UNIVERSIDADE DO VALE DO RIO DOS SINOS
PROGRAMA INTERDISCIPLINAR DE PÓS-GRADUAÇÃO EM
COMPUTAÇÃO APLICADA

AGrADC: Uma Arquitetura para Implantação e Configuração Autônomas de Aplicações em Grades Computacionais

por

SIDNEI ROBERTO SELZLER FRANCO

Dissertação submetida à avaliação como requisito parcial para a obtenção do grau de Mestre em Computação Aplicada

Orientador: Prof Dr. Marinho Pilla Barcellos
Co-orientador: Prof Dr. Luciano Paschoal Gaspary

São Leopoldo, Fevereiro de 2007

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Franco, Sidnei Roberto Selzler

AGrADC: Uma Arquitetura para Implantação e Configuração Autônomas de Aplicações em Grades Computacionais / por Sidnei Roberto Selzler Franco. — São Leopoldo: Ciências Exatas e Tecnologicas da UNISINOS, 2007.

75 f.: il.

Dissertação (mestrado) — Universidade do Vale do Rio dos Sinos. Ciências Exatas e Tecnológicas Programa Interdisciplinar de Pós-Graduação em Computação Aplicada, São Leopoldo, BR–RS, 2007. Orientador: Barcellos, Marinho Pilla; Co-orientador: Gaspary, Luciano Paschoal.

1. Grades Computacionais. 2. Computação Autônoma. 3. Auto-gerenciamento. I. Barcellos, Marinho Pilla. II. Gaspary, Luciano Paschoal. III. Título.

UNIVERSIDADE DO VALE DO RIO DOS SINOS
Reitor: Pe. Dr. Marcelo Fernandes de Aquino
Diretora da Unidade de Pós-Graduação e Pesquisa: Profª. Drª. Ione Bentz
Coordenador do PIPCA: Prof. Dr. Arthur Tórgo Gómez

Não me procures onde não queres que eu esteja.

Agradecimentos

Por mais que qualquer trabalho, nesse caso dissertação, seja difícil, sempre deixamos para escrever os agradecimentos no final. Na minha opinião, essa é a parte mais difícil. Isso porque durante todo o percurso foram várias as pessoas que ajudaram e contribuíram para a realização deste trabalho. Eis a questão, como quantificar e utilizar as palavras corretas para agradecer cada uma dessas pessoas?

Eu mereceria ser condenado à morte se não falasse da importância da minha família e mais, eu mereceria a forca se não sentisse uma profunda gratidão para com eles. Se eu contasse, acho que nesses dois anos, foram raros os dias que não falei com eles, principalmente a minha mãe. Posso garantir, com toda a certeza, que levo duas vidas, uma delas a cada instante do meu dia e outra quando fico sabendo das notícias da minha família. Agradeço MUITO a minha mãe (Leonida), meu pai (Celso), meus irmãos (Sandra e Silmar) e aos meus sobrinhos (Gabriela e Gabriel). Muito obrigado de coração!!! Podem até existir, mas não conheço família tão presente e atuante, que me deu suporte para tudo, sempre apoiando, ajudando e querendo ajudar mais ainda. Vocês são muito importantes na minha vida e fundamentais para que mais esse obstáculo fosse vencido. AMO MUITO VOCÊS!!

Agradecimentos a minha namorada, Fernanda, pela compreensão, afeto, dedicação e amor que tenho recebido desde que começamos a namorar. Você é muito especial para mim, AMO VOCÊ.

Agradeço a minha Nona (Dona Anélia), Padrinhos (Osi e Lurdes), e aos tios, tias e primos. Agradeço também aos meus amigos que sempre estiveram ou tentaram estar por perto ajudando.

Agradeço aos meus colegas do mestrado, obrigado pelas aulas, pelo companheirismo, pelos dias nos laboratórios e pelas conversas no café. Obrigado aos amigos Igor, Marlom, Juliano, Lucas, César, Etiene, Scheila e Glauco. Obrigado especial aos amigos Paulo, Otávio e Tiago pela amizade, companheirismo, jantares e brincadeiras.

É hora de agradecer aos orientadores, que no meu caso são três. Vou começar

agradecendo do último para o primeiro. Ao meu amigo e orientador 3, Prof. Marinho Barcellos, muito obrigado por se dispor a orientar um orientando desgovernado. Obrigado pelas aulas, conselhos, ensinamentos e paciência. Ao amigo e orientador 2 Prof. Gerson Cavalheiro, muito obrigado também pelas aulas, pelos conselhos e ensinamentos. E finalmente, ao meu amigo e orientador 1, Prof. Luciano Gaspary pelo constante incentivo, dedicação e companheirismo. Mesmo não tendo obrigação de continuar ao meu lado, ele sempre esteve indicando a direção a ser tomada nos momentos de maior dificuldade. Nos momentos que as minhas forças falhavam e eu parecia esmorecer, ele ajudava a reunir forças e ver uma luz no final do túnel. Agradeço pela confiança depositada no meu trabalho e, bem lá no início, na minha pessoa.

Sou grato também a Hewlett-Packard Brasil pela bolsa integral que viabilizou a realização do mestrado com dedicação exclusiva. Valeu muito pela oportunidade oferecida.

Agradeço principalmente a Deus por ter me dado forças e fé para vencer os obstáculos e chegar até aqui.

Desculpa a todos pela minha ausência, pelos convites recusados, pelos convites não feitos. Desculpa por não estar presente e não ter ajudado da forma que deveria e que eu gostaria.

Resumo

A implantação e a configuração de aplicações em grades computacionais são tarefas exaustivas e sujeitas a erros, ainda representando elo fraco do ciclo de vida de aplicações desta natureza. Para lidar com o problema, este trabalho propõe AGrADC, uma arquitetura para instanciação sob demanda de aplicações em grades que incorpora características da Computação Autônoma. Esta arquitetura instrumenta o processo de desenvolvimento de aplicações para grades computacionais, oferecendo ferramentas para definir (a) um fluxo de implantação, respeitando dependências entre componentes que compõem a aplicação, (b) parâmetros de configuração e (c) ações a serem executadas diante de situações adversas tais como falhas. O resultado desse processo, materializado na forma de um conjunto de descrições, é repassado a um motor de instanciação, que passa a autonomamente conduzir e gerenciar o processo de implantação e configuração.

Palavras-chave: Grades Computacionais, Computação Autônoma, Auto-gerenciamento.

TITLE: "AGrADC : An Architecture for Deployment and Autonomic Configuration of the Aplications in Grid Computing"

Abstract

Deployment and configuration of grid computing applications are exhaustive and error-prone tasks, representing a weak link of the lifecycle of grid applications. To address the problem, this work proposes AGrADC, an architecture to instantiate grid applications on demand, which incorporates features from the Autonomic Computing paradigm. This architecture improves the grid applications development process, providing tools to define (a) a deployment flow, respecting dependencies among components that comprise the application, (b) configuration parameters and (c) actions to be executed when adverse situations like faults arise. The result of this process, materialized in the form of a set of descriptions, is delivered to an instantiation engine, which starts to autonomously conduct and manage the deployment and configuration process.

Keywords: grid computing, autonomic computing, self-management.

Sumário

Lista de Abreviaturas

CDDLM	Configuration Description, Deployment, and Lifecycle Management
VO	Virtual Organization
API	Application Programming Interface
SDK	Software Development Kit
GGF	Global Grid Forum
OGSA	Open Grid Services Architecture
OASIS	Organization for the Advancement of Structured Information Standards
OGSI	Open Grid Services Infrastructure
GT	Globus Toolkit
XML	eXtensible Markup Language
WSRF	Web Services Resource Framework
GRAM	Globus Resource Allocation Manager
GSI	Grid Security Infrastructure
MDS	Monitoring and Discovery Service
FTP	File Transfer Protocol
TCP	Transmission Control Protocol
BoT	Bag-of-Tasks
NOW	Network of Workstations
WSDM	Web Services Distributed Management
AIDE	Autonomic Integrated Development Environment

LCFG	Local ConFiGuration system
SmartFrog	Smart Framework for Object Groups
LGPL	Lesser General Public License
CDL	Configuration Description Language
SOAP	Simple Object Access Protocol
ROST	Remote & hOt Service deployment with Trustworthiness
TNA	Trust Negotiation Agent
RHD	Remote Hot Deployment
SCC	Service Container Configuration
AGS	Adaptive Grid Service
AGSI	Adaptive Grid Service Instance
ARP	Adaptative Resource Provider
HAND	Highly Available DyNamic Deployment Infrastructure
DDC	Dynamic Deployer Core
JVM	Java Virtual Machine
DAM	Deploy Approach Manager
WSDD	Web Service Deployment Descriptor
WSDL	Web Services Description Language
SPM	Service Package Manager
EPR	End-Point Reference
RBAC	Role-Based Access Control

Lista de Figuras

Capítulo 1

Introdução

O uso de grades computacionais tem se expandido de forma gradual e consistente nos últimos anos [Simmons and Lutfiyya, 2005]. As infra-estruturas de grades computacionais são construídas a partir do compartilhamento de recursos distribuídos. Dentre seus objetivos, incluem-se: (a) reduzir a necessidade de atualização de equipamentos aproveitando computadores geograficamente dispersos; (b) fornecer desempenho superior na solução de problemas que exigem processamento intensivo; (c) tirar melhor proveito de recursos explorando capacidade ociosa [Kesselman and Foster, 1998]; (d) prover transparência no uso de recursos, considerando-os como um único e poderoso computador; (e) oferecer disponibilidade de acesso apenas a recursos específicos de um nodo [Nemeth and Sunderam, 2002]. Observa-se que com a evolução das tecnologias de grade, elas incorporam novas funcionalidades e melhoram os serviços oferecidos. Entre os avanços perseguidos, destaca-se a necessidade de automatizar tarefas associadas ao *middleware* de grades, possibilitando a implantação, a configuração e o gerenciamento de recursos e serviços de forma facilitada.

Executar uma aplicação de larga escala – com grande número de recursos e serviços distribuídos – requer que o ambiente seja devidamente implantado e configurado, atendendo, assim, às necessidades da aplicação. Esta tarefa tende a ser exaustiva e sujeita a erros, se apresentando como o elo fraco do ciclo de vida do processo de desenvolvimento de aplicações desta natureza. O problema é agravado pelo desejo de se executar aplicações cada vez mais específicas, que exigem a implantação e a configuração de arcabouços bastante peculiares. Nitidamente, a abordagem manual para realizar tais processos não é suficiente.

Para lidar com o problema, este trabalho propõe AGrADC (*Autonomic Grid Application Deployment & Configuration Architecure*), uma arquitetura para

instanciação de aplicações de grade permitindo que a infra-estrutura necessária para sua execução seja implantada, configurada e gerenciada sob demanda. O objetivo é propiciar que o arcabouço de software necessário para a execução de uma aplicação de grade seja instanciado no momento de sua invocação. Na arquitetura proposta, as ferramentas disponibilizadas permitem ao desenvolvedor definir (a) um fluxo de implantação, respeitando dependências entre componentes que compõem a aplicação, (b) parâmetros de configuração e (c) ações a serem executadas diante de situações adversas tais como falhas. O resultado desse processo, materializado na forma de um conjunto de documentos, é repassado a um *motor de instanciação*, que passa a autonomamente conduzir e gerenciar o processo de implantação e configuração.

O presente trabalho possui duas contribuições principais. Primeiro, definiu-se um serviço *inédito* para conduzir o processo de instanciação de aplicações em grades – o motor de instanciação – que incorpora características da Computação Autônoma, sendo capaz de executar procedimentos de contorno para lidar com problemas enfrentados ao longo do processo. Segundo, projetou-se e desenvolveu-se uma arquitetura de software plenamente alinhada com especificação proposta para a área, a CDDLM (*Configuration Description, Deployment, and Lifecycle Management*) [Bell et al., 2005], na qual o referido serviço foi acomodado.

O restante da dissertação está organizado da seguinte forma. O Capítulo 2 revisa o referencial teórico, incluindo esforços referentes à computação em grade, ao paradigma de computação autônoma e à especificação CDDLM. No Capítulo 3 são abordados os trabalhos relacionados, enquanto no Capítulo 4 é apresentada uma visão conceitual da AGrADC e são descritos os elementos que a compõem. Como prova de conceito, o Capítulo 5 aborda o protótipo desenvolvido. No Capítulo 6 é apresentada uma avaliação experimental da arquitetura. A dissertação é encerrada no Capítulo 7 com considerações finais e perspectivas de trabalhos futuros.

Capítulo 2

Referencial Teórico

Este capítulo revisa conceitos importantes e que constituem a base teórica desta dissertação. Inicialmente são descritas as grades computacionais, incluindo seu histórico e conceitos, estrutura básica e principais tecnologias, tendo como foco principal o Globus *toolkit*. Na Seção 2.2 são apresentados os principais conceitos referentes ao paradigma de Computação Autonôma, introduzindo sua arquitetura básica, seus componentes e tecnologias. A Seção 2.3 apresenta a especificação CDDLM que define mecanismos para encapsular informações de uma aplicação, permitindo implantar, configurar, e gerenciar o seu ciclo de vida em um ambiente de grades computacionais. Finalmente, na Seção 2.4 é apresentado um sumário do capítulo, tecendo considerações relacionadas às grades computacionais, à computação autônoma e à especificação CDDLM.

2.1 Grades Computacionais

Esta seção apresenta conceitos, características e requisitos das grades computacionais. Inicialmente é apresentado um breve histórico desta área de pesquisa, citando possíveis definições para o termo. Na seqüência (Sub-seção 2.1.2 é apresentada a estrutura básica de um sistema de grades computacionais. Na sub-seção seguinte (2.1.3) são listadas tecnologias de grade, onde o foco principal é a tecnologia de grades Globus, utilizada neste trabalho.

2.1.1 Histórico

A terminologia Grade Computacional (*Grid Computing*) surgiu como uma metáfora entre "Rede Elétrica" (*Electric Grid*) e "Grade Computacional"

(*Computational Grid*) [Kesselman and Foster, 1998]. A rede elétrica disponibiliza energia elétrica sob demanda, sendo esta a idéia das grades computacionais: possibilitar que os usuários simplesmente se conectem à grade e façam uso do poder computacional disponível (ex: capacidade de processamento e armazenamento, softwares e periféricos). Esse poder computacional é integrado e disponibilizado usando a rede mundial de computadores, a *Internet*.

Embora o termo *grade computacional* possua várias definições, de forma geral o propósito de uma grade é permitir o compartilhamento de recursos computacionais heterogêneos e geograficamente dispersos. De acordo com Kesselman [Kesselman and Foster, 1998], as grades computacionais foram pensadas na metade da década de 90 como uma infra-estrutura computacional distribuída para engenharias e ciências avançadas, fornecendo grande poder computacional à execução de aplicações dessas áreas. Krauter [Krauter et al., 2002] define grade computacional como sendo um sistema de rede de larga escala, tendo máquinas distribuídas através de múltiplas organizações e domínios administrativos.

Mediante a implantação de sistemas que dêem suporte ao escalonamento e ao gerenciamento de tarefas, um ambiente de grade faz uso da infra-estrutura de recursos disponíveis. As principais características das grades são [Baker et al., 2000, Cirne and Santos-Neto, 2005]:

- heterogeneidade: uma grade normalmente envolve uma diversidade de recursos heterogêneos, que podem estar dispersos por diversas plataformas e arquiteturas computacionais, bem como por diversos domínios administrativos;
- escalabilidade: uma grade pode crescer de poucos para milhões de recursos compartilhados, agregando serviços localizados em vários domínios, sem perda significativa de desempenho;
- compartilhamento: uma grade computacional não pode ser dedicada a uma aplicação de forma exclusiva por um determinado período de tempo;
- controle distribuído: tipicamente não há uma única entidade que tenha poder sobre toda a grade, ou seja, cada instituição implementa as políticas de uso dos recursos locais e não interfere na implementação feita por outras instituições participantes;
- dinamicidade ou adaptabilidade: é preciso considerar que a probabilidade de que recursos venham a falhar seja alta; assim, os escalonadores de recursos

e aplicações devem ser projetados para adaptarem o seu comportamento dinamicamente, objetivando extrair o máximo de desempenho dos recursos e serviços disponíveis.

Na seqüência é apresentada uma estrutura básica para a formação de grades, em que a interoperabilidade é considerada questão-chave.

2.1.2 Estrutura Básica

De forma geral um sistema de grades computacionais tem o objetivo de fazer uso de recursos ociosos (ex: capacidade de processamento e armazenamento), estejam eles numa mesma rede local, num mesmo domínio administrativo ou até mesmo em domínios administrativos diferentes. Um grupo formado pela união desses recursos ociosos para um determinado objetivo é chamado de Organização Virtual (*Virtual Organization* – VO) [Foster et al., 2001]. Independentemente da solução de grade que está sendo usada, seja ela resultado de um esforço acadêmico (ex: Globus, Legion, Condor, OurGrid) ou comercial (ex: Entropia, distributed.net), é possível identificar alguns componentes (serviços) que formam uma estrutura básica para a execução de computação em grades.

Existem muitas questões associadas ao termo *Virtual Organization*, como, por exemplo, segurança, alocação e contabilização de recursos, gerenciamento de dados e comunicação. Os esforços acadêmicos têm sido maiores na alocação de recursos e no gerenciamento dos dados; por sua vez, entre os esforços comerciais destacam-se as questões inerentes à segurança e à contabilização do uso de recursos. A contabilização é muito importante na solução comercial para associar valor aos recursos, ou seja, quantificá-los em moeda corrente.

Uma arquitetura de grade inicia com a perspectiva de que seja possível efetuar operações em VOs formadas por vários domínios administrativos, possibilitando o estabelecimento de relações de compartilhamento entre quaisquer potenciais participantes. Para que seja possível estabelecer, gerenciar e explorar dinamicamente o relacionamento dentro das VOs, é necessário identificar os principais componentes, seus objetivos e funções, e indicar como eles podem interagir uns com os outros. Dessa forma fica evidente a grande necessidade de interoperabilidade para as grades computacionais [Foster et al., 2001].

Entre as características de uma arquitetura de grades, é possível identificar e definir, primeiramente e com maior relevância, Protocolos (*Protocols*) e Serviços (*Services*) e, posteriormente, APIs (*Application Programming Interfaces*) e SDKs

(*Software Development Kits*) [Foster et al., 2001]. Essas características são comentadas a seguir.

