquitetura que possa ser implementada em outras grades computacionais não exigindo mudanças estruturais na grade hospedeira.

A arquitetura AutoMan acrescenta algumas novas entidades ao OurGrid: **Monitores**, entidades responsáveis pelo monitoramento de máquinas e serviços da grade; o **Aggregator**, usado para armazenar os valores das métricas coletados pelos monitores bem como prover mecanismos de consulta para todas as métricas armazenadas na sua base de dados; e a entidade que provê um nível de autotratamento (*self-healing*) para o OurGrid, denominada **Leukocyte**, a qual recupera a grade dos principais problemas e restaura seu funcionamento normal. Essas novas entidades e seu relacionamento com a arquitetura OurGrid são mostrados na Figura 2.1.

O monitor roda dentro das entidades do OurGrid capturando o seu estado atual, todos os eventos relacionados, e o estado da máquina. Cada Worker (também denominado UserAgent ou GUM), Peer e DiscoveryService (CorePeer) possui um monitor correspondente, que coleta e publica informações sobre os serviços e sobre as máquinas onde estes executam. A única entidade OurGrid que não é monitorada é o MyGrid (ou OurGrid *Broker*), porque este funciona no lado do cliente, não impactando assim no funcionamento normal da grade. As métricas que indicam mudanças significativas no estado de máquinas e/ou serviços são coletadas e enviadas para o Aggregator, o qual armazena as métricas

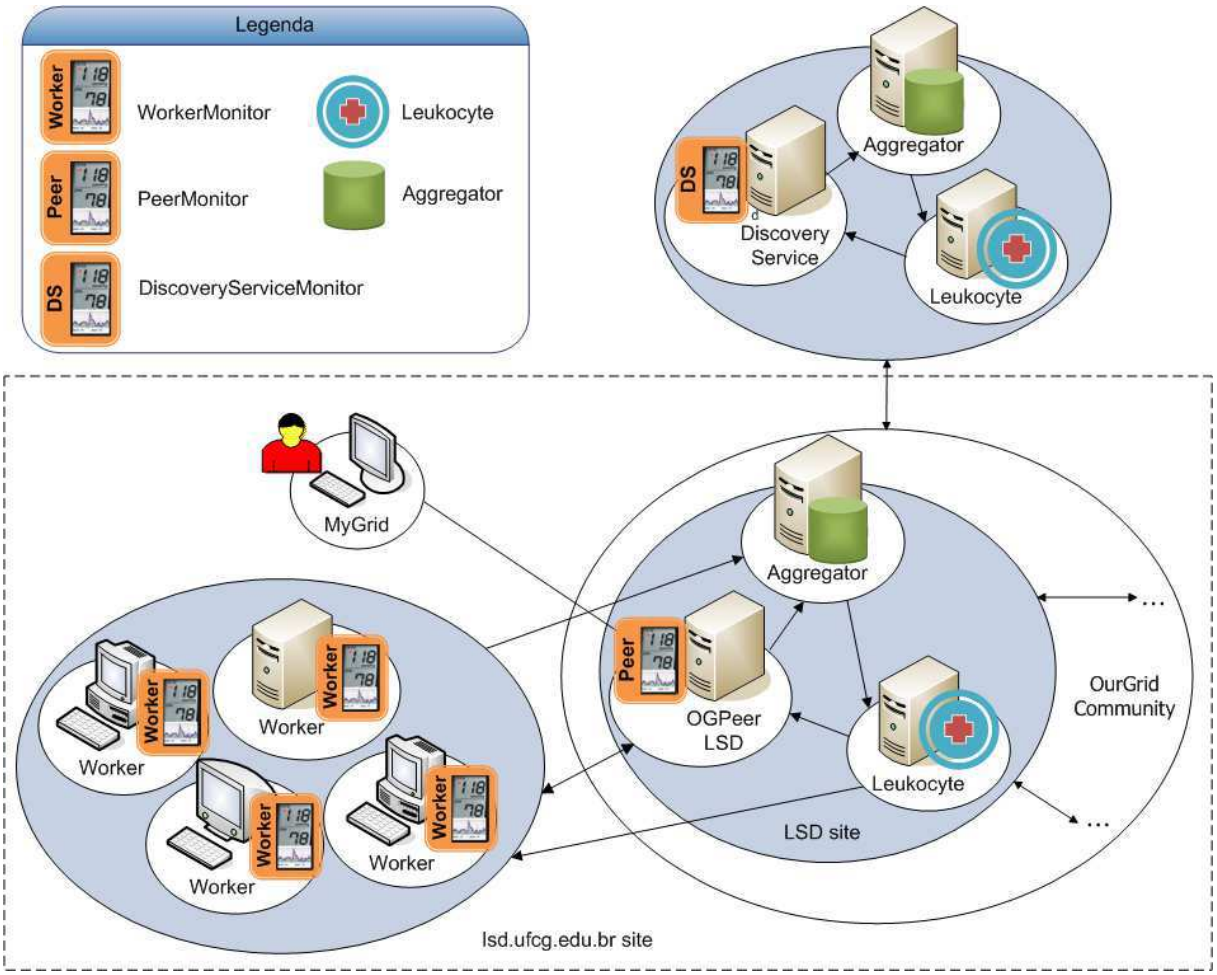

Figura 2.1: Entidades AutoMan

em sua base de dados.

Além de armazenar os dados enviados pelos monitores, o Aggregator também possui um serviço *publish-subscribe* [19] que permite que outros sistemas possam se inscrever como interessados em um conjunto de métricas, sendo automaticamente notificados de acordo com contextos estabelecidos em função das métricas e seus valores. O Aggregator também provê um procedimento padrão para consulta, permitindo consultas *ad hoc* à sua base de dados. Os Aggregators, por sua vez, podem ser inscritos como interessados em outros Aggregators, a fim de prover, eles mesmos, informações mais completas, evitando um tráfego excessivo de mensagens com informações de monitoramento.

O Leukocyte se inscreve como interessado nas métricas necessárias para prover a gerência dos serviços que são objeto de tratamento pelo AutoMan. Baseado nessas métricas, e dependendo dos valores destas, um de seus componentes internos, o *Detector*, executa alguns passos necessários para o diagnóstico correto do problema. Finalmente, um outro sub-componente, o *Effector*, atua restaurando o serviço ou trazendo o ambiente às condições adequadas de operação.

A Figura 2.2 ilustra a interação entre os elementos AutoMan durante o tratamento do evento *Peer Down*. Como se pode observar, o serviço Leukocyte se inscreve no Aggregator para a métrica PeerDown (2). Quando o Peer fica inativo (3) o **DiscoveryServiceMonitor** percebe o evento informando o ocorrido (3), e uma notificação é enviada para o **Aggregator** através do seu servidor (5,6). O **Aggregator** ao receber a métrica do **DiscoveryServiceMonitor** armazena-a na sua base de dados. Em seguida todos os serviços inscritos como interessados nessa métrica (e.g., Leukocyte) são notificados (7). Ao receber a métrica PeerDown, o **Leukocyte** verificando que o **Peer** está inoperante e baseado nas informações recebidas (e.g., nome e endereço do Peer inativo) inicia o processo de recuperação reativando o Peer (8).

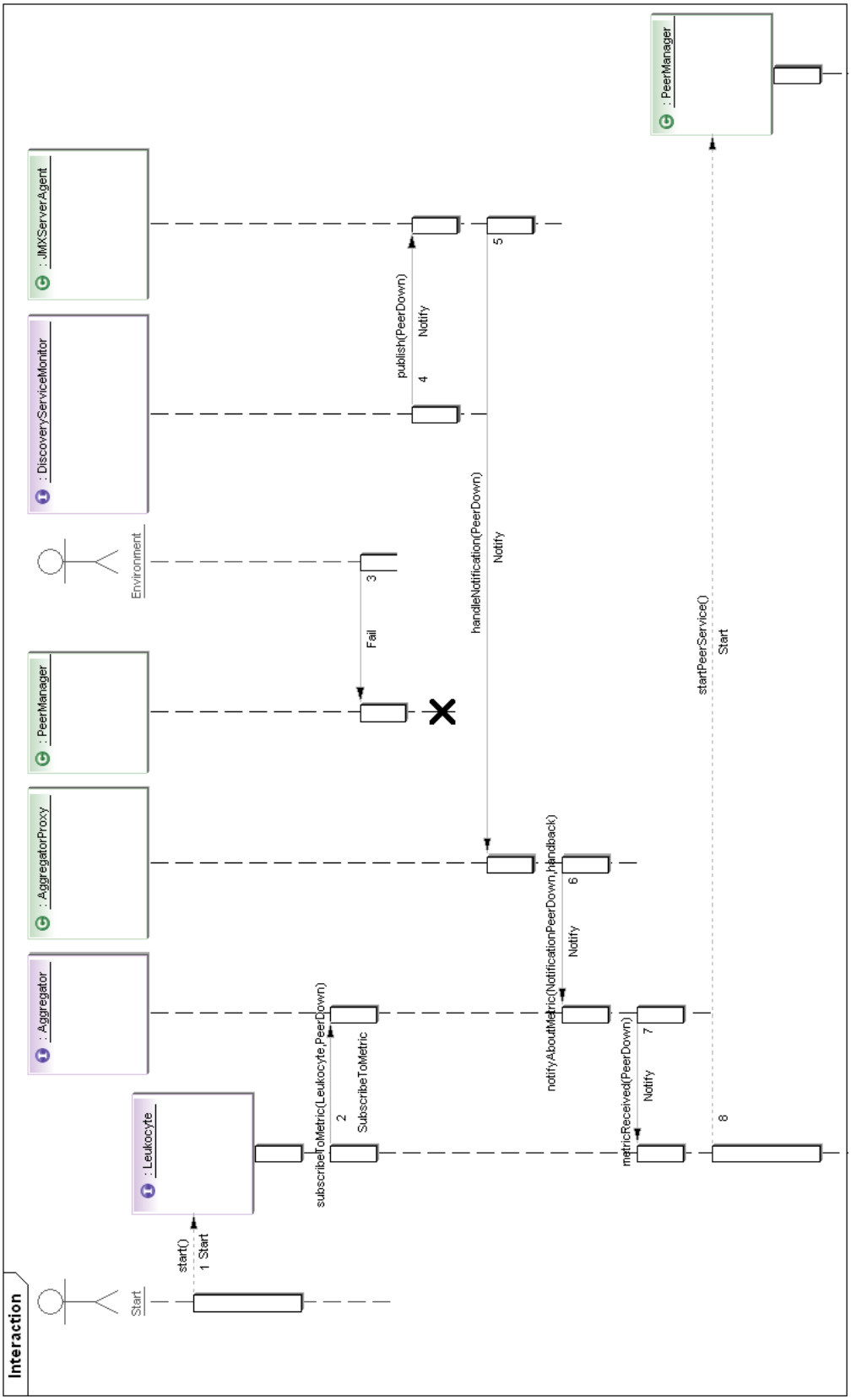

Figura 2.2: Interação entre os componentes AutoMan

Monitor

Os monitores obtêm os valores de métricas atuais acerca dos eventos importantes através dos sensores incorporados às entidades monitoradas. Quando ocorre a detecção de qualquer mudança nos valores de métricas pré-definidos, estes são enviados para o Aggregator através de uma camada utilizando *Java Management Extensions* (JMX) [40]. A tecnologia JMX é utilizada para o desenvolvimento de ferramentas distribuídas e dinâmicas para gerência e monitoração. JMX foi escolhida por conta do seu suporte a segurança e porque ela possui um mecanismo de pré-processamento que permite analisar as métricas antes de decidir enviá-las ou não (*e.g.*, poder-se-ia decidir enviar a mensagem somente quando ocorresse uma mudança significativa na métrica) evitando sobrecarga de comunicação. As métricas monitoradas no OurGrid foram escolhidas baseadas nas ações de monitoração que os administradores fazem para verificar o funcionamento do OurGrid tais como verificar o estado dos serviços do OurGrid, o estado da máquina em que esses serviços estão executando como também dos recursos que estas prover. Também são coletada métricas necessárias para a construção do *workload*, tais como, quando um *job* foi submetido, tempo de execução, dados transferidos.

Para obter as métricas da máquina que não são específicas da aplicação (*e.g.*, carga do processador), os monitores precisam de uma aplicação que armazene e publique informações da máquina. Nesse trabalho adotou-se o sistema de monitoração Ganglia [38]. Ganglia é um sistema de monitoração escalável e distribuído, projetado para sistemas de computação de alto desempenho como *clusters* e grades computacionais. A sua licença é GPL. O Ganglia monitora informações como carga da CPU, memória e disco em intervalos de tempo que podem ser definidos em seu arquivo de configuração. Seu agente de monitoração, que é instalado e configurado em cada máquina que se deseja monitorar, é chamado de *GMonD* (*Ganglia Monitoring Daemon*).

Outros sistemas de monitoração de máquina e rede como o Nagios [25] e o NWS (*Network Weather Service*) [49] foram estudados, porém, ambos possuíam um grau de complexidade maior na sua instalação pois era necessária instalação ou de um sistema de arquivo em rede (i.e., *Network File System*, NFS) ou de uma base de dados compartilhada. Outro fator importante levado em consideração para a escolha do Ganglia foi o formato e a maneira de aquisição dos dados monitorados pelo monitor. O formato que o Ganglia utiliza para enviar os dados é o XML [10] que é uma linguagem de marcação utilizada para descrever dados em arquivos. A fim de recolher as informações coletadas por este agente e enviá-las para a camada JMX, implementou-se o componente chamado **GmondDataCollector**. O GmondDataCollector é um serviço que se conecta ao monitor Gmond da sua máquina para ficar recebendo os dados monitorados. Esses dados são filtrados, considerando-se apenas as informações que são úteis para o AutoMan, e em seguida enviados para o Aggregator.

A Figura 2.3 ilustra a arquitetura básica do monitor. O componente **OurGrid modified** é o OurGrid com códigos de monitoração inseridos com o objetivo de fazer a interceptação em chamadas de códigos quando um evento monitorável ocorre e assim fazer a notificação. O código inserido na grade hospedeira é apenas uma linha contendo a chamada para o método que fará a manipulação e notificação do dado coletado no envento. O componente **JMX Publishing Engine** tem a função de enviar as métricas coletadas para o Aggregator utilizando o JMX. Os dados enviados para o Aggregator estão no formato XML. O AutoMan possui três tipos de monitores, um para cada entidade monitorável do OurGrid: WorkerMonitor, PeerMonitor e o DiscoveryServiceMonitor.

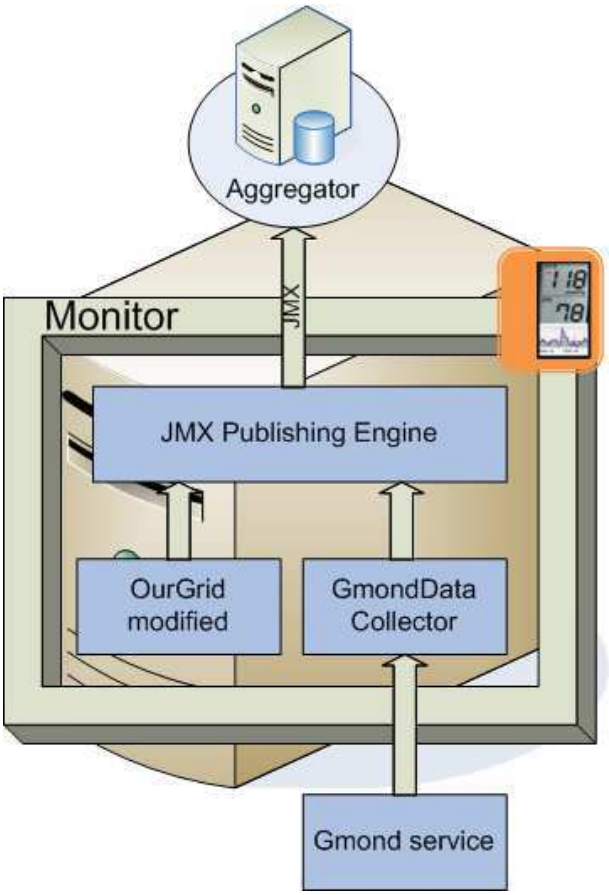

Figura 2.3: Entidade Monitor

Agregador de Dados

O Aggregator armazena as informações providas pelos monitores, permitindo a análise e a consulta das informações armazenadas a qualquer momento. Além disso, os Aggregators podem consultar uns aos outros sobre métricas armazenadas em múltiplas bases de dados.

O Aggregator possui um componente que utiliza JMX para capturar os valores de métricas enviados pelos monitores, e também outro componente que implementa uma interface para *Remote Method Invocation* (RMI) [16] que trata de tornar públicos os serviços do Aggregator.

O primeiro desses componentes é o *AggregatorProxy*. Ele é basicamente um procurador (*proxy*) para redirecionar os valores de métricas para um serviço que implementa uma interface RMI com os serviços do Aggregator. Este serviço é chamado de *AggregatorService*. Além de manipular os valores de métricas recebidos pelo *proxy*, o *AggregatorService* também tem a função de inscrever os interessados nas métricas, e também de fornecer um mecanismo de consulta à base de dados. Internamente, a base de dados é gerenciada pelo componente Metrics DataBase (MD). Como gerenciador da base de dados foi escolhido o *Db4O* [28]. O banco de dados *Db4O* foi escolhido por ser um banco de dados orientado a objetos, embarcado, com uma boa escalabilidade, de fácil uso e aprendizado e por ter licença GPL. Os objetos são armazenados no *DB4O* nativamente eliminando a complexidade extra e a perda de desempenho. O AutoMan armazena nele o objeto *metricsValues*, contendo as informações da métrica (*e.g.*, *timestamp*, origem da métrica, evento ocorrido).

O serviço *publish-subscribe* do Aggregator é provido pelo componente chamado *AggregatorPublisher* que é acessível pelo *AggregatorService*. Diversos serviços, como o Leukocyte ou outros Aggregators, podem se inscrever como interessados nas métricas armazenadas no Aggregator. A Figura 2.4 mostra a arquitetura do Aggregator.

Analisador e Atuador

Quando um organismo fica doente, seus leucócitos começam a lutar contra o atacante. No AutoMan a metáfora é exatamente a mesma: há um mecanismo similar, denominado Leukocyte. Um Leukocyte no AutoMan é capaz de diagnosticar e resolver certos problemas que podem comprometer a integridade do OurGrid. O Leukocyte age como um analisador e mecanismo de recuperação do sistema. O Leukocyte exporta uma interface RMI que é implementada pelo componente *LeukocyteServer* com o intuito de prover estes serviços. O Leukocyte se inscreve nas métricas necessárias para a gerência automática usando o serviço Aggregator Publisher, com a intenção de receber atualização dos valores de métricas requisitados.

