GPO - Um Middleware para Orquestração de Serviços em Grades Computacionais

Este exemplar corresponde à redação da Dissertação apresentada para a Banca Examinadora antes da defesa da Dissertação.

Campinas, 27 de Fevereiro de 2007.

Prof. Dr. Edmundo Roberto Mauro Madeira
(Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Instituto de Computação
Universidade Estadual de Campinas

GPO - Um Middleware para Orquestração de Serviços em Grades Computacionais

Carlos Roberto Senna[1]

Outubro de 2007

Banca Examinadora:

- Prof. Dr. Edmundo Roberto Mauro Madeira (Orientador)
- Prof. Dr. Bruno Richard Schulze
  LNCC - Laboratório Nacional de Computação Científica
- Profa. Dra. Maria Beatriz Felgar de Toledo
  IC – UNICAMP
- Prof. Dr. Luiz Eduardo Buzato (Suplente)
  IC – UNICAMP

[1]Suporte financeiro de: Bolsa do CNPq (processo 133.542/2003-3) 2004–2005

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Miriam Cristina Alves – CRB8a / 5094

Senna, Carlos Roberto
Se58g GPO – Um middleware para orquestração de serviços em grades computacionais/Carlos Roberto Senna -- Campinas, [S.P. :s.n.], 2007.

Orientador: Edmundo Roberto Mauro Madeira.
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Computação em grade (Sistema de computador). 2. Fluxo de trabalho. 3. Processamento distribuído. 4. Redes de computadores. I. Madeira, Edmundo Roberto Mauro. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

(mca/imecc)

Título em inglês: GPO – A middleware for service orchestration in computacional grids.

Palavras-chave em inglês (Keywords): 1. Grid computing (Computer systems). 2. Workflow. 3. Distributed process. 4. Computer networks.

Área de concentração: Redes de computadores.

Titulação: Mestre em Ciência de Computação

Banca examinadora: Prof. Dr. Edmundo Roberto Mauro Madeira (IC-Unicamp)
Prof. Dr. Bruno Richard Schulze (LNCC)
Profa. Dra. Maria Beatriz Felgar de Toledo (IC-Unicamp)

Data da defesa: 27/02/2007

Programa de Pós-Graduação: Mestrado em Ciência da Computação

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 27 de fevereiro de 2007, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Bruno Richard Schulze
LNCC – Lab. Nacional de Computação Científica.

Profa. Dra. Maria Beatriz Felgar de Toledo
IC – UNICAMP.

Prof. Dr. Edmundo Roberto Mauro Madeira
IC – UNICAMP.

200800418

Resumo

No ambiente colaborativo das grades computacionais são poucas as ferramentas para gerência de processos e serviços orientadas ao usuário. Esta dissertação apresenta o *Grid Process Orchestration (GPO)*, uma infra-estrutura que faz orquestração de serviços e processos em grades computacionais, permitindo ao usuário criar e gerenciar fluxos complexos, com tarefas fortemente acopladas, sem suporte adicional. O GPO é baseado na OGSA (*Open Grid Services Architecture*) e descreve os fluxos usando o conceito de orquestração de serviços *Web* aplicados aos serviços das Grades Computacionais.

A dissertação descreve a arquitetura da infra-estrutura proposta, detalha seus principais componentes, suas funcionalidades para gerência de fluxos e alguns aspectos do protótipo implementado. Além disso, propõe uma linguagem compacta para descrever os *workflows*. Uma aplicação exemplo é apresentada ilustrando as facilidades da infra-estrutura proposta.

Abstract

In the collaborative environment of the Computational Grids, there are few tools used for process management and user guide services. This work presents the Grid Process Orchestration (GPO), an infrastructure for service and process orchestration in computational grids, which allows to create and manage complex workflows composed of strong-coupled jobs with no additional support. The GPO is based on the Open Grid Services Architecture (OGSA) and describes workflows using Web Services orchestration concepts applied to computational grid services.

This work describes the architecture of the proposed infrastructure, detailing its main components, functionalities for flow management, and shows an implemented prototype. In addition to the architecture, it proposes a compact language for describing workflows. An application example is presented to illustrate the facilities of the proposed infrastructure.

Agradecimentos

A Glória e ao Pedro que são a razão de tudo.

Ao meu orientador, Professor Edmundo R. M. Madeira, pela paciência e perseverança, pela ajuda e por acreditar sempre que o objetivo seria atingido. Este trabalho não seria possível sem a sua participação.

Aos meus professores da graduação na UFSCar, José Hiroki Saito e Durval Makoto Akamatu, que mais uma vez apoiaram a minha iniciativa.

Ao Professor Hugo Fragnito, do Instituto de Física da UNICAMP, que além do apoio no mestrado também cedeu o seu laboratório permitindo minha participação em um projeto científico inovador.

Aos amigos Francis, Rose e Caio que agiram como catalisadores nessa minha jornada ao mundo científico.

Ao grande mentor e amigo Professor Marco Aurélio Pinheiro Lima, do Instituto de Física da UNICAMP, pelo apoio, incentivo e ensinamentos de como percorrer o novo caminho.

Aos colegas do Instituto de Computação pela ajuda concedida.

Ao Instituto de Computação pela oportunidade dada.

Ao CNPq que tornou este trabalho possível através da Bolsa de Mestrado oferecida.

"Poderoso ou miserável, homem ou mulher, trabalhador ou vagabundo, inteligente ou não, todo mundo tem o mesmo tempo à disposição. Pois tempo é vida, que não pode ser estocada nem renovada. E que, mesmo não vivida, se gasta e desaparece."

Jean-Louis Servan-Schreiber

Lista de Acrônimos

API *Application Programming Interface*
BoT *Bag of Tasks*
BSP *Bulk Synchronous Parallel*
CAS *Community Authorization Service*
CORBA *Common Object Request Broker Architecture*
EPR *Endpoint Reference*
GGF *Global Grid Forum*
GJD *WS-GRAM Job Description*
GPO *Grid Process Orchestration*
GPOL *Grid Process Orchestration Language*
GSI *Grid Security Infrastrucutre*
GSH *Grid Service Handle*
GSR *Grid Service Reference*
GWSDL *Grid Web Services Description Language*
HTML *Hyper Text Markup Laguage*
HTTP *Hyper Text Transfer Protocol*
IS *Information Services*
MPI *Message Passing Interface*
OASIS *Organization for the Advancement of Structured Information Standards*
OGSA *Open Grid Services Architecture*
OGSI *Open Grid Services Infrastruture*
QoS *Quality of Service*
RFT *Reliable File Transfer*
RPC *Remote Procedure Call*
RSL *Resource Specification Language*
SGW Sistemas de Gerenciamento de Workflow
SOA *Service Oriented Architecture*
SOAP *Simple Object Access Protocol*
SOC *Service-Oriented Computing*

UDDI *Universal Description Discovery and Integration*
URI *Uniform Resource Identifier*
VO *Virtual Organizations*
W3C *World Wide Web Consortium*
WfMC *Workflow Management Coalition*
WSCI *Web Service Choreography Interface*
WSDL *Web Services Description Language*
WS-GRAM *Web Service Globus Resource Allocation Manager*
WSIL *Web Services Inspection Language*
WSRF *Web Services Resource Framework*
WS-BPEL *Web Service Business Process Execution Language*
XML *eXtensible Markup Language*

Sumário

Resumo **iii**

Abstract **v**

Agradecimentos **vii**

Lista de Acrônimos **ix**

1 Introdução **1**
1.1 Motivação 1
1.2 Objetivos 2
1.3 Estrutura do Trabalho 3

2 Computação Orientada a Serviços e Grades Computacionais **5**
2.1 Computação Orientada a Serviços 5
2.1.1 Serviços *Web* 5
2.1.2 Composição de Serviços 10
2.1.3 Sistemas de *Workflow* 14
2.2 Grades Computacionais 16
2.2.1 Aplicações de Grades Computacionais 16
2.2.2 Tipos de recursos de uma grade 17
2.2.3 Arquitetura e tecnologias 18

3 Trabalhos Relacionados **27**
3.1 Grades Computacionais 27
3.1.1 Integrade 27
3.1.2 OurGrid 28
3.1.3 Globus Toolkit 28
3.2 Composição de Serviços 32
3.3 Gerência de Workflow em Grades Computacionais 33

3.3.1 Sistemas de Gerência 33
3.3.2 Linguagens 38

4 Linguagem Proposta: GPOL **43**
4.1 A Estrutura de um *Workflow* GPOL 45
4.2 Notação e Considerações Gerais 46
4.2.1 *Atividade* 46
4.2.2 *Expressão* 46
4.3 Elementos da Linguagem 47
4.3.1 *workflow* 47
4.3.2 *import* 48
4.3.3 *definitions* 49
4.3.4 *process* 50
4.3.5 *variables* e *variable* 50
4.3.6 *services* e *service* 51
4.3.7 *faultHandlers* 53
4.3.8 *invoke* 54
4.3.9 *assign* 55
4.3.10 *echo* 56
4.3.11 *return* 56
4.3.12 *exit* 57
4.3.13 *clear* 57
4.3.14 *if* 57
4.3.15 *while* 58
4.3.16 *sequence* 58
4.3.17 *flow* 59
4.3.18 *transfer* 59
4.3.19 *execute* 60
4.3.20 *javacode* 61
4.4 *Um exemplo de workflow GPOL* 62
4.5 *Esquemas XML GPOL* 64

5 Middleware Proposto: GPO **67**
5.1 Arquitetura GPO 67
5.2 *GPO Run* 68
5.3 GPO Maestro 69
5.3.1 GPO Maestro Factory Service 69
5.3.2 GPO Maestro Service Instance 71
5.4 A integração do GPO com o ambiente da Grade 72

5.4.1 O GPO integrado ao sistema de segurança da Grade 72
5.4.2 A integração do GPO com os escalonadores da Grade 73
5.5 Aspectos Sobre a Implementação Feita 73

6 Uma Aplicação da Grade: Fotoionização de Átomos 75
6.1 A Modularização da Fórmula 76
6.2 Visão Geral da Implementação do Workflow em GPOL 77
6.3 Detalhes sobre o Ciclo Somatório 80
6.4 Execução do *workflow* exemplo 82

7 Conclusão 85

Bibliografia 87

A Workflow GPOL para Fotoionização de Átomos 97

Lista de Tabelas

6.1 Relação entre os números quânticos 76

Lista de Figuras

2.1 Pilha de Protocolos dos Serviços *Web* 6
2.2 Exemplo de documento XML 7
2.3 Diagrama e exemplo de mensagem SOAP 8
2.4 Parte de um documento WSDL 9
2.5 Uso de um Serviço Web 10
2.6 WS-BPEL e os protocolos dos Serviços Web 13
2.7 Níveis da Grade 19
2.8 Arquitetura principal OGSA com OGSI 21
2.9 Componentes OGSI 22
2.10 WSRF com as especificações para Serviços *Web* 25

3.1 Componentes da arquitetura OurGrid 28
3.2 Evolução do GT e convergência com os padrões *Web* 29
3.3 Serviços e componentes do GT4 30
3.4 Uso do WS-GRAM na submissão de tarefas 31
3.5 Exemplo de submissão de tarefas em GJD 32
3.6 Condor na infra-estrutura Globus 35
3.7 Uma submissão com o Condor DAGMan 36
3.8 Camadas da arquitetura *Java CoG Kit* 37
3.9 Arquitetura GSFL 38
3.10 *Workflow Karajan* executando compilação Java remota 40
3.11 Componentes da gerência de *workflow* do projeto GriPhyN 41

4.1 Exemplo de *workflow* em GPOL 45
4.2 Submissão de tarefa definida com *WS-GRAM Job Description* 61
4.3 Operações do Serviço *MathFactoryService* e DAG da Expressão 62
4.4 Definições do *workflow* para a expressão matemática 63
4.5 O processo para executar a expressão matemática 64

5.1 A arquitetura GPO e seus componentes 67
5.2 Exemplo de execução com *GPO Run* 68

5.3 Relacionamento entre os componentes do GPO Maestro 70

6.1 Fórmula da seção de choque de fotoionização de átomos relativísticos . . . 75
6.2 Diagrama com as dependências entre os termos da fórmula 77
6.3 Visão geral do *workflow Fotoion* 78
6.4 Execução do *workflow Fotoion* usando o GPO Run 82

Capítulo 1

Introdução

1.1 Motivação

Cada vez mais as organizações necessitam reduzir seus ciclos de desenvolvimento, melhorar a integração e interação entre as suas áreas, diminuindo custos e se adaptando mais rapidamente a mudanças. A automação, a colaboração e os sistemas de distribuição de recursos podem ser usados para atingir esses objetivos [6].