- Protocolo: a definição de um protocolo especifica como um elemento de um sistema distribuído interage com outro para atingir um comportamento específico. Um protocolo também especifica a estrutura das informações trocadas durante essa interação. Os protocolos são os responsáveis por coordenar as interações entre componentes e não pela sua implementação. Ele é o mecanismo básico pelo qual usuários e VOs negociam e descobrem recursos, estabelecem, gerenciam e exploram as relações de compartilhamento. É importante que cada domínio administrativo dentro de uma VO tenha a possibilidade de estabelecer suas próprias políticas relacionadas aos seus recursos, sendo necessário ter isso especificado no protocolo. Um padrão aberto de arquitetura e protocolo facilita a extensibilidade, a interopabilidade, a portabilidade, o compartilhamento de código e a definição de interfaces de serviços.

- Serviços: um serviço pode ser definido como um protocolo padrão que responde e se comporta de acordo com sua implementação (*serviço = protocolo + comportamento*). A definição de padrões de serviços permite descobri-los, escalonar tarefas, fazer uso de computação, acessar dados ou fazer replicação destes e mais um leque de possibilidades. Isso permite que serviços sejam oferecidos para VOs, abstraindo detalhes específicos de implementação dos serviços, sistema operacional e arquitetura.

- APIs (*Application Programming Interfaces*) e SDKs (*Software Development Kits*): desenvolvedores devem ser capazes de implementar aplicações sofisticadas em ambientes complexos e dinâmicos, fornecendo facilidades para que o usuário seja capaz de operá-las. Padrões de abstração, APIs e SDKs aceleram o desenvolvimento de código e habilitam o seu compartilhamento. Cabe salientar que sem padrões de protocolo, a interoperabilidade é alcançada somente em nível de API, através de uma mesma implementação, tornando assim a solução não interessante para VOs.

Em função dessas características, a natureza de uma arquitetura em grade contempla a coexistência de vários protocolos e serviços (detalhes em [Foster et al., 2001]). Existem também muitos esforços no intuito de padronizar estes últimos, citando os esforços do GGF [GGF, 2006], da OGSA [OGSA, 2006] e da OASIS [OASIS, 2006].

2.1.3 Tecnologias

Grades são propostas com o intuito de compartilhar recursos computacionais heterogêneos que estão geograficamente dispersos, usando a infra-estrutura de rede disponível para integrá-los. Mesmo os recursos computacionais geograficamente distribuídos estando disponíveis, para que seja possível usá-los é necessário que seja montada uma infra-estrutura de software para dar suporte às funcionalidades de grades. Para apoiar a execução de aplicações são necessários sistemas que sejam responsáveis por escalonar e gerenciar os recursos disponíveis no ambiente e as aplicações a serem computadas. Atualmente, existem várias soluções que se destacam, entre elas cita-se o Globus [Globus, 2006], o Condor [Condor, 2006] e o OurGrid [Ourgrid, 2006]. Estas soluções são brevemente descritas a seguir.

Globus

O Globus [Globus, 2006] consiste em um *toolkit* de software, desenvolvido pela *Globus Alliance*, com o objetivo de oferecer suporte para a formação de ambientes de grade e para o desenvolvimento de aplicações a serem executadas nestes. O Globus *toolkit* é composto por um conjunto de serviços que oferece várias funcionalidades, incluindo: segurança, alocação de recursos, gerenciamento de dados e comunicação [Foster, 2006]. Uma das características apresentadas pelo Globus é a possibilidade de utilizar os serviços disponíveis de forma independente ou em conjunto. Desta forma, serviços com diferentes funcionalidades podem ser combinados de acordo com o ambiente onde serão inseridos e com as necessidades da aplicação. A possibilidade do uso parcial de Globus é um aspecto importante para a sua aceitação, pois possibilita, inicialmente, que funcionalidades básicas sejam utilizadas e, à medida que a aplicação necessite, novas funcionalidades sejam incorporadas.

No intuito de concretizar a visão da orientação a serviços, houve uma convergência de tecnologias da área de Computação de Alto Desempenho e de padrões bem consolidados pela indústria. Isso ocorreu através da união de tecnologias e conceitos de grades computacionais com os de *web services*. A partir disso, foi definida uma arquitetura de serviços básicos para a construção de uma infra-estrutura de grades computacionais baseada em serviços. Esta arquitetura foi denominada *Open Grid Services Architecture* (OGSA) [OGSA, 2006, Foster et al., 2002].

A definição da OGSA contempla a idéia de interconexão de sistemas e a criação de ambientes virtuais multi-institucionais. Além disso, os recursos que podem

ser agregados à grade são representados por serviços que são chamados de *Grid Services* [Foster et al., 2002]. Os *grid services* são essencialmente *web services* que seguem convenções estabelecidas na especificação da OGSA e suportam interfaces padronizadas para garantir algumas operações adicionais, como gerenciamento do ciclo de vida do serviço.

Após a definição do modelo da arquitetura e a identificação de serviços básicos através do padrão OGSA, fez-se necessária a especificação do comportamento desses serviços. A especificação denominada *Open Grid Services Infrastructure* (OGSI) descreve a infra-estrutura de serviços básica, no intuito de permitir a implementação do modelo de arquitetura definido pela OGSA. Assim, OGSA define *grid services* e OGSI especifica o comportamento deles.

Na versão 4 do Globus *toolkit* (GT4), alguns aspectos da especificação OGSI precisavam ser modificados/refinados devido à evolução da arquitetura de *web services*. Dentre os aspectos dessa evolução, pode-se citar o aumento da flexibilidade e da extensibilidade e uma ampla conformidade com mecanismos baseados em XML (*eXtensible Markup Language*), usados para descrever, descobrir e invocar serviços [Foster, 2006]. O padrão WSRF (*Web Services Resource Framework*) surge basicamente como resultado do refinamento de OGSI com o objetivo de aproveitar a existência dos novos padrões que surgiram para *web services* (ex: *WS-Addressing*, *WS-Notification*). Desenvolvido pela OASIS [OASIS, 2006], o WSRF especifica como é possível implementar serviços *web* com estado, além de determinar outras funcionalidades que os tornam mais adequados para trabalhar com aplicações de grade [Sotomayor, 2006].

Como um *middleware*, o GT4 disponibiliza uma plataforma padrão para a construção de serviços, necessitando de outros componentes e ferramentas que dêem suporte ao ambiente de grade formado. Esses componontes e ferramentas que integram e/ou agregam funcionalidades e facilidades ao Globus são agrupados no que Foster define como Ecossistema Globus (*Globus Ecosystem*) [Foster, 2006]. Dentre os componentes e ferramentas, cita-se os de maior relevância: *Gridbus*, *MPICH-G2*, *Grid Packaging Toolkit* e *Grid Portal Software*. Já entre os serviços oferecidos pelo Globus, os principais são:

- GRAM (*Globus Resource Allocation Manager*): fornece mecanismos para submissão, monitoramento e controle das aplicações que são executadas na grade;
- GSI (*Grid Security Infrastructure*): infra-estrutura através da qual são

oferecidos serviços de segurança, tais como autenticação, confidencialidade, integridade, controle de acesso e auditoria;

- MDS (*Monitoring and Discovery Service*): fornece informações sobre os recursos que compõem a grade, como disponibilidade e carga de CPU;
- GridFTP: protocolo para transferência de dados que estende o tradicional FTP (*File Transfer Protocol*); novas funcionalidades são adicionadas, tais como transferência em paralelo, usando várias conexões TCP (*Transmission Control Protocol*) entre origem e destino, e transferência *striped*, que usa conexões TCP entre várias origens e um destino, ou vice-versa.

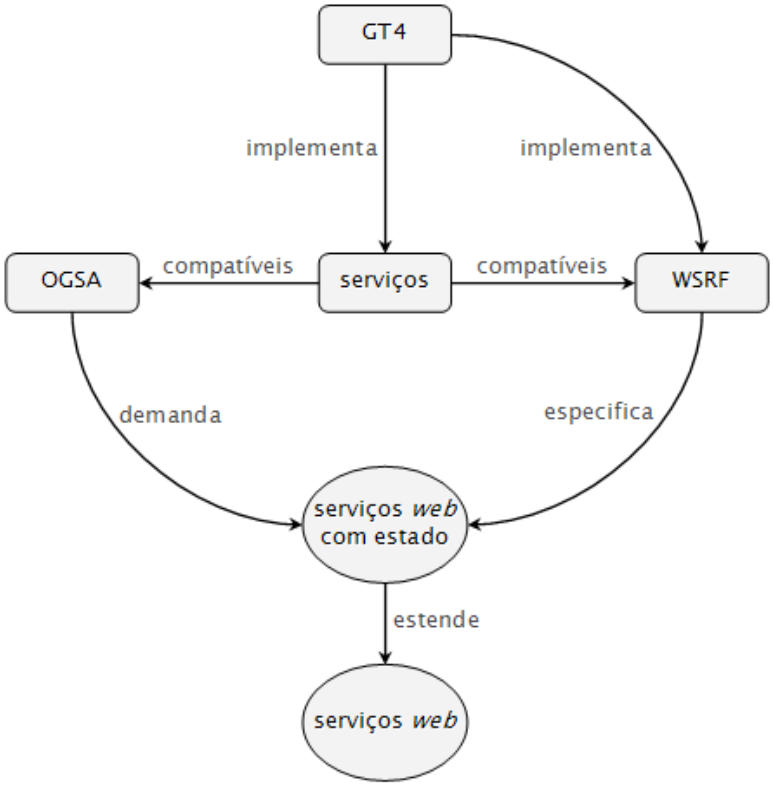

FIGURA 2.1 – Relação entre GT4, OGSA, WSRF e serviços *web*

A definição de padrões é crucial na concretização do conceito de grades computacionais. Os padrões permitem uma maior interoperabilidade, integrando serviços independentemente da plataforma, sistema operacional, ambientes de desenvolvimento e das próprias aplicações. Nesse sentido, com o uso de padrões como OGSA e WSRF, o Globus atinge grande interoperabilidade entre os serviços de grade. A Figura 2.1 ilustra as relações entre GT4, OGSA, WSRF e serviços

web. O GT4, além de uma implementação completa do WSRF, implementa também diversos serviços para grades. A maioria desses serviços é compatível com os requisitos da OGSA e também são implementados de acordo com o WSRF. Enquanto a OGSA necessita de serviços *web* com estado, o WSRF é o responsável por especificá-los, os quais são uma extensão dos serviços *web* tradicionais.

Além do Globus fornecer um arcabouço de vários componentes que podem ser combinados para o desenvolvimento de uma solução, ele permite que aplicações sejam desenvolvidas usando as linguagens Java, C e Python. Somando-se ao citado, a conformidade com padrões abertos tem feito o Globus conquistar cada vez mais adeptos, usuários e colaboradores, tornando-o assim uma das tecnologias de grade mais usadas. Todavia, a montagem de uma ambiente de execução de aplicações não é uma tarefa trivial, colaborando para isso a heterogeneidade de software e hardware, bem como a dificuldade de obter permissões de acesso aos recursos – na maioria das vezes dispersos por várias VOs, ficando restritos à atuação dos seus gerentes.

OurGrid

Ourgrid [Ourgrid, 2006] é uma solução de grade computacional e tem como objetivo principal a criação de um ambiente de execução para aplicações do tipo *Bag-of-Tasks* (BoT). Estas últimas são aplicações paralelas cujas tarefas são independentes umas das outras, ou seja, não precisam de comunicação entre si para realizarem suas funções. Devido a sua simplicidade, as aplicações BoT são usadas em uma variedade de cenários, como buscas maciças, mineração de dados, simulações de Monte Carlo, cálculo de fractais, biologia computacional e processamento de imagens. O OurGrid é desenvolvido por esforços colaborativos entre a Universidade Federal de Campina Grande (UFCG) e a Hewlett-Packard (HP).

O Ourgrid é uma solução "aberta", *free-to-join*: uma grade cooperativa na qual laboratórios doam seus recursos computacionais ociosos formando uma rede *peer-to-peer* e, com isso, esperam obter acesso aos recursos de outros laboratórios quando necessário. Na medida em que houver recursos disponíveis na grade, réplicas das tarefas que estão em execução podem ser criadas com o objetivo de tentar diminuir o tempo de computação da aplicação. Desta forma, réplicas de uma mesma tarefa são executadas simultaneamente, sendo que a primeira a ser concluída é computada e as demais abortadas.

Os três principais componentes que formam a solução Ourgrid são: *Mygrid*, *Peers* e *User Agents*. *MyGrid* é o componente responsável por escalonar, configurar e gerenciar as tarefas, atuando também como coordenador da grade. Já os *Ourgrid*

Peers têm como função principal organizar e prover recursos dentro de um mesmo domínio administrativo. Os componentes *OurGrid User Agents* executam em cada um dos recursos (*gums*), fornecendo o acesso necessário às requisições do *mygrid* e, além disso, dando suporte básico para a instrumentação e o rastreamento de falhas.

Condor

O sistema Condor [Condor, 2006], ou *Condor High Throughput Computing System*, é um gerenciador de recursos especializado para aplicações que exigem computação intensa. Produzido pelo *Condor Research Project* da Universidade de Wisconsin-Madison, seu objetivo é oferecer uma grande quantidade de poder computacional a médio e longo prazo (dias a semanas), utilizando recursos ociosos do ambiente. Os autores do sistema enfatizam que Condor objetiva alta vazão (*high throughput*) e não alto desempenho (*high performance*). As tarefas Condor são independentes para execução, ou seja, são tarefas BoT.

Condor foi inicialmente concebido para funcionar em NOWs (*Network of Workstations*). Uma NOW que executa Condor denomina-se *Condor Pool*. O elemento arquitetural mais importante de um *Condor Pool* é o *Matchmaker*, que aloca tarefas a máquinas pertencentes ao *Pool*. Tal alocação é baseada nas necessidades de cada tarefa e nas restrições de uso de cada máquina. Ao efetuar o casamento (*match*) entre os requisitos da tarefa e os atributos da máquina, o *matchmaker* notifica agentes que representam ambas as partes, que então interagem diretamente para realizar a execução da tarefa.

A habilidade de realizar a migração de processos (mecanismo de *checkpointing*) também é um aspecto interessante do Condor. Uma vez que um recurso necessite ser desocupado, o mecanismo de *checkpoint* salva transparentemente o estado de execução da tarefa. Isso permite que essa tarefa seja reexecutada em outra máquina a partir do ponto em que parou. No entanto, há restrições quando em ambientes bastante heterogêneos ou de aplicações que fazem uso de primitivas do sistema operacional.

Como não foi projetado para trabalhar com grades, Condor foi posteriormente estendido para tal finalidade. É possível citar duas formas de funcionamento em grades: *Flock of Condors* e *Condor-G*. Um *Flock of Condors* é uma grade formada por vários *Condor Pools*. A base para criação de um *flock* é um acordo de cooperação (de troca de recursos) entre dois *Condors Pools*, onde suas ligações são sempre em pares, não envolvem entidades centralizadoras e não alteram o software Condor original. As funcionalidades do *Flock of Condors* são implementadas por uma

máquina especial, chamada *Gateway*. Com ele, os recursos de outra máquina podem ser usados de forma transparente. Outra forma mais recente de funcionamento em grades, o *Condor-G* adota uma visão mais heterogênea de grades. Além de usar o *Condor Pool*, também utiliza recursos via tecnologia Globus.

2.2 Computação Autônoma

Esta seção apresenta os principais conceitos de Computação Autonôma. Ela inicia apresentando uma breve definição de computação autônoma, listando as suas características desejáveis (Sub-seção 2.2.1). Na seqüência, na Sub-seção 2.2.2, é apresentada a arquitetura básica de um sistema de computação autônoma e seus componentes. Na Seção 2.2.3, são apresentados esforços de pesquisa no intuito de alcançar características desejáveis à computação autônoma, evidenciando a importância e a emergência de tecnologias e soluções que buscam implementar essas características.

2.2.1 Definição

A necessidade de gerenciamento de recursos computacionais não é um problema novo para a Ciência da Computação. Por décadas componentes e sistemas têm evoluído muito e, com eles, a complexidade dos sistemas de controle, de compartilhamento de recursos e de gerenciamento das operações. A Computação Autônoma surge neste contexto como uma alternativa promissora para reduzir tal complexidade, consistindo na habilidade de um sistema ser mais auto-gerenciável [Ganek and Corbi, 2003].

O termo *Autonomic Computing* (Computação Autonôma) foi cunhado pelo então vice-presidente de pesquisas da IBM, Paul Horn, em uma palestra na Universidade de Harvard em março de 2001. O mesmo descreveu a Computação Autonôma como sendo um "grande desafio" ([Horn, 2001]) e chamou a atenção para o desenvolvimento de sistemas autônomos, apontando oito características chave desejáveis nesses sistemas. Essas características foram também apresentadas em [IBM, 2006, Ganek and Corbi, 2003, White et al., 2004, Salehie and Tahvildari, 2005] e são brevemente descritas a seguir.

1. *Auto-Definição:* capacidade do sistema conhecer a si próprio, devendo ser formado por componentes que possuam essa mesma capacidade.

2. *Auto-Configuração:* capacidade do sistema se adaptar automática e

dinamicamente a mudanças do ambiente. Provê características que permitem a inserção de novos componentes com a mínima intervenção humana. Auto-configuração consiste, ainda, na habilidade de sistemas inteiros se auto-configurarem.

3. *Auto-Otimização:* capacidade do sistema maximizar a alocação e a utilização dos recursos para satisfazer as requisições do usuário. Para tal, precisa ser capaz de ajustar seus parâmetros e monitorar seu desempenho.

4. *Auto-Recuperação:* capacidade do sistema detectar, diagnosticar e reparar problemas localizados, resultantes de erros ou falhas, tanto de hardware quanto de software.

5. *Auto-Proteção:* capacidade do sistema antecipar, detectar e recuperar-se de ataques. O objetivo é defender o sistema contra ataques maliciosos ou falhas em cascata ocorridas na tentativa de auto-recuperação.

6. *Sensibilidade ao Contexto:* capacidade do sistema de se adaptar às condições do ambiente onde ele está inserido. Projeta-se o sistema de maneira que possa controlar, gerenciar e alocar seus componentes e recursos de rede de forma a melhor atingir seus objetivos (ex: alto desempenho ou baixo custo).

7. *Interoperabilidade:* capacidade do sistema ser altamente portável considerando múltiplas plataformas e grande variedade de infra-estruturas de componentes. Para que isso seja possível, independente de desenvolvedor e tecnologia, é necessário que interfaces padrão sejam definidas e utilizadas. Assim, elementos autônomos podem ser contactados e/ou relacionarem-se.