Um componente do Leukocyte chamado *Detector* executa o diagnóstico analisando e verificando se esses valores recebidos indicam a existência de algum problema conhecido. Dependendo do problema, um outro componente chamado *Effector*, toma as medidas necessárias para reparar o problema. O *Effector* é provido de um conjunto de procedimentos para o processo de recuperação, que são invocados de acordo com as notificações recebidas do *Detector*. A recuperação é executada através de

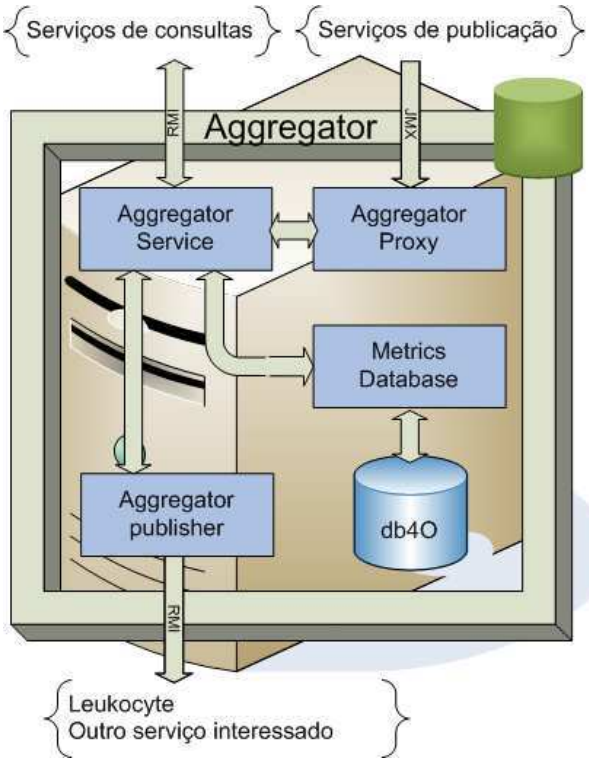

Figura 2.4: Entidade Aggregator

conexões seguras *Shell (i.e., ssh)* e invocando *scripts* correspondentes à falha que se deseja recuperar. Tais *scripts* podem ser usados para tornar o serviço funcional novamente ou para outras ações corretivas como, por exemplo, apagar o diretório temporário. Para executar esses *scripts*, os Leukocytes têm quer ser iniciados pelo usuário com direito administrativo nas máquinas que contêm os serviços OurGrid que utilizam esse Leukocyte para sua ressurreição ou rejuvenescimento do sistema. A Figura 2.5 mostra a arquitetura do *Leukocyte* e seus componentes internos.

A Tabela 2.1 mostra qual entidade OurGrid gerou o evento e as ações tomadas pelo Leukocyte para a recuperação do componente falho.

2.2.2 Implementação

O AutoMan foi implementado utilizando a tecnologia Java [27] pois as características de portabilidade são essenciais no sistema de monitoração desenvolvido. Um outro motivo para a escolha desta tecnologia foi o fato do OurGrid também ser desenvolvido em Java. Para fazer a infra-estrutura de comunicação e publicação dos dados utilizou-se RMI [16] e JMX [40]. O RMI para as chamadas remotas entre os componentes AutoMan ou entre um componente AutoMan e outro componente externo que venha a utilizar alguma informação que o sistema de possa prover (e.g., consulta às métricas

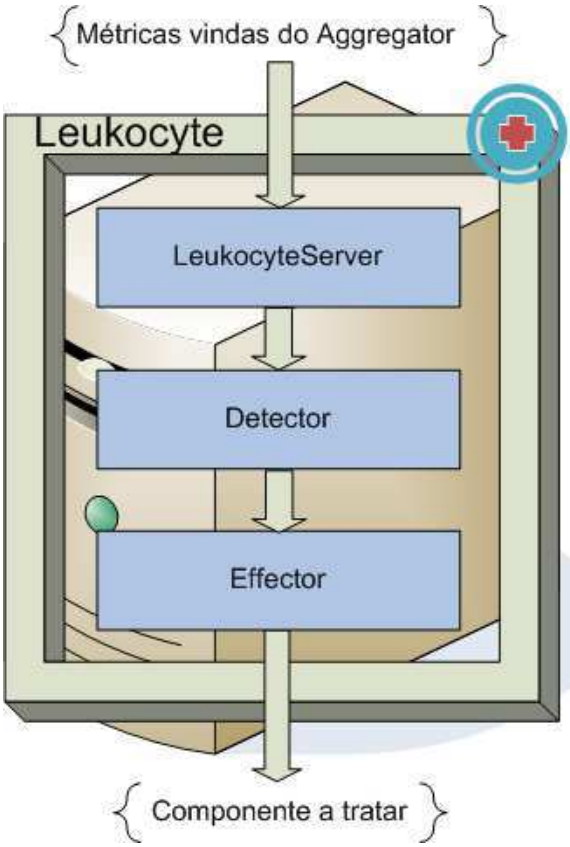

Figura 2.5: Entidade Leukocyte

armazenadas). O JMX foi usado para a publicação das métricas coletadas pelos monitores e envio destas para o Aggregator. Para o armazenamento das métricas o Aggregator usa o banco de dados DB4O como já foi citado anteriormente. A seguir serão apresentados alguns detalhes da implementação e os diagramas UML para as principais classes incluindo as suas funções.

Obtenção das Métricas

Como apresentado anteriormente, foram implementados três tipos de entidade de monitoração para a obtenção das métricas: PeerMonitor, WorkerMonitor e o DiscoveryServiceMonitor. Em seguida, serão apresentadas as métricas escolhidas para serem monitoradas e qual monitor coleta cada métrica. Algumas métricas coletadas correspondem aos estados das entidades do OurGrid e outras correspondem às características dos *jobs* submetidos à grade computacional. As métricas foram escolhidas para satisfazer os requisitos do sistema, porém pode-se acrescentar o monitoramento a outros eventos sem grandes dificuldades.

No que se refere ao Worker, as seguintes informações são coletadas: i) O status de cada tarefa executada na grade computacional; ii) A hora de início e de término (ou sua falha, se esse for o caso)

Serviço OurGrid	Evento	Ação
DiscoveryService	DiscoveryService fica inativo	Leukocyte chama um script que restaura o DiscoveryService.
DiscoveryService	Peer fica inativo	Leukocyte que chama um script que restaura o Peer.
Peer	Worker fica inativo	Leukocyte chama um script que restaura o worker.
Peer	$ storage fica cheio	Leukocyte chama um script que limpa o $storage.
Worker	/tmp fica cheio	Leukocyte chama um script que limpa o /tmp.

Tabela 2.1: Ações tomadas pelo Leukocyte

para cada tarefa; iii) Todas as mensagens de saída da tarefa (incluindo mensagens de erro); iv) As informações sobre arquivos utilizados pela tarefa (nome do arquivo, valor *hash* do MD5, localização, tamanho, tipo e tempo de transferência); v) O tamanho atual de diretórios utilizados pelas tarefas; vi) A linha de comando que foi enviada para a execução da tarefa.

A Tabela 2.2 mostra as métricas geradas pelo WorkerMonitor com uma breve descrição delas. A tabela completa pode ser encontrada no Apêndice A.

Métrica	Descrição da Métrica Gerada
Worker.ReplicaStarted	Início da execução do *job* na replica
Worker.ReplicaFinished	Término da execução do *job* na replica
Worker.TempDirSizeReport	Espaço livre no /tmp da máquina

Tabela 2.2: Métricas e ações do WorkerMonitor

Para o Peer, as seguintes informações são capturadas: i) Worker mudou de estado (*i.e.*, *Contact, Idle, In Use, Donated, Owner*); ii) Informações apresentadas para cada tarefa, incluindo quem solicitou os recursos (*i.e.*, o número de máquinas solicitadas); iii) Contabilidade do uso das máquinas da grade (máquinas, que correspondem a Workers) considerando os recursos doados e recebidos por outros Peers (i.e., NoF); iv) Máquinas OurGrid atribuídas aos usuários através dos *brokers* (*i.e.*, MyGrid).

A Tabela 2.3 mostra as métricas geradas pelo PeerMonitor com uma breve descrição delas. A tabela completa pode ser encontrada no Apêndice A.

Métrica	**Descrição da Métrica Gerada**
Peer.WorkerRequest	Mygrid requisita um Worker
Peer.WorkerReceived	Worker recebido por um Mygrid
Peer.Accounting	Recebe o balanço de favores da NOF de um Peer
Peer.WorkerDelivered	Worker devolvido pelo MyGrid
Peer.WorkerStatusChange	Mudança do estado do Worker (ativo/inativo)
Peer.StorageDirSizeReport	Espaço livre no $storage da máquina

Tabela 2.3: Métricas e ações do PeerMonitor

Para o DiscoveryService as métricas capturadas são: i) O estado (*i.e.*, Up ou Down) para todos os Peers conhecidos; ii) DiscoveryService *HeartBeat* (enviado para o Aggregator periodicamente para monitoração do *status* do DiscoveryService).

A Tabela 2.4 mostra as métricas geradas pelo DiscoveryServiceMonitor com uma breve descrição delas. A tabela completa pode ser encontrada no Apêndice A.

Métrica	**Descrição da Métrica Gerada**
DiscoveryService.PeerUp	Peer ficou ativo
DiscoveryService.PeerDown	Peer ficou inativo
DiscoveryService.DiscoveryServiceUp	DiscoveryService ficou ativo
DiscoveryService.DiscoveryServiceDown	DiscoveryService ficou inativo

Tabela 2.4: Métricas e ações do DiscoveryServiceMonitor

Como apresentado anteriormente, cada entidade Monitor organiza suas métricas e as envia usando notificações JMX. Isto foi implementado introduzindo em cada entidade monitorada do OurGrid chamadas para a classe *Monitor* passando as informações necessárias para a geração da métrica específica, dependendo do evento causador. Quando ativado, o monitor inicia um servidor JMX que publica os dados monitorados. Os serviços interessados, nesse caso o Aggregator, são notificados quando os dados são modificados.

Para introduzir monitoramento em novos serviços ou em novos eventos, primeiro cria uma assinatura na fachada do monitor que vai ser chamada pelo código da grade hospedeira. Em seguida

introduzir as regras de manipulação (semelhantes às apresentas nas Tabelas 2.2 2.3 2.4) dos dados monitorados na classe *MetricsRecordHandler* do monitor correspondente. Por fim, é necessário introduzir a chamada ao Monitor no código hospedeiro utilizando a fachada.

As métricas são enviadas no formato XML e são construídas pela classe *MetricsRecordHandler* e inseridas na classe *MetricRecord* que por sua vez será publicada pelo servidor JMX. A Figura 2.6 ilustra o XML gerado pelo *MetricsRecordHandler* do *PeerMonitor* durante a requisição de Workers para execução de um *job*: a linha 2 mostra o *id* do *job*; a linha 3 mostra se a requisição feita foi para o Peer local ou não; a linha três mostra o número de máquinas requisitadas; a linha 5 ilustra os requisitos do *job*; as linhas 7 e 8 ilustram respectivamente o nome e o endereço do Peer que está fazendo a requisição; e a linha 10 informa se a requisição foi local ou feita por um Peer de outra comunidade.

```
1.  <WorkerRequest>
2.      <requestID> 1 </requestID>
3.      <isLocal> true </isLocal>
4.      <numberOfMachines> 15 </numberOfMachines>
5.      <requestRequirements> os == linux </requestRequirements>
6.      <clientPeerList>
7.          <clientName> PeerLsd </clientName>
8.          <clientLocation> cherne.lsd.ufcg.edu.br </clientLocation>
9.      </clientPeerList>
10.     <requestSource> FROM_LOCAL_BROKER </requestSource>
11. </WorkerRequest>
```

Figura 2.6: XML gerado pelo MetricsRecordHandler

A Figura 2.7 apresenta o diagrama para as classes *PeerMonitor* e *PeerMetricsRecordHandler* ilustrando suas funções. As funções dependem das entidades e métricas que estão sendo monitoradas. Nesse caso é apresentado o monitor da entidade Peer.

Além das métricas específicas dos serviços OurGrid, existem ainda métricas relativas às próprias máquinas como, por exemplo, a memória principal, espaço em disco, e uso de CPU. Estas métricas são recolhidas através da classe *GMondDataCollector* invocada pelo serviço *GMondMonitor*. Este serviço é iniciado juntamente com a máquina que possuir uma entidade OurGrid, após a inicialização do serviço *Ganglia Monitor Daemon* (*i.e.*, GMonD). Quando o Ganglia captura alguma mudança nos valores de uma métrica, o *GMondMonitor*, que implementa a interface *GMondMetricValueChan-*

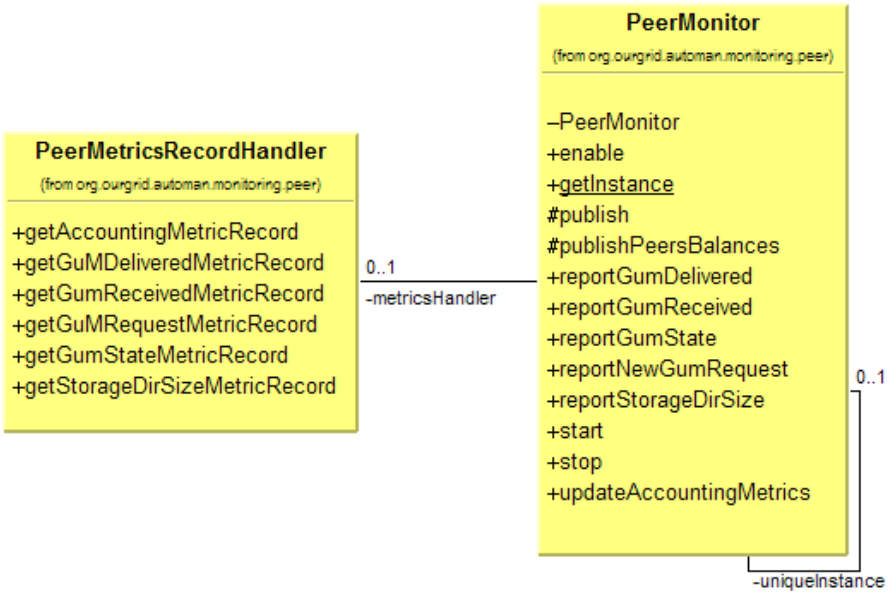

Figura 2.7: Principais classes do monitor do Peer

gedListener, é notificado do evento ocorrido e envia a métrica para a camada JMX. Por sua vez, a camada JMX envia a métrica no formato XML para o Aggregator.

Na Figura 2.8 é mostrado um exemplo dos dados XML enviados pelo GMonD. O formato XML basicamente é o seguinte: uma *tag* com o nome da máquina (HOST NAME) (Figura 2.8, Linha 3) mostrando de qual máquina foram coletados os dados, a hora em que foram coletados (REPORTED), dentre outras informações. Internamente a essa *tag* existem as *tags* com as informações monitoradas que contém o nome da métrica (METRIC NAME), valor da métrica (VAL), o tipo do valor exibido (TYPE), a unidade da métrica (UNIT) e outras informações. Como exemplo, na linha 4 da Figura 2.8 está a carga de 95.4% de uso da CPU no momento da coleta.

As principais classes responsáveis pelo tratamento dos dados relacionados às métricas de máquinas são ilustradas na Figura 2.9.

Armazenamento e Publicação dos Dados Monitorados

Como mencionado anteriormente, o Aggregator (ver Figura 2.11) é o componente responsável pelo armazenamento e publicação dos dados monitorados. O Aggregator segue uma abordagem básica de receber e armazenar as métricas produzidas pelas entidades OurGrid e pelos monitores GMonD. Todos os monitores possuem em seu arquivo de configuração o endereço lógico do Aggregator para onde enviar as métricas. A disseminação das mensagens monitoradas é realizada através da camada JMX para a classe *AggregatorProxy*, que implementa a interface *NotificationListener* do Aggregator

```
1. <?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
2. <GANGLIA_XML VERSION="2.5.7" SOURCE="gmond">
3. <HOST NAME="cherne.lsd.ufcg.edu.br" IP=" 150.165.85.97" REPORTED="1199828946"
   TN="0" TMAX="20" DMAX="0" LOCATION="LSD" GMOND_STARTED="1198847031">
4. <METRIC NAME="cpu_user" VAL="95.4" TYPE="float" UNITS="%" TN="41" TMAX="90"
   DMAX="0" SLOPE="both" SOURCE="gmond"/>
5. <METRIC NAME="machine_type" VAL="x86" TYPE="string" UNITS="" TN="617" TMAX="1200"
   DMAX="0" SLOPE="zero" SOURCE="gmond"/>
6. <METRIC NAME="disk_total" VAL="38.976" TYPE="double" UNITS="GB" TN="639"
   TMAX="1200" DMAX="0" SLOPE="both" SOURCE="gmond"/>
7. <METRIC NAME="mem_total" VAL="1028220" TYPE="uint32" UNITS="KB" TN="597"
   TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond"/>
8. <METRIC NAME="mem_free" VAL="728540" TYPE="uint32" UNITS="KB" TN="120" TMAX="180"
   DMAX="0" SLOPE="both" SOURCE="gmond"/>
9. <METRIC NAME="cpu_speed" VAL="1793" TYPE="uint32" UNITS="MHz" TN="251"
   TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond"/>
```

Figura 2.8: XML com as métricas vindas do GMonD

correspondente. Todo objeto que deseje publicar métricas terá que implementar a interface *MetricPublisher* (vide Figura 2.10).

O serviço *publish-subscribe* que o Aggregator oferece é exportado pela camada RMI implantada pela classe *AggregatorService*. O processo de manipulação dos inscritos (*subscribers*) (e.g., enviar notificações) é atribuído a classe *AggregatorPublisher*. Esta classe possui uma tabela *hash* com todos os inscritos e suas métricas de interesse. Todo objeto que deseje se inscrever como interessado em alguma métrica, terá que implementar a interface *MetricInterested* (vide Figura 2.10).