Organizações Virtuais e Colaboração

A Internet e seus padrões abertos acrescidos das facilidades disponíveis nos sistemas distribuídos permitem que as organizações possam compartilhar recursos, desde a fase inicial de desenvolvimento de produtos até as etapas finais do processo de negócio. A união desses conceitos e tecnologias resultou no aparecimento das *Organizações Virtuais* que são multi-institucionais e dinâmicas [15]. Uma organização virtual pode ser definida como *"uma rede de diferentes domínios de negócios administrativos, que cooperam compartilhando processos de negócio e recursos, para prover um* ***serviço*** *com valor agregado ao cliente"* [74, 68]. Em uma organização virtual, *serviço* passa a ser não só a base das aplicações como também a peça fundamental do processo de colaboração entre os participantes da organização.

Esse novo ambiente torna os padrões atuais de *workflow* limitados para suportar a interação entre os participantes nas organizações virtuais, notadamente no que se refere a coordenação e *composição* dinâmica de processos e serviços requeridos por essas organizações.

Computação Orientada a Serviços

A Computação Orientada a Serviços (*Service-Oriented Computing* - SOC) é o paradigma computacional que utiliza serviços como elementos fundamentais para o desenvolvimento de aplicações e soluções [22]. No ambiente orientado a serviços, as aplicações (agora definidas como serviços), devem tornar públicas as suas funcionalidades e capacidades, para que todos os participantes possam descobrí-las automaticamente. Esse novo paradigma faz dos serviços a base de construção de novas aplicações onde a *composição entre serviços* é um requisito fundamental para o desenvolvimento de aplicações. Na Sessão 2.1 desta dissertação estão apresentados maiores detalhes sobre Computação Orientada a Serviços. Neste capítulo também são mostrados os principais mecanismos para a Composição de Serviços, com destaque para a WS-BPEL (*Web Service Business Process Execution Language*), uma linguagem que serviu de base para a presente dissertação.

Grades Computacionais

Grades computacionais são ambientes para computação distribuída de alta performance que permitem o compartilhamento de recursos heterogêneos. A estrutura da grade gerencia a execução de tarefas e coordena a disponibilidade dos recursos e serviços que compõem o seu ambiente. É uma coleção de recursos computacionais distribuídos sobre uma rede como um grande sistema de *computação virtual* [1, 2], que permite o compartilhamento de recursos individuais e institucionais. A grade é a ferramenta ideal para a implementação das *organizações virtuais*.

O ambiente colaborativo das grades facilita a interação entre os usuários mas esse cenário requer um nível adicional de gerência de *workflow*, principalmente se levarmos em consideração que a maioria dos usuários não tem conceitos de programação ou conhecimento para a criação de *scripts*, que possam controlar com eficiência a execução de tarefas distribuídas, principalmente as que tem alto grau de inter-relacionamento.

Neste trabalho foi usada uma infra-estrutura de grade baseada na OGSA (*Open Grid Services Architecture*). Essa arquitetura une os conceitos de grades computacionais e serviços *Web*, e seus detalhes serão mostrados na Sessão 2.2 desta dissertação.

1.2 Objetivos

Este trabalho de Mestrado tem por objetivo o desenvolvimento do GPO (*Grid Process Orchestration*). O GPO é uma infra-estrutura para *workflow* controlável pelo usuário, que faz a orquestração de serviços e processos em grades computacionais. O GPO usa os conceitos das grades baseadas na OGSA, e acrescenta a essas grades as facilidades para orquestração de serviços oferecidas pela WS-BPEL.

As principais contribuições deste trabalho são:

- A proposição de uma linguagem para submissão de tarefas em grades computacionais baseadas em OGSA, denominada GPOL (*Grid Process Orchestration Language*).
- O projeto de uma infra-estrutura (GPO) para submissão de tarefas fortemente acopladas em grades computacionais.
- A implementação de um protótipo da infra-estrutura proposta e uma aplicação exemplo desse protótipo em computação científica.

1.3 Estrutura do Trabalho

Esta dissertação está organizada da seguinte forma:

- **Capítulo 2:** estão os conceitos básicos relativos ao trabalho. Na primeira parte apresenta os conceitos sobre computação orientada a Serviços *Web*, *workflow* e composição de serviços. Na segunda parte mostra os conceitos sobre Grades Computacionais, suas aplicações, as arquiteturas com destaque para a OGSA.
- **Capítulo 3:** são mostrados os trabalhos relacionados aos tópicos desta dissertação.
- **Capítulo 4:** descreve formalmente a linguagem para submissão de tarefas GPOL.
- **Capítulo 5:** apresenta a arquitetura GPO e descreve seus principais componentes.
- **Capítulo 6:** mostra uma aplicação onde a linguagem e a infra-estrutura são usadas.
- **Capítulo 7:** estão as conclusões sobre o trabalho, suas principais contribuições e possíveis extensões.

Capítulo 2

Computação Orientada a Serviços e Grades Computacionais

Este capítulo apresenta os conceitos básicos necessários para o entendimento desta dissertação e está dividido em duas partes. Na primeira parte estão os conceitos relacionados à Computação Orientada a Serviços. São introduzidos o conceito de serviço *Web*, os principais protocolos a ele relacionados e as maneiras para compô-los em processos executáveis. Na segunda parte estão os conceitos relacionados às Grades Computacionais com ênfase na arquitetura orientada a serviços.

2.1 Computação Orientada a Serviços

2.1.1 Serviços *Web*

Os serviços são os elementos fundamentais do paradigma computacional chamado de *Computação Orientada a Serviços (SOC - Service-oriented Computing)*, onde as aplicações são desenvolvidas como uma composição de serviços [22]. Ao aplicarmos os conceitos de SOC na Internet criamos os Serviços *Web*. Serviço *Web* (WS) define uma técnica para descrição de componentes de *software*, métodos para acessar esses componentes e métodos para localizar e identificar esses componentes. WS permite o desenvolvimento de aplicações distribuídas para a Internet. Os Serviços *Web* oferecem interoperabilidade universal, são baseados nos protocolos abertos da Internet, tem como foco a troca de mensagens e permitem a criação de aplicações com um baixo nível de integração (fracamente acopladas). O conceito WS é fundamental no trabalho aqui apresentado, pois os serviços das grades computacionais baseadas em OGSA são uma extensão de serviços *Web* [30].

"Um WS é um sistema de *software* projetado para suportar interoperabilidade da interação máquina-a-máquina sobre uma rede. Ele tem uma interface descrita em um

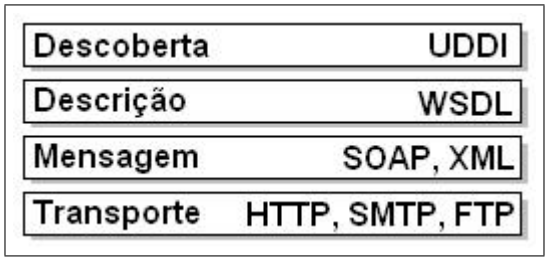

Figura 2.1: Pilha de Protocolos dos Serviços *Web*

formato processável pela máquina (especificamente WSDL). Outros sistemas interagem com um WS da maneira prescrita através de mensagens SOAP, tipicamente transportadas usando HTTP com conteúdo XML em conjunto com outros protocolos da Internet" [11]. SOAP provê mensagens entre o provedor do serviço e o requisitante. É um mecanismo que envolve a XML que define convenções para RPC (*Remote Procedure Call*) e convenções para mensagens, sendo independente do protocolo de transporte (HTTP por exemplo). SOAP é a forma com que os WSs são invocados. WSDL é um documento XML que descreve operações, seus parâmetros e tipos de dados requeridos. Um documento WSDL também contém informações sobre como acessar o serviço. O serviço é identificado por uma URI (*Uniform Resource Identifier*), mas pode ser uma entrada no UDDI (*Universal Description, Discovery and Integration*). A Figura 2.1 mostra a arquitetura WS.

A seguir estão detalhadas as especificações de linguagens e protocolos envolvidas em um Serviço *Web*.

- XML (*eXtensible Markup Language*), definida pelo W3C [12], é uma linguagem para descrever uma classe de objetos de dados chamados de *documentos XML* e também descreve de certa forma o comportamento dos programas processadores desses documentos [50]. XML é derivada da SGML (*Standard Generalized Markup Language*) [48] e tem as seguintes características:

  - Os documentos XML são construídos por unidades de armazenamento chamadas de *entidades*.
  - As *marcações* codificam uma descrição da organização e da estrutura lógica do documento.
  - XML usa DTD (*Document Type Definition*) ou um *XML Schema* [89] para descrever os dados, de maneira a serem auto-descritivos.

  Cada documento XML contém tanto a sua estrutura lógica como a sua estrutura física. Fisicamente o documento é composto por *entidades* e sempre começa pela *raiz* ou *documento entidade*. Logicamente o documento é composto por declarações,

```
<?xml version = "1.0" ?>
<artigo>
  <titulo>
     The Grid: A New Infrastructure for
     21st Century Science
  </titulo>
  <data>Fevereiro, 2002</data>
  <publicacao>Physics Today</publicacao>
  <autor>Ian Foster</autor>
  <resumo>
      Como as Grades Computacionais podem
      transformar a prática da ciência.
  </resumo>
</artigo>
```

Figura 2.2: Exemplo de documento XML

elementos, comentários e instruções de processamento. O exemplo da Figura 2.2 mostra um documento XML que descreve um artigo (<artigo> *raiz*), que tem outros marcadores que indicam o título (<titulo>), a data da publicação (<data>), onde foi publicado (<publicacao>), o autor (<autor>) e um resumo do artigo (<resumo>).

- SOAP (*Simple Object Access Protocol*) [87], recomendado pelo W3C (*World Wide Web Consortium*) é um protocolo baseado em XML e provê um mecanismo para troca de informações entre computadores em um ambiente distribuído e descentralizado. Mensagens SOAP são independentes de plataforma e de linguagem e podem transpor até mesmo redes protegidas por *Firewalls*. Uma mensagem SOAP é formada pelos elementos *Envelope* (obrigatório), *Header* (opcional) e *Body* (obrigatório) [17]. *Envelope* identifica o documento XML como sendo uma mensagem SOAP, *Header* (cabeçalho) pode conter informações adicionais da aplicação (assinatura digital, por exemplo) e *Body* (corpo) contém informações sobre uma requisição ou uma resposta. A Figura 2.3 mostra o esquema geral e um exemplo de mensagem SOAP [88].

- WSDL (*Web Services Description Language Version* 2.0) [19] é uma especificação que define como descrever um serviço *Web* através de uma gramática XML. WSDL define uma linguagem para descrever a funcionalidade (as operações ou métodos) e a localização do serviço, separando a descrição da funcionalidade abstrata dos detalhes de implementação. Uma descrição WSDL possui quatro informações fundamentais:

  - Informações sobre a interface descrevendo todas as funcionalidades disponíveis;
  - Os tipos de dados que podem ser trocados por todas as mensagens transacionais (requisições ou respostas);

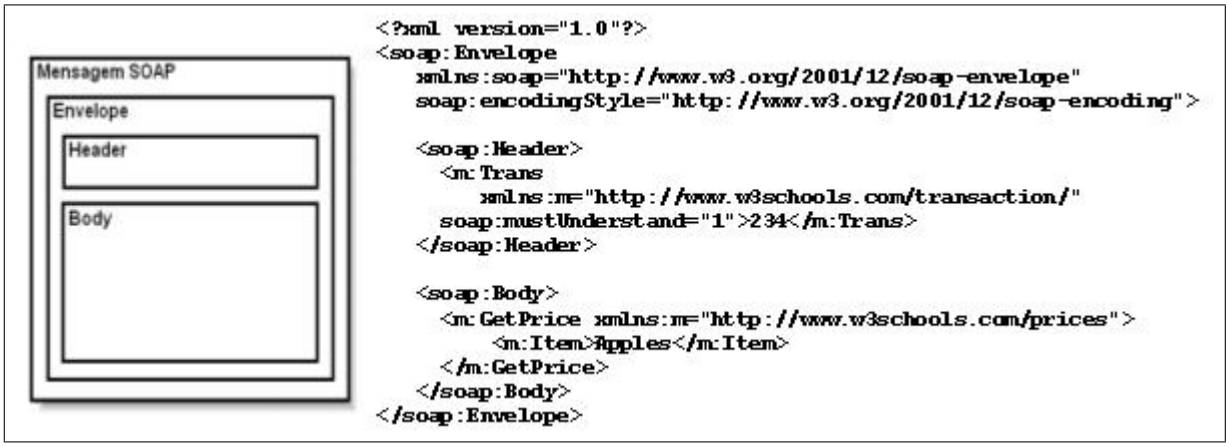

Figura 2.3: Diagrama e exemplo de mensagem SOAP

- O protocolo de transporte a ser usado; e
- O endereço para localização do serviço especificado.

Um documento WSDL descreve um serviço *Web* usando os seguintes elementos:

- <*definitions*>: deve ser o elemento raiz de todo documento WSDL. Ele define o nome do Serviço Web, declara os múltiplos *namespaces* usados e contém todos os outros elementos do serviço;
- <*types*>: são os tipos de dados que o Serviço usa;
- <*message*>: descreve as mensagens enviadas e recebidas pelo Serviço *Web*. Define o nome das mensagens e seus elementos integrantes;
- <*portType*>: combina os múltiplos elementos *message* formando uma operação. Um *portType* pode, por exemplo, combinar uma requisição e uma resposta formando uma operação simples, e pode também definir múltiplas operações;
- <*binding*>: protocolo de comunicação usado pelo Serviço *Web*;
- <*service*>: define o endereço para que o Serviço possa ser invocado, incluindo uma URI a ser usada pelo SOAP.