8. *Antecipação:* capacidade do sistema otimizar o uso e a busca por recursos de forma que usuários e aplicações possam utilizá-los em qualquer ambiente, sem a necessidade de requisições específicas e sem aumentar o nível de complexidade para o usuário.

Com conotação biológica, inspirada pelas funcionalidades do sistema nervoso humano, a Computação Autônoma sugere que os sistemas computacionais gerenciem a si mesmos tendo em vista objetivos de alto nível (*Políticas*).

2.2.2 Arquitetura

Uma arquitetura de computação autônoma deve buscar dentre os seus objetivos [White et al., 2004]: (a) oferecer interfaces externas e características que

permitam a um elemento individual ser autônomo, ou seja, auto-gerenciável, e (b) descrever como o sistema é composto, indicando como os elementos autônomos vão interagir para formar um sistema auto-gerenciável. Assim, na tentativa de descrever os componentes de uma arquitetura, bem como de identificar esforços e desafios da computação autônoma, Kephart [Kephart, 2005] divide o espaço de pesquisa em três domínios distintos: elementos autônomos, sistemas autônomos e interações homem-computador. Esses três domínios são explicados a seguir.

Elementos Autônomos

Elementos autônomos são blocos básicos para a construção de sistemas autônomos que, através de suas interações mútuas, produzem o todo do auto-gerenciamento [Kephart, 2005]. Elementos autonômos gerenciam seu comportamento interno e suas relações com outros elementos. Esse gerenciamento é feito de acordo com regras que determinam qual será o comportamento do elemento autônomo, interna e externamente [Kephart and Chess, 2003].

A interação entre os elementos é feita de forma restritiva e padronizada, sendo necessário para isso, a definição de *interfaces* padrão. Através delas, os serviços são descritos, descobertos e acessados. Para que seja possível atingir o auto-gerenciamento e a interoperabilidade entre os mais diversos componentes, é necessário que novas interfaces sejam definidas para atingir o comportamento autônomo desejado [White et al., 2004]. A padronização dessas interfaces facilita a interoperabilidade entre elementos desenvolvidos com diferentes tecnologias. Um exemplo são as interfaces definidas pela arquitetura OGSA, as quais padronizam os serviços de grades e, apesar de serem definidas com um objetivo pontual (grades computacionais), se aplicam para qualquer arquitetura orientada a serviços [White et al., 2004]. Na seqüência são apresentadas interfaces importantes aos elementos autônomos.

- *Interface de monitoramento e teste*: habilita um elemento para que ele possa ser monitorado por qualquer outro elemento que tenha estabelecido, com ele, relações administrativas. Essas interfaces podem ser utilizadas para controlar a quantidade de dados de *log* e *trace* que um elemento acumulou da sua própria execução e, além disso, para acessar dados com o objetivo de visualizá-los ou até mesmo alterá-los. Ademais, essas interfaces devem ser usadas para instruir um elemento a fazer o auto-teste e obter os resultados.

- *Interface do ciclo de vida*: habilita um elemento administrativo a gerenciar

a máquina de estados de um elemento, determinando quais políticas melhor se adequam para reger o ciclo de vida do elemento. Neste caso, o ciclo de vida de execução de uma tarefa seria a análise das dependências, o início, o monitoramento, a obtenção dos resultados e o seu término. Elementos que possuem suas atividades prolongadas, permanecendo ativos por longos períodos, monitoram os estados do seu ciclo de vida, com o objetivo de mantê-los executando.

- *Interface de políticas*: usada para o envio de novas políticas para elementos e para determinar quais devem ser usadas. A alteração das políticas está condicionada às permissões e/ou à confirmação de algum elemento.

- *Interface de negociação e binding (acordos)*: permite que um elemento requisite/forneça um determinado serviço. Quando ocorre uma requisição, o elemento recebe uma mensagem de confirmação ou erro, buscando assim reagir de acordo com as políticas para ele impostas. Esses tipos de interfaces são comuns em arquiteturas orientadas a serviços.

Para que seja possível ter auto-gerenciamento flexível, elementos autônomos devem suportar interfaces mais complexas, que permitem uma maior interação entre eles. Essas interfaces permitem que acordos de serviços (*WS-Agreement*) [GGF, 2006] sejam negociados e estabelecidos (ex: reserva e formação de relações de longo duração (*longer-term*).

Quando um elemento autônomo provê um serviço para outro elemento, diz-se que esses dois elementos possuem um Relacionamento (*Relationship*). Essas relações são formadas em tempo de execução e não durante o processo de implantação, podendo mudar a medida que mudanças de configuração ou requisito ocorrem. Isso é feito pelos próprios elementos sem a intervenção de administradores.

Relacionamento é a maneira pela qual os elementos autônomos são agrupados para formar os sistemas autônomos. Em geral os relacionamentos são formados como resultado da negociação entre elementos. Quando um elemento necessita de um serviço, ele procura outro elemento que forneça esse serviço e faz uma requisição. Para que a relação seja concretizada é necessário que os elementos sejam autorizados e que possuam os requisitos mínimos exigidos para o serviço.

A *integridade das interações* possibilita que um elemento autônomo controle seu próprio comportamento e controle as interações com outros elementos. A comunicação de um elemento autônomo e outro deve ser feita através da interface associada à especificação do serviço. Os dados trocados entre dois elementos

autônomos não podem ser acessados por outros elementos. O contrário representa um risco de segurança e pode impossibilitar a formação de sistemas autônomos.

Sistemas Autônomos

Mesmo que se tenha elementos auto-gerenciáveis, eles não garantem o auto-gerenciamento no nível de sistema. Para um sistema ser constituído de elementos autônomos, é necessário que esses sejam capazes de descobrir um ao outro, identificar outros elementos com quem podem se comunicar, agindo de forma coordenada para atingir os objetivos mútuos. Supondo que se deseje implementar uma solução de grade computacional e que todos os elementos autônomos estejam disponíveis, é necessário coordená-los para criar um *sistema autônomo de grades computacionais*.

A coordenação entre os elementos autônomos é realizada por elementos especializados que fornecem suporte às operações de integração, possibilitando assim que os sistemas autônomos sejam formados. A lista de elementos inclui *registry, sentinel, aggregator, broker* e *negotiator* [White et al., 2004], conforme comentado a seguir.

O mecanismo *Registry* é responsável por prover serviços que permitem aos elementos registrar e procurar por serviços. Além disso, o mecanismo *Registry* determina que tipos de acordos/relacionamentos podem ser fechados pelos elementos autônomos. Quando um elemento quer procurar outro de um determinado tipo, ele primeiro contata um *registry*, com o qual ele tem um relacionamento já estabelecido. Perguntado a respeito de elementos de um determinado tipo, o mecanismo de registro responde com uma lista de endereços. O elemento requisitante estabelece um relacionamento com o elemento da lista que lhe for mais conveniente, que melhor se adequar aos seus requisitos.

O mecanismo *Sentinel* (Sentinela) é responsável por monitorar os serviços e efetuar as ações cabíveis mediante o presente cenário. Um sentinela pode ser um mecanismo chave, desempenhando algumas funcionalidades (serviços) que necessitariam estar implementados nos elementos. Por exemplo, o sentinela pode ficar responsável por monitorar variáveis e alertar elementos autônomos em específico, caso limiares sejam atingidos.

O *Aggregator* é responsável por combinar dois ou mais elementos existentes. Sua finalidade é, de forma geral, melhorar o serviço que um elemento individualmente pode prover em quesitos como desempenho, custo e confiabilidade.

O *Broker* tem a função de facilitar as interações entre elementos e serviços. Este recebe requisições de elementos que não conseguem efetuar uma determinada

tarefa. O *broker* coordena a tarefa e retorna o resultado ao elemento requisitante. Um exemplo de uso desse tipo de mecanismo é no acesso a serviços de armazenamento de alta disponibilidade.

O último elemento que fornece suporte para operações em um sistema autônomo é o *Negotiator*. Ele é especializado em auxiliar elementos em negociações complexas, principalmente quando estes não possuem suporte para a operação. Um *negotiator* pode ser requisitado para testar parâmetros, auxiliando dessa forma os componentes em suas tomadas de decisão.

A essência dos sistemas autônomos é o auto-gerenciamento [Kephart and Chess, 2003]. Um sistema auto-gerenciável pode ser definido como um sistema que possui protocolos e serviços suficientes para atingir um estado estável após a ocorrência de uma anomalia, sem que seja necessária a intervenção humana [Van Moorsel, 2005]. A computação autônoma dá suporte para aplicações auto-gerenciáveis, possibilitando que estas controlem sua própria evolução, desde a fase de implantação e instalação, passando pelo gerenciamento do seu ciclo de vida, bem como pela recuperação, proteção e otimização de sua operação.

Kephart [Kephart, 2005] afirma que, alternativamente e em alguns casos mais particulares, um elemento autônomo pode ser identificado como sendo um agente de software e sistemas autônomos como sendo sistemas multi-agentes. *Unity* [Chess et al., 2004] é um exemplo de arquitetura baseada em agentes, em que cada componente é um elemento autônomo.

Interação homem-computador

A computação autônoma tem a intenção de reduzir a sobrecarga de gerenciamento imposta aos administradores, permitido que esses expressem seus objetivos e deixem que o sistema de gerenciamento cuide dos detalhes necessários para atingir os objetivos especificados. Para determinar como melhorar o suporte e suas práticas, é muito importante entender como os administradores gerenciam os sistemas atualmente. É importante também identificar as necessidades dos sistemas, bem como tentar entender o crescimento destes. As *políticas* são usadas para expressar os objetivos dos administradores e tem um papel essencial para concretizar a visão de computação autônoma, porque elas são a forma pela qual humanos expressam seus objetivos para sistemas autônomos.

Existem muitas iniciativas de pesquisa no intuito de desenvolver padrões para a especificação de políticas que possibilitem a integração de elementos autônomos, permitindo a estes interagir e associar-se. Por mais que existam mecanismos que

facilitem a integração de elementos para formar sistemas autônomos, as interações e o comportamento desses elementos será regido de acordo com as políticas definidas por humanos. Quanto mais precisas e abrangentes as políticas, maior será a autonomia do sistema.

2.2.3 Esforços em Direção à Computação Autônoma

Diversos esforços de pesquisa têm sido realizados no intuito de desenvolver sistemas computacionais com características da computação autônoma, bem como arcabouços que facilitem e dêem suporte para o desenvolvimento de sistemas. A seguir são apresentados quatro trabalhos relacionados bastante representativos: AIDE [AIDE, 2006], *Unity* [Chess et al., 2004], LCFG(ng) [LCFG, 2006] e SmartFrog [SmartFrog, 2006a].

Autonomic Integrated Development Environment

A IBM tem desenvolvido um conjunto de ferramentas de desenvolvimento que usam o padrão WSDM (*Web Services Distributed Management*) e que tem como objetivo facilitar o gerenciamento de TI. Tal objetivo é perseguido mediante uma arquitetura para o relacionamento entre recursos computacionais e aplicações de gerenciamento [AIDE, 2006]. Nessa arquitetura, um gerente autônomo se comunica com um ou mais recursos (servidores, bases de dados, etc.) através de *endpoints* gerenciáveis. Um *endpoint* é uma pequena camada de código que traduz os comandos da interface padrão em comandos específicos do produto. A seguir são apresentadas as ferramentas oferecidas pelo AIDE para o desenvolvimento de soluções.

O *IBM Manageability Endpoint Builder* inclui ferramentas e um ambiente de *run-time* para a construção de *endpoints*, o que permite aos produtos exporem suas interfaces gerenciáveis. Ele permite aos desenvolvedores implantar *endpoints* gerenciáveis nos ambientes Apache Axis ou OSGi. Com essas interfaces é possível que qualquer ferramenta que implemente o padrão WSDM ou um gerente autônomo construído pelo AIDE possa visualizar o estado dos recursos e fazer chamadas para modificá-los.

O *Manageable Resource Browser* foi uma adição feita na primeira atualização do AIDE. Ele permite que o usuário analise os recursos e os serviços. Usando o *manageable resource explorer*, um usuário pode visualizar e manipular as propriedades disponibilizadas pelo WSDM *endpoint* e além disso invocar operações válidas nele.

Finalmente o *IBM Manageability Endpoint Simulator*, primeiramente

conhecido como *IBM Touchpoint Simulator*, ajuda no desenvolvimento de gerentes autônomos emulando um recurso que implementa o padrão WSDM. O maior obstáculo no desenvolvimento de gerentes autônomos reside no fato de ser necessário recursos (*endpoints*) para que testes possam ser efetuados. Com o simulador é possível que os gerentes autônomos sejam testados com vários recursos e componentes, sem a necessidade de ter a estrutura previamente disponível.

Unity

Unity [Chess et al., 2004] é um projeto de pesquisa desenvolvido no Centro de Pesquisas Thomas J. Watson. Esse projeto de pesquisa tem por objetivo explorar o comportamento e os relacionamentos que irão permitir aos sistemas computacionais complexos se auto-gerenciarem.

Todos os componentes que compõem o *Unity* são elementos autônomos, sendo responsáveis por se auto-gerenciarem e fornecerem serviços para humanos e para outros elementos. Esses elementos podem ser: recursos computacionais (ex: base de dados, sistema de armazenamento, servidores), autoridades gerenciadoras – elementos de mais alto nível – ou elementos que monitoram/auxiliam os outros elementos no cumprimento de suas tarefas (ex: repositório de políticas, *sentinel*, *broker* e *registry*).

No *Unity* cada elemento autônomo é responsável por gerenciar seu comportamento interno, controlando os seus recursos e gerenciando suas próprias operações internas. Nestas operações, espera-se que os elementos possuam características desejáveis aos sistemas autônomos (vide detalhes na Sub-seção 2.2.1). Cada elemento estabelece e gerencia seus relacionamentos de acordo com seus objetivos. Os relacionamentos são concretizados através de interfaces específicas de cada elemento e/ou interfaces do padrão OGSA. A comunicação dos elementos é efetuada exclusivamente de acordo com as interfaces documentadas, sendo dessa forma possível controlar todas as interações entre os elementos [Chess et al., 2004].

Local ConFiGuration system

LCFG (*Local ConFiGuration system*) [LCFG, 2006] é um sistema para instalar e gerenciar as configurações de um grande número de computadores. Ele é apropriado para ambientes heterogêneos e bastante dinâmicos. LCFG foi desenvolvido pela Universidade de Edinburgh, sendo que o projeto teve início por volta de 1993. A versão aqui apresentada é chamada de LCFGng (*next generation*) por apresentar diferenças significativas em relação ao LCFG original

[Anderson, 2001, Anderson and Scobie, 2002].

Além de possuir uma linguagem de configuração, o LCFG possui um repositório central com as especificações de configurações, a partir de onde máquinas podem ser automaticamente instaladas e configuradas. Mudanças nessa central de especificações faz com que todos os nodos sejam automaticamente atualizados, segundo suas respectivas especificações. A proposta é que o sistema seja apropriado para ambientes em que as configurações são as mais diversas (variando de grandes servidores até *laptops*) e em que diferentes aspectos de configuração podem mudar freqüentemente, podendo estes serem gerenciados por várias pessoas [Anderson, 2001].

A configuração de um ambiente é descrita por *Source Files*, os quais ficam armazenados em um servidor central. Um *source file* não necessariamente corresponde a uma máquina ou a um componente. Ele geralmente descreve aspectos mais amplos de configuração que permitem a formação de unidades lógicas de gerenciamento. Como exemplo, cita-se unidades como "parâmetros para máquinas com Globus" e "parâmetros para máquinas com GridFTP". Dependendo dos parâmetros da unidade descrita, mais que um *source file* pode ser requisitado.

Cada máquina tem seu *Profile* que é gerado a partir da compilação dos *source files* específicos. O *profile* contém todos os parâmetros de configuração de uma máquina e é publicado através de serviços *web*. Quando o *profile* é alterado, o cliente recebe uma notificação e este, por sua vez, requisita o novo *profile* com as mudanças.

O componente *script* é o responsável por ler os parâmetros de configuração e executar as ações necessárias para implementar essas configurações no cliente. Periodicamente, o servidor recebe e disponibiliza informações do estado dos clientes.

Embora o LCFG seja usado em ambientes (ex: grades) com configuração estática, um mecanismo chamado de *contexts* permite que ele seja também usado para tarefas simples com configurações dinâmicas. *Contexts* pode ser usado para trocar o conjunto de configurações pré-definidas sem envolver o servidor de configurações (onde ficam todas as configurações usadas pelo LCFG). Isso potencialmente permite que nodos, que estão sobre o controle do LCFG, resolvam determinados conflitos de versões de pacotes mudando as configurações em tempo de execução [Smith and Anderson, 2004].

Smart Framework for Object Groups

SmartFrog [SmartFrog, 2006a] é uma tecnologia para a descrição de sistemas distribuídos. Ela consiste de coleções de componentes gerenciáveis e que cooperam entre si. O SmartFrog foi desenvolvido pelo HP Labs em Bristol, na Inglaterra, e seu arcabouço tem licença LGPL (*Lesser General Public License*).

SmartFrog consiste de uma linguagem para descrição das coleções de componentes, seus parâmetros de configuração e um ambiente de execução que ativa e gerencia os componentes para manter os sistemas executando. As seguintes funcionalidades são providas pelo SmartFrog [SmartFrog, 2006b]:

- configuração: permite descrever os componentes autônomos e integrá-los em uma aplicação distribuída, usando para isso uma única descrição;
- implantação: permite implantar uma configuração sobre um conjunto de recursos computacionais;
- ciclo de vida: permite assegurar o progresso de um componente dentro de seu ciclo de vida (implantar, iniciar e terminar) de maneira orquestrada, seguindo uma ordem, analisando e respeitando as suas dependências;
- descoberta e comunicação: permite que elementos localizem outros elementos em uma aplicação e comuniquem-se com eles, estabelecendo relacionamentos estáticos ou dinâmicos em tempo de execução.

SmartFrog tem sido bastante usado no domínio de computação utilitária para configuração de sistemas em larga escala. É importante deixar claro que o SmartFrog é um arcabouço para o desenvolvimento de soluções e não uma solução pronta. Os serviços, desenvolvidos em Java ou encapsulados, devem ser ativados pelo SmartFrog [SmartFrog, 2006a].