O agregador é também responsável pelo monitoramento do serviço DiscoveryService, isso porque nenhuma outra entidade OurGrid gera eventos relacionados ao DiscoveryService. Esse monitoramento é realizado pela classe *CorePeerHeartBeatMonitor* que manipula as métricas de sinalização enviadas periodicamente e constantemente (*heart beat*) pelo DiscoveryService. Se o monitor passar certo período de tempo sem receber o *heart beat*, um evento notificando que o DiscoveryService esta inativo é gerado, que por sua vez é enviado para o *AggregatorService*.

Para armazenar métricas no banco de dados, o *AggregatorService* invoca a classe *MetricsDatabaseFacade*, que delega a inserção à classe *DataBaseProcessor* (ver Figura 2.12).

As consultas são realizadas através da classe *MetricsDatabaseFacade* usando a classe *MetricQuery* que contém as informações sobre métricas (*i.e.*, serviço Ourgrid, evento ocorrido, intervalo de data, etc.) que se deseja fazer a procura. Como já citado, o banco de dados utilizado é o DB4O, o

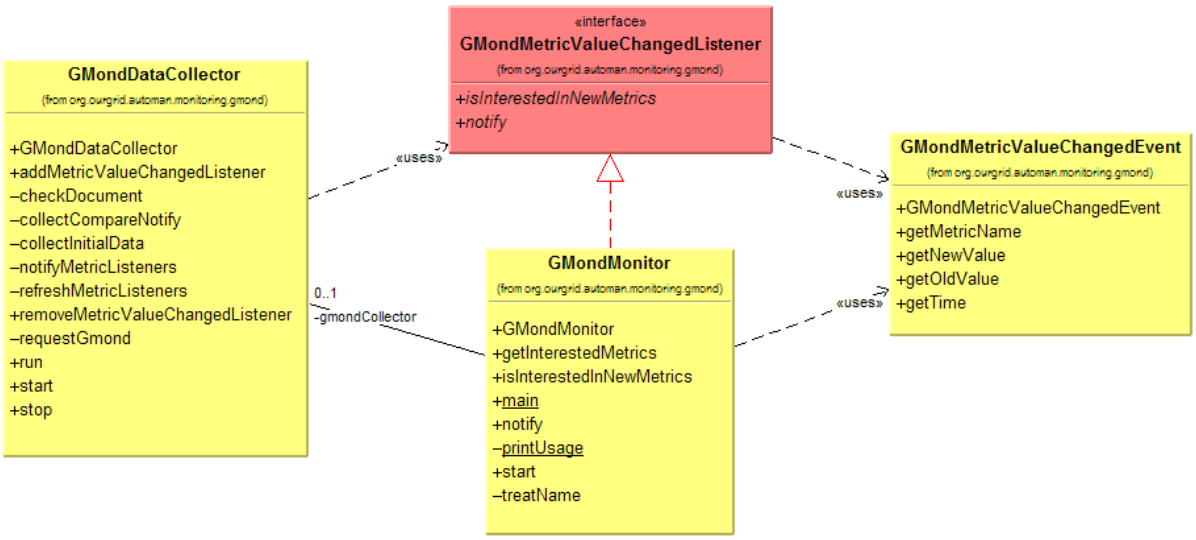

Figura 2.9: Principais classes para o tratamento dos dados monitorados pelo Ganglia

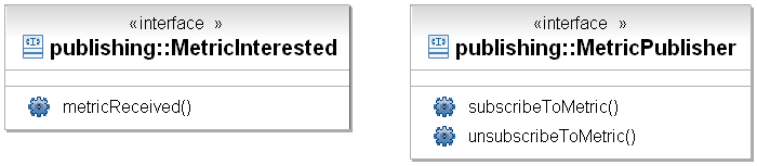

Figura 2.10: Interfaces MetricInterested e MetricPublisher

qual é orientado a objeto. Assim, o objeto armazenado é o *MetricRecord* (Ver Figura 2.13), criado pelas classes *MetricsHandles*. A fim de proporcionar uma implementação específica para manipular o DB4O (*i.e.*, acesso para inserção, consulta, exclusão das métricas), implementou-se a classe *SingletonDataBase*.

Recuperação das Entidades OurGrid

Durante a etapa de levantamento de requisitos para o desenvolvimento do AutoMan, observou-se a necessidade de mecanismos para iniciar automaticamente os serviços do OurGrid (*i.e.*, Peer, DiscoveryService e Workers) que poderiam ficar inativos. Atualmente, a gerência de várias comunidades OurGrid requer a intervenção humana. Os administradores das comunidades têm que verificar periodicamente o estado do sistema e, caso algum dos serviços estija inativo, o administrador terá que reiniciá-lo manualmente executando alguns *scripts* de inicialização.

A Limpeza de diretórios/arquivos temporários utilizados pelos Workers (*i.e.*,*$storage, /tmp*) deve ser feita automaticamente quando o espaço de armazenamento se encontra muito exíguo, ou quando uma réplica termina a execução da sua tarefa. Isto ocorre porque, uma vez que o local de armazena-

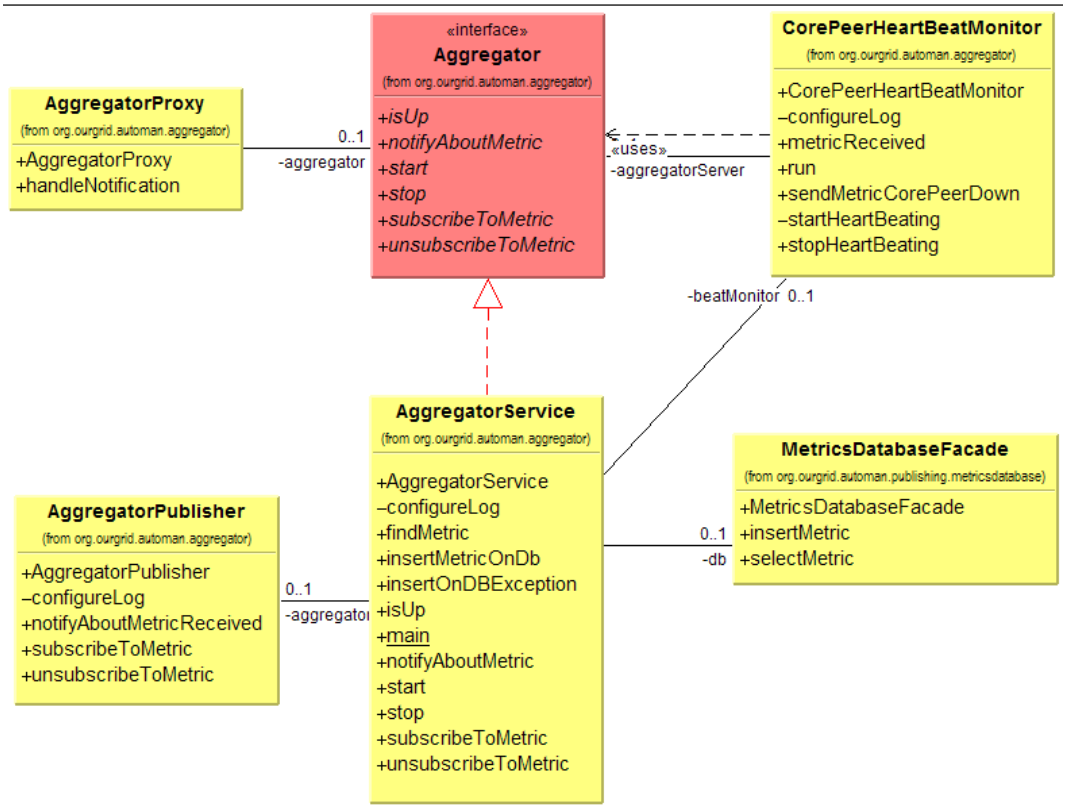

Figura 2.11: Principais classes do componente Agregador

mento temporário enche, as tarefas em execução falharão.

Outro problema que pode ocorrer é a falha periódica de determinados Workers, sempre durante a execução de tarefas de um determinado usuário da grade computacional. Atualmente quando isto ocorre, os usuários notificam ao administrador que suas tarefas estão sempre falhando em determinado Worker. O administrador, por sua vez, reinicia o Worker falho da máquina e analisa o *log* à procura do motivo da falha.

Às vezes, devido a problemas de comunicação ou *bugs* no *middleware* OurGrid, o serviço Peer falha em prover ao MyGrid o número de máquinas requisitadas por ele, mesmo quando estas estão disponíveis. Quando isto ocorre, os usuários normalmente contactam o administrador da comunidade que reinicia o serviço Peer. Uma das características desejáveis no desenvolvimento do sistema era o de automatizar o processo de re-inicialização dos serviços assim que um problema fosse detectado.

Portanto, o Leukocyte foi desenvolvido para ser o mecanismo para reiniciar os serviços e resolver os problemas que levam esses serviços a falharem. Por exemplo, o AutoMan possui um monitor que verifica a capacidade dos diretórios que são usados pelas tarefas do OurGrid e gera as métricas *Peer.StorageDirSizeReport* para reportar as informações do diretório *$storage* e a métrica

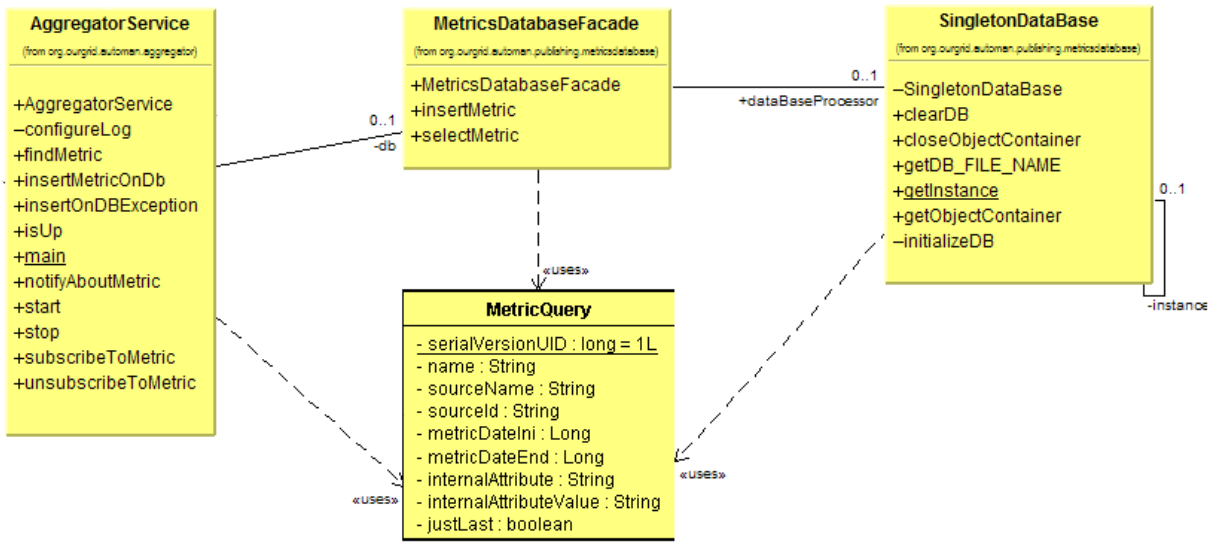

Figura 2.12: Principais classes da gerência do banco de dados do AutoMan

Worker.TempDirSizeReport para reportar as informações do diretório */tmp*. Assim que os diretórios chegam a uma percentagem de uso pré-estipulada, os arquivos temporários desses diretórios são removidos. Outras métricas e seu uso serão descritos em seguida.

A métrica *Peer.WorkerStatusChange*, que informa o estado do Worker (*i.e.*, *Contact, Idle, In Use, Donated, Owner*), é usada para verificar se é necessário reiniciar um Worker que sempre falha durante a execução das tarefas da grade computacional.

O AutoMan usa as métricas *DiscoveryService.PeerUp* e *DiscoveryService.PeerDown* (*i.e*, Peer ativo/inativo) para decidir quando irá reiniciar um Peer que pode não estar respondendo.

As métricas *DiscoveryService.DiscoveryServiceUp* e *DiscoveryService.DiscoveryServiceDown* informam quando o DiscoveryService não está respondendo. O DyscoveryService periodicamente envia uma mensagem(*i.e.*,*HeartBeat*) indicando que ele está ativo. A ausência desta mensagem indica que o DiscoveryService esta inativo e o AutoMan reinicia o serviço.

As principais classes do Leukocyte são ilustradas na Figura 2.14. O Leukocyte possui uma interface que estende as interfaces *MetricInterested* e *MetricPublisher* necessárias para a inscrição e recebimento das métricas do Aggregator.

A implementação do Leukocyte apresenta um servidor RMI, o *LeukocyteServer*, que implementa a interface *Leukocyte*. Ele tem a função de se inscrever nas métricas de interesse para sua gerência (ver Tabela 2.5), e receber do Aggregator as métricas nas quais ele se inscreveu como interessado. Ao receber alguma métrica, esta é repassada para a classe *Detector* que tem a função de analisá-la. Isto é feito comparando a métrica recebida com uma tabela de casos possíveis que o AutoMan pode

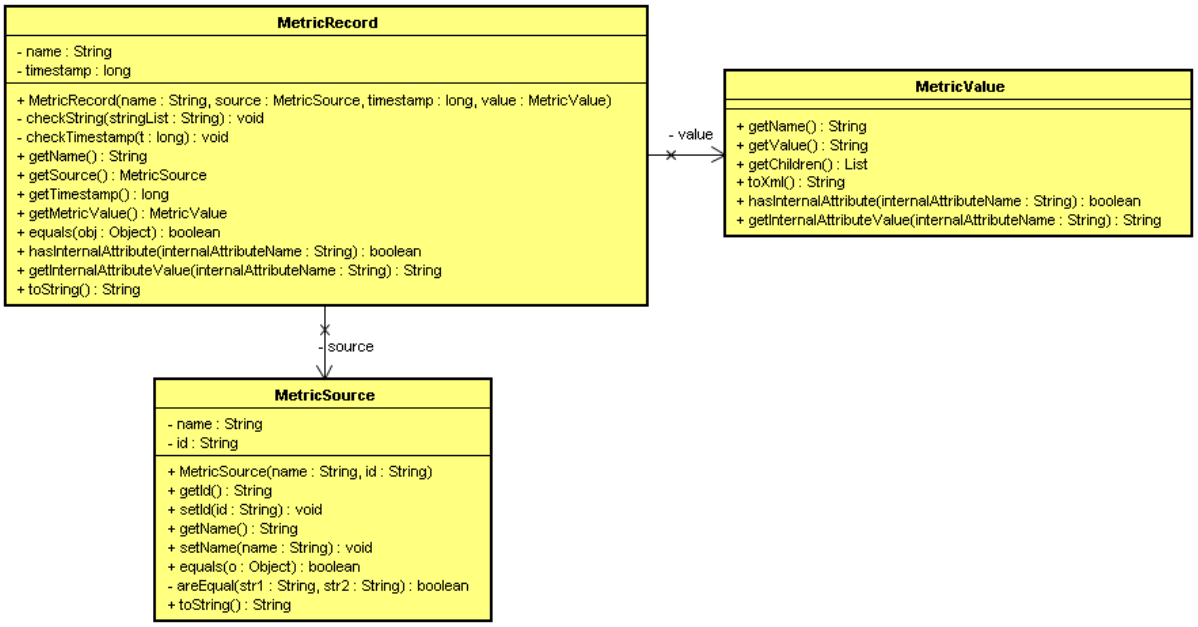

Figura 2.13: Classe MetricRecord e algumas classe relacionadas

recuperar (ver Tabela 2.5). Se a solução para o problema é encontrada, os dados necessários para isto são enviados para a classe *Effector*, que toma uma ação reparadora (chamada via *script*) específica para cada falha e componente. Toda vez que a classe *Effector* tenta recuperar um serviço falho (*i.e.*,Worker, Peer, DiscoveryService), ele fica monitorando a volta do serviço. Se em um período de tempo configurado, o serviço não for reativado, a ação reparadora é novamente executada.

Um exemplo dos processos de recuperação é a ressurreição dos serviços Peer, DiscoveryService (CorePeer) e Worker (UserAgents/GUM), que é executada através da chamada de *scripts* pelos respectivos métodos da classe *Effector*.

Para que o Leukocyte consiga detectar e recuperar um novo serviço é necessário seguir os seguintes passos: primeiro é cadastrado na lista de métricas de interesse do Leukocyte a métrica que possuí as informações sobre este serviço. No componente Detector esta métrica também é cadastrada na tabela com a sua respectiva solução. Depois no componente Effector, é implantado a chamada para o *script* passando as informações necessarias para o tratamento da falha e o por fim a criação do *script* que recuperará a falha.

2.3 AutoMan V2

Durante o desenvolvimento, implantação e uso da primeira versão do AutoMan (i.e., V1), foram feitas diversas observações dos comportamentos e características do sistema, onde foram constatados

Métrica	**Evento**	**Ação**
DiscoveryService.DiscoveryServiceDown	DiscoveryService fica inativo	Leukocyte chama um *script* que restaura o DiscoveryService.
DiscoveryService.PeerDown	Peer fica inativo	Leukocyte que chama um *script* que restaura o Peer.
Peer.WorkerStatusChange(Contact)	Worker fica inativo	Leukocyte chama um *script* que restaura o worker.
Peer.StorageDirSizeReport	$*storage* fica cheio	Leukocyte chama um *script* que limpa o $*storage*.
Worker.TempDirSizeReport	/tmp fica cheio	Leukocyte chama um *script* que limpa o /tmp.

Tabela 2.5: Tabela das métricas de Gerência do Leukocyte

pontos que não tinham sido previstos no projeto. Esses pontos foram: a dificuldade de implantação e configuração do sistema AutoMan na grade computacional; pontos únicos de falhas que poderiam comprometer todo o sistema de gerência e monitoramento; e erros ocasionados pelo sistema de recuperação.

A nova versão desenvolvida, além de possuir os requisitos gerados por essas observações, também agregou outras características que melhorariam o AutoMan tais como: a diminuição da intrusão do código de monitoramento do AutoMan no código do sistema a ser monitorado; a implantação de alguns padrões de programação que diminui a complexidade do entendimento do código para futuras atualizações e aumentariam a facilidade de implantar o AutoMan como sistema de monitoração e gerência automática em outras grades computacionais.

A decisão de se desenvolver uma nova versão para a grade computacional que estava em desenvolvimento (i.e., OurGrid Versão 4.0) em vez de se atualizar a versão antiga, foi tomada pelo motivo de mostrar que o sistema AutoMan pode ser empregado em outras grades computacionais - e possivelmente até em outros sistemas distribuídos - e assim fazer uma nova avaliação verificando se a implantação do AutoMan em um outro sistema também obteria resultados satisfatórios em relação à disponibilidade como na versão anterior.