A Figura 2.4 mostra uma parte de um documento WSDL [17]. O elemento *definitions* indica *HelloService* como o nome do Serviço, cuja descrição está em *http://www.ecerami.com/wsdl/HelloService.wsdl*. O elemento *portType* define a operação *sayHello*, que usa as mensagens *SayHelloRequest* e *SayHelloResponse*.

- UDDI (*Universal Description, Discovery and Integration*) [21] é uma especificação para descrever, descobrir e integrar Serviços *Web*. Ele é uma parte importante nos

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloService"
        targetNamespace="http://www.ecerami.com/wsdl/HelloService.wsdl"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:tns="http://www.ecerami.com/wsdl/HelloService.wsdl"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">

        <message name="SayHelloRequest">
                <part name="firstName" type="xsd:string"/>
        </message>
        <message name="SayHelloResponse">
                <part name="greeting" type="xsd:string"/>
        </message>

        <portType name="Hello_PortType">
                <operation name="sayHello">
                        <input message="tns:SayHelloRequest"/>
                <output message="tns:SayHelloResponse"/>
                </operation>
        </portType>

        . . .

 </definitions>
```

Figura 2.4: Parte de um documento WSDL

protocolos para Serviços *Web* pois permite que as companhias publiquem e encontrem os Serviços. O núcleo da especificação UDDI é destinada à construção de um registro distribuído de negócios e Serviços *Web*, onde os dados são armazenados em formato XML. Esse núcleo inclui também APIs (*Application Programming Interfaces*) para procurar dados já existentes e para publicar novos dados. As informações contidas em um registro de negócio UDDI são divididas em três categorias principais [17]:

- *White Pages* (Páginas Brancas): Inclui informações gerais sobre uma determinada empresa, como por exemplo, o nome da empresa, a descrição do seu negócio principal, o seu endereço e telefone comercial;
- *Yellow Pages* (Páginas Amarelas): Inclui a classificação do negócio ou do serviço oferecido;
- *Green Pages* (Páginas Verdes): Contém informações técnicas sobre um Serviço *Web*. Geralmente inclui uma referência para uma especificação externa e um endereço para invocar o Serviço *Web*.

Um processo típico para invocar um WS é mostrado na Figura 2.5. Em 1 o Cliente consulta o registro UDDI para descobrir onde está o Serviço desejado. Em 2, o Registro UDDI que está no Servidor A responde à requisição de localização indicando que o Serviço está no Servidor B. O Cliente pergunta ao Servidor B como é possível invocar o Serviço

(em 3), e o Servidor B responde com a descrição WSDL correspondente (em 4). De posse da descrição do Serviço, o Cliente pode gerar um *stub* cliente, incluí-lo na aplicação, e usar o *stub* todas as vezes que for necessário invocar o WS. A aplicação faz uma chamada local ao *stub* cliente que envia uma requisição SOAP. Essa requisição é enviada através do HTTP ao *stub* servidor (em 5). O WS *container* recebe a requisição e converte para que a implementação do serviço possa entendê-la. O resultado da operação é passado ao *stub* servidor que o transforma numa resposta SOAP, e envia de volta usando o HTTP (em 6). O *stub* cliente recebe a resposta e passa para a aplicação de forma que esta possa entendê-la e apresentá-la ao Cliente.

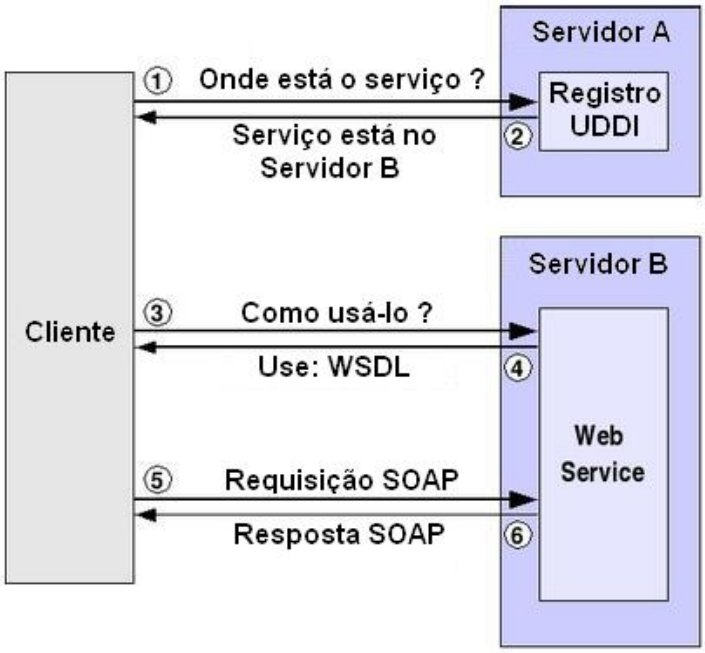

Figura 2.5: Uso de um Serviço Web

2.1.2 Composição de Serviços

Os Serviços *Web* estão migrando de seu modelo inicial e estão sendo usados em interações mais robustas que fazem parte dos processos de negócio [74]. Nessa nova fase os WSs são compostos ou agregados de maneira a formar aplicações distribuídas que envolvem vários participantes. Esta composição de WSs pode ser feita através de conceitos como *Orquestração* ou *Coreografia*. A *Coreografia* estabelece protocolos para definir como deve ser a troca de mensagens entre os WSs participantes do processo de negócio. A *Orquestração* define um fluxo de interações entre os WSs em um processo.

Em uma composição de Serviços, independente do conceito usado, alguns aspectos são importantes;

- A ordem das interações entre os Serviços;
- As regras que devem orientar a seqüência de interações;
- A relação entre as mensagens trocadas pelos WS participantes;
- Os aspectos relevantes ao iniciar e encerrar o processo; e
- As possibilidades para restaurar interações já realizadas.

Os conceitos de *Coreagrafia* e *Orquestração* atendem a maioria dos requisitos citados. A seguir são detalhados esses conceitos e duas especificações correspondentes.

1. **Coreografia**

   Coreografia define protocolos para a troca de mensagens entre um conjunto de serviços *Web*, onde cada um dos serviços tem igual responsabilidade na troca dessas mensagens. Na coreografia todos os participantes são responsáveis pelo processo, ou seja, não é centralizado. Uma forma de descrever uma coreografia é o WSCI (*Web Services Choreography Interface*).

   WSCI

   É uma especificação [8] feita pela Sun, SAP, BEA e Intalio que define uma linguagem baseada em XML para colaboração (ou coreografia) entre serviços *Web*. Define as mensagens entre os serviços e os participantes no processo de colaboração. A especificação suporta correlação de mensagens, regras de seqüência, tratamento de exceções, transações, e a dinâmica na colaboração. O aspecto fundamental do WSCI é que este só descreve o procedimento visível entre os serviços *Web*. WSCI não endereça a definição do processo executável de negócio. Cada documento WSCI descreve um dos participantes na troca de mensagens, e não há um processo único que gerencie a interação. WSCI pode também ser visto como uma camada adicionada no topo da pilha dos serviços *Web*. WSDL pode ser usada para descrever os pontos de entrada em cada serviço disponível, e WSCI pode descrever as interações entre essas operações WSDL. WSCI suporta transações e tem tratamento para exceções.

2. **Orquestração**

   Orquestração descreve como serviços *Web* podem interagir com outros em nível de mensagens, incluindo a lógica de negócio e a ordem de execução das interações entre eles. Essas interações podem estender as aplicações e as organizações, resultando

num modelo de processo transacional com um ciclo de vida maior. Orquestração trata processos executáveis de negócios que podem interagir interna e externamente com um serviço *Web*. Na orquestração o processo é sempre controlado a partir da perspectiva de uma das partes do negócio. O termo orquestração de serviços *Web* pode ser usado para descrever a criação de processos de negócios, executáveis ou colaborativos, que usem tais serviços. A orquestração deve ser dinâmica e flexível. Flexibilidade pode ser obtida através da clara separação entre a lógica do processo e os serviços usados. Essa separação pode estar dentro do *motor* de orquestração, que manipula o fluxo do processo, chamando os serviços apropriados e determinando os próximos passos a serem completados. Uma linguagem para definição de uma orquestração é a WS-BPEL (*Web Service Business Process Execution Language*).

WS-BPEL

É uma especificação feita inicialmente por IBM, Microsoft e BEA, e que está sendo padronizada pela OASIS (*Organization for the Advancement of Structured Information Standards*) [57], que modela o procedimento dos serviços *Web* na interação do processo de negócio [93]. Define uma gramática baseada em XML para descrever a lógica de controle, requerida para coordenar os serviços participantes no fluxo do processo. Essa gramática é interpretada e executada pelo *motor* de orquestração, que é controlado por um dos participantes. Como mostra a Figura 2.6, WS-BPEL é uma camada no topo da WSDL, com WSDL definindo as operações específicas e WS-BPEL definindo a ordem de execução das operações. Documentos WS-BPEL interagem com WSDL de três maneiras:

- Todo processo WS-BPEL é exposto como um serviço usando WSDL. A WSDL descreve os pontos de entrada e saída públicos para o processo;
- Os tipos de dados WSDL são usados com os processos WS-BPEL para descrever as informações passadas entre as requisições; e
- WSDL pode ser usada como referência externa dos serviços requeridos através do processo.

Uma composição WS-BPEL é um processo de negócio, ou um *workflow*, que interage com um conjunto de Serviços *Web* [93]. A composição é formada por duas partes. Na primeira parte (<*definitions*>), são feitas as definições das mensagens que serão trocadas, dos serviços e operações usados (<*portTypes*>) e os tipos de ligação entre os parceiros (<*partnerLinkType*>). A segunda parte (<*process*>), é a estrutura do processo de negócio, onde são definidos os parceiros (<*partnerLinks*>) e as atividades entre eles. Cada passo do processo é chamado de ***atividade***. Há uma variedade de atividades básicas, tais como:

Figura 2.6: WS-BPEL e os protocolos dos Serviços Web

- <*invoke*> Usada para invocar uma operação em um Serviço *Web*.
- <*receive*> Espera por uma mensagem para então executar outra atividade.
- <*reply*> Gera uma resposta para uma operação de entrada e/ou saída.
- <*wait*> Espera um tempo determinado.
- <*assign*> Copia dados de um local para outro.
- <*throw*> Indica que algo errado ocorreu.
- <*terminate*> Encerra o processo.

Essas atividades básicas podem ser combinadas de forma complexa usando as atividades estruturadas, tais como:

- <*if*> Escolha de opções.
- <*while*> Definição de ciclos.
- <*sequence*> Define uma seqüência de passos e a ordem na qual devem ser executados.
- <*flow*> Define que um grupo de passos devem ser executados em paralelo (simultaneamente).

Um Serviço *Web*, pela própria definição, não mantém o estado. WS-BPEL usa um mecanismo chamado de *correlação de mensagens* para manter o estado de uma interação [74]. Os campos principais nas mensagens de dados são identificados e usados para relacionar as mensagens trocadas entre os parceiros (<*message*> e <*variables*>).

WS-BPEL oferece também suporte para o tratamento de erros através do uso de mecanismos de compensação e de tratamento de falhas. Os tratadores de falhas usam o modelo estruturado similar ao *"try-catch"* de *Java*, e permitem *compensar* a falha através da indicação de ações a serem tomadas quando estas ocorrem. Esse modelo de compensação está relacionado aos protocolos definidos pelas especificações *WS-Transaction* [7] e *WS-Coordination* [60] (Figura 2.6). *WS-Coordination* e *WS-Transaction* tratam de coordenação confiável e transacional de WSs, e podem ser usados de forma isolada ou em conjunto com WS-BPEL [22].