2.3 Configuration Description, Deployment, and Lifecycle Management

A especificação CDDLM (*Configuration Description, Deployment, and Lifecycle Management*) [Bell et al., 2004] define mecanismos para encapsular informações de uma aplicação de grade, que podem ser utilizadas para a sua implantação em uma ou mais estações-alvo de uma infra-estrutura de grade. Essas informações descrevem os componentes que formam a aplicação e seus

parâmetros de configuração, bem como o fluxo de implantação desses componentes [Bell et al., 2005]. Adicionalmente, CDDLM descreve interfaces para a instanciação e o gerenciamento dos componentes que integram a aplicação [Loughran, 2005].

A especificação CDDLM é organizada em três documentos: linguagem de descrição de componentes, modelo de componentes e API de implantação [Bell et al., 2004]. A Figura 2.2 ilustra o escopo e as relações entre a especificação de cada documento. A definição das aplicações de grade é feita através da linguagem de descrição de componentes. Já o modelo de componentes define as relações entre componentes e destes com os recursos. Uma implementação da API de implantação CDDLM tem como finalidade gerir o processo de instanciação da aplicação de acordo com a sua descrição. Esses três documentos são apresentados na seqüência.

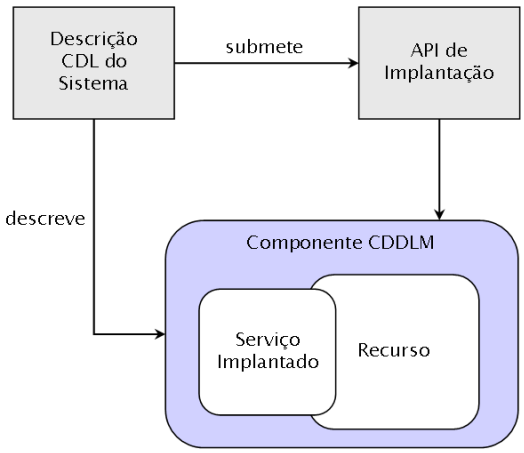

FIGURA 2.2 – Arquitetura do Modelo de Componentes do CDDLM

Linguagem de Descrição de Configuração

A Linguagem de Descrição de Configuração (*Configuration Description Language* – CDL) [Bell et al., 2005] fornece um conjunto de declarações que permite descrever componentes (blocos básicos de implantação) e suas dependências. A linguagem CDL é baseada em XML e oferece maneiras para definir propriedades dos componentes (nomes, valores e tipos) e associar valores a elas. Ademais, CDL é capaz de representar dependências entre os valores dos parâmetros de configuração, garantindo a sua consistência. O fluxo de implantação também pode ser descrito, instruindo a implementação CDDLM a executar a seqüência correta de operações para implantar, iniciar, executar, recuperar, parar e remover os componentes.

Modelo de Componentes

O modelo de componentes CDDLM *Component Model* [Schaefer, 2005] define os componentes como sendo unidades básicas de implantação. Além disso, o modelo descreve o comportamento e as interfaces que permitem aos componentes fornecerem e utilizarem serviços. Os componentes, suas dependências e o fluxo de implantação são descritos segundo a linguagem de descrição CDL. Os serviços implementados pelos componentes são encapsulados em *containers* (pacotes) de implantação, facilitando a sua instanciação e o seu gerenciamento. O ciclo de vida desses componentes é dirigido por uma máquina de estados, que define a evolução no processo de instanciação.

API de Implantação

A interface de implantação, denominada *Deployment* API [Loughran, 2005], especifica métodos a serem invocados para implantar e configurar os componentes, bem como para gerenciar seu ciclo de vida. A API segue o padrão WSRF e utiliza o serviço *web* SOAP (*Simple Object Access Protocol*) para implantação das aplicações em uma ou mais estações-alvo.

2.4 Sumário

Este capítulo apresentou o referencial teórico do presente trabalho. Na Seção 2.1 foram apresentados conceitos de grades computacionais que, de forma geral, consistem em tecnologias para propiciar o compartilhamento de recursos computacionais heterogêneos, os quais estão geograficamente distribuídos. Também foram citados aspectos inerentes às grades (heterogeneidade, escalabilidade, compartilhamento, controle distribuído e dinamicidade ou adaptabilidade). Ainda nesta seção, foi apresentada uma estrutura básica de grades e exemplos de middleware que oferecem o suporte necessário para a execução de aplicações nas grades (ex: Globus *toolkit*).

A Seção seguinte (Seção 2.2) iniciou com a definição de computação autônoma. Além disso, foram apresentadas considerações sobre uma possível arquitetura de computação autônoma, a qual é baseada em componentes autônomos e suas interações, formando os sistemas autônomos. Também foram descritos os principais esforços no intuito de desenvolver aplicações com algum tipo de suporte à computação autônoma.

A Seção 2.3 apresentou a especificação CDDLM, que fornece uma linguagem para a descrição da configuração de serviços, a determinação da hierarquia entre eles. Além disso, descreve métodos para a instanciação de ambientes de grade. Como foi mencionado, o processo de instanciação compreende funcionalidades para implantar, configurar e gerenciar os serviços que compõem uma aplicação.

Atualmente as iniciativas de pesquisa em grades computacionais buscam cada vez mais incorporar mecanismos que permitam a automatização de suas operações. De encontro a essa necessidade, o paradigma de computação autônoma define características desejáveis para que essa autonomicidade seja atingida. Os esforços existentes lidam com apenas algumas das características desejáveis à computação autônoma. Trata-se de uma área de pesquisa nova e em ampla ascensão, possuindo crescentes esforços tanto no meio acadêmico como na indústria. Uma das vertentes desses esforços, a especificação CDDLM, sinaliza para a padronização e o desenvolvimento de características de computação autônoma para as tecnologias de grade.

Capítulo 3

Trabalhos Relacionados

A área de implantação e configuração dinâmicas de serviços e aplicações tem sido alvo de pesquisas recentes, gerando como resultado diversas propostas para lidar com o problema. Nesta seção apresenta-se trabalhos relacionados, abordando-se, primeiro, implantação e configuração dinâmicas de serviços em geral e na seqüência, de aplicações de grade.

Em [Talwar et al., 2005] é apresentada uma análise comparativa de soluções manuais, baseadas em *scripts*, em linguagens e em modelos para implantação de serviços, constituindo uma referência-chave para mapear a área. Os autores avaliam escalabilidade, complexidade e expressividade das possíveis soluções e concluem que as baseadas em modelos são as mais promissoras por lidarem bem com escalabilidade e complexidade, aspectos estes cruciais para implantação de aplicações distribuídas de grande porte. A implantação dinâmica de serviços também foi investigada em contextos outros que não grades computacionais, como J2EE [Reverbel et al., 2004] e Web Services [Benatallah et al., 2002]. Em [Rauch et al., 2000] foi proposta a clonagem de partições de sistema para implantação de software, abordagem reconhecidamente *cara* computacionalmente por exigir substituição de toda a imagem do sistema operacional. Seguindo na mesma direção, em [Keahey et al., 2005] é proposto o emprego da tecnologia de máquinas virtuais para implantar ambientes com diferentes serviços e configurações.

Os principais trabalhos correlatos focados em grades computacionais são os apresentados em [Sun et al., 2005], [Weissman et al., 2005] e [Qi et al., 2007]. O primeiro propôs uma solução segura para implantação dinâmica de serviços WSRF (*Web Services Resource Framework*). O segundo introduziu uma arquitetura baseada no Tomcat para incorporar a funcionalidade de implantação dinâmica de serviços ao Globus Toolkit versão 3 (GT3), com a restrição de operar em nível de

container. O terceiro trabalho propôs mudanças no núcleo do Tomcat para tornar sua implementação mais leve e propiciar implantação em nível de *serviço*. Em paralelo a esses trabalhos, o GGF (*Global Grid Forum*) tem despendido esforços para padronizar a especificação CDDLM (*Configuration Description, Deployment, and Lifecycle Management*), que constitui uma abordagem baseada em modelos para configuração, implantação e gerenciamento do ciclo de vida de aplicações de grades (descrita na Seção 2.3). A seguir são detalhados os três trabalhos recém citados e que possuem estreita relação com a proposta desta dissertação.

3.1 ROST

Este trabalho propõe uma solução segura para implantação dinâmica de serviços WSRF (*Web Services Resource Framework*) em ambientes distribuídos [Sun et al., 2005]. Para facilitar a distribuição dos serviços, a sua implantação remota e em tempo de execução é uma característica altamente desejável. Outro requisito importante é a segurança, particularmente considerando a implantação em domínios diferentes. O ROST (*scheme of Remote & hOt Service deployment with Trustworthiness*) é uma proposta para distribuição remota, segura e em tempo de execução dos serviços.

As atualização das configurações do ambiente é feita dinamicamente em tempo de execução, evitando que o sistema tenha que ser reiniciado. ROST faz uso de mecanismos para negociação entre os componentes com o objetivo de assegurar que a distribuição dos serviços seja feita de forma segura.

A arquitetura ROST é ilustrada na Figura 3.1. Ela é composta por diversos componentes, sendo os principais TNA, RHD e SCC. O componente TNA (*Trust Negotiation Agent*) é responsável pela segurança entre o mecanismo de implantação (*deployer*) e o cliente (*container*). O componente RHD (*Remote Hot Deployment*) é responsável pela implantação dos serviços remotos em tempo de execução. Já o componente SCC (*Service Container Configuration*) fornece vários serviços de configuração para o *container*.

O ROST tem como ponto forte a segurança oferecida a requisições e trocas de informações entre os elementos da grade. No entanto, o trabalho não descreve o seu comportamento diante de possíveis falhas e dependências. Além disso, a implantação é feita em nível de *container*, o que limita as possibilidades de gerenciamento dos serviços implantados.

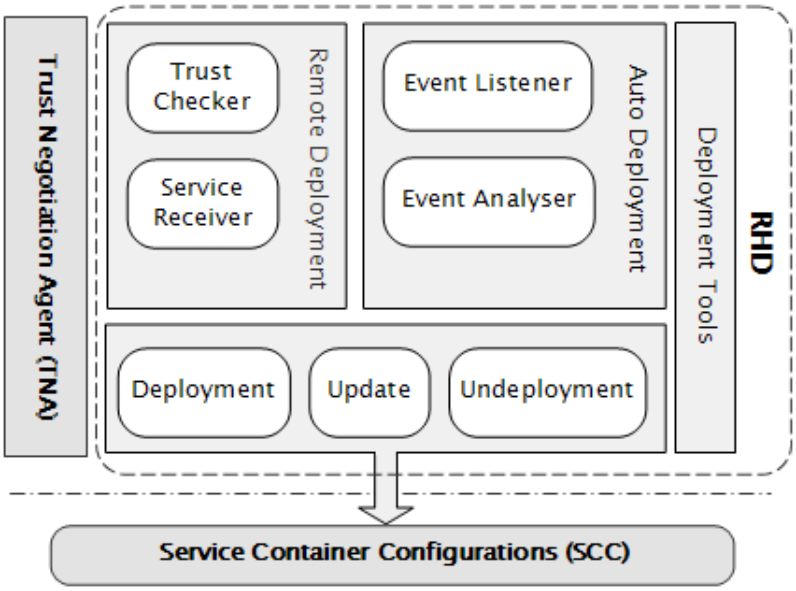

FIGURA 3.1 – Arquitetura ROST

3.2 Arcabouço para adaptação dinâmica de serviços em grades

O projeto descrito em [Weissman et al., 2005] trata do problema da adaptação dinâmica de serviços em grades. A necessidade de adaptação surge quando ambos, recursos e serviços, possuem demandas variáveis. O trabalho cita características consideradas chave no âmbito do problema: (a) uma arquitetura de grade dinâmica baseada na OGSA, que suporte a implantação dinâmica de serviços; (b) *middleware* de gerenciamento de recursos que decida dinamicamente como recursos podem ser alocados para atender determinadas requisições; (c) uma estrutura dinâmica de empréstimo (*leasing*) para decidir quantos e quais recursos serão reservados a um serviço, garantindo a sua execução no futuro; (d) um modelo robusto que permita descrever serviços que sejam sensíveis às mudanças da grade.

O serviço AGS (*Adaptive Grid Service*) é uma abstração oferecida para os serviços de grade para que estes possam adaptar-se às mudanças de requisições e disponibilidade de recursos. De forma resumida, a Figura 3.2 apresenta os componentes da AGS, sendo os três principais: *frontend*, *deployer* e *back-end*. O componente AGS *frontend* é um cliente que requisita recursos e toma decisões sobre onde as requisições devem ser executadas. Já o componente AGS *deployer* é o responsável pelo escalonamento, decidindo em que estações devem ser implantados os serviços. Além disso, é responsável por manter informações dos serviços

já implantados e executando. O componente *back-end*, por sua vez, consiste de um AGS *factory* que contém o código atual de cada serviço e responde a cada requisição criando uma instância chamada de AGSI (*Adaptive Grid Service Instance*). Adicionalmente, o *back-end* é dinamicamente implantado ou hospedado pelos serviços fornecidos pelo provedor dinâmico de recursos (*Adaptative Resource Provider* – ARP).

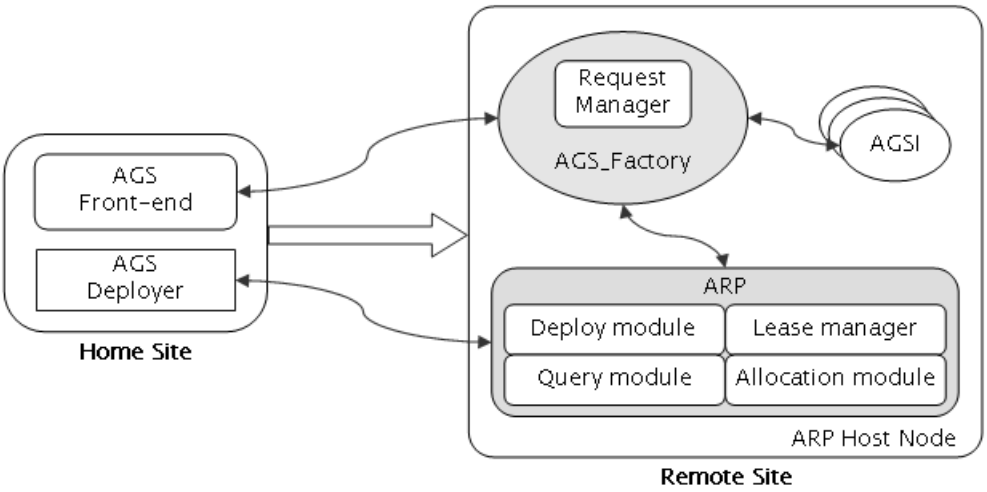

FIGURA 3.2 – Arquitetura de serviços dinâmicos

A implementação do serviço de implantação dinâmica é realizada usando o Tomcat, que foi modificado para que fossem incorporadas funcionalidades de implantação dinâmica de serviços ao Globus Toolkit versão 3 (GT3). As operações estão restritas a nível de *container*. Além disso, o trabalho não menciona nenhum mecanismo para o tratamento de falhas ou operações mal sucedidas, o que pode comprometer o processo de implantação dinâmica.

3.3 HAND

HAND (***H**ighly* ***A**vailable* *DyN**a**mic* ***D**eployment* *Infrastructure*) [Qi et al., 2007] propõe uma infra-estrutura para permitir implantação de serviços em ambientes dinâmicos de grades baseada no *Java web services Core* do Globus Toolkit 4.

Duas abordagens de implantação são oferecidas por HAND: em nível de serviço (*Service-level* – HAND-S) e em nível de *container* (*Container-level* – HAND-C). Em uma implantação em nível de serviço, um ou mais serviços existentes podem ser instalados, ativados e/ou reativados de forma unitária. Já em uma implantação em

nível de *container*, a instalação de qualquer novo serviço envolve a reinstalação e reconfiguração de todos os serviços hospedados no *container*.

A Figura 3.3 apresenta as três divisões do HAND e seus respectivos componentes, que são descritos a seguir.

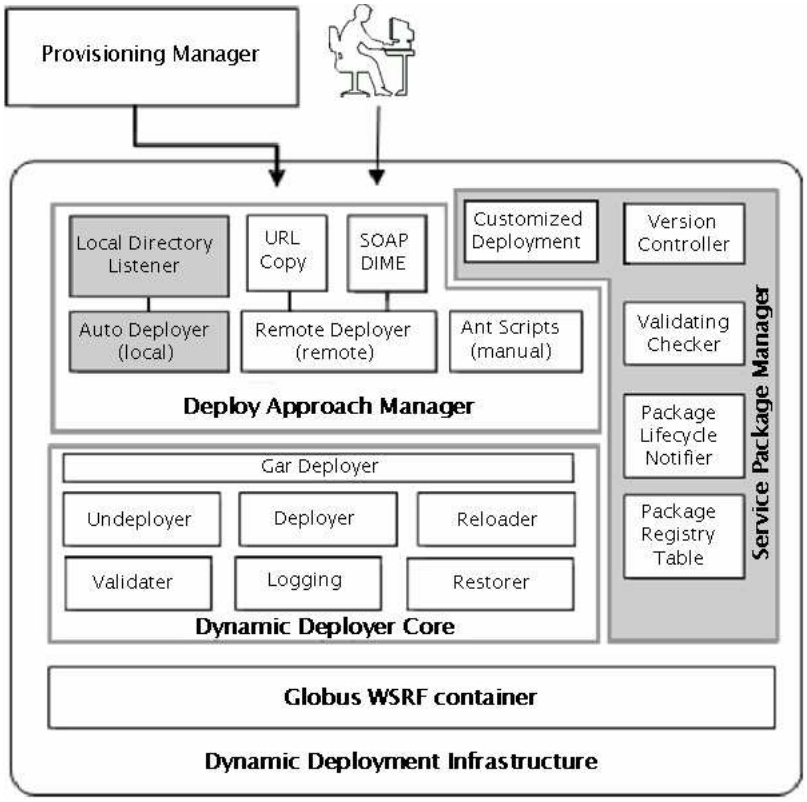

FIGURA 3.3 – Módulos de implantação dinâmica

- *Dynamic Deployer Core* (DDC): é a parte principal na realização da implantação dinâmica. DDC é o responsável por ativar e desativar os serviços e *containers* e, além disso, por atualizar o contexto da JVM (*Java Virtual Machine*). Como apresentado na Figura 3.3, DDC possui sete componentes, sendo que *GAR deployer* é o responsável pelas ações de implantação. *Undeployer*, *reloader* e *deployer* são responsáveis por ações de recarga que são passadas do DAM para o mecanismo de implantação. As ações de recarga podem atualizar bibliotecas de serviços e arquivos de configuração. O componente *validator* é o responsável por verificar a corretude dos arquivos GAR que estão sendo implantados, prevenindo a ocorrência de falhas. Já o componente *logging* é responsável por armazenar informações sobre operações

de execução e recarga. Finalmente, o componente *restorer* é um mecanismo de *backup* que, em caso de erro, ajuda o *container* a retornar ao estado anterior.