Essa nova versão do AutoMan aumentou a tolerância a falhas do sistema de monitoração diminuindo os seus pontos mais suscetíveis a falhas. A complexidade de configuração também foi diminuída com a implementação do sistema de configuração semi-automático. Além disso, a disponibilidade do OurGrid 4.0 também aumentou, como será descrito no Capítulo 3. A seguir será

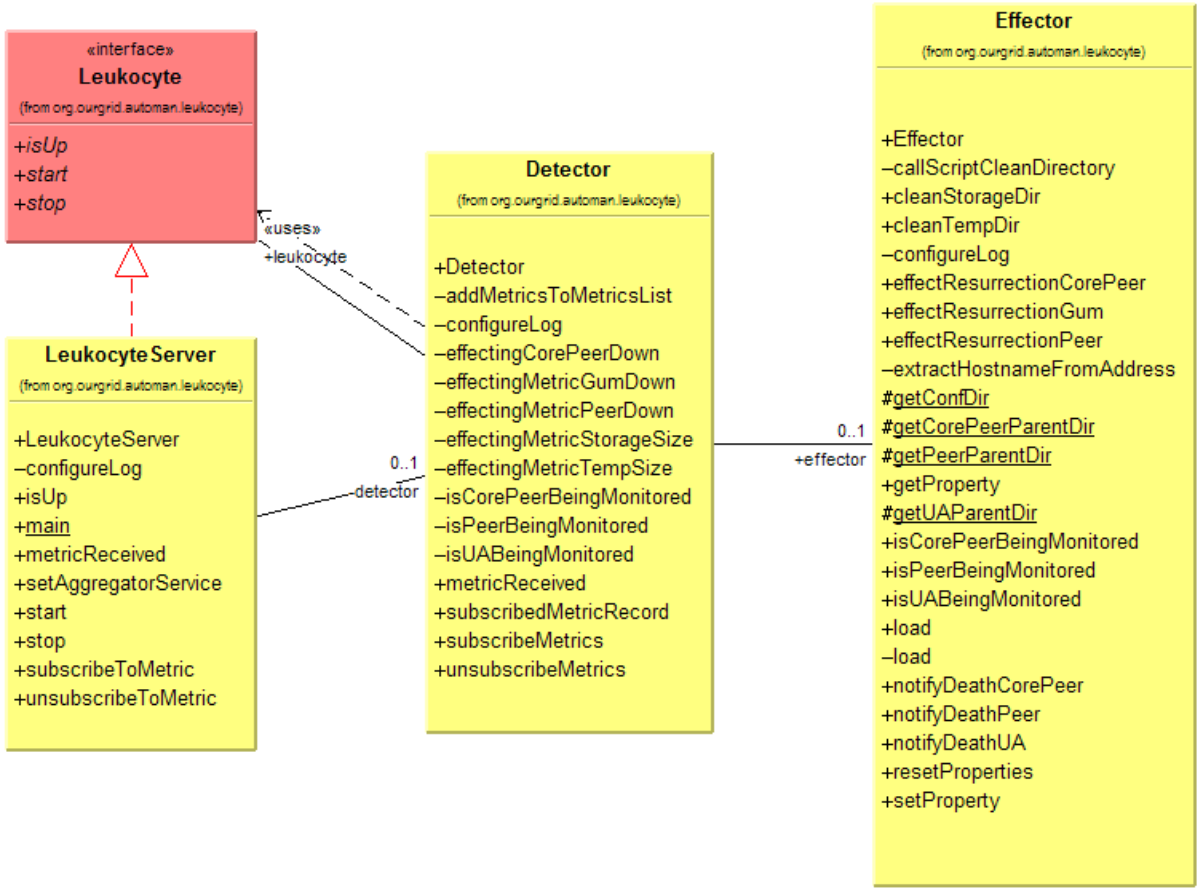

Figura 2.14: Principais classes do componente Leukocyte

apresentada a nova arquitetura (Seção 2.3.1) e as mudanças feitas na implementação (Seção 2.3.2) para a nova versão do AutoMan.

2.3.1 Arquitetura

A nova versão do AutoMan para o OurGrid 4.0 passou por algumas modificações na arquitetura original (ver Figura 2.1). O AutoMan V1 possui alguns pontos de falhas que comprometem todo o sistema. Um desses pontos de falha está na própria entidade Leukocyte, responsável pela recuperação dos componentes monitorados pelo AutoMan. Quando este falha, o sistema fica sem a habilidade de recuperar as entidades falhas do OurGrid. Também existe o problema de escalabilidade. O número de entidades que precisam ser recuperadas pode ser grande o suficiente para deixar a entidade Leukocyte muito lenta e/ou incapaz de funcionar corretamente (*i.e.*, o número limite de conexões *SSH* abertas na máquina foi excedido).

A solução encontrada para resolver esses problemas foi a replicação dos Leukocytes. Foi colo-

cado um serviço Leukocyte em cada máquina que possui um serviço OurGrid monitorado e/ou um serviço AutoMan como mostrado na Figura 2.15. Deste modo a reação à falha é mais rápida e o problema de escalabilidade melhorado. Porém ainda existe o problema da falha do Leukocyte. A solução encontrada foi implementar uma nova função no Aggregator. O Aggregator agora além das suas funções já descritas, tem a função de monitorar e reativar o Leukocyte se necessário. Nesta versão o Aggregator é o poto único de falha pois se ele falhar o sistema está comprometido, uma provável solução é discutida na Seção 5.1.

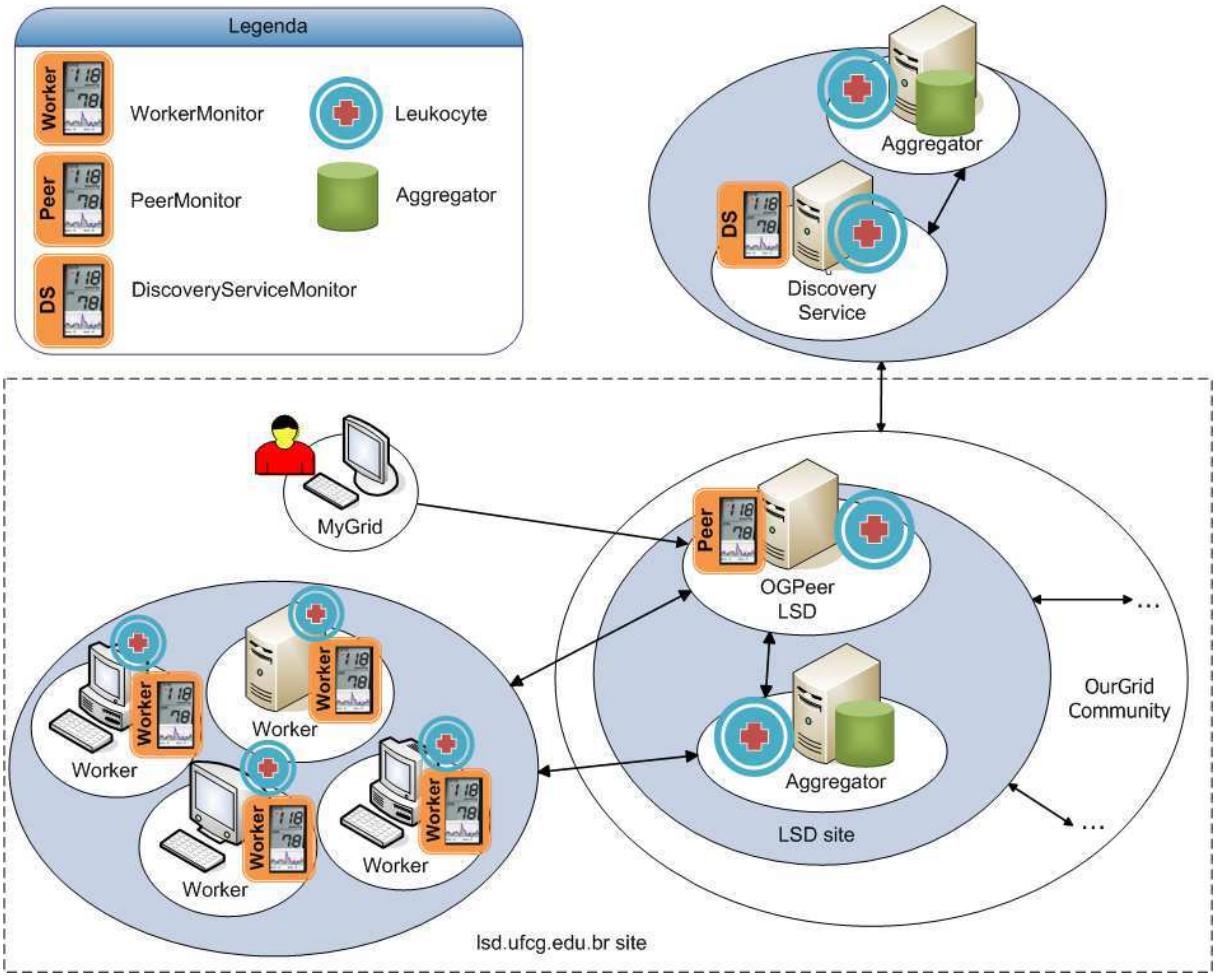

Figura 2.15: Entidades AutoMan V2

Monitor

Das três entidades monitor (*i.e.*, PeerMonitor, WorkerMonitor e DiscoveryServiceMonitor) apenas o WorkerMonitor e o PeerMonitor sofreram modificação nos seus requisitos. A nova grade computacional em que o novo AutoMan seria implantado passou por diversas modificações se comparada à

sua versão anterior, sendo que uma delas afeta diretamente a coleta de uma das métricas. Uma das métricas corresponde às informações coletadas sobre o espaço disponível no diretório *$storage*, que é o diretório usado para armazenamento de arquivos comuns entre os *jobs*. Na versão anterior do OurGrid essa métrica é coletada na entidade Peer que monta esse diretório, agora quem monta esse diretório são os Workers.

O outro requisito a ser implementado na nova versão do AutoMan é a diminuição da complexidade de configuração. Na primeira versão, existias um arquivo de configuração para cada entidade AutoMan que tinha que ser preenchida pelo administrador com informações de nome da máquina que está sendo monitorada, endereço lógico, local que no disco em que o serviço monitorado está instalado entre outros. Com a grande escalabilidade da grade, poderia ser necessário preencher centenas ou milhares de arquivos com isto, a introdução de erros é inevitável. Como solução, foi implementado um método de configuração semi-automática nos monitores. Quando os monitores são iniciados, os mesmos coletam as seguintes informações: diretório de instalação, nome da máquina e endereço lógico do *host* em que o serviço monitorado está instalado. Estas informações são armazenadas em um arquivo em um diretório padrão para serem usados pelos Leukocytes quando necessário.

Agregador de Dados

O Aggregator foi a entidade que teve a mudança mais impactante. Como citado, o Aggregator (ver Figura 2.16) agora tem a função de monitorar e reviver o Leukocyte. Esta função foi adicionada no Aggregator e não em um Leukocyte pelo motivo de que para fazer com que um Leukocyte tivesse destreza de recuperar outro, se fazia necessário garantir que ele tenha acesso à máquina onde se encontra o Leukocyte falho. No entanto, o Aggregator como entidade central, tem a comunicação garantida a todos os Leukocytes que nele se inscreveram como interessados. Podendo desta forma acessar as máquinas para recuperar o Leukocyte falho.

Isso foi feito adicionando o componente *Effector* - o mesmo do Leukocyte - para fazer a recuperação. Quando o Aggregator tenta enviar uma métrica para uma Leukocyte e não consegue, ele assume que o mesmo está inativo e toma uma ação reparadora.

A Figura 2.17 ilustra um diagrama de seqüência com a interação ocorrida quando um Leukocyte fica inativo e o Aggregator nota quando tenta enviar uma métrica. O Leukocyte é ativado (1) e em seguida se inscreve como interessado em certas métricas (no caso o WorkerDown) no Aggregator (1.1); se o Leukocyte falha (2); O PeerMonitor notifica para o Aggregator que o Worker ficou inativo (3); O Aggregator, por sua vez, tenta contactar o Leukocyte responsável pelo Worker falho (4). Como o Leukocyte esta inativo, o envio da notificação falha (4) e o Aggregator toma a ação de recuperar

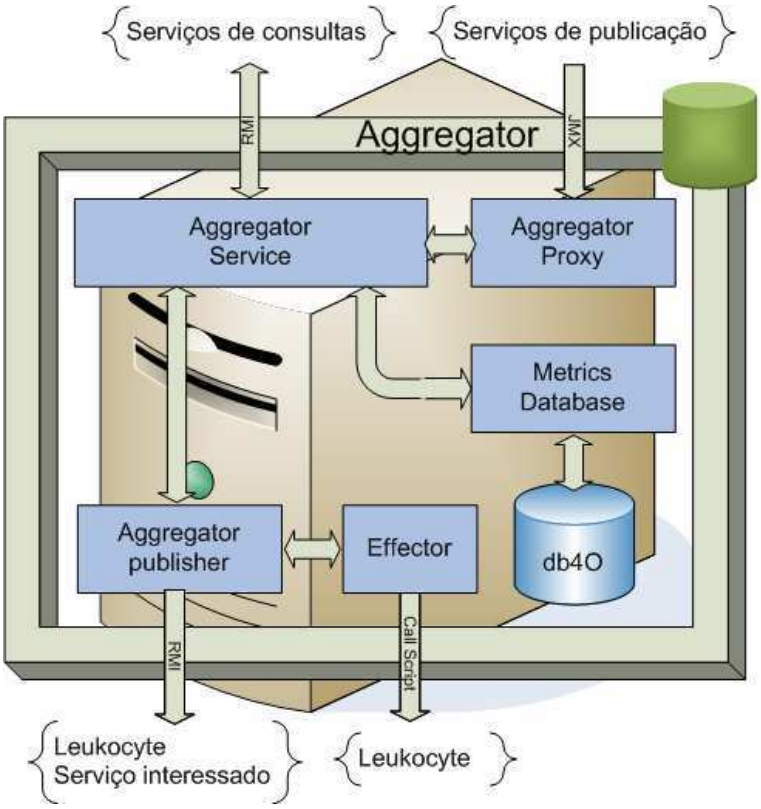

Figura 2.16: Nova Entidade Aggregator V2

o Leukocyte falho (5); Quando ativo novamente, o Leukocyte recupera todas a entidades pelas quais ele é responsável (5.1), no caso o Worker da sua máquina.

Analisador e Atuador

A entidade Leukocyte teve apenas um novo requisito: ter a habilidade de recuperar outro Leukocyte. Esta habilidade foi implementada nos sub-componentes *Detector* e *Effector* pois, como discutido, estes componentes agora também são usados pelo Aggregator.

2.3.2 Implementação

Esta Seção apresenta as implementações desenvolvidas para satisfazer os novos requisitos do AutoMan V2. Na implementação do AutoMan V2 foram aplicados alguns padrões de programação objetivando melhor desempenho e a facilidade de entendimento em uma possível evolução do sistema.

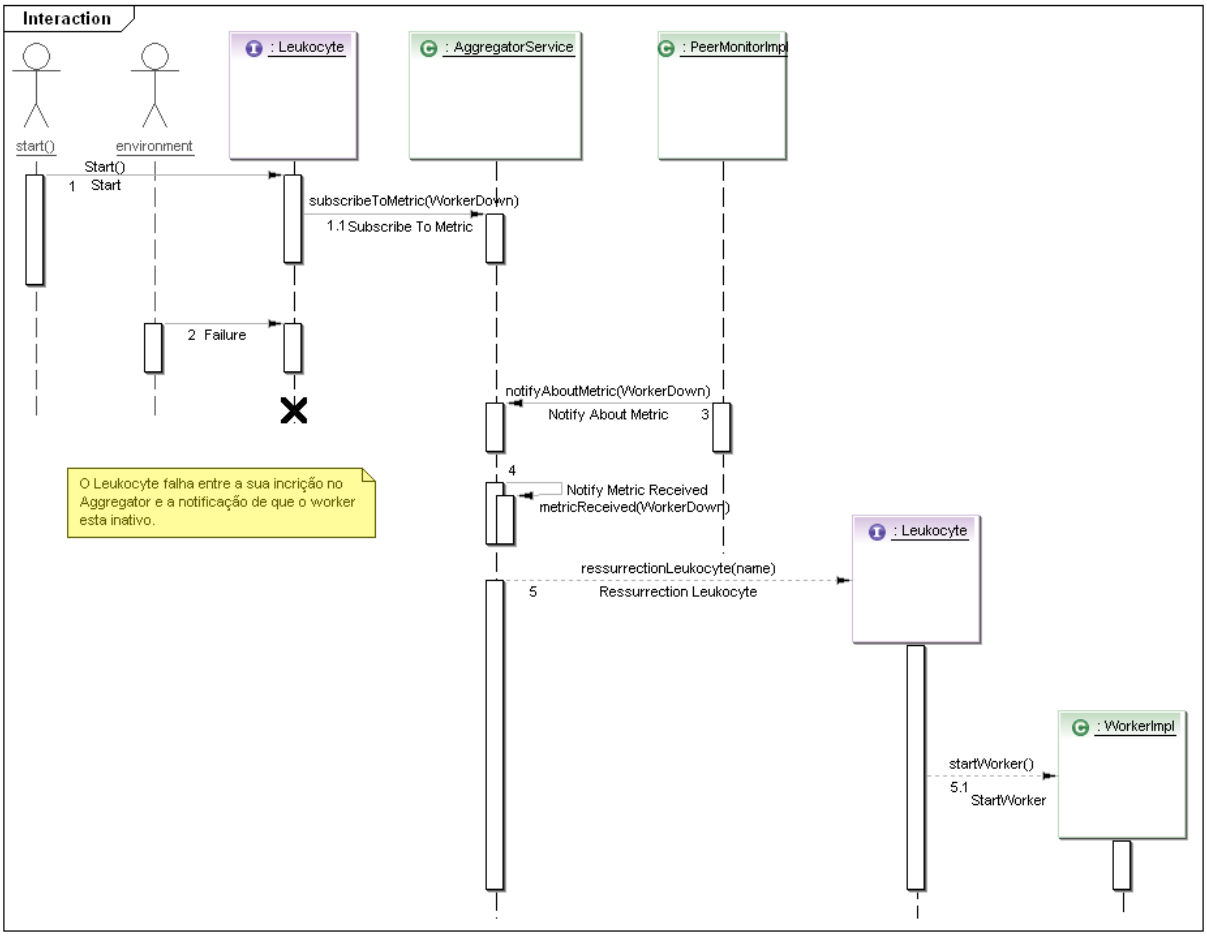

Figura 2.17: Interação entre o Leukocyte e o Aggregator

Obtenção das Métricas

Com a mudança do OurGrid, a métrica que informa o tamanho e espaço livre do diretório *$storage* deixou de ser monitorada pelo PeerMonitor e passou a ser monitorada pelo WorkerMonitor. Como já foi discutido na Seção 2.2.2, acrescentar/remover métricas ao Monitor podem ser feitas sem grandes dificuldades.