Cada uma das especificações tem propósitos bem definidos na composição de Serviços:

- WS-BPEL permite a composição de WSs em novos Serviços através de construtores de modelagem de processos bem definidos;
- *WS-Transaction* define um mecanismo para interoperabilidade transacional entre domínios WS e provê um meio para compor aplicações com QoS [35] transacionais. Define também dois tipos particulares de coordenação. Um para transações de curta duração (*WS-AtomicTransaction*) [58], e outro para transações de negócio que são de longa duração (*WS-BusinessActivity*) [59];
- *WS-Coordination* é um *framework* geral para implementação de tipos específicos de coordenação, onde a coordenação requer um contexto compartilhado; e

A combinação dessas três especificações permite que o modelo de composição WS-BPEL possa ser estendido com capacidade de coordenação distribuída.

3. WS-BPEL x WSCI

   Comparando as duas especificações é possível notar que WS-BPEL enfatiza a criação do processo executável de negócio, enquanto que WSCI concentra-se na troca de mensagens entre os serviços. WSCI é mais colaborativo, requerendo cada participante na troca de mensagens que define a interface WSCI. WS-BPEL é mais introspectivo, descrevendo um processo executável a partir da perspectiva de um dos participantes.

2.1.3 Sistemas de *Workflow*

O termo *workflow* pode ser definido como uma representação específica de um processo, cujo mecanismo formal de coordenação entre atividades, aplicações e processos participantes, pode ser controlado através de um sistema de informação chamado de *Sistema de Gerência de Workflow* (SGW) [54, 72]. Um SGW é definido pela WfMC (*Workflow Management Coalition*) [94, 54] como: "*Um sistema que define, cria e gerencia a execução*

de workflows através do uso de software, executado por um ou mais motores de workflow (workflow engines), responsáveis por interpretar as definições do processo, interagir com os participantes, e quando requerido, invocar ferramentas e aplicações". Os principais termos relacionados com SGW são [78]:

- *Processo:* é a descrição de uma seqüência de tarefas ou atividades que devem ser executadas;
- *Atividades:* representam tarefas, que são atribuídas a determinados *papéis*. Uma atividade pode ser composta por uma ou mais (sub)tarefas. Uma atividade mais complexa pode ser representada por um processo, neste caso, chamado de sub-processo do processo que a contém; e
- *papéis:* funções ou cargos associados a pessoas ou programas.

Para coordenar e modelar atividades de longa duração, os SGWs devem satisfazer vários requisitos, cujos principais são apresentados a seguir [78]:

- *Capacidade de encaminhar documentos:* boa parte das operações realizadas em um SGW trata o envio e recebimento de documentos entre os diferentes usuários do sistema.
- *Suporte a compartilhamento de dados:* várias aplicações cooperativas necessitam compartilhar dados e informações.
- *Composição de atividades:* atividades podem ser agrupadas, compondo sub-processos que podem ser utilizados como atividades de outros processos.
- *Capacidade de integração com diferentes aplicações existentes:* muitas tarefas podem exigir aplicações auxiliares. O SGW deve fornecer meios para o uso dessas aplicações e permitir o intercâmbio de documentos e dados gerados através delas.
- *Interoperabilidade entre diferentes SGWs.*
- *Suporte a recuperação de falhas:* um SGW deve ser capaz de lidar com falhas de forma transparente, maximizando a disponibilidade do sistema e escondendo, dentro do possível, o tratamento de erros dos usuários.
- *Monitoramento:* em processos longos e complexos são necessários mecanismos que possibilitem determinar o estado corrente dos processos em execução. As informações obtidas através desses mecanismos constituem a base para ações de gerenciamento, fornecendo dados para detecção de falhas e gargalos de desempenho no sistema.

- *Tratamento de Exceções.*
- *Reconfiguração Dinâmica:* devido à característica de longa duração de muitas atividades e à necessidade de tratamento de exceções em processos, os SGWs devem permitir a mudança dinâmica das definições de processos em andamento. A reconfiguração dinâmica (*dynamic change*) consiste em alterar o plano de um processo enquanto suas instâncias estão em execução.

Os conceitos de *workflow* e orquestração de processos e serviços são extremamente importantes no contexto das Grades Computacionais. A execução de um *workflow* na Grade é mais complexa devido à dinamicidade e heterogeneidade natural desse ambiente. Além disso, um dos principais focos de uso das Grades é o compartilhamento de recursos de forma segura e colaborativa em múltiplas instituições geograficamente distribuídas. Nesse ambiente, a gerência de *workflow* torna-se mais complexa devido à comunicação entre processos que transpõe as fronteiras institucionais, requerendo suporte a múltiplas políticas de segurança [2].

2.2 Grades Computacionais

Grades computacionais são ambientes para computação distribuída de alto desempenho que permitem o compartilhamento de recursos heterogêneos. Conceitualmente uma Grade consiste no compartilhamento coordenado de recursos por organizações multi-institucionais e dinâmicas [31, 76]. Esse compartilhamento pode ser de arquivos, acesso direto a computadores, *software* e dados, por exemplo. Ele requer controle sofisticado, com provedores e consumidores definindo claramente o que é compartilhado, quem pode acessar, enfim, as condições para que o compartilhamento ocorra. O conjunto de indivíduos e/ou instituições definido através dessas regras de compartilhamento forma as *Organizações Virtuais* (VO - *Virtual Organizations*) [31].

A seguir são apresentadas as aplicações mais comuns das Grades Computacionais (Subseção 2.2.1), seus tipos de recursos (Subseção 2.2.2) e considerações sobre a arquitetura (Subseção 2.2.3).

2.2.1 Aplicações de Grades Computacionais

Grades computacionais podem ser aplicadas em:

- **Exploração de recursos sub-utilizados:** [49] uma das aplicações imediatas da grade é executar uma aplicação em qualquer máquina disponível [42]. Se a máquina onde a aplicação normalmente é executada estiver ocupada, a grade procura uma que

esteja disponível para executá-la. A grade deve encontrar um recurso computacional que atenda às necessidades da aplicação em *software* e *hardware*, que mantenha o desempenho. Essa capacidade da grade em encontrar recursos pode ser estendida não só a processamento, mas também a recursos de armazenamento, comunicação, impressão, dentre outros.

- **Supercomputação distribuída:** aplicações que necessitam de grandes recursos computacionais podem usar a grade para agregar esses recursos e permitir a sua execução, a qual não seria possível num único sistema (uma máquina isolada). Dependendo de como a grade foi organizada, podem ser agregados recursos espalhados num país, ou simplesmente uma *workstation* que está em outro departamento da companhia.

- **Computação de alto desempenho:** a grade pode ser usada para organizar e distribuir [44] um grande número de tarefas independentes, aumentando a velocidade de processamento e execução. Essa facilidade pode ser usada em computação científica, problemas de criptografia, simulações, projeto de microprocessadores, etc.

- **Computação sob demanda:** algumas máquinas na grade podem ter recursos especiais, normalmente caros, disponibilizados a todos os participantes da grade. Recursos como impressoras de alta qualidade, gravadores de DVD, equipamentos científicos como microscópios eletrônicos, entre outros. Para o uso desses recursos a grade prevê reserva e agendamento.

- **Uso intensivo de dados:** aplicações cujo foco seja sintetizar novas informações a partir de dados em repositórios geograficamente distribuídos, bibliotecas digitais e bancos de dados, com uso intensivo de processamento e comunicação para captação e análise dos dados. Aplicações experimentais de física onde são usados terabytes de dados por dia, fotografias astronômicas e sistemas meteorológicos fazem uso intenso de dados.

- **Colaboração:** através dos recursos da grade é possível integrar trabalhos desenvolvidos em vários locais por diversas pessoas, formando o ambiente ideal para colaboração [55, 32].

2.2.2 Tipos de recursos de uma grade

A Grade é uma coleção de máquinas que são referenciadas como nós, recursos, membros, enfim de diversas maneiras [49, 75]. Alguns desses recursos podem ser usados por todos os usuários, enquanto que outros tem aplicações mais restritivas. De forma geral os recursos de uma Grade podem ser:

- **Computacionais:** são os processadores das máquinas que compõem a grade. Esses processadores são heterogêneos, tendo velocidades, arquitetura, ambiente operacional, memória, sistema de armazenamento e facilidades para comunicação que podem ser distintos ou não. Esses recursos computacionais podem ser usados simplesmente para executar uma aplicação numa máquina disponível, podem ser usados por uma aplicação que possa ser distribuída nos diversos processadores, ou aplicações que devam ser executadas repetidas vezes em diferentes máquinas.

- **Armazenamento:** Cada máquina na grade disponibiliza sistemas de armazenamento para uso comum. Pode ser memória associada a um processador, que permita a uma aplicação manipular uma massa maior de dados que a permitida na máquina onde está sendo executada, ou armazenamento secundário onde a grade torna transparente para a aplicação o sistema de organização de arquivos do recurso compartilhado.

- **Comunicação:** Recursos de comunicação são disponibilizados aos integrantes da grade. Máquinas onde haja conexão com Internet podem ser usadas, assim como canais rápidos de comunicação com outras máquinas.

- ***Softwares* e licenças:** Pela facilidade natural de execução remota, a grade permite o compartilhamento de programas.

- **Reserva, agendamento e procura:** A grade é responsável por enviar e executar uma tarefa num dado recurso computacional. Permite que seja feita uma reserva do recurso, indicando quando deverá ser executado. Cabe à grade também encontrar o recurso com os requisitos necessários à aplicação.

- **Gerência:** A grade possui protocolos para gerenciar os recursos compartilhados, e que são responsáveis por negociações de segurança, monitoramento, enfim o controle dessas operações compartilhadas feitas nos recursos individuais.

2.2.3 Arquitetura e tecnologias

Uma grade é constituída por um infra-estrutura de *hardware* e componentes de *software* que são usados para interconectar os diferentes computadores. Esses componentes suportam o fluxo de informação entre os sistemas e provêem um conjunto básico de serviços de conectividade, segurança, avaliação de desempenho e gerência. Uma grade pode ser representada por quatro níveis [31], conforme pode ser visto na Figura 2.7:

- **Aplicações:** são as aplicações executadas pelo usuário na grade: cosmologia, química, engenharia, clima, combustão, etc.

- **Serviços coletivos:** serviço de diretório, diagnóstico e monitoramento.
- **Recursos e protocolos de conectividade:** protocolos, autenticadores, segurança, instrumentação, gerência de recursos, eventos, pesquisa de recursos, etc.
- **Fábrica:** armazenamento, suporte de rede, computadores, *displays*, controle de serviços locais, etc.

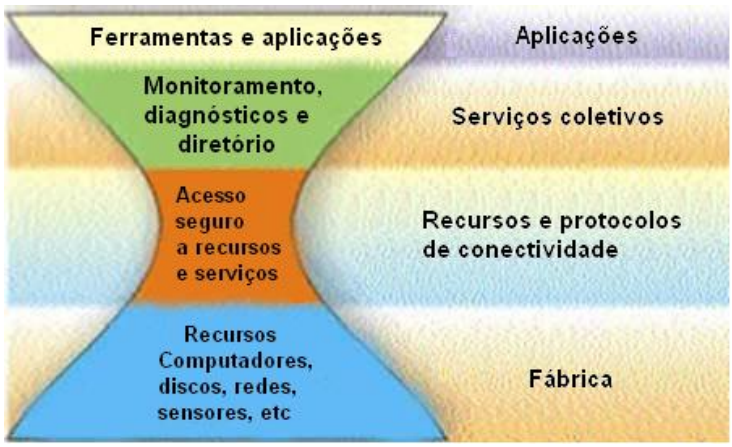

Figura 2.7: Níveis da Grade

A computação em Grade requer tecnologias que devem incluir [49]:

- Suporte para execução de programas em várias plataformas;
- Uma infra-estrutura de segurança;
- Tratamento para movimentação, réplica e uso compartilhado de dados;
- Recursos para descoberta; e
- Recursos para gerência.

Para cada uma dessas áreas existe uma variedade de tecnologias e padrões disponíveis. Algumas delas são relevantes para o presente trabalho e são detalhadas a seguir.

Serviços da Grade

Uma interface de serviço associada a um recurso da Grade é conhecida como um *Serviço da Grade (SG)*. Um recurso e seu estado são controlados e gerenciados através do SG no ambiente da Grade. Um SG pode acessar mais que um recurso e múltiplos SGs podem

acessar o mesmo recurso. Além disso um SG pode criar uma nova instância do recurso, usando para isso o conceito de *Fábrica* das Grades. Vários recursos da Grade podem interagir ou integrar outros recursos dependendo dos requisitos da aplicação, lembrando que essa interação está ocorrendo no ambiente heterogêneo da Grade. Dessa forma é necessário um *framework* que possa abstrair os detalhes de implementações específicas do ambiente através, por exemplo, de um sistema de mensagens entre Grades. A arquitetura SOA oferece tal *framework* e pode ser usada como base na integração de recursos heterogêneos em vários níveis da arquitetura das Grades.

Apesar dos SGs serem implementados usando a tecnologia dos Serviços *Web* existem diferenças fundamentais entre eles. O Serviço *Web* é persistente e isso é usado em sua localização e uso. Já um SG é concebido para recursos virtuais em um ambiente dinâmico. Um SG pode ser transiente ou persistente, podendo ser criado e destruído dinamicamente, muito diferente de um Serviço *Web* que está acessível conforme a descrição do seu WSDL correspondente. Essas características dos SGs mostradas tem forte impacto na gerência, descoberta e uso dos SGs.

OGSA

A OGF (*Open Grid Forum*) [63] usou os princípios SOA como base da OGSA (*Open Grid Service Architecture*) [30, 64], que provê um *framework* para implementar uma Grade. Na OGSA todos os recursos, sejam físicos ou lógicos, são modelados como Serviços da Grade. Isso permite que os SGs usem o modelo de mensagem dos Serviços *Web*, seu serviço de descrição e de descoberta. Assim, um SG em OGSA é um Serviço *Web* potencialmente transiente, baseado nos protocolos da Grade e que usa WSDL.

OGSA define uma semântica uniforme para exposição dos Serviços da Grade. Define mecanismos padronizados para criar, nomear e descobrir instâncias de serviços transientes da grade. Provê transparência de localização e tratamento para múltiplos protocolos para essas instâncias. Suporta facilidades integradas às plataformas nativas nos ambientes computacionais. As interfaces dos serviços OGSA são descritas em WSDL, incluindo mecanismos para criação e composição de sistemas sofisticados de distribuição e gerência do ciclo de vida, alterações e notificações dos serviços, além da definição dos protocolos de comunicação. O registro UDDI e documentos WSIL (*Web Service Interface Language*) [13] são usados para localizar o serviço, e o protocolo de transporte SOAP é usado para conectar dados e aplicações que acessam os serviços da grade.

No contexto da Grade um recurso representa um *estado* ou um *dado* e provê alguma capacidade útil através de uma *interface*. Essa interface associada a um recurso define um grupo de operações lógicas que podem ser invocadas pelos clientes. Os conceitos emergentes da arquitetura orientada a serviço [22] (SOA - *Service Oriented Architecture*) podem ser aplicados na definição das interfaces dos recursos da Grade transformando-as

em interfaces padronizadas de Serviços *Web*. Entretanto os WSs não *mantêm o estado*, isto é, nenhuma informação entre duas chamadas a um WS é armazenada. Ocorre que a manutenção do estado de um recurso ou serviço nas Grades é freqüentemente necessária, sendo esta uma diferença importante entre Serviços *Web* e Serviços da Grade.

OGSA oferece *gerência de estado* através da qual os serviços da grade podem manter seu estado interno. Serviços interagem com outros, através da troca de mensagens, baseada na invocação de serviços. O estado interno de um serviço pode ser garantido, permitindo aplicações orientadas a transações. OGSA possui também mecanismos para capturar as informações de estado associadas a um serviço numa operação que causou falha. Quando uma falha ocorre, a mensagem *keepalive* é interrompida indicando que o serviço não foi executado. O serviço de grade automaticamente encontra outro recurso computacional para substituí-lo.

OGSA possui um mecanismo para troca de mensagens assíncronas (*notificações*) que avisam os clientes sobre eventos que ocorreram, como por exemplo, mudanças nas propriedades, eventos associados ao tempo de vida, etc.

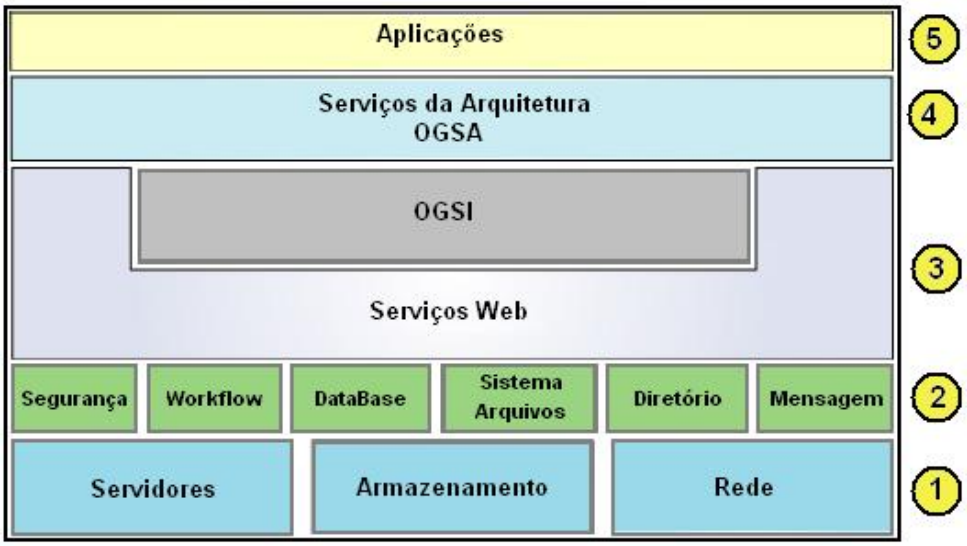

Figura 2.8: Arquitetura principal OGSA com OGSI

A Figura 2.8 mostra na camada 1 os recursos físicos da Grade, na camada 2 os recursos lógicos, na camada 3 os Serviços *Web* e no centro a implementação OGSI, na camada 4 os principais serviços da arquitetura OGSA e na camada 5 as aplicações [85].

OGSI

O OGF estendeu os conceitos da OGSA definindo interfaces específicas para vários serviços que implementam as funções definidas através da OGSA, criando a OGSI (*Open Grid*

Services Infrastructure) [65]. OGSI define mecanismos para criação, gerência e troca de informações entre os Serviços da Grade, além de providenciar definições WSDL para essas interfaces. A *Globus Alliance* usou as especificações OGSA e OGSI na implementação do *Globus Toolkit* versão 3 [37].

A especificação OGSI define facilidades para os Serviços da Grade que incluem [49]:

- Manutenção de estado;
- Interação sobre o estado;
- Facilidades para criação de novas instâncias dos Serviços da Grade;
- Gerência do ciclo de vida;
- Troca de informações (*notificação*) e
- Serviço de grupos.

O modelo OGSI especifica os Serviços da Grade através da GWSDL (*Grid Web Services Definition Language*), que é uma extensão da WSDL. Isto é, OGSI usa os principais conceitos dos padrões para Serviços *Web* mas não é compatível com os mesmos. A especificação OGSI 1.0 define as interfaces apresentadas a seguir e ilustradas na Figura 2.9:

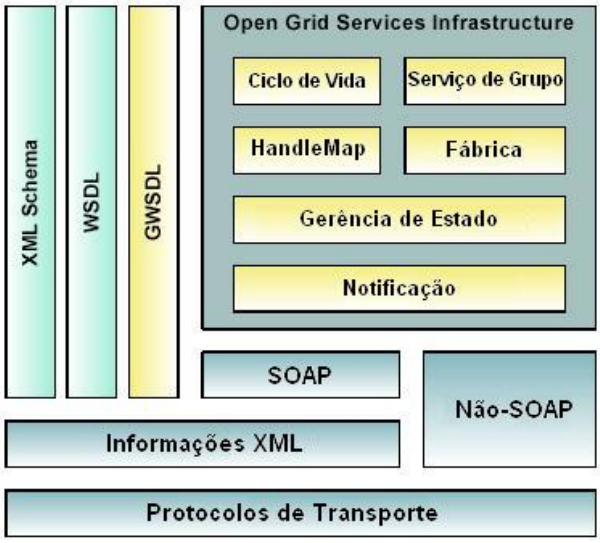

Figura 2.9: Componentes OGSI

- *GridService*: É a interface que encapsula o modelo de Serviço. Ela é especificada através da GWSDL.

- *Fábrica*: É a operação padrão para criação de instâncias de Serviços da Grade.
- *HandleMap*: Os métodos OGSI de criação de instâncias retornam um *handle*. Esse *handle* indica uma referência que então pode ser usada pelo cliente para comunicar com o recurso via Serviço da Grade. Essa interface provê esse funcionalidade através do GSH (*Grid Service Handle*) e do GSR (*Grid Service Reference*).
- *Notificação*: São três interfaces que compõem o serviço de notificação: *NotificationSource*, *NotificationSubscription* e *NotificationSink*.
- *Serviço de Grupo*: São três interfaces que compõem o serviço de grupos: *ServiceGroup*, *ServiceGroupRegistration* e *ServiceGroupEntry*.

WSRF

A implementação dos Serviços da Grade através da extensão de alguns Serviços *Web* feita por OGSI, e a contínua evolução dos Serviços *Web* tem dificultado o caminho inicialmente desejado da união dessas tecnologias [24, 69]. Com o objetivo de integrá-las foi criada a especificação WSRF (*Web Services Resource Framework*) [61]. WSRF, proposto pela OASIS [56], é um conjunto de especificações, que define o relacionamento entre Serviços *Web* (que não mantêm *estado*) com recursos que mantêm *estado*. Em termos gerais, WSRF propõe padrões para:

- Recursos: definições, tempo de vida e propriedades;
- Serviço de grupo: que pode ser visto como uma coleção de recursos;
- Tratamento de falhas;
- Notificações e
- Tópicos.

De forma geral a convergência de OGSI para WSRF resulta em:

- Um Serviço da Grade é transformado em um recurso de acordo com a especificação *WS-Resource*;
- Melhora na definição das funções e interfaces quebrando as funcionalidades em especificações individuais; e
- Usa a especificação *WS-Notification* em Serviços da Grade para construir notificações sobre mudança de estado usando Serviços *Web*.

WS-Resource [61] é uma construção que modela recursos que mantém estado (*stateful resource*) usando *frameworks* da arquitetura para Serviços *Web*. De acordo com WSRF, um recurso que mantém estado deve:

- Ter seu *estado de dados* descrito por um documento XML;
- Ter um *ciclo de vida* bem definido; e
- Ser *conhecido* para ser acessado por Serviços *Web*.

Um recurso modelado através de construções *WS-Resource* pode ser implementado como um arquivo, um sistema de arquivos, registros em um banco de dados ou mesmo em uma estrutura residente em memória.

A composição entre um recurso que mantém estado e um Serviço *Web* é chamada de um *WS-Resource*, e o componente gerado por essa composição tem outros requisitos. Um Serviço *Web* expõe sua interface através da construção *portType*, sendo através dele que o Serviço *Web* torna disponível suas operações ao cliente. Um Serviço *Web* torna-se um *WS-Resource* quando a sua definição *portType* (no arquivo WSDL correspondente) é associada com uma representação XML das propriedades que mantêm o estado, descritas conforme o *framework WS-Resource*.

Em WSRF o estado é mantido implicitamente via subsistema com o qual o Serviço *Web* interage, ou seja, a gerência da instância e do estado é delegada a um componente externo. A implementação WSRF passa implicitamente o identificador do recurso quando o cliente interage com um *WS-Resource*. Para isso WSRF usa um conjunto de convenções tais como XML, WSDL e *WS-Addressing*.