- *Deploy Approach Manager* (DAM): recebe como entrada um arquivo GAR, o qual consiste de um descritor de implantação (*Web Service Deployment Descriptor* – WSDD), de um descritor de recursos WSRF (*Web Service Definition Language* – WSDL) e da aplicação propriamente dita. DAM fornece três mecanismos de implantação: (a) *Auto Deployer* permite a implantação e remoção de um GAR de um *container* simplesmente movendo-o para um diretório específico; (b) *Ant Scripts* têm como objetivo a implantação e remoção de GARs; (c) *Remote Deployer* é um serviço que segue o padrão WSRF e permite implantação remota, provendo operações de *upload, download, deploy, undeploy* e *reload*.

- *Service Package Manager* (SPM): é um serviço opcional no HAND que possui características de gerenciamento em alto nível. Os componentes *Package Lifecycle* e *Package Registry Table* são os responsáveis por manter informações necessárias para a implementação em nível de serviços. Os componentes restantes são: (a) *Version Control*, responsável por controlar as versões dos serviços; (b) *Customized Deployment*, permite que usuários remotos submetam seu próprios *scripts* de implantação; e (c) *Validating Checker*, que é similar ao *Validator* do DDC, responsável por tarefas mais complexas de avaliação de dependências e conflitos entre os serviços.

O HAND, através de mudanças no núcleo do Tomcat, possui implementação mais leve e permite a implantação em nível de serviços. Ao ampliar a granularidade de implantação, os autores obtiveram como resultado uma infra-estrutura capaz de lidar com ambientes altamente dinâmicos sem comprometer disponibilidade e capacidade de atendimento de requisições. Contudo, HAND carece de mecanismos para lidar com situações adversas, como falhas no processo de instanciação da aplicação.

3.4 Sumário

Este capítulo apresentou os principais trabalhos correlatos. A Seção 3.1 apresentou uma solução segura para implantação dinâmica de serviços WSRF. Já a seção 3.2 apresentou um arcabouço para adaptação dinâmica de serviços em grades, com a restrição de atuar em nível de *container*. Na Seção seguinte (Seção 3.3),

apresentou-se o HAND, que propõe uma infra-estrutura para permitir implantação de serviços em ambientes dinâmicos de grades e possibilita a implantação em nível de *container*.

Como pôde ser observado, os avanços realizados na área se concentraram em prover mecanismos para interagir individualmente com estações-alvo, oferecendo uma forma sistemática e interoperável para instanciar componentes de uma aplicação de grade. Contudo, não abordaram como orquestrar a implantação de uma aplicação completa, envolvendo diversos recursos e serviços distribuídos, tampouco se preocuparam em lidar com situações adversas tais como indisponibilidade de estações selecionadas para implantação e erros de configuração – foco da arquitetura apresentada neste trabalho.

Capítulo 4

Arquitetura AGrADC

Este capítulo descreve a arquitetura proposta para a instanciação de aplicações de grades computacionais. Inicialmente, na Seção 4.1, é apresentada uma visão geral da arquitetura, enunciando seus elementos e os fluxos de informação existentes entre eles. Na Seção 4.2 descreve-se como deve ser realizada a especificação de cenários de instanciação. Na Seção 4.3 detalha-se o funcionamento dos componentes da arquitetura, enfatizando a funcionalidade fornecida pelo motor de instanciação – principal componente da AGrADC – para propiciar auto-configuração e auto-recuperação no processo de instanciação de aplicações. Por fim, na Seção 4.4 é apresentado um sumário do capítulo.

4.1 Visão Geral

AGrADC (Autonomic Grid Application Deployment & Configuration Architecure) consiste em uma arquitetura para instanciação de aplicações de grade que permite que a infra-estrutura necessária para sua execução seja implantada, configurada e gerenciada sob demanda. O objetivo é propiciar que o arcabouço de software necessário para a execução de uma aplicação de grade seja instanciado no momento de sua invocação. Através da arquitetura proposta, ao desenvolvedor são oferecidas maneiras para definir (a) um fluxo de implantação, respeitando dependências entre componentes que compõem a aplicação, (b) parâmetros de configuração e (c) ações a serem executadas diante de situações adversas tais como falhas.

A arquitetura AGrADC é composta por quatro elementos: (a) *aplicação de gerenciamento*, (b) *repositório de componentes*, (c) *motor de instanciação* e (d) *serviços de instanciação*. A Figura 4.1 ilustra uma visão geral da

arquitetura instanciada em uma infra-estrutura de grade composta por três domínios administrativos A, B e C.

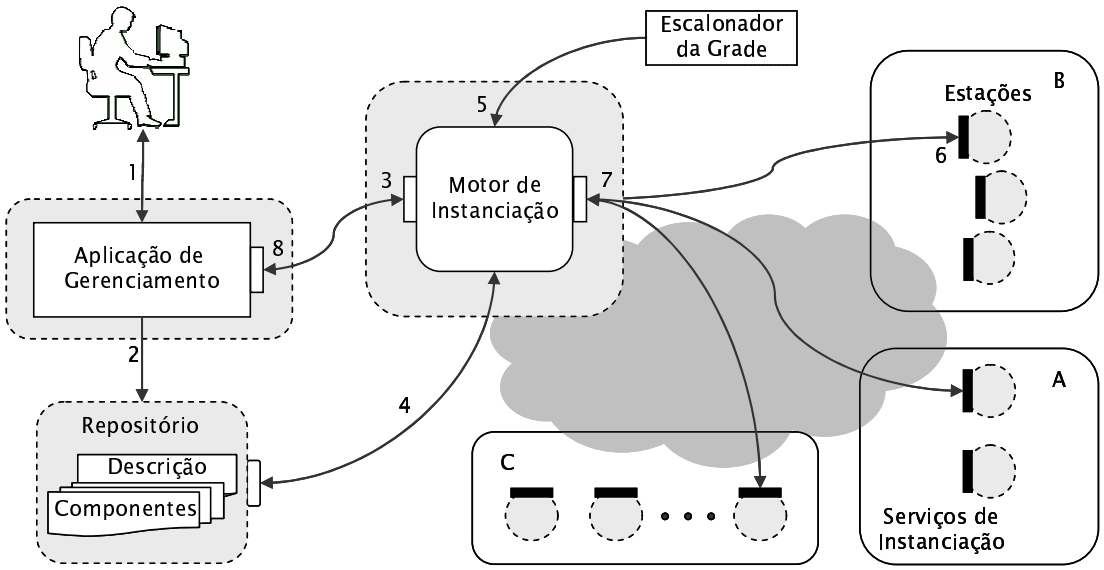

FIGURA 4.1 – Arquitetura AGrADC e interação entre seus elementos

A aplicação de gerenciamento permite que o desenvolvedor defina os componentes da aplicação, especifique o fluxo de implantação e configuração, bem como requisite a instanciação da aplicação na infra-estrutura de grade. Os componentes e os roteiros de implantação são armazenados em um repositório. O motor de instanciação tem por função receber invocações da aplicação de gerenciamento e orquestrar a instanciação do ambiente de execução solicitado. Por fim, os serviços de instanciação – hospedados em todas as estações da grade – fornecem interfaces que as tornam aptas a executar implantação, configuração e gerenciamento de componentes.

A interação entre os elementos da arquitetura se dá da seguinte forma. Inicialmente, o desenvolvedor define – usando a linguagem CDL – os componentes que fazem parte de sua aplicação (ex: banco de dados, servidor http e *grid service*), a seqüência de implantação a ser respeitada e os parâmetros de configuração (fluxo 1 na Figura 4.1). O resultado desta etapa é a geração de um conjunto de arquivos CDL e componentes, que são armazenados no repositório (fluxo 2). A próxima etapa consiste na solicitação de instanciação da referida aplicação ao motor de instanciação (3), que é acompanhada do identificador da localização do arquivo de descrição da aplicação. Ao receber a solicitação, o motor de instanciação recupera este arquivo (4), interpreta-o e inicia o processo de instanciação.

À medida que o motor identifica os componentes no arquivo de descrição, os mesmos vão sendo recuperados do repositório (fluxo 4 na Figura 4.1). Com base nas informações fornecidas pelo escalonador da grade (5) em relação aos recursos disponíveis, o motor determina em que estações cada componente será instanciado e interage com os serviços de instanciação das estações escolhidas (6). A interação prevê operações para implantar, configurar e gerenciar os componentes. O resultado dessas operações – sucesso ou falha – é informado ao motor mediante notificações geradas pelos serviços de instanciação (7). A partir de políticas expressas junto ao motor (explicadas na Seção 4.3), o mesmo reage autonomamente avançando o processo de instanciação ou executando um procedimento de contorno. Por fim, ao término do processo de instanciação, o motor notifica a aplicação de gerenciamento sobre o resultado (8). Neste momento, se o processo resultou bem sucedido, o ambiente de execução da grade está pronto para executar a aplicação.

A representação de cenários de instanciação é crucial para a arquitetura, considerando a diversidade de aplicações e, sobretudo, de requisitos de execução demandados pelas mesmas. Sem os cenários (e a infra-estrutura para instanciá-los dinamicamente), as aplicações ficariam restritas aos serviços disponíveis previamente nas estações da grade ou exigiriam um grande esforço operacional (e *ad-hoc*) para preparar o ambiente desejado. A próxima seção aborda a funcionalidade provida pela linguagem CDL para especificação dos referidos cenários.

4.2 Representação de Cenários de Instanciação

A descrição de um cenário de instanciação é realizada com o uso da linguagem de descrição de configuração (CDL) e consiste na identificação dos componentes e seus parâmetros de configuração, bem como na determinação do fluxo de implantação a ser respeitado. Antes de detalhar a funcionalidade da linguagem, um exemplo de cenário é apresentado gráfica e textualmente nas Figuras 4.2 e 4.4, respectivamente. O cenário é composto por três componentes: (a) base de dados (*Hsqldb*); (b) servidor de aplicações Java para *web* (*Tomcat*); e (c) aplicação de grade (*GridApp*). O exemplo ilustra as dependências entre esses componentes, indicando que *Hsqldb* e *Tomcat* podem ser instanciados em paralelo, e somente ao término desse passo *GridApp* pode ser instanciada.

A representação visual ilustra o grafo de dependências entre os componentes, onde as setas contínuas indicam a ordem de implantação, as setas pontilhadas indicam eventos ou referências a parâmetros de configuração e os números indicam

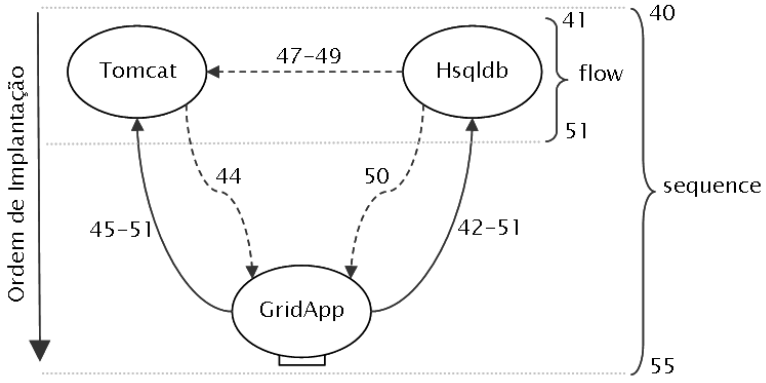

FIGURA 4.2 – Representação gráfica de um cenário de instanciação

a linha ou o intervalo onde a descrição é encontrada na especificação textual. A especificação em CDL, por sua vez, define os componentes que formam a aplicação de grade, os seus parâmetros e o fluxo de instanciação do cenário. A especificação é organizada em dois blocos: (a) definição e configuração de componentes (`cdl:configuration`) e (b) definição da ordem de instanciação dos componentes (`cdl:system`), explicados a seguir.

No bloco de definição e configuração (Figura 4.4, linhas 9 a 37) são definidos os componentes da aplicação de grade, os quais serão associados, posteriormente, aos recursos disponíveis. Os componentes são definidos em blocos XML e a cada um deles são associados os parâmetros de configuração. Além de representar serviços (ex: *Tomcat*, linhas 17 a 22, e *Hsqldb*, linhas 23 a 30), os componentes podem representar parâmetros a serem compartilhados por outros componentes, como é o caso de *DBConnection*, linhas 10 a 16.

O número de parâmetros de cada um dos componentes é peculiar às suas necessidades. São três as maneiras de associar valores para cada um dos parâmetros dos componentes: (a) definir o valor no momento da declaração do parâmetro (ex: linha 20, `<port> 8080 </port>`); (b) informar que o valor será definido no momento da instanciação (ex: linha 19, `<hostname cdl:lazy="true"/>`); ou (c) determinar que o valor do parâmetro deve ser aquele associado à referência indicada (ex: linha 47, `<dbport cdl:ref="Hsqldb:/port"/>`). Neste último exemplo, determina-se que o componente *Tomcat* herdará do componente *Hsqldb* a configuração da porta em que o banco de dados estará disponível.

No bloco de definição da ordem de instanciação dos componentes, a ordenação é determinada pela hierarquia com que os componentes são declarados no bloco

`cdl:system` (Figura 4.4, linhas 39 a 61). A hierarquia entre os componentes é estabelecida através das seguintes regras.

- *Sequence*: indica que os componentes devem ser instanciados de acordo com a ordem léxica. Por exemplo, nas linhas 40 a 55 é especificado que a instanciação dos componentes *Tomcat* e *Hsqldb* deve ser finalizada antes do processo de instanciação de *GridApp* ser iniciado.

- *Inverse*: operação inversa da seqüência, é empregada para indicar a ordem de remoção dos componentes em um processo de *Undeploy* (linhas 56 a 60).

- *Flow*: é utilizada quando não é imposta uma ordem na instanciação entre componentes, ou seja, não existem dependências. Na Figura 4.4, linhas 41 a 51, *Tomcat* e *Hsqldb* podem ser instanciados em qualquer ordem.

- *Switch*: permite que o fluxo de instanciação seja alterado, dependendo de valores associados a variáveis de condição. A Figura 4.3 ilustra um trecho CDL que descreve a condição de escolha entre instanciar o componente `Apache2` ou o componente `Tomcat5`. Na condição expressa na linha 2 (`cdl:ref=“DB:/port”=“80”`), caso o parâmetro *port* do componente *DB* tenha o valor igual a *80*, o componente *Apache2* deve ser instanciado, caso contrário, o componente *Tomcat5* será instanciado (linhas 5 a 7).

```
1 <cmp:switch lifecycle=''initialization''>
2    <cmp:case cdl:ref=''DB:/port''=''80''>
3       <Apache2 />
4    </cmp:case>
5    <cmp:otherwise>
6       <Tomcat5 />
7    </cmp:otherwise>
8 </cmp:switch>
```

FIGURA 4.3 – Representação textual do construtor *switch*

```
<?xml verson="1.0" encoding="UTF-8"?>
<cdl:cdl xmlns="http://cddlm.org/gridApp"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns:cdl="http://www.gridforum.org/namespaces/2005/02/cddlm/CDL-1.0"
         xmlns:t="http://cddlm.org/component-model-example"
         xmlns:cmp="http://www.gridforum.org/cddlm/components/2005/02"
         targetNamespace="http://cddlm.org/gridApp">

  <cdl:configuration>
      <DBConnection>
         <JNDIName />
         <hostname />
         <dbport />
         <username />
         <password />
      </DBConnection>
      <Tomcat cdl:extends="c:Component">
         <data>http://cddlm.unisinos.br/repository/Tomcat-5.0.gar </data>
         <hostname cdl:lazy="true" />
         <port>8080</port>
         <dbconnection cdl:extends="DBConnection" />
      </Tomcat>
      <Hsqldb cdl:extends="c:Component">
         <cmp:CommandPath>org.cddlm.service.Hsqldb</cmp:CommandPath>
         <data>http://cddlm.unisinos.br/repository/Hsqldb.gar </data>
         <hostname cdl:lazy="true" />
         <port>3306</port>
         <username>userDB</username>
         <password>passwordDB</password>
      </Hsqldb>
      <GridApp>
         <t:application>http://cddlm.unisinos.br/repository/GridApp.gar</t:application>
         <t:applicationPath>/GridApp</t:applicationPath>
         <t:dbname>jdbc/Hsqldb</t:dbname>
         <t:hostname />
      </GridApp>
   </cdl:configuration>

   <cdl:system>
      <cmp:sequence lifecycle="initialization">
         <cmp:flow lifecycle="initialization">
            <Hsqldb>
               <hostname cdl:lazy="true" />
            </Hsqldb>
            <Tomcat>
               <hostname cdl:lazy="true" />
               <dbport cdl:ref="Hsqldb:/port" />
               <username cdl:ref="Hsqldb:/dbuser" />
               <password cdl:ref="Hsqldb:/dbpassword" />
            </Tomcat>
         </cmp:flow>
         <GridApp>
            <hostname cdl:lazy="true" />
         </GridApp>
```

```
55          </cmp:sequence>
56          <cmp:reverse lifecycle="terminate">
57              <GridApp />
58              <Tomcat />
59              <Hsqldb />
60          </cmp:reverse>
61      </cdl:system>
62  </cdl:cdl>
```

FIGURA 4.4 – Representação textual de um cenário de instanciação

4.3 Elementos da Arquitetura

Esta seção apresenta, com mais detalhes, os quatro elementos da arquitetura AGrADC .

4.3.1 Aplicação de Gerenciamento

A aplicação de gerenciamento permite ao desenvolvedor definir os componentes da aplicação e especificar o fluxo de implantação e configuração (exemplo na Figura 4.4), bem como requisitar a instanciação da aplicação na infra-estrutura de grade. Além disso, é através dela que os componentes e os roteiros de implantação são enviados para o repositório de componentes. A aplicação de gerenciamento também oferece suporte para que componentes e descrições sejam carregados e salvos localmente, facilitando assim a sua reutilização.