Nesta versão do monitor, foi implementado um mecanismo responsável por fazer a configuração semi-automática do sistema AutoMan. Este sistema faz a coleta das informações necessárias para que o Leukocyte possa fazer a recuperação das entidades falhas. Tais informações são: os serviços monitoráveis pelo AutoMan da máquina, o caminho do diretório em que os serviços estão instalados e o endereço lógico da máquina. Antes, estas informações eram preenchidas pelo administrador no momento do sistema ser instalado, aumentando a possibilidade de erros provocando um mau funcionamento do sistema de monitoração. O método implantado para realizar esta função foi o *createPropertiesForMonitoring* na classe *RemoteMonitor* do monitor referente. Essas classes podem

ser vistas na Figura 2.18.

Figura 2.18: Principais classe do PeerMonitor V2

Em relação à redução da dificuldade no caso da implantação do AutoMan em outras grades computacionais, foi implementada uma fachada (RemoteMonitor) para fazer a conexão de forma mais simples e menos intrusiva entre o código do monitor e o código do sistema hospedeiro.

Armazenamento e Publicação dos Dados Monitorados

A entidade Aggregator tem duas novas funções, a monitoração e recuperação do Leukocyte. Quando o Aggregator (Figura 2.19) tenta notificar um Leukocyte através da chamada RMI e esta falha, o *AggregatorPublisherThread* lança uma exceção e manda as informações para a classe *AggregatorMetricsRecordHandler* que gera uma métrica informando que o Leukocyte está inativo (Leukocyte-

Down). Essa métrica é armazenada no banco de dados e repassada para a classe *Effector* que tomará as medidas necessárias para a ressurreição do Leukocyte falho.

Figura 2.19: Principais Classes do Aggregator V2

Outra novidade implantada é um *pool de threads* que se encarrega de enviar as mensagens com as métricas para as entidades interessadas. Esta estratégia foi adotada para melhorar o desempenho e a escalabilidade do Aggregator. Quando a classe *AggregatorService* recebe uma métrica que vai ser repassada para outras entidades, ele envia esta métrica para a classe *AggregatorPublisher* que utiliza um *pool de threads* (classe *AggregatorPublisherThread*) para fazer o repasse.

Recuperação das Entidades OurGrid

A implementação do Leukocyte, como citado anteriormente, teve apenas um novo requisito que foi o de poder reviver outro Leukocyte.

Figura 2.20: Novas Classes: Effector e Leukocyte

Na classe *Detector* passaram a ser considerados também as falhas e soluções relativas a esse novo evento a ser tratado. Na classe *Effector* foi implementado um método *effectRessurrection* necessário para fazer a ressurreição do Leukocyte falho (ver Figura 2.20).

2.4 Lições Aprendidas

Ao implementar e utilizar uma solução automática para gerência de grades, algumas lições foram aprendidas:

- **Atenção com arquivos de configuração**: **É muito fácil introduzir erros nesses arquivos**.

Gerência automática normalmente exige que várias configurações sejam feitas. Percebeu-se que é fácil introduzir erros em arquivos de configuração. Por exemplo, no AutoMan há arquivos de configuração indicando as máquinas onde um dado serviço foi instalado e o diretório de instalação dos serviços OurGrid. Sempre que se alterava a grade computacional, necessitava-se alterar esses arquivos de configuração. Várias vezes percebeu-se que erros de configuração, ou arquivos de configuração incompletos, eram as causas para o não restabelecimento automático de alguns serviços. Na segunda versão, as informações nos arquivos de informações foram reduzidas consideravelmente, já que nesta versão os arquivos são gerados automaticamente.

- **Não assumir que o *software* que está sendo monitorado está livre de *bugs***: Observou-se que considerar isso leva a uma solução de gerência automática falha. No AutoMan, por exemplo, parte da monitoração era feita através de instrumentação de código e dependia de mecanismos de detecção de falhas próprios da aplicação. Porém, percebeu-se que havia um comportamento anômalo do *software* nessa detecção. Por esse motivo, às vezes o sistema detectava a falha de serviços que estavam executando sem problemas, ou então não percebia a falha de recursos que estavam de fato indisponíveis. Isso acontecia, por exemplo, com o DiscoveryService (CorePeer) ao monitorar as falhas dos Peers. Desta forma, o Leukocyte não era notificado sobre falhas e não conseguia automaticamente reiniciar os serviços correspondentes. Para evitar tal problema, fez-se necessário o uso de mecanismos replicados para monitorar elementos via instrumentação de código ao invés de inserir código de monitoração em um único ponto ou serviço. Incluindo monitoração de *heartbeats* (como foi feito com o DiscoveryService (CorePeer)) na entidade Peer, poderíamos ter evitado o problema citado anteriormente.

- **Gerência Automática não substitui totalmente a gerência manual**: É muito comum que determinadas situações sejam esquecidas quando se projeta um sistema de gerência automática e uma pessoa é capaz de perceber comportamentos estranhos que uma máquina não conseguiria. No nosso caso, isso aconteceu quando os serviços ficavam indisponíveis, mas nenhuma notificação sobre isso era gerada devido a comportamentos anômalos do sistema. Além disso, erros de configuração em ferramentas de gerência automática também podem ser comuns, e muitos erros só poderão ser percebidos por humanos.

- **Mecanismos de recuperação devem ser escolhidos cuidadosamente**: Os mecanismos da nossa implementação do serviço Leukocyte eram baseados em *secure shell* (*i.e.*, *SSH*). Percebeu-se que essa não era a melhor alternativa e que outros mecanismos para o restabelecimento de serviços poderiam ser selecionados. Outras abordagens como *Smart Framework*

for Object Groups (SmartFrog) [26] ou *Configuration Description, Deployment and Lifecycle Management* (CDDLM) [7] poderiam ter sido mais úteis e acreditamos que deveriam ser exploradas já que focam em configuração e gerência de ciclo de vida de componentes. Erros comuns que obtivemos ao utilizar SSH foram o *"Too many open files in system"* ou conexões SSH recusadas devido a mudanças no identificador das máquinas (isso era comum quando se atualizava o sistema operacional de algumas máquinas). Devido a problemas como estes, alguns serviços deixaram de ser restabelecidos como se esperava. Na segunda versão do AutoMan apesar de ainda ser usada a solução baseada em *secure shell*, conseguiu-se diminuir os impactos negativos pois com a nova arquitetura as chamadas *SSH* feitas estão no mesmo domínio administrativo e a escalabilidade foi aumentada com a distribuição dos Leukocytes.

Capítulo 3

Avaliação

Este Capítulo apresenta os resultados e discussões das avaliações realizadas em ambas as versões do AutoMan. A Seção 3.1 apresenta os resultados da primeira versão do AutoMan e a Seção 3.2 apresenta os resultados da segunda versão do AutoMan.

3.1 Avaliação do AutoMan V1

Para avaliar o **AutoMan V1**, utilizaram-se duas variantes do OurGrid (uma com e outra sem o AutoMan V1) submetidas a dois conjuntos de experimentos. O primeiro conjunto mediu o impacto da monitoração no desempenho geral da grade. Foram comparados o desempenho do OurGrid com e sem o sistema de monitoração, com o propósito de avaliar a degradação no desempenho devido aos procedimentos de monitoração e gerência. O segundo conjunto de experimentos comparou as duas variantes do OurGrid para identificar a indisponibilidade média dos serviços OurGrid, dando uma idéia do quanto o AutoMan pode melhorar a disponibilidade de serviços da grade.

No OurGrid uma máquina da grade é considerada indisponível caso ela não esteja apta a executar uma tarefa da grade em um dado instante. Quando uma máquina está desligada, com defeito, ou é reiniciada em um sistema operacional no qual o serviço do Worker não está instalado, ela está de fato indisponível, todavia isso não implica que a grade em si está falhando.

3.1.1 Medindo o *Overhead* do AutoMan

Para medir o impacto do processo de monitoração do AutoMan no desempenho geral do sistema, foi desenvolvido um teste unitário (*junit*) que submete uma quantidade pré determinada de *jobs* e mede o tempo que ele levou para ser executado. Em um ambiente controlado (*i.e.* , máquinas com tráfego de

rede regular, executando o sistema operacional Linux), um *Job* foi submetido 500 vezes na grade para cada variante do OurGrid (uma com o serviço de monitoração habilitado e a outra sem). Os resultados mostraram que o impacto do processo de monitoração no desempenho do OurGrid foi mínimo. Em média, as tarefas levaram 1,86% mais tempo para rodar quando o sistema de monitoração estava habilitado, porque os Monitores, Aggregators e Leukocytes competiram pelos mesmos recursos da grade hospedeira (*e.g.* tarefas submetidas à grade). Usando o método T-Test [31], utiliado para calcular a probabilidade associada determinar a probabilidade de que duas amostras são significativas, com intervalo de confiança de 95%, concluímos que os resultados coletados para o tempo de execução para as duas variantes do OurGrid não possuem diferenças significativas. Isso é ilustrado na Figura 3.1, que mostra os tempos de execução dos *Jobs* submetidos à grade computacional.

Figura 3.1: Tempo de execução com e sem AutoMan

3.1.2 Indisponibilidade dos serviços OurGrid

Para verificar o quanto o AutoMan pode otimizar o uso de recursos perecíveis (maximizando o uso de máquinas), foram consideradas as seguintes métricas: indisponibilidade média das três entidades monitoradas (*i.e.*, Peer, Worker e DiscoveryService/CorePeer) em máquinas com e sem o AutoMan. A comunidade possuía 31 máquinas, uma com os serviços DiscoveryService e Peer e outra 30 máquinas com os serviços Worker. A indisponibilidade media considerada foi a do tempo que o serviço ficava indisponível entre à falha até sua recuperação. Dois tipos de indisponibilidade foram considerados:

contornável e inevitável. Uma indisponibilidade contornável é a que pode ser minimizada com a aplicação do AutoMan, como nos momentos em que um serviço falha devido a algum problema, e pode ser rapidamente reiniciado. A indisponibilidade inevitável é a que o AutoMan é incapaz de qualquer ação efetiva, como quando uma máquina é desligada, ou então quando o usuário decide mudar a máquina para outro sistema operacional.

Para medir a indisponibilidade, foi implementada uma ferramenta que coleta os intervalos de indisponibilidade a partir dos *logs* do OurGrid. Para verificar se uma indisponibilidade é inevitável, outra ferramenta similar para obter intervalos de indisponibilidade usando os *logs* do serviço de monitoração do *Nagios* [25] foi implementada. Desta forma, pode-se verificar se um determinado serviço esteve indisponível porque a máquina em si estava indisponível. Depois de coletar esses intervalos, a lista de máquinas foi filtrada para remover as que não estavam sendo sempre monitoradas pelo Nagios, ou que não estavam sempre inclusas na grade. Esse processo foi executado nas duas variantes do OurGrid (*i.e.*, com e sem o AutoMan). Adicionalmente, foram coletados valores de pico e analisadas as circunstâncias em que eles surgiram.

Considerando a comunidade usando AutoMan, embora ela tenha sido menor que a comunidade OurGrid real, foi emulado um cenário real pela submissão de *jobs* pertencentes a usuários regulares do OurGrid. Além disso, foram executadas injeções de falha através do encerramento de processos de entidades OurGrid para observar o tempo necessário para o serviço ser restabelecido automaticamente.

No ambiente OurGrid que foi analisado, há dois grupos de máquinas: o primeiro são as máquinas que integram a grade durante todo o tempo; o segundo corresponde às que estão normalmente usando um sistema operacional onde os serviços OurGrid não podem ser executados, ou máquinas que não estão disponíveis porque estão sendo operadas pelo usuário.

Para obter a indisponibilidade correspondente à variante regular do OurGrid (*i.e.*, sem AutoMan), somente dados de máquinas do primeiro grupo foram considerados, porque são as mais usadas e tem mais períodos válidos de indisponibilidade de serviços. Os dados para a variante regular do OurGrid foram obtidos dos *logs* observados num período de três meses. Para cada mês foi feito o seguinte: para cada máquina que integrava o OurGrid naquele mês, foi computado o período médio de indisponibilidade e depois disso, os resultados relativos aos três meses foram comparados. Estes resultados são apresentados na Tabela 3.1.

Para obter os dados de indisponibilidade da segunda variante do OurGrid (*i.e.*, com AutoMan), criou-se uma comunidade de teste do OurGrid, e se selecionou alguns *jobs* que são usualmente submetidos no ambiente OurGrid real. Escolheu-se os *jobs* que geraram mais indisponibilidade para

Service	Sem AutoMan				Com AutoMan			
	Contornável		Inevitável		Contornável		Inevitável	
	Média	Pico	Média	Pico	Média	Pico	Média	Pico
Worker	72900	318570	611809	60795.9	600	853	0	0
Peer	20073	29237	0	0	0	0	4	1
DiscoveryService	990	4719	0	100	103	311	4	4

Tabela 3.1: Tempos de Indisponibilidade (em segundos)

a grade no passado (*i.e.*, aqueles que demandam mais recursos de máquina como CPU, memória e espaço de disco). Depois que essa comunidade estava operando foram injetadas falhas nas entidades DiscoveryService (CorePeer), no Peer e Worker para observar os serviços AutoMan diagnosticando e tratando os problemas. Para injetar falhas decidiu-se simplesmente por matar os processos correspondentes a essas entidades do OurGrid.

Comparando as duas variantes do OurGrid, pode-se observar o seguinte: a indisponibilidade média dos Workers caiu de 72900 s para 600 s, e com picos de indisponibilidade caindo de 318570 s para 853 s. Nesta avaliação foi considerada a média de indisponibilidade real (*i.e.*, 318570 s) ao invés da média de indisponibilidade da máquina (*i.e.*, 611809 s), porque a indisponibilidade das máquinas totalmente aptas a serem usadas pela grade é menor que a indisponibilidade geral das máquinas.

Observou-se que o tempo médio para que o Worker esteja disponível na grade é em torno de 10 minutos, porque este é o tempo para que o sistema de detecção de falhas do OurGrid perceba a recuperação do Worker, mas de fato o serviço estava recuperado antes disso conforme foi observado no processo de monitoração da máquina. Registrou-se um tempo insignificante entre a injeção da falha e a reação do AutoMan para restabelecer o serviço. Além disso, na variante regular do OurGrid observaram-se alguns períodos longos de indisponibilidade durante os finais de semana, porque não havia administradores para restabelecer manualmente os serviços.

Para os Peers, a indisponibilidade caiu de 20073 s para aproximadamente 0 s (os dados obtidos indicam um tempo menor do que 1 s para cada indisponibilidade), com pico de indisponibilidade caindo de 29237 s para aproximadamente 0 s.

O serviço DiscoveryService teve uma queda de indisponibilidade de 990 s para 103 s, com picos de indisponibilidade variando de 4719 s para 311 s.

3.2 Avaliações do AutoMan V2

Para avaliar o **AutoMan V2**, foram usadas duas variantes do OurGrid, uma com e outra sem o AutoMan. O experimento teve como objetivo medir e comparar a disponibilidade média entre os componentes das duas variantes do OurGrid, dando uma idéia do quanto o AutoMan pode melhorar a disponibilidade de serviços da grade. Nesse experimento não foram separados os dois tipos de indisponibilidade que podem afetar o OurGrid, indisponibilidade de um serviço do OurGrid que o AutoMan pode recuperar e a indisponibilidade física da máquina que o AutoMan não pode contornar, pois queríamos avaliar o comportamento em um ambiente real onde o indisponibilidade do serviço não distingui seu motivo.

Com o sistema AutoMan ativado, um serviço da grade indisponível é detectado e recuperado em dez minutos se esta falha for contornável. Este tempo de detecção de falhas pode ser configurado.

3.2.1 Indisponibilidade dos serviços OurGrid

Para verificar o quanto o AutoMan pode otimizar o uso de recursos perecíveis (maximizando o uso de máquinas), foram consideradas as seguintes métricas: indisponibilidade média das três entidades monitoradas (Peer, Worker e DiscoveryService) em máquinas com e sem o AutoMan. Para obter os dados necessários para avaliação, foi implementada uma ferramenta semelhante à ferramenta usada na avaliação do AutoMan V1 que faz a coleta dos intervalos de indisponibilidade dos serviços nos registros (*logs*) gerados pelo OurGrid.

A comunidade analisada foi implementada no Laboratório de Sistemas Distribuídos (*www.lsd.ufcg.edu.br*). As comunidades possuíam uma máquina com o serviço DiscoveryService, uma máquina com o serviço Peer e 35 máquinas com o serviço Worker instalado. Em ambas as comunidades, com e sem AutoMan, foram utilizadas as mesmas máquinas com os mesmos serviços OurGrid instalados. Pesquisadores do laboratório submetiam constantemente tarefas que geravam altas cargas de processamento no OurGrid (*e.g.*,simulações) no decorrer de todo experimento para ambas as comunidades (*i.e.*, com e sem AutoMan).

Foram coletados e analisados *logs* de cinco semanas consecutivas para cada variante OurGrid. A variante sem AutoMan teve sua coleta dada entre 30 de julho e 2 de setembro de 2007, e a variante com AutoMan deu entre 24 de setembro e 28 de outubro de 2007.

Comparando as duas variantes do OurGrid, pode-se observar o seguinte: a indisponibilidade média do DiscoveryService foi de $93,75\%$ sem AutoMan e com o AutoMan a indisponibilidade caiu para $26,95\%$ como apresentado no gráfico da Figura 3.2. Esses altos valores de indisponibi-

lidade do discoveryService se deram pela pouca preocupação dos administradores com o serviço DiscoveryService, pois este serviço não é crucial quando a comunidade OurGrid possui apenas um Peer. Como supracitado, o DiscoveryService é um serviço de descoberta de Peers. Outro motivo para estes valores foram as costantes faltas de energia devido à ampliação da rede elétrica da universidade nesses três meses de experimentos, afetando os resultados de ambas as versões do AutoMan.