WS-Addressing [92] padroniza o endereço que representa um Serviço *Web*, cuja representação é chamada de EPR (*Endpoint Reference*). EPR é usado em WSRF para representar a informação contextual que disponibiliza a comunicação do cliente com o *WS-Resource*. Um EPR contém o endereço do Serviço *Web* e informações das propriedades do recurso, que incluem um identificador para a instância.

A criação de um identificador de recurso é análoga à criação de uma nova instância de um *WS-Resource*, que por sua vez é análoga à criação de um Serviço da Grade (conforme OGSA). A criação de uma nova instância de um *WS-Resource* envolve:

- Criar uma nova instância de um recurso;
- Associar um novo identificador à nova instância do recurso; e
- Criar uma associação entre a nova instância do recurso e o Serviço *Web* correspondente.

A reorganização feita em OGSI visando a convergência entre os Serviços da Grade e Serviços *Web* resultou em cinco novas especificações WSRF e três especificações na família *WS-Notification*. Um resumo dessas especificações é apresentado a seguir e o seu relacionamento com as especificações para Serviços *Web* é mostrado na Figura 2.10 [25].

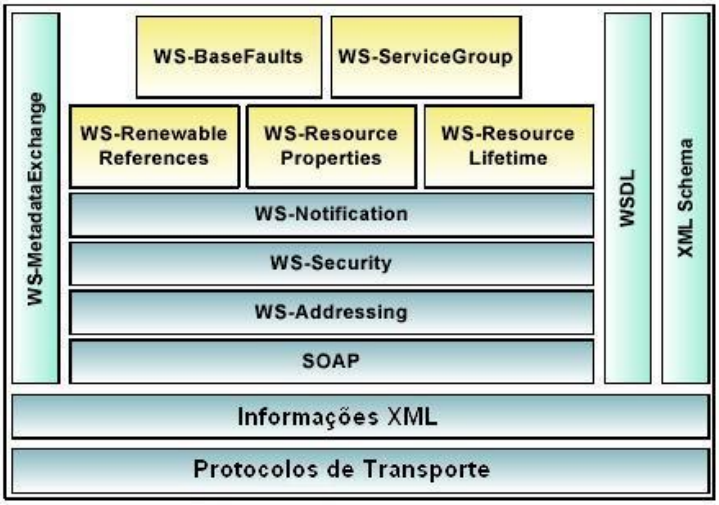

Figura 2.10: WSRF com as especificações para Serviços *Web*

Resumo das especificações WSRF [61]

- *WS-ResourceProperties:* descreve recursos associados a Serviços *Web* para produzir *WS-Resources*. Define também como as propriedades dos *WS-Resources* são acessadas, alteradas e eliminadas.

- *WS-ResourceLifeTime:* define como destruir *WS-Resources* imediatamente ou indicar a futura destruição.

- *WS-RenewableReferences:* anota um EPR com as informações de segurança necessárias para a sua recuperação quando a referência corrente tornar-se inválida.

Resumo das especificações *WS-Notification* [62]

- *WS-ServiceGroup:* cria e usa coleções de Serviços *Web* através de referências.

- *WS-BaseFault:* descreve os tipos básicos de falhas para futuros relatórios de erro.

- *WS-BaseNotification:* define operações dos Serviços *Web* que definem regras de notificação.

- *WS-BrokeredNotification:* define operações de Serviços *Web* para um *broker* de notificação.
- *WS-Topics:* Define um mecanismo para organizar e agrupar itens de interesse para subscrição chamado de tópicos (*topics*). São usados em conjunto com o mecanismo de notificação definido em *WS-BaseNotification*.

Comparando as especificações OGSI e WSRF é possível observar [24]:

- Os padrões para recursos e os conceitos de *WS-Resource* substituem a interface *GridService* definida pela especificação OGSI 1.0.
- Os conceitos GSH e GSR foram substituídos pelo padrão *WS-Addressing*, onde EPR é equivalente a GSH e GSR.
- WSRF define um *framework* padrão para *notificação* permitindo que Serviços *Web* e Serviços da Grade possam compartilhar notificações.

É possível implementar Serviços da Grade baseados em Serviços *Web* (OGSA) usando tanto OGSI como WSRF. A *Globus Alliance* usou WSRF com OGSA em sua implementação do *Globus Toolkit Versão 4* (GT4) [38].

Capítulo 3

Trabalhos Relacionados

Neste capítulo são apresentados alguns trabalhos que estão diretamente relacionados com esta dissertação, na área de Grades Computacionais, Composição de Serviços e Gerência de *Workflow* em Grades Computacionais.

3.1 Grades Computacionais

A seguir são mostrados os projetos Intergrade e OurGrid, *middlewares* para Grades em uso no Brasil. Depois é apresentado o *Globus Toolkit* que foi usado como plataforma para o desenvolvimento do GPO, e por esse motivo será apresentado de forma mais detalhada.

3.1.1 Integrade

O projeto Integrade [47] é um esforço de várias universidades (USP, PUC-Rio, UFMS, UFG e UFMA) na construção de um *middleware* para Grade Computacional que usa o tempo ocioso de computadores pessoais para processamento intensivo (*Grades Oportunistas*) [43]. É um *software* aberto multi-plataforma, orientado a objetos, feito em CORBA [67] que permite aplicações de longa duração, seqüências paramétricas (saco de tarefas - BoT (*Bag of Tasks*)) e paralelas (modelos BSP (*Bulk Synchronous Parallel*) e MPI (*Message Passing Interface*) [14]).

Implementa tolerância a falhas através de pontos de salvaguarda (*checkpoints*), que são incorporados às aplicações pelo uso de uma biblioteca de *checkpoint*. Provê uma interface gráfica para monitoramento dos recursos da Grade e das aplicações em execução.

No entanto requer alteração nas aplicações (compilação, no mínimo) para que estas possam usar as facilidades da tolerância a falhas. Além disso é orientado a aplicações, não sendo possível usar os conceitos de Composição de Serviços.

3.1.2 OurGrid

OurGrid [83] é um projeto desenvolvido pela UFCG em colaboração com a HP (*Hewlett Packard*) que visa uma solução para execução de aplicações BoT em Grades Computacionais. OurGrid é *software* livre sobre o qual se criou uma Grade aberta, ao qual instituições e usuários podem aderir livremente. Ele está focado em aplicações BoT, incluindo *data mining*, pesquisas massivas, varredura de parâmetros, simulações, fractais, biologia computacional e computação gráfica [4].

A solução OurGrid é composta por três componentes (mostrados na Figura 3.1): o *broker* MyGrid, a comunidade OurGrid e a segurança *Swan*. MyGrid escalona aplicações BoT e permite a seu usuário combinar todas as máquinas a que ele tem acesso na sua "*Grade Pessoal*". A comunidade OurGrid forma uma rede *peer-to-peer* que promove o compartilhamento de recursos em escala mundial. O *Swan* aproveita as características das aplicações BoT para prover um *sandboxing* que garante a execução segura de código desconhecido oriundo da Grade [84].

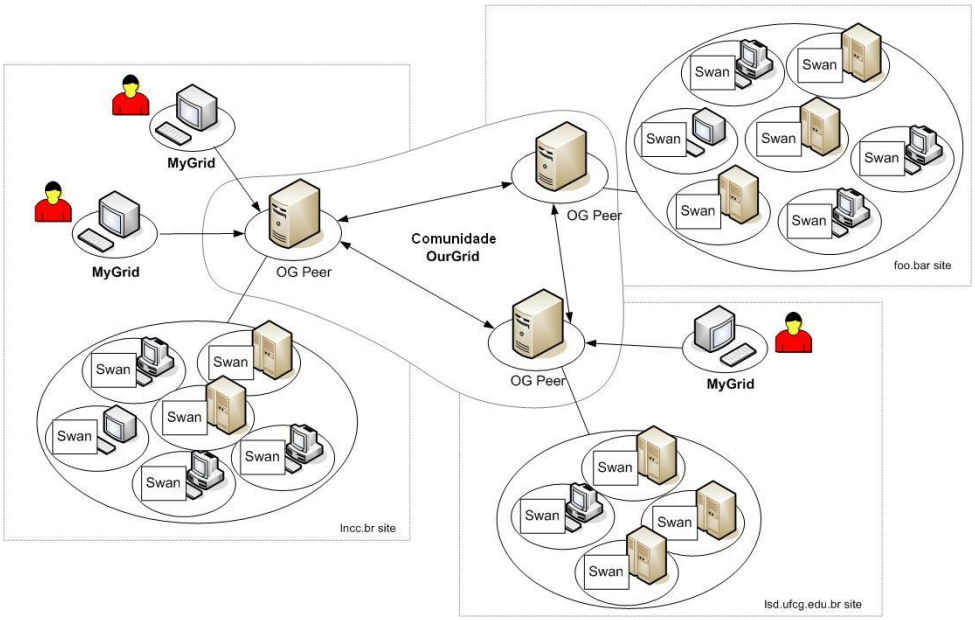

Figura 3.1: Componentes da arquitetura OurGrid

Os objetivos do projeto OurGrid são a simplicidade e as aplicações BoT, não apresentando características nativas que favoreçam a Composição de Serviços.

3.1.3 Globus Toolkit

O *Globus Toolkit* (GT) é uma ferramenta de *software* (*open source*), desenvolvido pela *Globus Alliance* que pode ser usado para construir sistemas de Grades e também para o

desenvolvimento de aplicações para as mesmas. Muitas companhias e organizações usam esse *toolkit* como base na implementação de Grades de vários tipos [36].

O GT foi evoluindo, gerando novos padrões, e convergindo para os padrões *Web* ao longo do tempo, como mostrado na Figura 3.2. Sua versão 3 (GT3 [37]) implementa a arquitetura OGSA com OGSI, enquanto que a versão 4, a atual (GT4 [38]), implementa OGSA com WSRF. Como neste trabalho foi usada a versão GT4 como plataforma para o desenvolvimento do GPO, esta versão está detalhada a seguir.

Figura 3.2: Evolução do GT e convergência com os padrões *Web*

O GT4 provê um conjunto de serviços de alto nível divididos em cinco categorias, sendo que cada categoria tem componentes baseados em Serviços *Web* (WS) e componentes que ***não são baseados*** em Serviços *Web* (NWS) (e servem principalmente para manter a compatibilidade com versões anteriores do GT). As cinco categorias são: componentes comuns de execução, segurança, gerência de dados, serviços de monitoramento e descoberta, e gerência de execução. As categorias e seus respectivos componentes estão relacionados na Figura 3.3 [29].

A seguir estão detalhados os componentes relevantes a esta dissertação.

- **Java WS Core**

  Java WS Core consiste de um conjunto de APIs e ferramentas que implementam os padrões WSRF e *WS-Notification* em linguagem *Java*. Esses componentes atuam como base para vários serviços que o GT4 provê, tais como, WS-GRAM, RFT (*Reliable File Transfer*), IS (*Information Services*), CAS (*Community Authorization Service*), *Delegation* e serviços WSRF personalizados (Figura 3.3). *Java WS Core* também contém a biblioteca básica para desenvolvimento de Serviços da Grade [80].

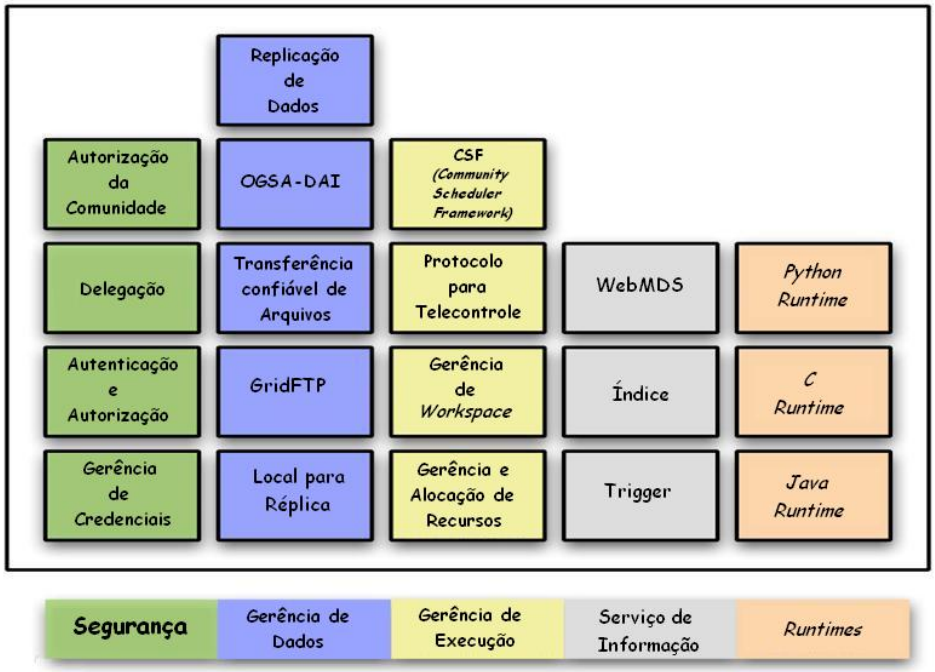

Figura 3.3: Serviços e componentes do GT4

Essa biblioteca e as ferramentas que compõem o *Java WS Core* foram usadas no desenvolvimento do GPO.

- **WS-GRAM**

  O WS-GRAM (*Web Service Globus Resource Allocation Manager*) é o Serviço que provê a execução e a gerência de tarefas na Grade [39]. Quando uma tarefa é submetida pelo cliente, a requisição é enviada ao *host* remoto como uma mensagem SOAP, sendo executada pelo serviço WS-GRAM localizado nesse *host* remoto. Para executar as submissões, o WS-GRAM usa outros serviços da Grade, como o *Delegation* para tratar aspectos de segurança (credenciais) e RFT-GridFTP para enviar e receber os dados da tarefa. O uso dos serviços de gerência do WS-GRAM combinados com os outros componentes do GT4 estão ilustrados na submissão típica que é mostrada na Figura 3.4. O serviço WS-GRAM usa *WS-Notification* para retornar as informações sobre a execução das tarefas. Além disso, o serviço WS-GRAM é capaz de submeter tarefas através de escalonadores [10] locais como *Platform LSF* [71] ou Condor [81].

  O WS-GRAM é formado por um conjunto de serviços que estão incorporados ao núcleo do GT4. Esses serviços são:

- ***ManagedJob***: Cada tarefa submetida é exposta como um recurso específico do serviço genérico *ManagedJob*, isto é, para cada tarefa uma instância desse serviço é criada, ficando responsável pela sua execução. O serviço provê uma interface que é usada para o monitorar o estado da execução da tarefa. Essa interface também é usada para encerrar a tarefa, e consequentemente encerrar (*destruir*) o recurso (a instância) *ManagedJob* criado.
- ***ManagedJobFactory***: Cada elemento computacional acessado através do escalonador local é exposto como um recurso distinto do serviço genérico *ManagedJobFactory*. Esse serviço provê uma interface para criar recursos *ManagedJob*, de acordo com o tipo de recurso apropriado para a tarefa alvo, no escalonador local. *ManagedJobFactory* é a *Fábrica* que produz as instâncias do *ManagedJob*.

Esse desenho arquitetônico foi usado na concepção do GPO. O GPO tem um serviço *Fábrica* que cria instâncias responsáveis pela execução de cada *workflow* submetido.

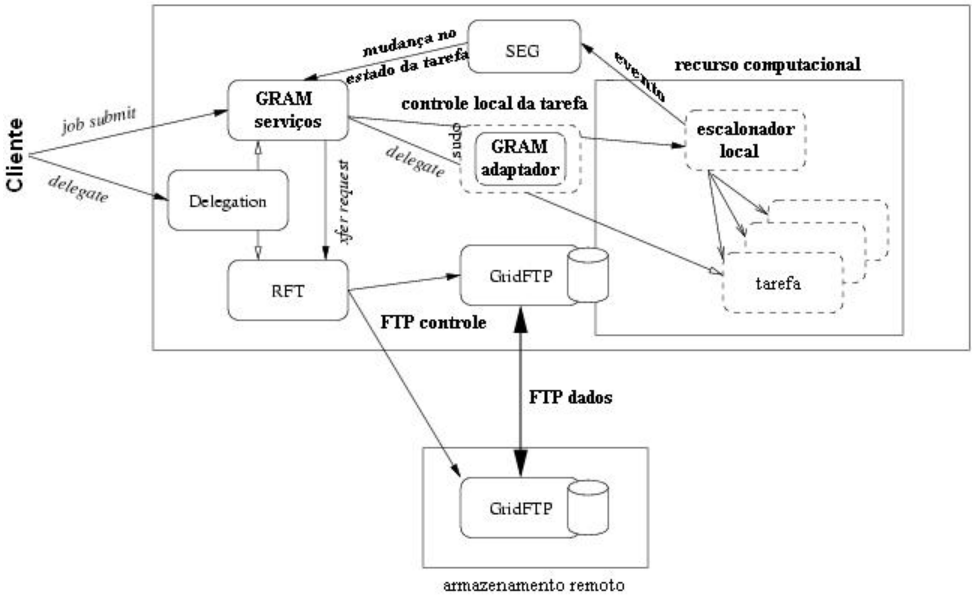

Figura 3.4: Uso do WS-GRAM na submissão de tarefas

O WS-GRAM recebe as submissões de tarefas, descritas através da GJD (*WS-GRAM Job Description*), e requisita as execuções nas máquinas remotas, verificando os aspectos relativos à segurança da comunicação, autenticando a submissão de tarefas.

GJD [40] é uma linguagem baseada em XML. Ela contém um conjunto restrito de diretivas que especificam o ambiente operacional, tempo de execução e argumentos,

dentre outras, conforme mostra a Figura 3.5. GJD é restrita no apoio ao usuário como ferramenta para *workflow*, além de não oferecer recursos para a execução de Serviços da Grade. GJD oferece recursos apenas para a execução de aplicações.