Através de mecanismos de análise da corretude da sintaxe da descrição, a aplicação de gerenciamento permite que erros de descrição sejam detectados ainda na fase de desenvolvimento da aplicação. Uma vez finalizada a descrição e de posse da implementação dos componentes da aplicação, o processo de encapsulamento é iniciado. O próximo passo é definir o identificador (*endereço*) do motor de instanciação e iniciar o processo de instanciação dos componentes da aplicação na infra-estrutura de grade disponível. Apesar da aplicação de gerenciamento não atuar como orquestradora do processo de instanciação (tarefa desempenhada pelo motor de instanciação, vide Sub-seção 4.3.3), ela oferece ao proprietário da aplicação a possibilidade de visualizar a evolução do processo de instanciação.

A aplicação de gerenciamento busca auxiliar o usuário da grade no desenvolvimento de sua aplicação. Ela funciona como um *front-end* que simplifica as tarefas atualmente necessárias à formação de ambientes para a execução de aplicações de grade, bem como para o desenvolvimento dos seus componentes.

A aplicação de gerenciamento interage com o motor de instanciação através da invocação da operação `Deploy`, usada para requisitar a instanciação de uma aplicação, e `Undeploy`, utilizada para cancelar o processo de instanciação. Outras operações também são oferecidas, permitindo que propriedades do motor e da aplicação implantada sejam consultadas. `GetResourceProperties` pode ser usada pela aplicação para recuperar o valor de uma propriedade específica (ex: `state`). GetMultipleResourceProperties, por sua vez, permite que o valor de múltiplas propriedades sejam obtidas com somente uma invocação. Da mesma forma, é possível realizar assinaturas por propriedades (`Subscribe(<Property>)`). Assim que o valor associado à propriedade em questão mudar, a aplicação de gerenciamento recebe uma notificação da mudança contendo os valores antigo e novo.

Do modo como a arquitetura foi proposta, não existe a diferenciação entre quem utiliza a aplicação de gerenciamento para desenvolver a aplicação de grade e quem a utiliza somente para invocar sua instanciação. Em ambientes onde existe essa diferença, seria interessante incorporar à aplicação de gerenciamento mecanismos que permitam definir e assegurar papéis aos usuários. Os usuários seriam associados aos papéis de acordo com as ações autorizadas. O modelo RBAC (*Role-Based Access Control*) poderia ser utilizado para modelar os referidos papéis.

4.3.2 Repositório de Componentes

O repositório de componentes é, como o próprio nome indica, o local onde são armazenados os componentes e as descrições das aplicações, em linguagem CDL. O objetivo de usar um repositório é estimular o compartilhamento de componentes, evitando assim esforços redundantes e desnecessários e diminuindo o tempo de desenvolvimento de aplicações. O uso de especificações padrão (ex: CDDLM) para descrever os componentes, suas configurações e suas dependências é um fator positivo que agrega à AGrADC características como interoperabilidade e compartilhamento de informações. Dependendo da sua finalidade e especificidade, o repositório usado pode ser de domínio público ou de acesso restrito.

À medida que o desenvolvedor descreve os componentes, suas configurações e dispõe de sua implementação, estes são encapsulados e armazenados no repositório de componentes. O mesmo acontece com a descrição da aplicação de grade, que é armazenada no mesmo repositório. Os componentes podem ser recuperados pela aplicação de gerenciamento para modificação, bem como pelo motor de instanciação, no momento da implantação da aplicação.

4.3.3 Motor de Instanciação

O motor de instanciação consiste em um serviço responsável por orquestrar a instanciação da infra-estrutura de software necessária para executar aplicações de grade. O motor dispensa a intervenção humana no processo de implantação, configuração e gerenciamento dos componentes que formam as aplicações, suprindo uma lacuna crítica até então não abordada em trabalhos anteriores.

Invocado pela aplicação de gerenciamento, o motor oferece dois métodos: *Deploy* e *Undeploy*. Acompanha a invocação do primeiro o identificador da localização do arquivo de descrição da aplicação a ser implantada, enquanto o segundo método é seguido da EPR (*End-Point Reference*) da aplicação a ser removida. No outro extremo, o motor de instanciação lança mão dos métodos fornecidos pela *Deployment* API, introduzida na Seção 2.3, para interagir com os serviços de instanciação.

A instalação de uma aplicação consiste na instanciação dos seus componentes, respeitando a ordem informada no arquivo de descrição. A instanciação de cada componente, por sua vez, é executada através dos passos descritos na Seção 2.3 e ilustrados na Figura 4.7. Para conduzir, de maneira controlada, esse processo complexo, o motor mantém uma máquina de estados para cada componente, que reflete a situação atual da instanciação de cada um deles.

A Figura 4.5 ilustra um *snapshot* do processo de instanciação da aplicação introduzida anteriormente na Seção 4.2. Observe que três máquinas de estados são utilizadas, sendo que aquela que controla a instanciação do componente *Hsqldb* aparece em destaque. O referido componente já foi carregado (*uploaded*) na estação-alvo e configurado. No momento o motor acaba de invocar o método *run()* e recebe, de forma assíncrona, uma mensagem de notificação do tipo *RunFault* informando que a operação não foi bem sucedida. Concomitantemente à tentativa de instanciação do componente *Hsqldb*, o motor está executando o mesmo procedimento para o componente *Tomcat*, já que a especificação CDL autoriza tal paralelismo.

Os estados em que se encontram os componentes e as mensagens de notificação, recebidas em resposta às assinaturas feitas por propriedades desses componentes, podem ser usados para expressar como o motor de instanciação deve se comportar diante de situações adversas. Tal comportamento – especificado pelo desenvolvedor da aplicação ou pelo gerente da infra-estrutura de grade mediante *políticas de ação* [Kephart and Walsh, 2004] – confere à arquitetura características da Computação Autônoma tais como auto-configuração e auto-recuperação.

Políticas de ação ditam ações que devem ser executadas sempre que o

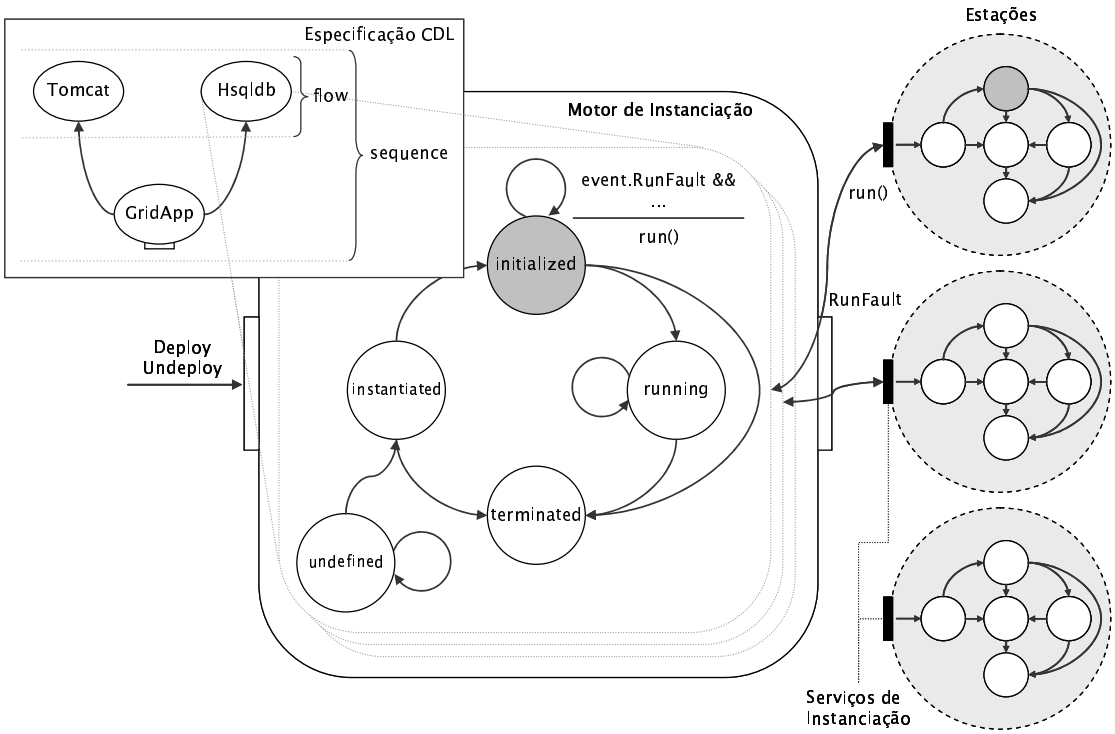

FIGURA 4.5 – Máquina de estados do motor de instanciação

sistema se encontra em determinado estado, sendo representadas na forma *ON (Estado) IF (Condição) THEN (Ação)*. Para que o motor de instanciação apresente comportamento racional, as políticas devem cobrir cada estado relevante definindo ações de acordo com condições pré-estabelecidas. A Figura 4.6 ilustra um conjunto de políticas definidas para reger o comportamento do motor.

```
1  ON UndefinedState {
2
3    IF (event.NonAvailableHost)
4    THEN terminate()
5
6    IF (event.HostUnreachable && event.HostUnreachable:ContFailed < 3)
7    THEN create()
8
9    IF (event.HostUnreachable && event.HostUnreachable:ContFailed >= 3)
10   THEN terminate()
11
12   IF (event.DeploymentFault && event.DeploymentFault:ContFailed < 2)
13   THEN create()
14
15   IF (event.DeploymentFault && event.DeploymentFault:ContFailed >= 2)
16   THEN terminate()
17
18    ...
```

```
19 }
20
21 ON InstantiatedState {
22
23   IF (event.ConfigurationFault && event.ConfigurationFault:ContFailed < 3)
24   THEN initialize()
25
26   IF (event.ConfigurationFault && event.ConfigurationFault:ContFailed >= 3)
27   THEN terminate()
28
29   ...
30 }
31
32 ON InitializedState {
33
34   IF (event.RunFault && event.RunFault:ContFailed < 3)
35   THEN run()
36
37   IF (event.RunFault && event.RunFault:ContFailed >= 3)
38   THEN terminate()
39
40   ...
41 }
42
43 ON RunningState {
44
45   IF (event.TestFault && event.TestFault:ContFailed < 3)
46   THEN ping()
47
48   IF (event.TestFault && event.TestFault:ContFailed >= 3)
49   THEN terminate()
50
51   ...
52 }
```

FIGURA 4.6 – Representação de políticas

As políticas ilustradas cobrem os quatro principais estados, de um total de cinco, do ciclo de vida de um componente: *undefined* (linhas 1 a 19), *instantiated* (linhas 21 a 30), *initialized* (linhas 32 a 41) e *running* (linhas 43 a 52). Por exemplo, quando o motor procura implantar um componente (estado *undefined*), algumas situações podem ocorrer:

- o escalonador pode não ter estação a oferecer (*NonAvailableHost*), condição que faz com que o motor interrompa a implantação do componente (*terminate()*) e, por conseqüência, a instanciação de toda a aplicação;
- o motor pode não conseguir contactar a estação determinada pelo escalonador (*HostUnreachable*), condição que leva o motor a tentar implantar o componente novamente na mesma ou em uma estação alternativa;

- o motor pode enfrentar dificuldades na interação com o serviço de instanciação na implantação dos arquivos referentes ao componente que está sendo instanciado na estação (*DeploymentFault*), condição que induz o motor a repetir a tentativa.

Observe que o estado que deve ser alcançado pela execução de uma determinada ação não é explicitamente especificado em políticas de ação [Kephart and Walsh, 2004]. Assume-se que o autor das políticas o conhece. No caso do motor de instanciação, a execução bem sucedida de uma ação faz com que a instanciação do componente avance ao estado seguinte, considerando a máquina de estados apresentada na Figura 4.5. Por outro lado, execuções mal sucedidas de uma ação fazem com que ou não haja avanço de estado (no caso de tentativas de procedimentos de contorno) ou o componente avance para o estado *terminated*.

4.3.4 Serviços de Instanciação

Os serviços de instanciação são hospedados em todas as estações da grade (estações-alvo), fornecendo interfaces que as tornam aptas a receber requisições para executar o processo de implantação e configuração de componentes. Além disso, essas interfaces permitem que seja realizado o gerenciamento dos componentes implantados.

Através das interfaces disponibilizadas pelos serviços de instanciação, os métodos que compõem o processo de instanciação dos componentes são invocados (`create()`, `initialize()`, `run()` e `teminate()`). À medida em que os métodos vão sendo invocados pelo motor de instanciação, o componente sendo implantado muda seu estado, respeitando a sequência de transições conforme ilustrado na Figura 4.7. Instanciação e inicialização representam a implantação (*upload*) e a configuração do componente, respectivamente. Quando no estado *running*, o componente encontra-se funcional; ao ocorrer uma falha em qualquer momento do ciclo de vida do componente, seu estado passa a ser *failed*. Por fim, *terminated* é o estado final de um componente. Mudanças bem sucedidas de estado ou problemas enfrentados ao longo do processo dão origem a mensagens de notificação, seguindo um padrão como o *WS-Notification* [Graham et al., 2005].

Os serviços de instanciação geram notificações que permitem ao motor monitorar e atuar no processo de instanciação dos componentes. As notificações listadas na Figura 4.6 são: (a) `DeploymentFault`, indica a ocorrência de uma falha na transição para o estado `Instantiated`; (b) `ConfigurationFault`, indica que o processo de configuração não foi concluído com sucesso; (c) `RunFault`, indica a

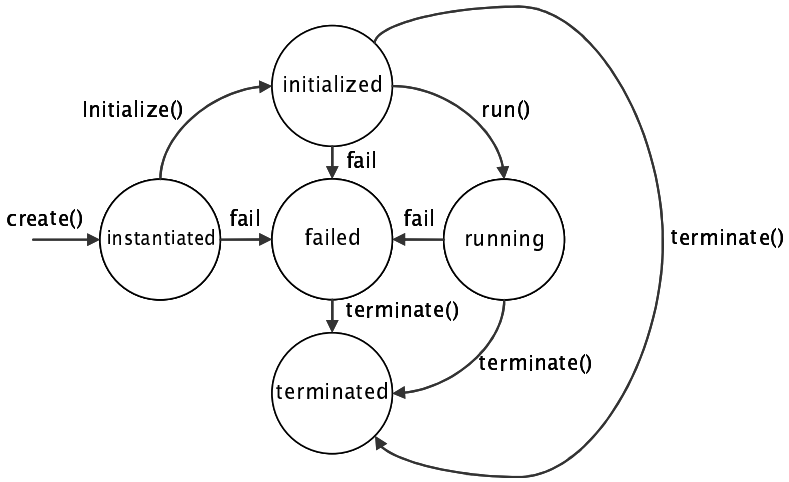

FIGURA 4.7 – Máquina de estados do ciclo de vida de um componente

ocorrência de erro na tentativa de executar a aplicação de grade; e (d) `TestFault`, indica que os mecanismos de teste detectaram algum tipo de anomalia na execução de uma aplicação, por exemplo, excesso de tempo na execução de uma tarefa.

4.4 Sumário

Este capítulo apresentou a arquitetura proposta. A Seção 4.1 descreveu uma visão geral da arquitetura AGrADC citando seus elementos, suas principais funcionalidades e características. Além disso, foram descritas as interações entre os elementos da arquitetura.

A Seção 4.2 mostrou como é realizada a descrição de um cenário de instanciação usando a linguagem de descrição de configuração (CDL). Na seqüência (Seção 4.3) foram detalhados os elementos que compõem a arquitetura, com ênfase no motor de instanciação, elemento responsável por orquestrar o processo de instanciação de ambientes de execução.

A arquitetura AGrADC possui duas contribuições principais. Primeiro, define um serviço *inédito* para conduzir o processo de instanciação de aplicações em grades – o motor de instanciação – que incorpora características da Computação Autônoma, sendo capaz de executar procedimentos de contorno para lidar com problemas enfrentados ao longo do processo. Segundo, a arquitetura de software projetada está plenamente alinhada com especificação proposta para a área, CDDLM.

Capítulo 5

Implementação

Um protótipo da arquitetura proposta foi desenvolvido com o objetivo de avaliar sua viabilidade técnica. Este capítulo descreve aspectos relacionados com a implementação desse protótipo, incluindo informações sobre cada um dos elementos que compõe a arquitetura AGrADC. A Seção 5.1 descreve a aplicação de gerenciamento, que oferece funcionalidades para o desenvolvimento e a gerência de aplicações de grades, e o repositório de componentes. Na Seção 5.2 detalha-se a implementação do motor de instanciação, que é o reponsável por orquestrar o processo de instanciação. Por fim, a Seção 5.3 descreve o serviço de instanciação que foi adaptado segundo as especificações CDDLM.

5.1 Aplicação de gerenciamento e repositório de componentes

A aplicação de gerenciamento foi implementada na linguagem Java, permitindo descrever os componentes que compõem uma aplicação de grade (e suas dependências) e verificar, com o apoio da API JDOM [JDOM, 2006], a correção dos arquivos de descrição XML-CDL [Tatemura, 2005] resultantes.

A Figura 5.1 ilustra a GUI da aplicação de gerenciamento, criada com o objetivo de facilitar o trabalho do desenvolvedor. Nela é possível criar, carregar, editar e salvar descrições de componentes, seus parâmetros e suas dependências. A aplicação de gerenciamento permite, ainda, gerar pacotes GAR (*Grid Archive*) de componentes – formato compatível com o Globus *toolkit* 4 [Globus, 2006] – e acompanhar a evolução do processo de instanciação da aplicação. A GUI da aplicação de gerenciamento é dividida em três regiões, descritas a seguir.