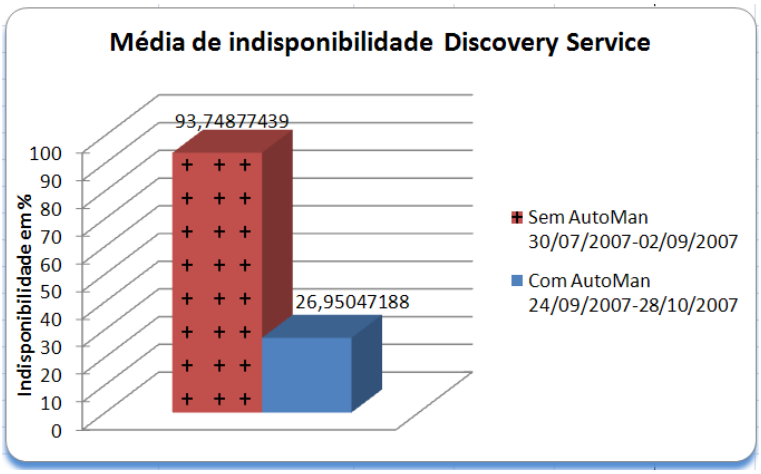

Figura 3.2: Indisponiblidade Média do DiscoveryService

O Gráfico da Figura 3.3 apresenta a indisponibilidade do DiscoveryService durante as cinco semanas. No decorrer das duas primeiras semanas, o DiscoveryService sem AutoMan esteve praticamente 100% indisponível. Na terceira semana, o serviço ficou aproximadamente 87% do tempo indisponível havendo um aumento para 99% na quarta semana. Na quinta semana a indisponibilidade caiu para 80%.

Com o AutoMan, a indisponibilidade foi decrescendo de 46% na primeira semana para $0(zero)\%$ na terceira semana e permanecendo assim até a quarta semana. Entre a quarta e quinta semana a indisponibilidade aumentou para aproximadamente 47%.

Para os Peers, a indisponibilidade média caiu de $14,37\%$ sem o AutoMan para $2,98\%$ com o AutoMan (vide Figura 3.4).

Como pode-se observar no gráfico da Figura 3.5 , o Peer sem AutoMan durante as duas primeiras semanas ficou $0(zero)\%$ indisponível tendo um aumento para 39% na terceira semana. Nas duas ultimas semanas a indisponibilidade decaiu até chegar aos 9% na quinta semana. A grande variação de indisponibilidade durante a segunda e quinta semana se deu pelo motivo da falta de energia, principalmente durante os fins de semanas.

O Peer com AutoMan teve $0(zero)\%$ de indisponibilidade na primeira semana aumentando para 15% na segunda semana. Na terceira semana a indisponibilidade caiu para $0(zero)\%$ e permaneceu

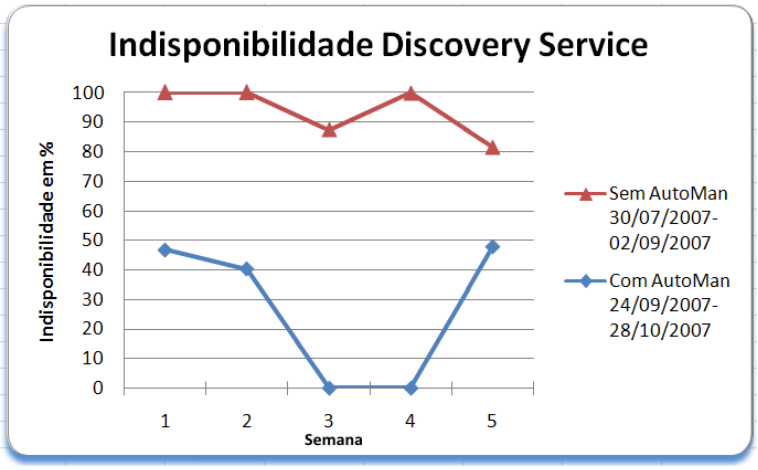

Figura 3.3: Indisponiblidade Semanal do DiscoveryService

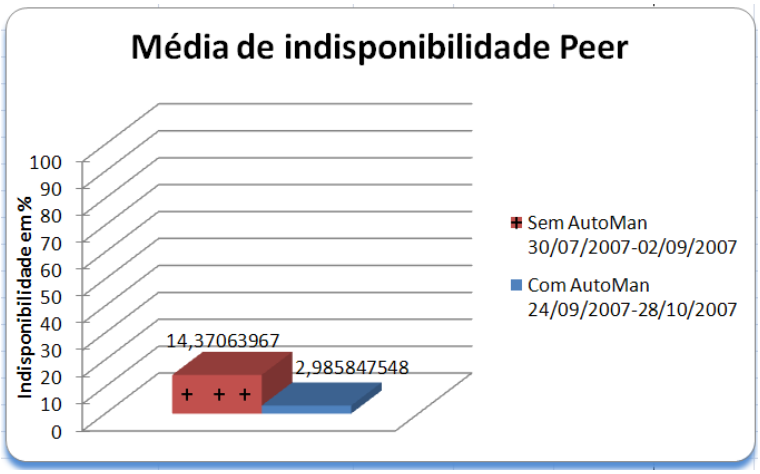

Figura 3.4: Indisponiblidade Média do Peer

assim até o final das cinco semanas do experimento.

Os Workers tiveram uma queda da indisponibilidade média de $30,17\%$ sem AutoMan para $4,47\%$ com o AutoMan (vide Figura 3.6).

O gráfico apresentado na Figura 3.7 ilustra a indisponibilidade durante as cinco semanas do experimento do Worker com/sem AutoMan. Os Workers sem AutoMan durante as três primeiras semanas ficaram em média 35% indisponível. Na quarta semana ocorreu uma diminuição para 20%, tendo na quinta semana uma pequena alta para 25% de indisponibilidade.

Com o AutoMan, a indisponibilidade média nas três primeiras semanas foi de aproximadamente 2% havendo um aumento para 7% nas duas ultimas semanas.

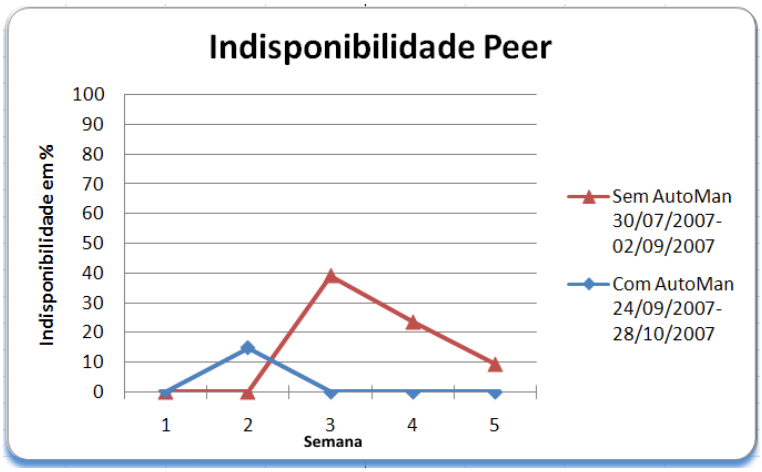

Figura 3.5: Indisponiblidade Semanal do Peer

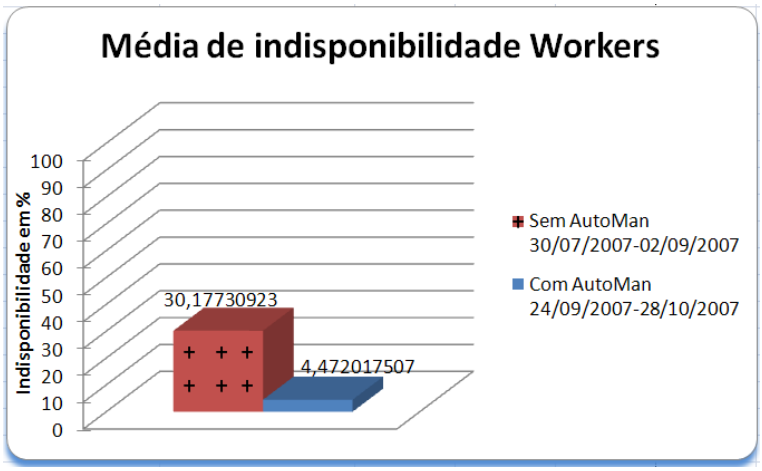

Figura 3.6: Indisponiblidade Média do Worker

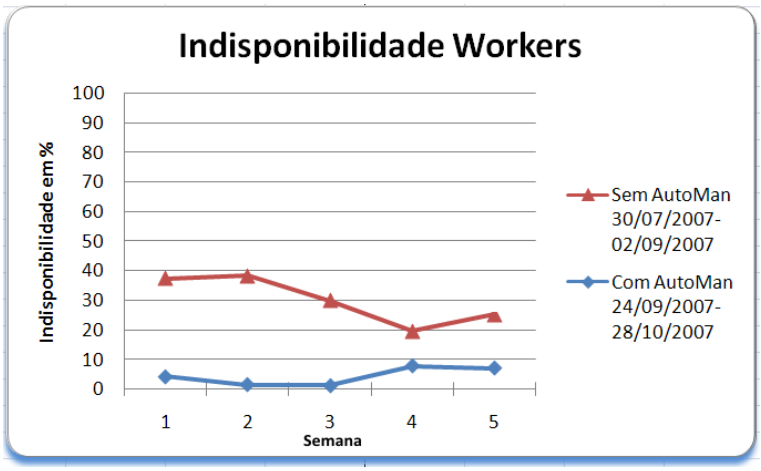

Figura 3.7: Indisponiblidade Semanal dos Workers

Capítulo 4

Trabalhos Relacionados

Este Capítulo aborda os trabalhos de gerência automática e monitoramento para grades computacionais. O Capítulo está dividido em duas Seções. A Seção 4.1 apresenta padrões de arquiteturas de monitoramento para grades computacionais e a Seção 4.2 apresenta alguns sistemas de gerência e monitoramento.

4.1 Arquiteturas de Monitoramento

As subseções seguintes discutem as propostas de padrões de arquiteturas de monitoramento para grades computacionais.

4.1.1 Grid Monitoring Architecture

Grid Monitoring Architecture (GMA) [5] é um padrão de arquitetura de monitoração para grades computacionais definido pelo *Open Grid Forum* (OGF) [20] - antigo Global Grid Forum (GGF) - com o objetivo de propiciar a interoperabilidade entre diversos sistemas de monitoração estabelecendo uma terminologia padrão e descrevendo uma especificação mínima de um sistema de monitoramento de grades computacionais. O GMA se baseia na arquitetura produtor/consumidor e em serviços de diretório. A arquitetura consiste em três interfaces distintas: *Directory Service*, *Consumer* e *Producer* (apresentados na Figura 4.1).

- O *Directory Service*: tem a função de publicação e descoberta de informação.
- O *Producer*: tem a função de coletar os dados da grade computacional (e.g., sensores, topologia da rede) e prover a informação coletada para os *Consumers*.

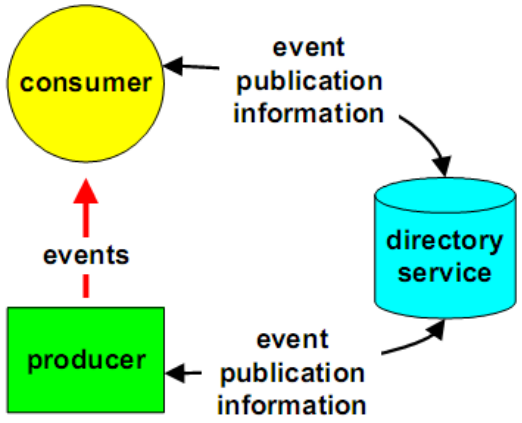

Figura 4.1: Componentes da Arquitetura GMA [5]

- Os *Consumers*: recebem/requisitam a informação provida pelos *Producers* e as publica para o *Directory Service*.

Além de *Producers* e *Consumers*, a arquitetura GMA também provê componentes intermediários (*i.e.*, *intermediaries*), os quais implementam ambas as interfaces.

Todas as informações trocadas entre os componentes são descritas na forma de eventos contendo os dados organizados em uma estrutura específica definida por um esquema (*event schema*) [29]. *Producers* e *Consumers* publicam no *Directory Service* sua existência e informações sobre eventos que eles produzem/consomem, protocolos aceitos e mecanismos de segurança. Os *Consumers* também utilizam o *Directory Service* para procurar *Producers* específicos, tão como os *Producers* utilizam-no para procurar *Consumers* interessados em seus serviços. Qualquer um dos dois componentes pode iniciar a interação com o componente descoberto.

Pode-se mapear os componentes da arquitetura AutoMan com os componentes da arquitetura GMA da seguinte forma. Os eventos gerados pelo GMA são equivalentes às métricas do AutoMan que também seguem um esquema de organização das informações contidas na métrica. Os Monitores do AutoMan seriam equivalentes aos *Producers* do GMA. O Aggregator do AutoMan possui as três interfaces. A interface *Directory Service*, pois ele conhece todos os *Producers* e *Consumers*, a interface *Producers*, pois ele envia as métrica recebidas para os *Consumers* inscritos e a interface *Consumer*, pois ele recebe as métricas enviadas pelos Monitores. O Leukocyte seria o *Consumer*, já que ele se inscreve em certas métricas no Aggregator para o recebimento quando estas métricas são geradas.

Na arquitetura GMA, existem três tipos de interações possíveis entre *Producers* e *Consumers*: *query/response*, *publisher/subscribe* e *notification*. Na interação *query/response*, o iniciador da in-

teração, que tem que ser um *Consumer*, envia um pedido a um *Producer* indicando quais eventos gostaria de receber informações, e o *Producer* por sua vez retorna com uma única resposta. Na interação *publisher/subscribe*, quem inicia o processo contacta o "servidor", indicando o interesse em alguns tipos de eventos. Na inicialização da interação também são enviados parâmetros de controle como, por exemplo, periodicidade e destino do evento. O produtor envia esses eventos até que o interessado cancele sua inscrição. Na interação *notification*, o iniciador transfere as informações para o consumidor uma única vez.

O AutoMan também possui esses três tipos de interação entre seus componentes, a interação *publisher/subscribe* se dá entre Aggregators e Leukocytes, entre Aggregators e outros serviços desenvolvidos (*e.g.*, página da *web* com informações coletadas pelo AutoMan) e entre Aggregators. A interação *query/response* é feita entre o Aggregator com serviços de consultas (*e.g.*, consulta de histórico de um certo serviço). E a interação *notification* é encontrada na interação entre os Monitores e os Aggregator.

Os dados utilizados para construir eventos são adquiridos de diversas fontes. Sensores em hardware e/ou software que coletam métricas em tempo real (e.g., carga de CPU, memória), de banco de dados, e de outros sistemas de monitoração. O *Producer* pode estar associado a qualquer informação disponível através de eventos GMA, tornando-os acessíveis para os *Consumers* (vide Figura 4.2).

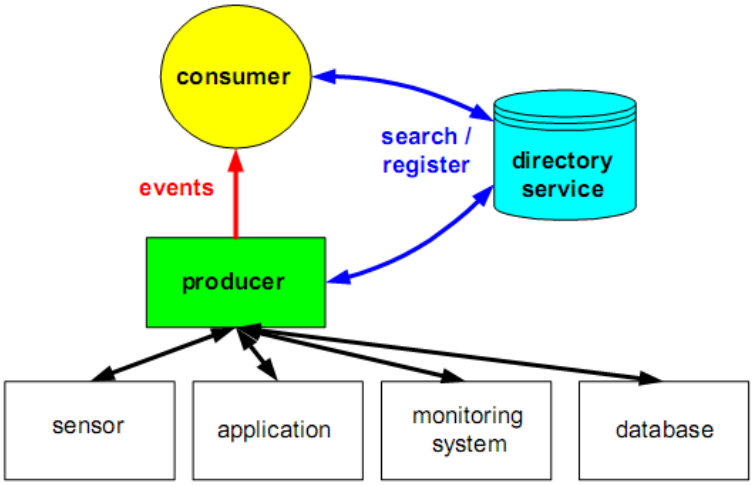

Figura 4.2: Origem dos Dados do GMA [5]

O AutoMan também coleta as métricas de diversas fontes como já discutido no Capitulo 2, no entanto, se poderia portar as interfaces *Consumer* e *Producer* do GMA para o AutoMan. Com essa implementação o AutoMan poderia interagir com outros sistemas de monitoramento baseados na arquitetura GMA como sistema R-GMA [15] do European DataGrid [43].

As interfaces GMA não especificam como ocorrerá a interação entre a fonte de dados e os *Producer*, apenas especifica como enviar eventos para os *Consumers*.

Os *Consumers* podem utilizar as informações recebidas para diferentes propósitos. Alguns exemplos de aplicação de *Consumers* são: um monitor que coleta informações e apresenta em interfaces*web*; um repositório de dados, que armazena as informações para utilização posterior; um sistema recuperador, que pode tomar alguma ação reativa de acordo com o evento recebido.

A Figura 4.3 apresenta um exemplo de como os componentes GMA podem ser utilizados em um cenário típico [5].