```
<job>
   <executable>/bin/echo</executable>
   <argument>Essa e uma string exemplo</argument>
   <argument>Globus esta aqui</argument>
   <stdout>${GLOBUS_USER_HOME}/stdout</stdout>
   <stderr>${GLOBUS_USER_HOME}/stderr</stderr>
</job>
```

Figura 3.5: Exemplo de submissão de tarefas em GJD

- ***Community Scheduler Framework***

  CSF (*Community Scheduler Framework*) é um conjunto (*open source*) de Serviços da Grade que podem desempenhar as funções de um meta-escalonador. O CSF tem as capacidades básicas de um escalonador e pode também ser usado como ferramenta para o desenvolvimento de escalonadores comunitários. O CSF disponibiliza uma interface simples para vários gerenciadores de recursos como *Platform LSF* [71], *Altair PBS* [1] e *Condor* [81]. Ele pode ser usado em conjunto com o GRAM e outros escalonadores, mas também não oferece facilidades para coordenar a execução de Serviços da Grade diretamente.

3.2 Composição de Serviços

Existem vários trabalhos sobre a composição de serviços. Os que estão apresentados a seguir discutem o uso de WS-BPEL no ambiente da Grade Computacional.

Em [18] é proposta uma arquitetura que usa BPEL4WS (versão 1.0 WS-BPEL) para definir processos de *workflow* que envolvam serviços das Grades OGSA-OGSI. É descrito um protótipo que mostra como a arquitetura proposta pode ser usada na modelagem, ou na orquestração de serviços da Grade com o uso de BPEL4WS. O trabalho destaca a dificuldade no endereçamento de serviços da Grade OGSA-OGSI (GSH e GHR) em BPEL4WS e mostra um exemplo de orquestração construído com essa linguagem.

Em [79] são discutidos os benefícios e os desafios do uso de WS-BPEL nas Grades. Mostra como o problema de endereçamento em Grades OGSA-OGSI contrasta com o *WS-Addressing* usado nos Serviços *Web*. Nesse aspecto a integração com OGSA-WSRF é natural pois ambos usam *WS-Addressisng*. O trabalho propõe o uso de "*pseudos parceiros*

(partners)" para facilitar a integração entre WS-BPEL e os serviços já existentes da Grade. O autor conclui que o modelo de execução WS-BPEL pode ser usado na execução de *workflows* nas Grades, mas são necessárias adaptações para que WS-BPEL possa atender a todos os requisitos.

Em [95] são apresentadas as principais características de WS-BPEL para orquestração de Serviços *Web*, e os benefícios de seu uso em ambientes SOA. O trabalho discute o uso de WS-BPEL com WSRF, destacando o relacionamento entre Serviços *Web*, recursos que mantêm o estado, e aspectos sobre a gerência do ciclo de vida desses recursos. É apresentado um exemplo de orquestração de Serviços, escrito em WS-BPEL, que usa o GT4 como implementação de WSRF.

3.3 Gerência de Workflow em Grades Computacionais

O ambiente heterogêneo provido pelas Grades Computacionais torna a gerência de *workflow* mais complexa, sendo necessário, por exemplo, gerenciar as múltiplas políticas de segurança com regras definidas pelas *organizações virtuais* participantes da Grade. Em grades orientadas a serviços como OGSA é necessário um novo conceito para gerenciar diretamente os serviços da grade sem a necessidade da construção de aplicações cliente para usá-los em sua plenitude.

A WfMC [94] define um *sistema de workflow* como sendo: "*Um sistema que define, cria e gerencia a execução de workflows através do uso de um software, executando um ou mais motores de workflow, capazes de interpretar a definição do processo, interagindo com os workflows participantes, invocando o uso de ferramentas e aplicações*".

Pela definição é possível perceber a necessidade de uma linguagem apropriada que descreva o *processo de negócio* e permita expor claramente as *regras e procedimentos* que devem ser respeitados. A definição destaca também o *motor de workflow* como a parte mais importante na interpretação e execução dos *workflows*. Nessa seção são examinados os principais trabalhos sobre gerência de *workflow* em Grades, analisadas as suas características e as linguagens propostas para descrever os *workflows*.

3.3.1 Sistemas de Gerência

A comunidade das Grades tem proposto várias soluções de propósito geral para gerência de *workflow*. XCAT [34], baseado na especificação CCA (*Common Component Architecture*), propõe um arquitetura com componentes distribuídos. *Triana* [82] oferece um modelo para programação visual a partir de módulos pré-definidos. Em [97] são discutidos e propostos algoritmos para composição de serviços e escalonamento direcionado a *workflow* em Grades. Permite composição, visualização, verificação, escalonamento e

monitoramento na execução dos *workflows*. Seu foco é na composição de serviços, não sendo ideal para escalonamento de tarefas, além disso foi desenvolvido para o GT3 (OGSA com OGSI). Em [23] é apresentada a *Grid Workflow Infrastructure*, baseada em OGSA e que usa os conceitos de WS-BPEL.

Outros trabalhos estão destacados a seguir:

- **GridFlow**: é um sistema de gerência de *workflow* baseado em agentes [16]. Ele tem uma arquitetura em duas camadas, uma para gerência de *workflow* na "Grade global" e outra camada para gerência e escalonamento de *sub-workflows* locais. Ele oferece uma interface gráfica para facilitar a composição dos elementos do *workflow* e um portal para o usuário fazer submissões.

- **GridAnt**: tem por objetivo ajudar o usuário na orquestração de atividades e suas dependências expressas em XML. Ele usa o *software* Ant [5] como motor no processamento de *workflows* [2] e tem tarefas pré-definidas para autenticação, transferência de arquivos e execução de tarefas. GridAnt permite reutilizar especificações de *workflow* baseadas em XML inclusas no Ant, que podem descrever execuções paralelas e seqüências. GridAnt é constituído por quatro componentes: um motor de *workflow*, um ambiente de execução, um vocabulário de *workflow* e um monitor de *workflow*. GridAnt apresenta várias limitações. Ele não permite execuções concorrentes, não oferece suporte iterativo que mostre o estado da execução, tem dificuldades para expressar condições e sua escalabilidade no paralelismo é limitada.

- **WebFlow**: é um sistema multi-nível para computação distribuída de alto desempenho com arquitetura formada por três camadas [9]. A camada superior é uma ferramenta baseada na *Web* para programação visual e monitoramento. Ela permite ao usuário compor novas aplicações usando componentes já existentes através da técnica de *arrastar e soltar*. A camada central é um servidor *Web* distribuído de fluxos implementado com extensões *Java* para servidores HTTP. A camada mais baixa usa o *Java CoG Kit* [53] para interface com a Grade, visando computação de alto desempenho. WebFlow usa o WS-GRAM como interface para submissão de tarefas no GT, adicionando a esse software alguma facilidade para programação visual.

- **Condor**: Condor é um *batch queueing system* para gerenciar tarefas de computação intensa [81]. Reuni as funções de um escalonador para Grade, um gerente de recursos e um sistema de balanceamento de carga num só produto. Condor-G [33] é a união das tecnologias Condor e Globus (GT). Globus é responsável pela infraestrutura da Grade enquanto que o Condor-G é usado para submissão e alocação de

tarefas, recuperação de erros (pontos de verificação) e pela criação de um ambiente amigável de execução. A Figura 3.6 mostra a integração entre as duas tecnologias.

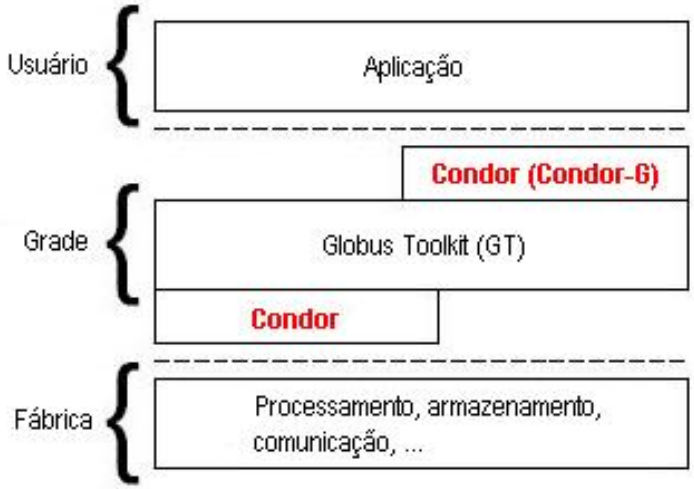

Figura 3.6: Condor na infra-estrutura Globus

Outro componente do projeto Condor é o meta-escalonador DAGMan (*Directed Acyclic Graph Manager*), um serviço para execução de múltiplas tarefas com dependências, na forma declarativa. A representação dessas dependências é especificada como parte de um DAG (*Directed Acyclic Graph* - Grafo Acíclico Direcionado), onde os nós representam programas e as arestas representam as dependências entre esses programas. Os DAGs são definidos através de diretivas armazenadas em arquivos de texto, onde cada nó representa uma tarefa *Condor*, que usa alguns arquivos como entrada e gera outros arquivos como saída da execução. Os arquivos são executados pelo uso do comando *condor_submit_dag* <*dagfile*>. O lado esquerdo da Figura 3.7 mostra o DAG representado pelas diretivas que estão no lado direito.

As vantagens do DAGMan são a simplicidade na especificação do *workflow* através de arquivo texto e a sua integração com o escalonador Condor-G. Como desvantagens não oferece uma representação XML das descrições, é um sistema fechado, o que dificulta a integração com outros escalonadores, não oferece uma interface gráfica e é direcionado para tarefas apresentando restrições quando usado na gerência de serviços *Web* e serviços da Grade.

- **Pegasus**: O principal diferencial do Pegasus [27] é que enquanto os outros sistemas usam *brokers* e estratégias de escalonamento, o Pegasus usa o conceito de *workflow abstrato*. Um *workflow abstrato* detalha os componentes do fluxo de forma genérica, não identificando os recursos ou a sua localização. Na execução desse *workflow*, o

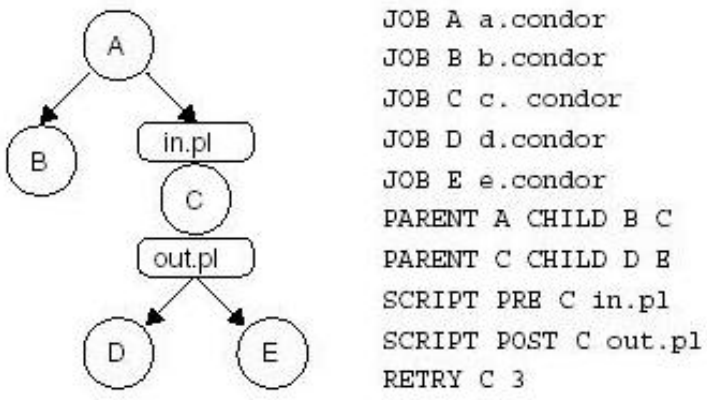

Figura 3.7: Uma submissão com o Condor DAGMan

sistema escolhe dentro dos recursos da Grade a melhor alternativa para executar o componente, de acordo com os recursos disponíveis na Grade. É uma alternativa interessante para "*Grades Oportunistas*". Pegasus pode alterar o *workflow abstrato* conforme o fluxo de dados necessário. O *workflow concreto* resultante é passado ao sistema de execução, como DAGMan, por exemplo. Em outras palavras o Pegasus analisa o *workflow abstrato* de acordo com os recursos disponíveis na Grade, gera o *workflow efetivo* na forma de um DAG, e submete esse DAG a um "*batch scheduler*" para ser executado. O Pegasus também pode endereçar apenas partes do *workflow* a cada vez, usando para isso, técnicas de planejamento "*just-in-time*". O Pegasus usa alguns Serviços do GT no processamento. Ele usa o *Serviço MDS* para descobrir recursos, usa o *Serviço GSI* para autenticação e usa o *Serviço WS-GRAM* para submissão tarefas remotas.

- **CoG Kits**: não é um sistema de gerência de *workflow*, mas um conjunto de ferramentas de *software* (*open source*) com *frameworks* de alto nível que permitem usar, desenvolver e administrar Grades [41]. Estão disponíveis para GT (3.2 e 4) e linguagens *Java* e *Python*, dentre outras opções, e provêem implementações GSI baseadas em *Java*, *GridFTP*, *myProxy* e clientes WS-GRAM. Aplicações cliente desenvolvidas com o kit têm a vantagem de permitir o acesso aos recursos da Grade mesmo que as bibliotecas e APIs não estejam instaladas.

  Um desses kits é o *Java CoG Kit* [53], cuja arquitetura é mostrada na Figura 3.8. Ele possui uma camada de provedores responsáveis pela ligação entre as aplicações e os *middlewares* da Grade, uma camada de abstração de *workflow*, uma camada de *frameworks de workflow* que faz a gerência de tarefas e de dados, uma camada de projeto e monitoramento de *workflow*, e a camada de aplicação que inclui suporte ao desenvolvimento de portais.

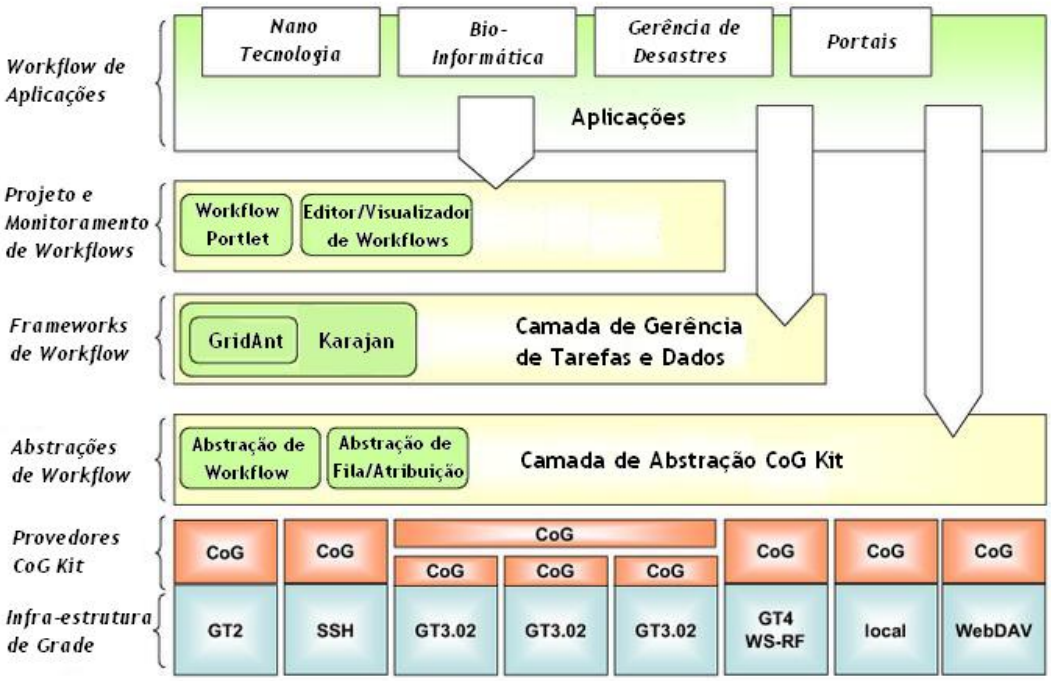

Figura 3.8: Camadas da arquitetura *Java CoG Kit*

O *Java CoG Kit Workflow* oferece três possibilidades para que usuários da Grade definam *workflows*. A primeira é um motor de execução e uma linguagem baseada em XML que juntos formam o *Karajan*. *Karajan* é uma evolução do *GridAnt*. A segunda possibilidade é uma abstração que faz parte do conjunto de APIs (requer desenvolvimento), e a terceira (um protótipo no momento) é uma linguagem tipo *shell script* para a definição de *workflows* simplificados. Através da abstração o *Kit* oferece interoperabilidade entre as várias versões da Grade (GT2, 3 e 4).

O *Karajan* [46] oferece tratamento de falhas, pontos de verificação, e *workflows* dinâmicos e distribuídos. Seu motor de execução é baseado em um modelo de eventos e usa notificação para acompanhar o estado da execução do *workflow*.

3.3.2 Linguagens

A escolha ou definição de uma linguagem para que o usuário possa descrever os *workflows* é fundamental. A linguagem deve descrever o *processo de negócio* e expor claramente as *regras e procedimentos* que devem ser respeitados. A seguir são apresentadas algumas linguagens propostas para *workflow*.

- **GSFL**: *Grid Services Flow Language* é uma linguagem baseada em XML que permite descrever *workflows* envolvendo Serviços em Grades baseadas em OGSA [52]. Sua arquitetura, mostrada na Figura 3.9, é composta por um conjunto de modelos. Um *workflow* GSFL deve preencher as informações conforme esses modelos. Os mais significativos são:

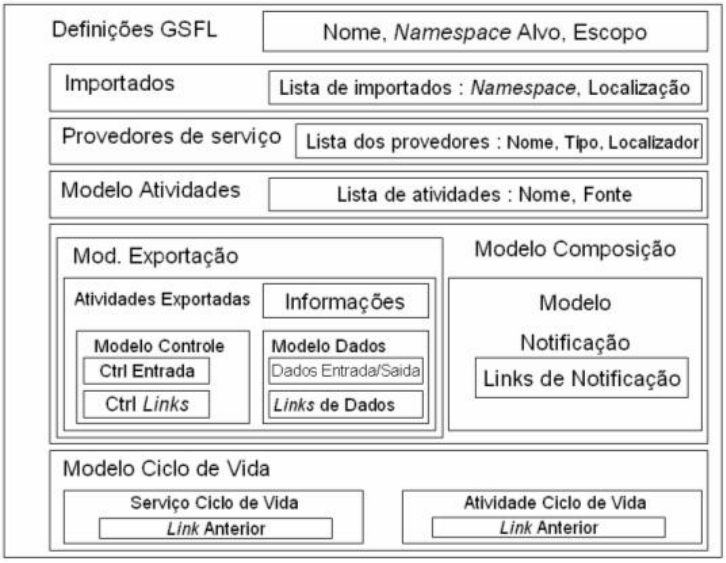

Figura 3.9: Arquitetura GSFL

  - **Modelo provedores de serviços**: contém a lista de serviços que fazem parte do *workflow*. Especifica os provedores do serviço, indicados através do nome, tipo e o localizador (que pode ser estático, fábrica ou registro).
  - **Modelo de atividade**: especifica as atividades a serem executadas, indicando o nome e a referência para a operação que implementa a atividade (serviço, o *portType* e a operação).
  - **Modelo de Composição**: descreve a composição dos serviços participantes. Descreve o fluxo de controle e de dados entre as operações dos serviços. É divido nos modelos *Export* (define as atividade que expõem operações) e *Notification* (define a comunicação direta entre os serviços).

- **Modelo de ciclo de vida**: define o ciclo de vida dos serviços e das atividades do *workflow*. Ordena os serviços como um DAG usando *links* de precedência, e também ordena as atividades através de *links* de precedência.

GSFL claramente usa WS-BPEL como base e incorpora diretivas para atender aos requisitos das Grades, como por exemplo, o localizador indicar o serviço como fábrica. Foi originalmente projetada para Grades OGSA com OGSI (GT3).

- **JSDL**: *Job Submission Description Language* é uma especificação do OGF para descrever os requerimentos de uma tarefa computacional para submissões em recursos das Grades [66]. Em JSDL os documentos são descritos em XML e tem seus marcadores (elementos) definidos conforme um "*XML Schema*". Esses documentos são organizados da seguinte forma: *JobDefinition* é o elemento raiz e que contém um filho obrigatório chamado *JobDescription* onde estão os elementos que descrevem a tarefa. Esses elementos são *JobIdentification*, *Application*, *Resource* e *DataStaging*.

  A JSDL foi projetada para descrever a submissão de uma tarefa individual a ser executada em um recurso. Ela pode ser usada como parte integrante de uma linguagem de *workflow* que envolve, não só a submissão de tarefas, mas também as dependências entre elas. Além disso *workflows* estão envolvidos com outras atividades como tratamento de falhas e monitoramento da execução. JSDL não é apropriada para descrições de *workflow*.

- **Karajan**: parte integrante do *Java CoG Kit*, é um *framework* composto de uma linguagem e um motor de *workflow*. A linguagem projetada oferece duas possibilidades de uso. Uma possibilidade é usar uma sintaxe baseada em XML, com origem no GridAnt, com facilidades como estruturas de controle. Outra opção é usar uma sintaxe simplificada para gerar *scripts*, limitada mas que oferece um mecanismo para prototipação ágil. Uma opção pode ser convertida na outra. A linguagem provê primitivas para seqüências genéricas, execuções paralelas, execuções condicionais, variáveis e suporta tipos de dados comuns. Tem alguns recursos diferenciados que foram criados para atender a requisitos específicos das Grades. Esses recursos são:

  - ***Providers***: é um marcador (*tag*) que permite abstrair o *middleware* da Grade onde está o recurso a ser usado na execução. O programador não necessita levar em conta as particularidades no uso dos recursos. É uma facilidade importante. Por exemplo, no GT3 os recursos usam endereçamento baseado em duas referências (GSH e GSR), enquanto que o GT4 usa apenas uma referência (EPR).

- ***Element***: permite a definição de novos elementos na linguagem. É semelhante à definição de funções em linguagens procedurais.
- ***Task***: é uma abstração que permite fácil adaptação aos diversos protocolos do *middleware* da Grade. Três marcadores podem ser usados nesse caso. O marcador *execute* (execução de uma tarefa em um *host* remoto), *transfer* (transferência de arquivos, geralmente relacionados à execução) e *authenticate* (para autenticação, no GT usado com GSI).
- ***Namespace***: define elementos que podem ser usados como prefixos na indicação do caminho do recurso. O *Namespace* criado pode ser usado como qualquer variável dentro do *workflow*.

A Figura 3.10 mostra um *workflow Karajan* que executa uma compilação *Java* remota usando a abstração *task*.