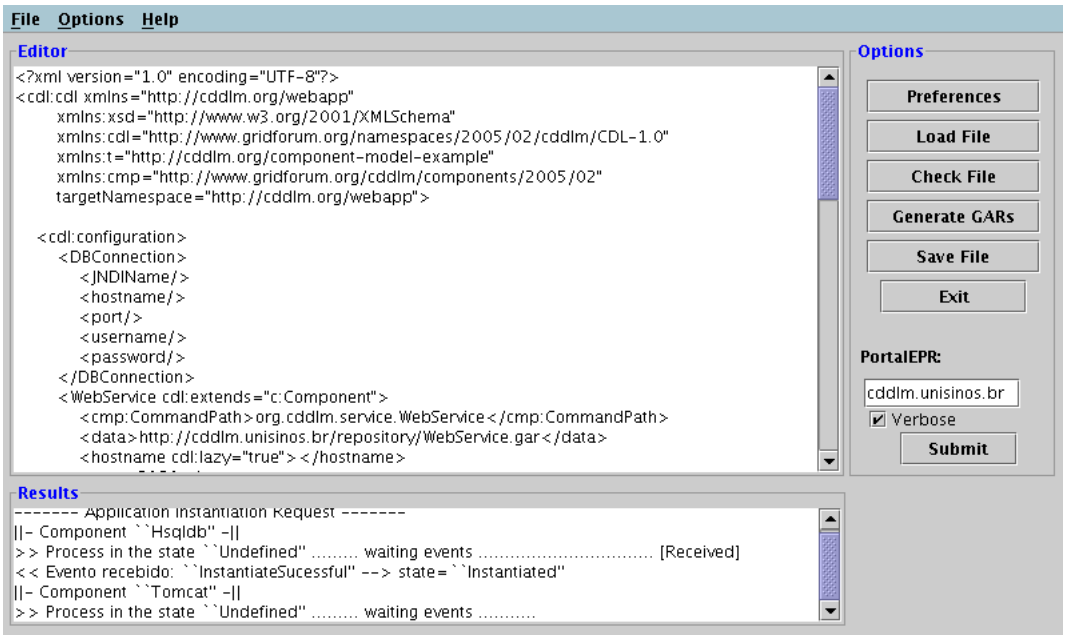

FIGURA 5.1 – Captura de tela da Aplicação de Gerenciamento

A região `Editor` fornece a funcionalidade de criação e edição de descrições CDL. Já a região `Options` fornece funcionalidades relacionadas ao arquivo de descrição, como carregar, salvar e verificar a sua conformidade com a linguagem CDL. Também é possível configurar o caminho do repositório e da implementação dos componentes, bem como gerar os seus pacotes. Ademais, o endereço do motor de instanciação pode ser definido, bem como o nível de detalhes desejado no acompanhamento do processo de instanciação. Por fim, a região `Results` mostra a evolução do processo de instanciação.

O desenvolvimento da aplicação é iniciado pela criação ou edição da descrição dos componentes e do fluxo de implantação (opções `Load File` e `Save File`). A corretude desse passo é verificada pela opção que permite verificar se a descrição está de acordo com a linguagem CDL (`Check File`). O próximo passo é especificar nas preferências da aplicação o local onde é encontrada a implementação dos componentes e o endereço do repositório de componentes (`Preferences`). A opção `Generate File` gera os pacotes GAR dos componentes, que são salvos no repositório de componentes. De posse da descrição da aplicação e dos pacotes GAR, o próximo passo é definir o endereço do motor de instanciação e escolher o nível de detalhes na visualização dos resultados, na opção `verbose`.

A aplicação de gerenciamento implementa um *listener* para receber as notificações, que são mostradas na região `Results`. Nessa região, na Figura 5.1, é

ilustrado o processo de instanciação do componente `Tomcat`, o qual inicia no estado `Undefined` e, ao receber a notificação de sucesso na transição de estados, passa para `Instantiated`.

Os arquivos de descrição e os componentes são armazenados em um servidor de aplicações *web* (*Apache Tomcat*), o repositório de componentes. Este permite o *download* e *upload* de componentes e descrições de ambientes. Além disso, implementa restrições de acesso que permitem ao desenvolvedor escolher quais componentes compartilhar e com quem compartilhar.

5.2 Motor de Instanciação

O motor de instanciação tem como característica principal a conformidade com a especificação CDDLM. Também desenvolvido em Java, o motor disponibiliza interfaces para a interação com a aplicação de gerenciamento e com os serviços de instanciação. A interface oferecida à aplicação de gerenciamento permite invocar e acompanhar a instanciação de aplicações seguindo a especificação WSDM (*Web Services Distributed Management*) [Bullard and Vambenepe, 2005], tendo sido desenvolvida usando o arcabouço Muse [Muse, 2006], versão 2.0, da *Apache Software Foundation*. Para interagir com os serviços de instanciação, a atual versão do protótipo utiliza a API de implantação fornecida pelo Globus *toolkit* versão 4. O motor implementa, ainda, um *listener* para receber mensagens de notificação geradas pelos serviços de instanciação.

O processo de instanciação, orquestrado pelo motor, é baseado no arquivo de descrição CDL. A API JDOM é empregada para ler, interpretar e armazenar em memória as descrições das aplicações. Além de descrever os componentes e seus parâmetros, a descrição CDL também expressa as dependências entre os componentes da aplicação de grade. Respeitando essas dependências, o motor implementa uma máquina de estados (conforme ilustrado na Figura 4.5) para cada componente.

As transições na máquina de estados são regidas pela invocação de operações. Essas retornam notificações, que são utilizadas para o acompanhamento da evolução do processo de instanciação. Se o método de transição foi executado com sucesso, uma notificação é gerada para sinalizar a troca de estado. Já na ocorrência de falhas, as notificações geradas podem ser utilizadas para recuperar a falha (ex: DeploymentFault, Figura 4.6 linha 12). Tais mensagens de notificação são tratadas por políticas, como aquelas apresentadas na Figura 4.6, que indicam o estado, a

condição e o método que deve ser invocado para tratá-los.

A implementação das políticas é feita através de um arquivo de propriedades, o que permite que políticas sejam criadas, modificadas e removidas dinamicamente sem a necessidade de recompilar o motor de instanciação. A representação é feita da seguinte forma: `<estado_atual>.<notificação>=<ação>`. A Figura 5.2 ilustra algumas das políticas descritas em um arquivo de propriedades, sendo estas lidas pelo motor através de métodos Java específicos para o tratamento de arquivos de propriedades. À medida que as notificações chegam, são concatenados no motor o estado atual e a notificação que ocorreu, gerando o nome da propriedade. Por exemplo, quando no estado `UndefinedState`, se a notificação `NonAvailableHost` é recebida pelo motor, a concatenação (separada pelo caracter ".") resulta na propriedade `UndefinedState.NonAvailableHost`, como pode ser visto na linha 5.

```
1  # ''0'' sem contador associado
2  # ''m3'' indica contador menor que 3
3  # ''M3'' indica contador maior ou igual a 3
4
5  UndefinedState.NonAvailableHost=terminate(0)
6  UndefinedState.HostUnreachable=create(m3)
7  UndefinedState.HostUnreachable=terminate(M3)
8  UndefinedState.DeploymentFault=create(m2)
9  UndefinedState.DeploymentFault=terminate(M2)
10 ...
11
12 InstantiatedState.ConfigurationFault=initialize(m5)
13 InstantiatedState.ConfigurationFault=terminate(M5)
14 ...
15
16 InitializedState.RunFault=run(m3)
17 InitializedState.RunFault=terminate(M3)
18 ...
19
20 RunningState.TestFault=ping(m3)
21 RunningState.TestFault=terminate(M3)
22 ...
```

FIGURA 5.2 – Arquivo de políticas

As ações associadas às mensagens de notificação são representadas como ilustrado na Figura 5.2. Às ações são associados os valores do contador de cada notificação. A representação do valor do contador é feita de três formas: (a) à ação é indicada a passagem do valor 0, o que indica a inexistência de contador para essa propriedade (ex: `terminate(0)`, linha 5); (b) `ação(mn)` indica que enquanto o valor do contador da notificação for menor que `n` essa ação deve ser executada pelo motor

(ex: na linha 8, `create(m2)` indica que enquanto o contador associado à notificação `DeploymentFault` for menor que 2, a ação `create` será executada); e (c) `ação(Mn)` indica que essa deve ser a ação escolhida quando o valor do contador for igual ou maior que `n`. O valor do contador associado a cada uma das notificações é gerenciado pelo motor de instanciação, lembrando que as mesmas notificações podem ocorrer em estados diferentes, sendo assim necessária a sua diferenciação.

Assim que o motor de instanciação recebe a invocação para instanciar uma aplicação, ele requisita ao repositório o arquivo de descrição que define os componentes. A partir do grafo formado pelas dependências entre os componentes, o motor requisita cada um dos componentes à medida que eles vão sendo habilitados no grafo de instanciação (`getComponent(name_component)`). O motor de instanciação rege o processo de instanciação invocando operações como `initialize()`, `create()`, `run()` e `terminate()`, entre outras. A implantação de um componente caracteriza-se basicamente pela transferência da sua implementação para a estação-alvo, através da operação `instantiate()`, e a transmissão dos arquivos, realizada via protocolo SOAP. Já o processo de configuração (operação `initialize()`) consiste na leitura do CDL da aplicação, localização da definição do componente em questão e análise dos componentes da aplicação e seus parâmetros. Concluído o processo de configuração, o componente disponibiliza os seus serviços (operação `run()`). O processo de implantação da aplicação tem seqüência, respeitando sempre as dependências entre os componentes. Uma vez executadas as tarefas destinadas à aplicação de grade, o resultado é enviado para a aplicação de gerenciamento e o ambiente de execução pode ser desfeito. O arquivo de descrição determina a ordem com que os componentes serão removidos (tag `<reverse>`). Assim, a infra-estrutura de grade utilizada é liberada para novas alocações.

5.3 Serviços de Instanciação

Os serviços de instanciação são serviços *web* que implementam as interfaces e os mecanismos previstos na especificação CDDLM (introduzidos na Seção 2.3). Atualmente encontram-se em desenvolvimento três implementações de referência deste serviço: uma no contexto do projeto BizGrid (por Fujitsu, NEC e Hitachi), uma pela empresa Softricity e outra em um esforço conjunto entre a HP e a UFCG. Como o foco deste trabalho reside em investigar técnicas para orquestrar a instanciação de aplicações completas, envolvendo recursos e serviços distribuídos, optou-se por não repetir esforços já em andamento, sem no entanto abrir mão

de propor solução alinhada com a referida especificação e os serviços por vir. Nesse contexto, adotou-se temporariamente o serviço de implantação fornecido pelo Globus *toolkit* versão 4, acrescido de funcionalidades que permitem a implantação, a configuração e o gerenciamento de componentes emulando a especificação CDDLM.

À medida que o processo de instanciação evolui, os serviços de instanciação geram notificações para o motor de instanciação. Em cada uma das estações-alvo, os serviços de instanciação mantêm uma máquina de estados para o componente que está sendo implantado, que corresponde aquela ilustrada na Figura 4.5.

Capítulo 6

Avaliação Experimental

Este capítulo apresenta a avaliação experimental realizada com o protótipo da arquitetura. AGrADCfoi implantada em um ambiente real de grade (executando Globus versão 4) com o objetivo de avaliar sua capacidade para (a) instanciar aplicações sob demanda, seguindo rigorosamente especificações de dependência e configuração, e (b) reagir autonomamente diante de situações adversas geradas sinteticamente. Além disso, procurou-se estimar – ainda que de maneira preliminar – os ganhos proporcionados por AGrADC no tempo de instanciação de aplicações em comparação com soluções mais convencionais, que não oferecem mecanismos para implantação e configuração autônomas.

6.1 Ambiente de Teste

Um ambiente de teste foi criado para avaliar a arquitetura e suas interações com um ambiente real de grade. A Figura 6.1 ilustra o ambiente, no qual quatro estações foram utilizadas para disponibilizar recursos à grade e uma para instalar a aplicação de gerenciamento, o repositório de componentes e o motor de instanciação. As estações pertencentes à grade possuem como infra-estrutura básica o Globus 4 e possuem suporte ao serviço de implantação HAND.

No intuito de gerar uma carga de trabalho no ambiente proposto, foi desenvolvida uma aplicação sintética que realiza operações de soma e subtração. Para fins de teste, o componente `GridApp` mantém em tabelas da base de dados `Hsqldb` os últimos valores resultantes das operações requisitadas. No momento que uma nova operação é requisitada, o componente `GridApp` recebe um valor, passado como parâmetro da operação, sendo o outro consultado na base de dados.

A aplicação sintética foi desenvolvida com base nos componentes ilustrados na

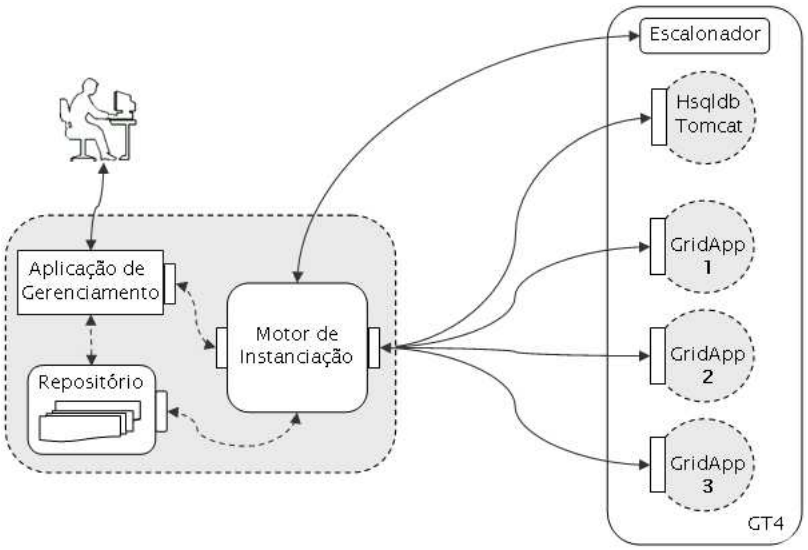

FIGURA 6.1 – Ambiente de teste

Figura 4.4. No arquivo de descrição da aplicação, a definição dos componentes e suas configurações permaneceram inalteradas. Já a descrição do fluxo de implantação pode ser observada na Figura 6.2, onde é especificada a ordem que deve ser respeitada para a instanciação dos componentes. A descrição determina que a instanciação dos componentes `Hsqldb` e `Tomcat` seja feita em uma mesma estação-alvo (linhas 5 e 8, `labo01`) e do componente `GridApp`, em três outras estações-alvo distintas, usando o construtor `lazy` (linhas 15, 18 e 21) para que o escalonamento seja feito de acordo com os recursos disponíveis na grade no momento da instanciação. A aplicação foi descrita de tal forma que as três instâncias de `GridApp` são configuradas para utilizarem a mesma base de dados `Hsqldb` e o mesmo servidor de aplicações `Tomcat`.

```
1   <cdl:system>
2     <cmp:sequence lifecycle="initialization">
3       <cmp:flow lifecycle="initialization">
4         <Hsqldb>
5           <hostname>labo01</hostname>
6         </Hsqldb>
7         <Tomcat>
8           <hostname>labo01</hostname>
9           <dbport cdl:ref="Hsqldb:/port" />
10          <username cdl:ref="Hsqldb:/dbuser" />
11          <password cdl:ref="Hsqldb:/dbpassword" />
12        </Tomcat>
13      </cmp:flow>
14      <GridApp1>
15        <hostname cdl:lazy="true" />
16      </GridApp1>
17      <GridApp2>
```

```
18              <hostname cdl:lazy="true" />
19            </GridApp2>
20            <GridApp3>
21              <hostname cdl:lazy="true" />
22            </GridApp3>
23        </cmp:sequence>
24        <cmp:reverse lifecycle="terminate">
25            <cmp:flow lifecycle="terminate">
26              <GridApp3 />
27              <GridApp2 />
28              <GridApp1 />
29            </cmp:flow>
30            <Tomcat />
31            <Hsqldb />
32        </cmp:reverse>
33    </cdl:system>
```

FIGURA 6.2 – Descrição do fluxo de implantação usado na avaliação experimental

No intuito de "validar" o protótipo desenvolvido segundo a arquitetura AGrADC, três experimentos foram executados no ambiente de teste descrito. No primeiro, nenhum tipo de falha é provocada. Já no segundo e terceiro experimentos, foram provocadas falhas na implantação do componente `GridApp1` e na operação de configuração do componente `Hsqldb`. A diferença entre esses dois experimentos está no fato de que somente um deles acaba lançando mão das políticas de ação.

6.2 Resultados Obtidos

Em um primeiro instante, foi executado um experimento no ambiente de teste sem que fosse inserido nenhum tipo de falha. A Figura 6.3 ilustra o funcionamento do processo de instanciação, caracterizando as dependências respeitadas no fluxo de implantação. A duração do processo de instanciação é quantificada em unidades de tempo (*ut*), atribuídas a cada estado do ciclo de vida de um componente, sendo que no experimento em questão são necessárias 9 *uts* para a instanciação de uma aplicação. Se fosse quantificado, o tempo total do ciclo de vida de uma aplicação seria igual à soma do tempo de instanciação mais o tempo da execução da aplicação.

No intuito de avaliar o comportamento da arquitetura diante de situações adversas enfrentadas ao longo do processo de instanciação de aplicações, foram executados, na seqüência, dois experimentos com o mesmo ambiente e aplicação de teste. Duas falhas foram artificialmente provocadas: de implantação (`Deployment Fault`) e de configuração (`Configuration Fault`), descritas em detalhes na seqüência.

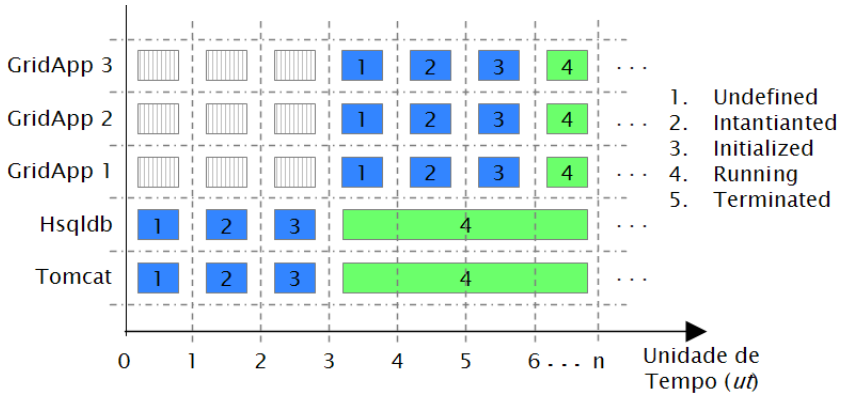

FIGURA 6.3 – Instanciação de uma aplicação

A falha de implantação foi artificialmente gerada da seguinte forma: o motor de instanciação requisita ao escalonador da grade uma lista de estações que podem ser utilizadas para a instanciação dos componentes `GridApp`. Esse escalonador é instruído a enviar uma lista de 4 estações, sendo que na primeira da lista, propositalmente, o serviço de instanciação foi configurado de forma que o diretório de implantação ficasse sem permissão de escrita. Alocadas as estações-alvo, o serviço de instanciação, executando na estação na qual o componente `GridApp1` deveria ser implantado, retorna uma mensagem de notificação informando a ocorrência de uma falha de implantação (`DeploymentFault`), gerada pela impossibilidade de escrever no diretório apropriado.