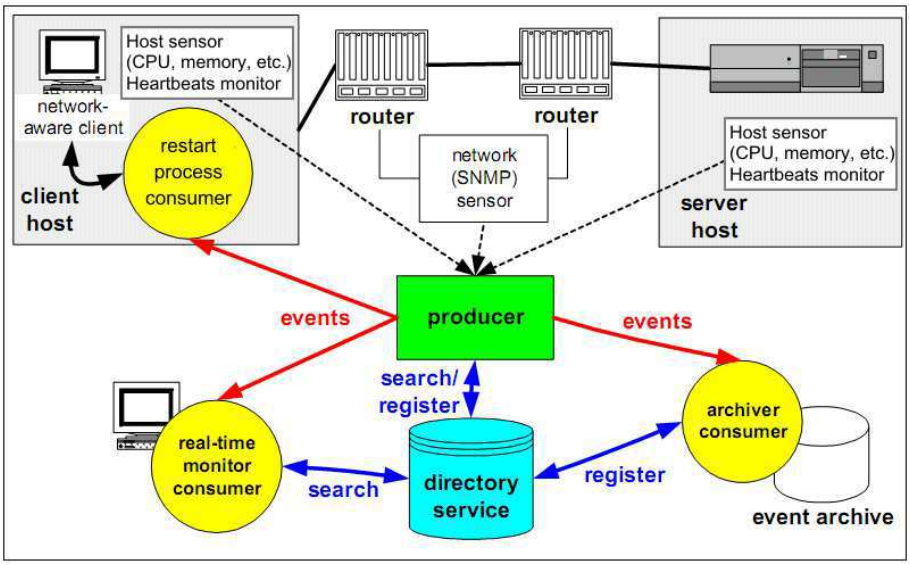

Figura 4.3: Exemplo de uso da Arquitetura GMA [5]

Os dados são coletados em dois *hosts* e nos dois roteadores que interligam as redes. Os sensores de *hosts* e da rede e o monitor de *heartbeats* são geradores dos dados gerenciados por um *Producer*. O *Producer* registra a disponibilidade dos eventos provenientes dos *hosts* e da rede no *Directory Service*. Um *Consumer* que faz a monitoração em tempo real subscreve-se para receber todos os eventos disponíveis, possibilitando a visualização dos eventos em tempo real e a análise dos recursos monitorados. Um *Consumer* que tem a funcionalidade de restabelecer serviços falhos recebe eventos sempre que alguma entidade falhar, independentemente da máquina em que o monitor está executando. O *Producer* também envia os eventos para um repositório (*evente archive*).

Um ponto importante a se observar é que uma boa distribuição dos componentes de um sistema de monitoramento distribuídos aumenta a sua escalabilidade. *Producer* e *Directory Service* adicionais podem ser utilizados para o balanceamento de cargas no caso de muitos consumidores requisitarem o mesmo evento. No AutoMan para aumentar a escalabilidade, os Aggregators podem trocar infor-

mação diminuindo a carga e também foi colocado um Leukocyte em cada máquina que possua um serviço recuperável.

4.1.2 Reporting Grid Service

Reporting Grid Service (ReGS) [4] define um conjunto de serviços da grade para *logging*, *tracing* e monitoração de aplicações em ambientes *Open Grid Services Architecture* (OGSA) [21]. O ReGS é uma arquitetura inspirada no GMA que define as interfaces e o comportamento de dois tipos de *intermediaries* (*i.e.*, componente que pode consumir e prover dados): um componente que realiza a filtragem de informação (*Filtering Component*) e um componente para armazenamento e entrega de mensagens (*Storage and Delivery Component*).

Em um cenário provável, um produtor (*Menssage Producer*) gera dados para um consumidor (*Menssage Consumer*) que poderá utilizar esses dados em algum momento. Na maioria dos casos a quantidade de dados coletados é grande e a informação retirada para uso do sistema é pequena. Sendo assim, é necessário um mecanismo para filtrar os dados gerados e controlar realmente o que deve ser mantido (*Filtering Service*). Além disso, diferentes tipos de dados podem possuir diferentes características de consumo. Alguns dados tem tempo de utilidade predeterminado assim tendo a necessidade de consumi-los antes de atingir este tempo (*i.e.*, dado para monitoração em tempo real), e outros tem que ser armazenados por muito tempo (*i.e.*, dados para auditoria). Então, é necessário um mecanismo que permita a criação de repositórios diferentes (*Baskets*) baseados nas características dos dados armazenados.

No AutoMan, a camada JMX e os manipuladores de métricas dos monitores filtram as informações necessárias para o AutoMan. Porém, o AutoMan possui apenas um tipo de repositório não separando os tipos de métricas quanto as suas características (*i.e.*, *Storage and Delivery Service*). A Figura 4.4 apresenta os componentes relacionados nesse cenário e as suas interações.

Outra característica da arquitetura ReGS é que as mensagens trocadas (equivalente aos eventos GMA) entre os componentes são definidas por um esquema XML. Nesse esquema, determina-se, entre outras coisas, o nível de prioridade da mensagem (*i.e.*, *DEBUG,ERROR, WARN, FATAL*) e o seu *timestamp*.

As mensagens do AutoMan também são padronizadas no formato XML, no entanto elas não possuem prioridade. Essa característica, poderia aumentar o tempo de reação do sistema em certas situações (*e.g.*, qual entidade recuperar primeiro).

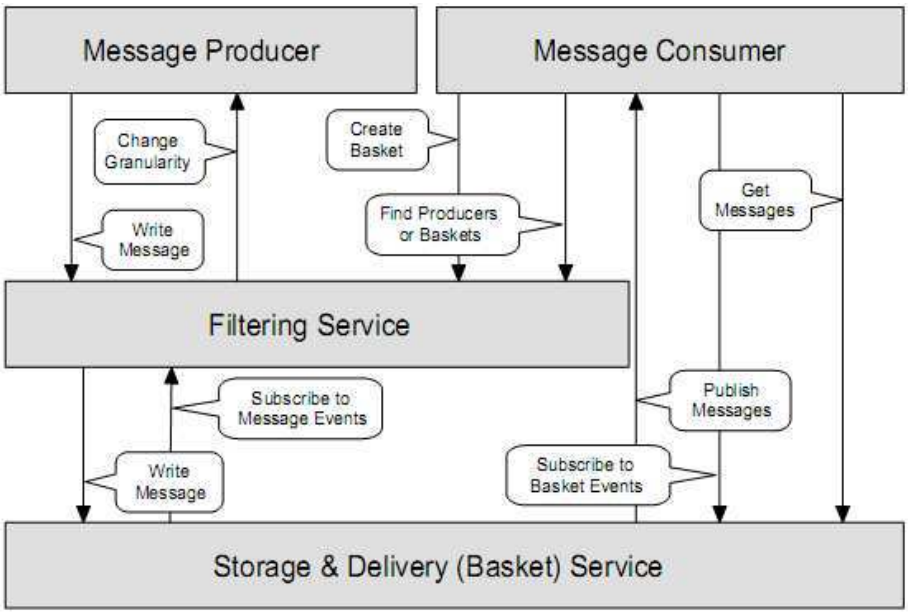

Figura 4.4: Interação dos Componentes ReGS [4]

4.2 Sistemas de Monitoramento

As subseções seguintes discutem alguns sistemas de monitoramento para grades computacionais e sistemas distribuídos.

4.2.1 Reconfiguração Dinâmica de Grades

O processo de reconfiguração faz parte da gerência e monitoração dos sistemas distribuídos. A reconfiguração é necessária na gerência manual e automática das grades porque as entidades podem estar executando em sistemas heterogêneos ou em sistemas administrativos diferentes.

Inúmeras abordagens de configuração de grades [47] são resultados de experimentos feitos em dois grandes projetos de grades: o TeraGrid [44] e o European DataGrid [43]. No European DataGrid, arquivos padrão de configurações são invocados por *scripts*. No TeraGrid, um conjunto de configurações básicas é desenvolvido e deve ser aceito por todos os participantes, e um monitor verifica se as configurações estão corretas. O AutoMan também utiliza arquivos de configurações e *scripts*. Na primeira versão do AutoMan, esses arquivos eram preenchidos pelos administradores, gerando muitos erros de configuração. Porém, na segunda versão, esse problema foi resolvido criando um sistema de autoconfiguração. Também se observaram outras abordagens de configuração automática como SmartFrog [26] e o padrão CDDLM [7] que poderão ser exploradas para avaliar a possibilidade e o ganho obtido ao combiná-las com o AutoMan.

O *framework* SmartFrog foi desenvolvido nos laboratórios da HP Bristol com o intuito de facilitar a instalação e configuração de aplicações em ambientes distribuídos. O SmartFrog apresenta uma linguagem que pode ser compreendida por humanos com o objetivo de descrever a configuração de um sistema e sua distribuição. A fim de se ter um ambiente com o SmartFrog capaz de instalar os componentes, o administrador de sistemas terá que instalar os seus agentes em todas as máquinas. Uma vez que isto esteja feito, o usuário terá que descrever os arquivos contendo as configurações dos componentes a serem instalados e onde eles deverão ser executados.

O padrão CDDLM foi desenvolvido baseado no SmartFrog e provê um conjunto de especificações para automatizar a instalação e gerência no ciclo de vida dos sistemas utilizando *Web Services*. Existem duas diferenças principais entre o CDDLM e o SmartFrog. A primeira é que o CDDLM usa uma linguagem baseada em XML para descrição dos componentes. A segunda é que a infra-estrutura de comunicação do SmartFrog é baseada no protocolo RMI e o CDDLM utiliza o protocolo SOAP [48] para as trocas de mensagens.

4.2.2 Gerência da Rede

A gerência da rede onde a grade está executando também tem recebido atenção da comunidade [42]. Essa gerência é feita comparando as regras de rede com as regras que definem o comportamento normal da grade e onde estão localizados os recursos da mesma. Essa abordagem é relevante e poderá complementar o AutoMan, que em sua versão atual não trata de gerência de rede, mas possui toda a infra-estrutura para a coleta de dados necessária ao seu gerenciamento.

4.2.3 Grid Resource Monitoring

O sistema *Grid Resource Monitoring* (GridRM) [6] é um sistema aberto e distribuído, desenvolvido com o objetivo de monitorar os recursos da grade computacional, não sendo intrusivo. Isto é, a grade computacional hospedeira não necessita sofrer modificações. Porém, ele só faz a monitoração de recursos como, por exemplo, CPU, memória, dispositivos de rede, *storages*, não monitorando os eventos e estados da aplicações como o AutoMan o faz.

O GridRM possui dois níveis hierárquicos, o global e local. O nível global se baseia em GMA, onde cada *gateway* se registra nos serviços de diretórios como *intermediarie* provendo/consumindo informações de/para outros *gateways*. Esta troca de informação entre gateways (*e.g.*, comunidades OurGrid) é feita no AutoMan através da entidade Aggregator. No entanto, a integração com arquitetura GMA pode ser uma opção com resultados interessantes, já que daria a possibilidade de

interação com outros sistemas que utilizem a arquitetura GMA. Em nível local, os *gateways* monitoram os recursos locais através de SNMP [13]. Este tipo de monitoração de recursos locais é feita pelo AutoMan utilizando uma integração com o sistema Ganglia. Outra diferença do sistema GridRM para o AutoMan é que o GridRM não possui mecanismos de recuperação.

4.2.4 GridICE

O GridICE [2] foi implementado em projetos como DataTag [37] e EGEE [24] com o objetivo de facilitar a administração de grades computacionais tendo como foco a monitoração da infra-estrutura. Ele provê informações sobre o status e a utilização da Organizações Virtuais (OVs) e recursos, como também gera estatísticas, históricos dos eventos e alertas em tempo real através de uma interface *web*.

Atualmente o OurGrid possui um WebStatus (vide *http://status.ourgrid.org/*) contendo apenas informações de disponibilidade dos serviços (*i.e.*, o Peer está ativo/inativo, os estados do Worker (*Contact, Idle, In Use, Donated, Owner*)). Poder-se-ia desenvolver uma interface *web* que se comunicaria com o serviço Aggregator, do AutoMan, para adquirir informações em tempo real da grade computacional (*i.e.*, carga do processador e memória utilizada durante a execução de um *job*, estados das máquinas e serviços) como também informações do histórico através da consulta ao banco de dados.

4.2.5 Scalable Sensing Service

O sistema *Scalable Sensing Service* (S3) [50] apresenta um serviço de escalonador para configuração e gerência em tempo real para grandes sistemas de rede. A arquitetura proposta é composta por um conjunto de sensores variados que medem ou monitoram tanto métricas de rede quanto dos nós. A informação obtida via sensores é utilizada como base para agregar dados medidos a partir das máquinas individuais de forma escalável. Este trabalho difere do AutoMan porque ele aborda apenas a monitoração e não inclui mecanismo de gerência automática. Além disso, ele só foca em métricas de máquinas e de rede, não explorando a infra-estrutura e os serviços específicos da grade como faz o AutoMan.

4.2.6 ChinaGrid

Considerando o aspecto monitoração, o trabalho da grade *China Grid* [35] também está relacionado ao AutoMan. Ele apresenta um sistema de monitoração de grade para o ChinaGrid. O objetivo deste sistema é monitorar o desempenho dos *brokers* (máquinas que rodam os *jobs*) e os estados dos

jobs rodando na grade. As informações de desempenho da grade (e.g., uso de memória e *cpu*) estão relacionadas às máquinas da grade, e são obtidas utilizando o sistema *Ganglia* [38]. Além disso, utiliza-se também uma camada com JMX para as trocas de mensagens de monitoração. Quando um *job* é submetido à grade, uma métrica contendo informações sobre esse job é gerada e enviada para os consumidores de dados. Isso é feito utilizando o padrão *Web Services Distributed Management* (WSDM) [41]. O AutoMan também usa JMX e conceitos básicos de serviços que consomem dados providos por monitores (padrão *publish-subscribe* [19]). Atualmente o AutoMan não é baseado em Web Services, futuramente pretende-se migrar o OurGrid e o AutoMan para uma arquitetura orientada a serviço. Outra semelhança entre o China Grid e o AutoMan é o fato de ambos utilizarem o sistema de monitoração Ganglia para obter informações de monitoração que não são específicas da aplicação. Já que não são suficientes as informações sobre as máquinas (obtidas através do Ganglia) e as informações dos *jobs* submetidos à grade, o AutoMan também monitora eventos em serviços específicos da grade. Tais informações são utilizadas para detectar mau funcionamento desses serviços.

4.2.7 HP Self-Aware and Control

O HP *Self-Aware and Control* (SAC) [12], desenvolvido pela Hewllet-Packard^{TM}, é uma arquitetura de detecção de falhas adaptável ao ambiente. Ele consiste em sistema de auto-tratamento (*self-healing*) baseado em Redes *Bayesianas* [46] - desenvolvidas para fornecer uma avaliação probabilística de falha - que são inseridos em serviços arbitrários dando-lhes a capacidade de se automonitorarem e fornecer dados sobre a sua própria "saúde". Esses dados podem ser usados por outros sistemas de monitoração ou pelo próprio serviço, que pode se autocorrigir.

A Figura 4.5 apresenta o modelo lógico do sistema SAC. Os *Sensors* coletam as informações sobre os serviços (da mesma forma como os monitores no AutoMan) e enviam para os *Measures* armazenar (equivalente ao Aggregator do AutoMan). Os *Sensors* podem enviar os dados coletados para qualquer *Measures*. *Evaluators*, por sua vez, processam a informação contida nos *Measures* e retornam uma opinião sobre a "saúde"da entidade com relação as informações consideradas. No último nível, tem-se os componentes da Rede *Bayesiana*, que combinam as opiniões individuais de cada *Evaluator* e retornando um resultado probabilístico sobre a "saúde"do serviço.

O AutoMan não possui nenhum método probabilístico ou de aprendizado como o HP SAC. Porém, uma mudança no Leukocyte - que utiliza uma tabela de casos já conhecidos e que ele pode recuperar - para possuir algum sistema de inteligência artificial como a metodologia de Raciocínio Baseado em Casos (RBC) [1]. Com isto, o AutoMan poderia formar tabelas dinâmicas de falhas e soluções que "evoluiriam"a cada tentativa de solucionar um novo problema comparando as simi-

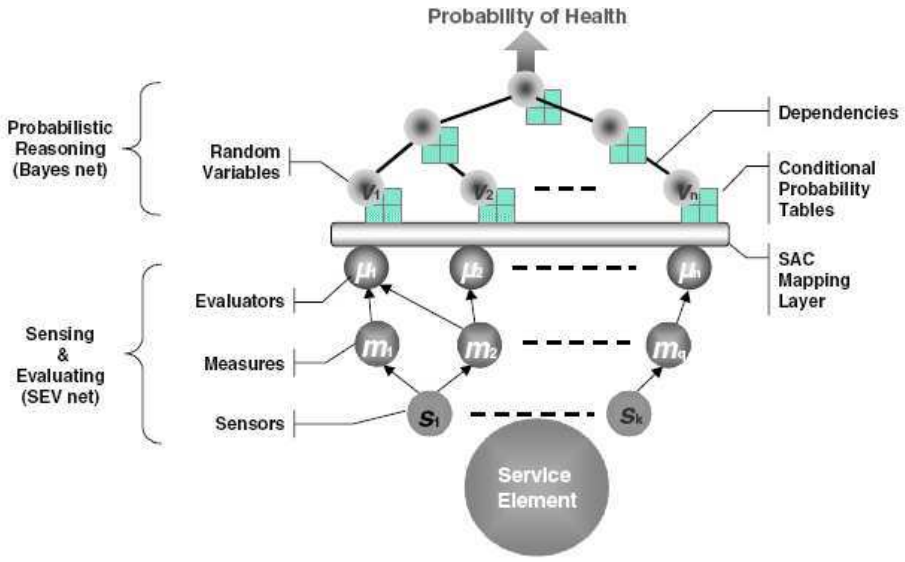

Figura 4.5: HP SAC *Health Engine* [12]

laridades com problemas já conhecidos e adaptando as soluções existentes para solucionar novos problemas.

4.3 Comparação entre Sistemas de Monitoramento

Esta Seção apresenta a Tabela 4.1 que faz um comparativo entre alguns dos sistemas de monitoramento citados na Seção 4.2 em relação aos tipos de recursos monitorados, aos tipos de dados coletados, impacto no sistema monitorado, adaptabilidade e escopo de atuação.

Como apresentado o Network Configuration não possui intrusão porém, ela possui uma baixa adaptabilidade e atua na detecção de falhas e configuração. O GridRM detecta falhas utilizado-se de dados SNMP da rede, deste modo, não tendo nenhuma intrusão. No GridICE monitora recursos e aplicação porém apenas detectando falhas. O S3 monitora a rede através de sensores SNMP implementados provendo uma alta adaptabilidade. No ChinaGrid são monitorados recursos e aplicações através de monitores implementados atuado na detecção de falhas e no escalonamento. O HPSAC monitora e corrige automaticamente as falhas com uma alta adaptabilidade porém, ela possui um alto nível de intrusão. O AutoMan monitora os recursos e aplicações detectando falhas e corrigindo-as quando possível, isto com uma boa adaptabilidade e baixa intrusão.