Já na falha de configuração, os componentes `Hsqldb` e `Tomcat` são implantados paralelamente até o momento em que uma falha de configuração do componente `Hsqldb` é detectada pelo motor de instanciação (notificação `ConfigurationFault`). Essa falha foi provocada previamente ocupando-se a porta solicitada pelo `Hsqldb` para disponibilizar seus serviços.

O primeiro experimento está ilustrado na Figura 6.4. Configurado com as políticas de ação, o motor recebeu uma mensagem de notificação indicando a ocorrência de uma falha de implantação (`DeploymentFault`). O motor, então, consultou as políticas de ação (Figura 5.2, linhas 10 e 11) e executou o processo falho novamente. O processo falha novamente na estação em que estava sendo executado e o processo de instanciação do componente nessa estação é abortado. O motor de instanciação verifica a disponibilidade de outra estação e inicia novamente o processo de instanciação do componente `GridApp1`. A falha `ConfigurationFail` ocorre quando o componente `Hsqldb` tenta configurar uma porta já ocupada. O motor de instanciação foi instrumentado para ao receber a notificação de falha de

configuração, executar uma chamada de sistema que liberasse a porta requisitada pela base de dados. Na nova tentativa feita, como indica a política associada a essa notificação (linhas 19 e 20), o processo de instanciação conseguiu ter prosseguimento.

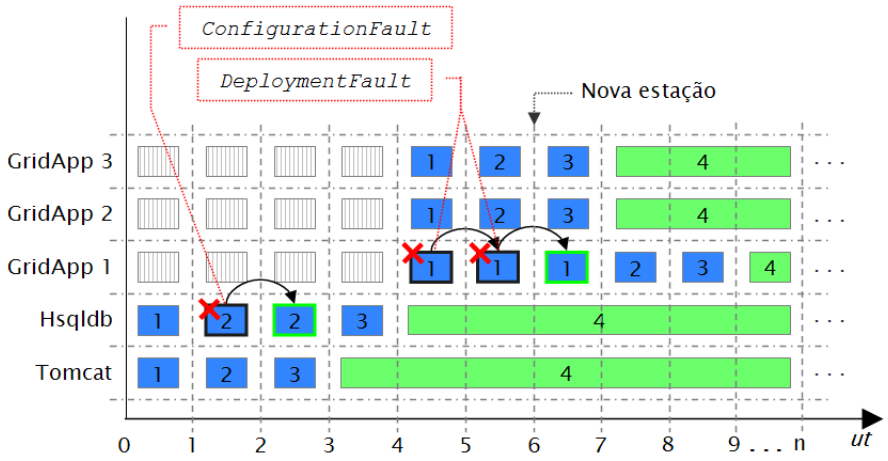

FIGURA 6.4 – Processo de instanciação usando políticas de ação

Já no segundo experimento, ilustrado na Figura 6.5, o mesmo processo de instanciação é executado, porém sem o uso das políticas de recuperação de falha. Isso implica que, na ocorrência das falhas, todo o processo de instanciação executado até o momento seja desfeito e iniciado novamente, independente da falha. Quando ocorre uma falha de implantação, o atraso observado é pequeno já que o processo ainda não tinha evoluído significativamente; já na falha de configuração, a sobrecarga causada pelo término e reinício do processo de instanciação pode ser grande.

Comparando as Figuras 6.4 e 6.5, pode-se observar que, para o cenário ilustrado, o ganho proporcionado por AGrADCno tempo de instanciação da aplicação, em comparação com solução sem suporte à implantação e configuração autônomas, foi de $4uts$. Este valor pode ser considerado bastante significativo, dado que a aplicação possui um pequeno número de componentes. Estima-se que esse ganho seja maior em aplicações com muitos componentes e sujeitas a múltiplas falhas (em função da dinamicidade das grades).

Nos experimentos realizados, a arquitetura comportou-se tal e qual esperado, respeitando a especificação no arquivo de configuração e concluindo com sucesso a instanciação. Observou-se também que a instanciação baseada em políticas possibilita que uma falha seja corrigida pontualmente, sem a necessidade de refazer todo o processo de um componente. Isso introduz uma sobrecarga significativa, que é propagada a todos os componentes dependentes. Sem o uso das políticas,

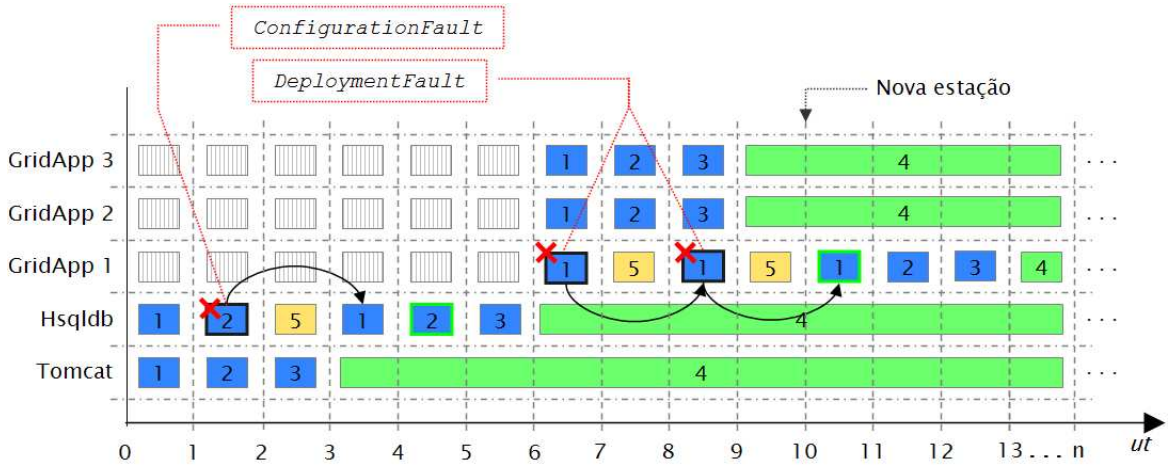

FIGURA 6.5 – Processo de instanciação sem as políticas de ação

independente do estado onde a falha ocorrer, o processo de instanciação deve ser reiniciado, aumentando o tempo necessário para concluí-lo.

Capítulo 7

Considerações Finais

Esta dissertação apresentou uma arquitetura para implantação, configuração e gerenciamento do ciclo de vida de aplicações de grades computacionais. Enquanto os trabalhos relacionados têm se concentrado na investigação de técnicas para permitir a carga e a configuração remota de componentes em estações de grade [Sun et al., 2005, Weissman et al., 2005, Qi et al., 2007], este trabalho assume tal problemática como (próxima de) resolvida e avança na direção de uma solução para orquestrar o processo de instanciação de aplicações de grade complexas. Para tal, propõe um serviço *inédito* que assume o papel até agora desempenhado por operadores humanos. Ao permitir a especificação de políticas para regular seu comportamento, o serviço é capaz de executar procedimentos de contorno para lidar com problemas enfrentados ao longo do processo de instanciação, imprimindo suporte à auto-configuração e auto-recuperação.

Salienta-se que o foco deste trabalho reside na instalação e na configuração da infra-estrutura de software necessária para implantar uma aplicação, incluindo os binários desta aplicação. A instanciação (ou invocação) da aplicação propriamente dita não faz parte do escopo e, portanto, não foi tratada. Para realizar esta tarefa, é possível utilizar ferramentas de *workflow* para grades, que têm por função invocar, de maneira coordenada, os componentes da aplicação.

Os resultados obtidos apontam que a arquitetura proposta pode ser aplicada em situações reais, fato comprovado pela implementação de um protótipo capaz de conduzir a instanciação de uma aplicação real. Tão logo sejam finalizadas as implementações de referência da especificação CDDLM para executar nas estações-alvo – o que deve se concretizar nos próximos meses – a arquitetura, com ajustes mínimos, estará habilitada para operar em total conformidade com a especificação. Esta característica, apesar de não ser mandatória, é importante para promover

interoperabilidade entre soluções desenvolvidas na área de grades computacionais.

Os ganhos que podem ser obtidos com a aquitetura proposta dependem de dois fatores: (a) das políticas definidas para lidar com situações adversas e (b) da granularidade dos eventos gerados pelos serviços de instanciação. As políticas definidas para tratar os eventos recebidos têm a finalidade de otimizar a execução de ações relacionadas à instanciação da aplicação de grade em questão. Sendo assim, quanto maior a acurácia das políticas, mais eficiente será o comportamento do processo de instanciação.

Como trabalhos futuros pretende-se realizar um conjunto mais extensivo de experimentos para caracterizar as limitações do emprego da arquitetura para instanciar aplicações de mais larga escala, bem como determinar os tempos envolvidos nesse processo. Pretende-se, ainda, explorar outras alternativas para representação de comportamento autônomo, tais como *políticas de objetivo* e *funções de utilidade*, visando prover um formalismo de mais alto nível para o desenvolvedor da aplicação ou gerente da infra-estrutura de grade.

Adicionalmente, como trabalhos futuros pretende-se tratar também a questão da segurança necessária ao processo de implantação e configuração dos componentes da aplicação. Uma das alternativas para abordar essa questão consiste no emprego de uma camada a mais na arquitetura, seguindo o padrão WS-Security.

Bibliografia

[AIDE, 2006] AIDE (2006). Ibm autonomic integrated development environment. http://www.alphaworks.ibm.com/tech/aide.

[Anderson, 2001] Anderson, P. (2001). The Complete Guide to LCFG. http://www.lcfg.org/doc/lcfg-guide.pdf.

[Anderson and Scobie, 2002] Anderson, P. and Scobie, A. (2002). LCFG - the Next Generation. In *UKUUG Winter Conference*. UKUUG.

[Baker et al., 2000] Baker, M. A., Buyya, R., and Laforenza, D. (2000). The Grid: International Efforts in Global Computing. In *International Conference on Advances in Infrastructure for Electronic Business, Science, and Education On the Internet (SSGRR 2000)*.

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

[Bell et al., 2005] Bell, D., Kojo, T., Goldsack, P., Loughran, S., Milojicic, D., Schaefer, S., Tatemura, J., and Toft, P. (2005). Configuration Description, Deployment, and Lifecycle Management (CDDLM) Foundation Document. GGF. http://www.gridforum.org/documents/GFD.50.pdf.

[Benatallah et al., 2002] Benatallah, B., Dumas, M., Sheng, Q. Z., and Ngu, A. H. H. (2002). Declarative Provisioning and Peer-to-Peer Provisioning of Dynamic Web Services. In *IEEE International Conference on Data Engineering (ICDE 2002)*, pages 297–308.

[Bullard and Vambenepe, 2005] Bullard, V. and Vambenepe, W. (2005). Web Services Distributed Management: Management Using Web Services (MUWS

1.1) Part 1. Standard. OASIS. http://docs.oasis-open.org/wsdm/wsdm-muws1-1.1-spec-os-01.pdf.

[Chess et al., 2004] Chess, D. M., Segal, A., Whalley, I., and White, S. R. (2004). Unity: experiences with a prototype autonomic computing system. In *IEEE International Conference on Autonomic Computing (ICAC 2004)*, pages 140–147.

[Cirne and Santos-Neto, 2005] Cirne, W. and Santos-Neto, E. (2005). Grids Computacionais: Da Computação de Alto Desempenho a Serviços sob Demanda. In *Mini-curso no 23º Simpósio Brasileiro de Redes de Computadores.*

[Condor, 2006] Condor (2006). Condor Project Home Page. http://www.cs.wisc.edu/condor.

[Foster, 2006] Foster, I. (2006). Globus Toolkit Version 4: Software for Service-Oriented Systems. *IFIP International Conference on Network and Parallel Computing (NPC 2006)*, 21(4):513–520.

[Foster et al., 2002] Foster, I., Kesselman, C., Nick, J., and Tuecke, S. (2002). The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. In *Global Grid Forum*, volume 22.

[Foster et al., 2001] Foster, I., Kesselman, C., and Tuecke, S. (2001). The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications (IJHPCA 2001)*, 15(3):200.

[Ganek and Corbi, 2003] Ganek, A. and Corbi, T. (2003). The Dawning of the Autonomic Computing Era. *IBM Systems Journal*, 42(1):5–18.

[GGF, 2006] GGF (2006). Global Grid Forum Home Page. http://www.gridforum.org.

[Globus, 2006] Globus (2006). Globus Toolkit Project Home Page. http://www.globus.org.

[Graham et al., 2005] Graham, S., Hull, D., and Murray, B. (2005). Web Services Base Notification 1.3 (WS-BaseNotification). Standard. OASIS. http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.htm.

[Horn, 2001] Horn, P. (2001). Autonomic Computing: IBM's Perspective on the State of Information Technology. http://www.research.ibm.com/autonomic/manifesto/.

[IBM, 2006] IBM (2006). Autonomic computing. http://www.research.ibm.com/autonomic/overview.

[JDOM, 2006] JDOM (2006). JDOM Home Page. http://www.jdom.org.

[Keahey et al., 2005] Keahey, K., Foster, I., Freeman, T., Zhang, X., and Galron, D. (2005). Virtual Workspaces in the Grid. In *International Euro-Par Conference (Euro-Par 2005)*, pages 421–431.

[Kephart, 2005] Kephart, J. O. (2005). Research challenges of autonomic computing. In *international conference on Software engineering (ICSE 2005)*, pages 15–22. ACM Press.

[Kephart and Chess, 2003] Kephart, J. O. and Chess, D. M. (2003). The Vision of Autonomic Computing. *Computer*, 36(1):41–50.

[Kephart and Walsh, 2004] Kephart, J. O. and Walsh, W. E. (2004). An Artificial Intelligence Perspective on Autonomic Computing Policies. In *IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2004)*, pages 3–12.

[Kesselman and Foster, 1998] Kesselman, C. and Foster, I. (1998). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers.

[Krauter et al., 2002] Krauter, K., Buyya, R., and Maheswaran, M. (2002). A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing. In *Software: Practice and Expirience*, volume 30, pages 135–164.

[LCFG, 2006] LCFG (2006). Local ConFiGuration system Home Page. http://www.lcfg.org.

[Loughran, 2005] Loughran, S. (2005). Configuration Description, Deployment, and Lifecycle Management. CDDLM Deployment API. Draft 2005-02-25. GGF. http://xml.coverpages.org/CDDML-Deployment-API-SpecificationDraft20050308.pdf.

[Muse, 2006] Muse (2006). Muse Project Home Page. http://ws.apache.org/muse/.

[Nemeth and Sunderam, 2002] Nemeth, Z. and Sunderam, V. (2002). A Formal Framework for Defining Grid Systems. In *IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2002)*, pages 188–197.

[OASIS, 2006] OASIS (2006). Organization for the Advancement of Structured Information Standards Home Page. http://www.oasis-open.org.

[OGSA, 2006] OGSA (2006). Open Grid Services Architecture Home Page. http://www.globus.org/ogsa.

[Ourgrid, 2006] Ourgrid (2006). Ourgrid Project Home Page. http://www.ourgrid.org.

[Qi et al., 2007] Qi, L., Jin, H., Foster, I., and Gawor, J. (2007). HAND: Highly Available Dynamic Deployment Infrastructure for Globus Toolkit 4. In *Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2007)*. (to appear).

[Rauch et al., 2000] Rauch, F., Kurmann, C., and Stricker, T. M. (2000). Partition Repositories for Partition Clonning – OS Independent Software Maintenance in Large Clusters of PCs. In *IEEE International Conference on Cluster Computing*, pages 233–242.

[Reverbel et al., 2004] Reverbel, F., Burke, B., and Fleury, M. (2004). Dynamic Deployment of IIOP-Enabled Components in the JBoss Server. In *Component Deployment: Second International Working Conference (CD 2004)*, pages 65–80.

[Salehie and Tahvildari, 2005] Salehie, M. and Tahvildari, L. (2005). Autonomic Computing: Emerging Trends and Open Problems. In *Workshop on Design and Evolution of Autonomic Application Software (DEAS 2005)*, volume 30, pages 1–7.

[Schaefer, 2005] Schaefer, S. (2005). Configuration description, deployment, and lifecycle management - component model. Technical Report 1, Global Grid Forum.

[Simmons and Lutfiyya, 2005] Simmons, B. and Lutfiyya, H. (2005). Policies, Grids and Autonomic Computing. In *Workshop on Design and Evolution of Autonomic Application Software (DEAS 2005)*, pages 1–5.

[SmartFrog, 2006a] SmartFrog (2006a). Smartfrog – smart framework for object groups. http://www.hpl.hp.com/research/smartfrog.

[SmartFrog, 2006b] SmartFrog (2006b). A smartfrog tutorial. http://www.smartfrog.org/releasedocs/smartfrogdoc/SmartFrogTutorial.pdf.

[Smith and Anderson, 2004] Smith, E. and Anderson, P. (2004). Dynamic Reconfiguration for Grid Fabrics. In *IEEE/ACM International Workshop on Grid Computing (GRID 2004)*, pages 86–93. IEEE Computer Society.

[Sotomayor, 2006] Sotomayor, B. (2006). The Globus Toolkit 4 Programmer's Tutorial. http://gdp.globus.org/gt4-tutorial.

[Sun et al., 2005] Sun, H., Zhu, Y., Hu, C., Huai, J., Liu, Y., and Li, J. (2005). Early Experience of Remote and Hot Service Deployment with Trustworthiness in CROWN Grid. In *International Workshop on Advanced Parallel Processing Technologies (APPT 2005)*, pages 301–312.

[Talwar et al., 2005] Talwar, V., Milojicic, D., Wu, Q., Pu, C., Yan, W., and Jung, G. (2005). Approaches for Service Deployment. *IEEE Internet Computing*, 9(2):70–80.

[Tatemura, 2005] Tatemura, J. (2005). Configuration Description, Deployment, and Lifecycle Management. XML Configuration Description Language Specification Version 1.0. Draft 03-05-2005. GGF. https://forge.gridforum.org/.

[Van Moorsel, 2005] Van Moorsel, A. P. A. (2005). Grid, Management and Self-Management. *Computer*, 48(3):325–332.

[Weissman et al., 2005] Weissman, J., Kim, S., and England, D. (2005). A Framework for Dynamic Service Adaptation in the Grid: Next Generation Software Program Progress Report. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS 2005)*.

[White et al., 2004] White, S. R., Hanson, J. E., Whalley, I., Chess, D. M., and Kephart, J. O. (2004). An Architectural Approach to Autonomic Computing. In *International Conference on Autonomic Computing (ICAC 2004)*, pages 2–9.