Sistema	Recurso Monitorado	Dado Coletado	Intrusão	Adaptabilidade	Escopo de atuação
Network Configuration[42]	Rede	Monitores implementados	Sem	Baixa	Detecção de falhas + configuração
GridRM [6]	Recursos + Rede	Dados SNMP	Sem	Sem	Detecção de falhas
GridICE [2]	Recursos + Aplicação (Disponibilidade)	Plugins de monitoramento implementados	Sem	Média	Detecção de falhas
S3 [50]	Rede	Sensores SNMP	Sem	Alta	Detecção de falhas
China Grid [35]	Recursos + Aplicação	Monitores implementados	Média	-	Detecção de falhas + escalonamento
HP SAC [12]	Aplicação	Qualquer métrica	Muito Alta	Muito Alta	Auto Monitoração + Auto Correção
AutoMan	Recursos + Aplicação	Monitores implementados	Baixa	Alta	Detecção de falhas + correção

Tabela 4.1: Comparação entre Sistemas de Monitoramento

Capítulo 5

Conclusões e Trabalhos Futuros

Este trabalho apresentou a arquitetura e a implementação de uma solução, denominada AutoMan, para a gerência automática de grades computacionais. Realizou-se também uma avaliação de desempenho e disponibilidade da grade hospedeira. Além disso, são apresentadas algumas lições decorrentes do processo de implementação e operação do AutoMan.

Observou-se que a gerência manual ou semi-automática atrasa o restabelecimento dos serviços da grade, aumentando assim seu período de indisponibilidade. Isto normalmente acontece porque parte da gerência depende da responsividade de um administrador humano. Conseqüentemente, quando situações adversas acontecem, é possível que ninguém esteja disponível para restabelecer os serviços da grade em um determinado instante, e recursos como ciclos de CPU podem ser desperdiçados.

AutoMan incorpora um sistema de monitoração e gerência que é capaz de resolver uma grande parte dos problemas de indisponibilidade no OurGrid sem introduzir perdas de desempenho consideráveis na grade computacional [11]. AutoMan age tentando restabelecer a operação normal da grade o mais rápido possível.

Com o AutoMan, a disponibilidade da grade computacional OurGrid aumentou significativamente sem acarretar uma degradação no desempenho geral da grade (como foi apresentado no Capítulo 3). Conseqüentemente, necessita-se menos recursos humanos (*i.e.*, administradores), além do que, observa-se melhor qualidade do serviço prestado pela grade sem um aumento no seu custo de manutenção e gerência.

Apesar de se ter aplicado a solução ao OurGrid, ela também pode ser usada em outros sistemas distribuídos, mesmo os que não sejam baseados em grades. Em geral, necessita-se nesses sistemas para manter vários componentes distribuídos e a geração e coleta de métricas sobre o seu funcionamento e ações automáticas de gerência seriam úteis.

5.1 Trabalhos Futuros

Com a evolução do AutoMan V1 para o AutoMan V2 conseguiu-se uma grande melhora principalmente na tolerância a falhas. No entanto, depois de utilizar o AutoMan e executar vários experimentos, identificou como trabalhos futuros algumas mudanças que podem ser feitas para maximizar a funcionalidade e aumentar a tolerância a falhas e escalabilidade dessa ferramenta. Várias destas melhorias foram abordadas no Capítulo 4, porém outras melhorias também foram observadas e devem ser consideradas.

Um trabalho futuro consiste em separar completamente o código de gerência da grade e o código principal do sistema hospedeiro usando técnicas como Programação Orientada a Aspectos [18] para evitar o entrelaçamento e espalhamento do código.

Outro trabalho futuro que aumentaria significativamente a escalabilidade e a tolerância à falhas do AutoMan seria a mudança de paradigma de comunicação do atual RMI para JIC. Com esta mudança o AutoMan se beneficiaria das mesmas vantagens que o OurGrid obteve na versão 4.0 (*e.g.*, mais escalável, flexível) e também do mecanismo de tolerância à falhas que o JIC provê.

Outra melhoria no AutoMan seria na sua arquitetura e na interação entre os seus componentes. O componente Leukocyte do AutoMan além de ser a entidade atuadora seria responsável por coletar os dados monitorados. Esse dados seriam analisados e uma ação seria tomada imediatamente caso necessário, com isso a reação seria mais rápida e a troca de mensagens entre os componentes diminuiria consideravelmente. Os dados recebidos seriam repassados para o Aggregator. Outra característica nova seria a rede de monitoramento entre Leukocytes. Os Leukocytes monitorariam outros Leukocytes seguindo uma ordem de requisitos formando um ciclo. Deste modo, se um Leukocyte falhasse o Leukocyte precursor (responsável pelo seu monitoramento), iria notar a falha e tomaria uma ação reparadora. Com esta rede de monitoramento a tolerância a falhas aumentaria bastante, pois mesmo que todo o sistema de monitoramento e sistema hospedeiro falhasse e apenas um Leukocyte permanecesse ativo, este conseguiria reativar todo o sistema novamente.

Bibliografia

[1] A. Aamodt and E. Plaza. Case-Based Reasoning. *Proc. MLnet Summer School on Machine Learning and Knowledge Acquisition*, pages 1–58, 1994.

[2] C. Aiftimiei, S. Andreozzi, G. Cuscela, N. De Bortoli, G. Donvito, S. Fantinel, E. Fattibene, G. Misurelli, A. Pierro, GL Rubini, et al. GridICE: Requirements, Architecture and Experience of a Monitoring Tool for Grid Systems. *Proceedings of the International Conference on Computing in High Energy and Nuclear Physics (CHEP2006), Mumbai, India*, 13:17, 2006.

[3] N. Andrade, F. Brasileiro, W. Cirne, and M. Mowbray. Discouraging free-riding in a peer-to-peer cpu-sharing grid. *Proceedings of 13th IEEE International Symposium on High-Performance Distributed Computing (HPDC13), Honolulu, Hawaii*, pages 4–9, June 2004.

[4] Y. Aridor, D. Lorenz, B. Rochwerger, B. Horn, and H. Salem. Reporting Grid Services (ReGS) Specification. *IBM Haifa Research Lab*, 2003.

[5] R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, B. Tierney, and R. Wolski. A Grid Monitoring Architecture. *Global Grid Forum/GMA Working Group Document (http://www.ogf.org/documents/GFD.7.pdf)*, 03 2002.

[6] M. Baker and G. Smith. GridRM: an extensible resource monitoring system. *Cluster Computing, 2003. Proceedings. 2003 IEEE International Conference on*, pages 207–214, 2003.

[7] D. Bell, T. Kojo, P. Goldsack, S. Loughran, D. Milojicic, S. Schaefer, J. Tatemura, and P. Toft. Configuration Description, Deployment, and Lifecycle Management (CDDLM). *GGF, Foundation Document*, 2004.

[8] F. Berman, G. Fox, and A.J.G. Hey. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley, 2003.

[9] F. Brasileiro, E. Araujo, W. Voorsluys, M. Oliveira, and F. Figueiredo. Bridging the High Performance Computing Gap: the OurGrid Experience. *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*, pages 817–822, 2007.

[10] T. Bray, J. Paoli, C.M. Sperberg-McQueen, et al. Extensible Markup Language (XML) 1.0. *W3C Recommendation*, 6, 2000.

[11] C. Brennand, M. Spohn, A. Coelho, A. Dantas, F. Brasileiro, G. Pereira, D. Candeia, G. Germoglio, and F. Santos. AutoMan: Gerência Automática no OurGrid. *Brazilian Workshop on Grid Computing and Applications. 25º Simpósio Brasileiro de Redes de Computadores*, 2007.

[12] A. Bronstein, J. Das, M. Duro, R. Friedrich, G. Kleyner, M. Mueller, S. Singhal, and I. Cohen. Self-aware services: using Bayesian networks for detecting anomalies in Internet-based services. *Integrated Network Management Proceedings, 2001 IEEE/IFIP International Symposium on*, pages 623–638, 2001.

[13] JD Case, M. Fedor, ML Schoffstall, and J. Davin. RFC1157: Simple Network Management Protocol (SNMP). *Internet RFCs*, 1990.

[14] W. Cirne, F. Brasileiro, N. Andrade, L. Costa, A. Andrade, R. Novaes, and M. Mowbray. Labs of the World, Unite. *Journal of Grid Computing*, 4(3):225–246, September 2006.

[15] A. Cooke, A.J.G. Gray, L. Ma, W. Nutt, J. Magowan, M. Oevers, P. Taylor, R. Byrom, L. Field, S. Hicks, et al. R-GMA: An information integration system for Grid monitoring. *Lecture notes in computer science*, pages 462–481.

[16] D. Curtis. White Paper: Java, RMI and CORBA. *Object Management Group, January*, 1997.

[17] D.P. da Silva, W. Cirne, and F.V. Brasileiro. Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids. *Euro-Par 2003 Parallel Processing: 9th International Euro-Par Conference, Klagenfurt, Austria, August 26-29, 2003: Proceedings*, 2003.

[18] T. Elrad, R.E. Filman, and A. Bader. Aspect-oriented programming: Introduction. *Communications of the ACM*, 44(10):29–32, 2001.

[19] P.T.H. Eugster, P.A. Felber, R. Guerraoui, and A.M. Kermarrec. The Many Faces of Publish/-Subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[20] Open Grid Forum. Open Grid Forum Web Page. *http://www.ggf.org/*, 2008.

[21] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Open Grid Service Infrastructure WG, Global Grid Forum, June*, 22:2002, 2002.

[22] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Int'l J. Supercomputer Applications*, 15(3), 2001.

[23] XMPP Standards Foundation. Extensible Messaging and Presence Protocol (XMPP). *http://www.xmpp.org/*, 2008.

[24] F. Gagliardi, B. Jones, F. Grey, M.E. Bégin, and M. Heikkurinen. Building an infrastructure for scientific Grid computing: status and goals of the EGEE project. *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, 363(1833):1729–1742, 2005.

[25] E. Galstad. Nagios version 2.0 documentation, 09 2004.

[26] P. Goldsack, J. Guijarro, A. Lain, G. Mecheneau, P. Murray, and P. Toft. SmartFrog: Configuration and Automatic Ignition of Distributed Applications. *HP Openview University Association conference*, 2003.

[27] J. Gosling. *The Java Language Specification*. Addison-Wesley Professional, 2000.

[28] Rick Grehan. Embedding the db4o Object-Oriented Database. *Linux J.*, 2006(142):5, 2006.

[29] D. Gunter and J. Magowan. Discovery and Monitoring Event Descriptions WG. *Global Grid Forum/DAMED Working Group Document (http://www-didc.lbl.gov/damed/documents/GFD.025.pdf)*, 01 2004.

[30] D. Hughes, G. Coulson, and J. Walkerdine. Free riding on Gnutella revisited: the bell tolls? *Distributed Systems Online, IEEE*, 6(6), 2005.

[31] R. Jain. *The Art of Computer Systems Performance Analysis*. Wiley, 1991.

[32] J. Joyce, G. Lomow, K. Slind, and B. Unger. Monitoring distributed systems. *ACM Transactions on Computer Systems (TOCS)*, 5(2):121–150, 1987.

[33] CERN Large Hadron Collider (LHC). LHC Computing Grid Project. *http://lcg.web.cern.ch/LCG/*, 2008.

[34] A. Lima, W. Cirne, F. Brasileiro, and D. Fireman. A Case for Event-Driven Distributed Objects. *OTM Confederated International Conferences, CoopIS, DOA, GADA, and ODBASE*, pages 1705–1721, November 2006.

[35] R. Liu, W. M. Zheng, Y. Chen, K. Farkas, and D. Milojicic. Automating the Access to Monitoring Data in ChinaGrid. *13th HP OVUA*, 2006.

[36] M. Mansouri-Samani and M. Sloman. Monitoring distributed systems. *Network, IEEE*, 7(6):20–30, 1993.

[37] J.P. Martin-Flatin and P.V.B. Primet. High-speed networks and services for data-intensive Grids: The DataTAG Project. *Future Generation Computer Systems*, 21(4):439–442, 2005.

[38] M.L. Massie, B.N. Chun, and D.E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, 30(7):817–840, 2004.

[39] R. Medeiros, W. Cirne, F. Brasileiro, and J. Sauve. Faults in grids: why are they so bad and what can be done about it? *Grid Computing, 2003. Proceedings. Fourth International Workshop on*, pages 18–24, 2003.

[40] S. Microsystems. JMX white paper, 1999.

[41] Stephen Morris. The Web Services Distributed Management (WSDM) Standard, 02 2006.

[42] R. Neisse, L.Z. Granville, M.J.B. Almeida, and L.M.R. Tarouco. On Translating Grid Requirements to Network Configurations through Policy-Based Management. *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 350–354, 2004.

[43] The DataGrid Project. EU DataGrid Project. *http://eu-datagrid.web.cern.ch/eu-datagrid/*, 2008.

[44] D.A. Reed. Grids, the TeraGrid, and Beyond. *Computer*, 36(1):62–68, 2003.

[45] E. Santos-Neto, W. Cirne, F. Brasileiro, and A. Lima. Exploiting Replication and Data Reuse to Efficiently Schedule Data-Intensive Applications on Grids. *Job Scheduling Strategies for Parallel Processing: 10th International Workshop, JSSPP 2004, New York, NY, USA, June 13, 2004: Revised Selected Papers*, 2005.

[46] C. Skaanning, F.V. Jensen, U. Kjærulff, P. Pelletier, and L. Rostrup-Jensen. Printing system diagnosis: A Bayesian network application. *Workshop on Principles of Diagnosis, Cape God, Massachussets*, 1998.

[47] E. Smith and P. Anderson. Dynamic Reconfiguration for Grid Fabrics. *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 86–93, 2004.

[48] Erl. Thomas. Introduction to Web Services Technologies: SOA, SOAP, WSDL and UDDI, 09 2004.

[49] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, 15(5-6):757–768, 1999.

[50] P. Yalagandula, P. Sharma, S. Banerjee, S. Basu, and S.J. Lee. S3: A Scalable Sensing Service for Monitoring Large Networked Systems. *Proceedings of the 2006 SIGCOMM workshop on Internet network management*, pages 71–76, 2006.

[51] S. Zanikolas and R. Sakellariou. A taxonomy of grid monitoring systems. *Future Generation Computer Systems*, 21(1):163–188, 2005.

Apêndice A

Tabelas das Métricas Geradas pelos Monitores

A seguir são apresentadas as tabelas contendo todas as métricas geradas pelos Monitores com suas respectivas descrições.

Tabela A.1: Métricas geradas pelo PeerMonitor

Nome da métrica	Valor	Descrição do Valor	Descrição da Métrica
Peer.WorkerRequest	REQUEST_ID	Id da requisição de um Worker	Peer requisita Workers para a execução do job
	REQUEST_IS_LOCAL	Requisição foi feita de um Mygrid da comunidade ou de outro Peer	
	REQUEST_NUMBER_ OF_MACHINES	Número de máquinas requisitadas	
	REQUEST_REQUIREMENTS	Atributos da requisição	
	REQUEST_CLIENT_NAME	Nome do requisitante (Peer de origem)	
	REQUEST_CLIENT_LOCATION	Endereço de rede	
	REQUEST_SOURCE	MyGrid que originou a requisição	
Peer.WorkerReceived	WORKER_NAME	Nome do Worker	Worker recebido por um Peer (que foi descrito no job submetido pelo MyGrid)
	WORKER_LOCATION	Endereço do Worker	
	PEER_NAME	Nome do Peer requisitante	
	PEER_LOCATION	Endereço do Peer requisitante	

Tabela A.1: Métricas geradas pelo PeerMonitor

	WORKER_RECEIVED_ WORKER_OWNER_NAME	Nome do Peer o qual Worker faz parte	
	WORKER_RECEIVED_ WORKER_OWNER_LOCATION	Endereço do Peer o qual Worker faz parte	
	REQUEST_ID	Id do *job* que vai ser executado	
Peer.Accounting	ACCOUNTING_PEER_NAME	Nome do Peer	Recebe o balanço de favores da NoF de um Peer
	ACCOUNTING_PEER_ LOCATION	Endereço do Peer	
	ACCOUNTING_PEER_ BALANCE	Balanço de favores do Peer	
Peer.WorkerDelivered	WORKER_NAME	Nome do Worker	Worker devolvido pelo MyGrid
	WORKER_LOCATION	Endereço do Worker	
	PEER_NAME	Nome do Peer requisitante	
	PEER_LOCATION	Endereço do Peer requisitante	
	WORKER_DELIVERED_ CLIENTE_NAME	Nome do Peer o qual Worker faz parte	
	WORKER_DELIVERED_ CLIENTE_LOCATION	Endereço do Peer o qual Worker faz parte	
	REQUEST_ID	Id do *job* que vai ser executado	
Peer.WorkerStatusChange	WORKER_NAME	Nome do Worker	Mudança do estado do Worker (ativo/inativo)
	WORKER_LOCATION	Endereço do Worker	
	WORKER_STATUS_CHANGE_ WORKER_STATE_NAME	Estado para qual o Worker mudou (*Contact, Idle, In Use, Donated, Owner*)	

Nome da métrica	**Valor**	**Descrição do Valor**	**Descrição da Métrica**
DiscoveryService.PeerUp	PEER_NAME	Nome do Peer	Peer ativo
DiscoveryService.PeerDown	PEER_NAME	Nome do Peer	Peer inativo
DiscoveryService.DiscoveryServiceUp	DISCOVERYSERVICE_NAME	Nome do DiscoveryService	DiscoveryService ativo
DiscoveryService.DiscoveryServiceDown	DISCOVERYSERVICE_NAME	Nome do DiscoveryService	DiscoveryService inativo

Tabela A.2: Métricas geradas pelo DiscoveryserviceMonitor

Nome da métrica	Valor	Descrição do Valor	Descrição da Métrica
Worker.ReplicaStarted	REP_STARTED_REMOTE_COMMAND	Comando executado	Inicio da execução do job na réplica
Worker.ReplicaFinished	REP_FINISHED_EXIT_VALUE	Valor de saída ao término	Termino da execução do job na réplica
	REP_FINISHED_STD_OUT	Valor de saída padrão ao término	
	REP_FINISHED_ERR_OUT	Valor de saída de erro padrão ao término	
Worker.TempDirSizeReport	TEMP_DIR_SIZE	Tamanho do /tmp	Espaço livre no /tmp da máquina
	TEMP_FREE_SPACE	Espaço livre no /tmp	
	PLAYPEN_ROOT	Endereço do /tmp	
	WORKER_NAME	Nome do Worker	
Worker.StorageDirSizeReport	STORAGE_DIR_SIZE	Tamanho do $storage	Espaço livre no $*storage* da máquina
	STORAGE_FREE_SPACE	Espaço livre no $*storage*	
	STORAGE_DIR	Endereço do $*storage*	
	WORKER_NAME	Nome do Worker	

Tabela A.3: Métricas geradas pelo WorkerMonitor