```
<project name="RemoteCompilation">
   <include file="cogkit.xml"/>
   <task:authenticate provider="GSI"/>
   <set name="host"
        value="hot.mcs.anl.gov"/>
   <set name="path"
        value="/usr/local/j2sdk1.4.2"/>
   <task:transfer desthost="{host}"
         provider="gridftp"
         srcfile="Test.java" />
   <task:Execute host="{host}" provider="gt4"
         executable="{path}/bin/javac"
         arguments="Test.java"/>
</project>
```

Figura 3.10: *Workflow Karajan* executando compilação Java remota

Como pode ser observado no exemplo, a linguagem do *Karajan* tem vários recursos importantes, mas está muito mais próxima de um programador do que de um usuário da Grade. Além disso não oferece recursos para a execução direta de Serviços da Grade.

- ***XML-based Description Language***: é uma linguagem e um ambiente de execução para orquestração de tarefas em Grades [73]. O trabalho propõe uma ferramenta flexível com recursos avançados para monitoramento, tratamento de falhas, liberação de recursos e que permita o desenvolvimento de aplicações robustas. Usa o *Java CoG Kit* como plataforma de desenvolvimento e propõe uma linguagem baseada no *Karajan*, com o objetivo principal de atender desenvolvedores e usuários

avançados. A linguagem oferece suporte a vários tipos de tarefas como: execução computacional (<*job*>), transferência de arquivos (<*filetransfer*>), invocação de serviços (<*service*>), envio de e-mails (<*mail*>), e uso de *scripts* e objetos *Java* (<*java*>, <*jobject*>). A linguagem apresenta algumas limitações não oferecendo, por exemplo, estruturas para controle de fluxo (*if*, *while*, etc). Além disso é orientada a desenvolvedores o que dificulta o seu uso pelo usuários menos experientes.

Projeto *GriPhyN*

Um trabalho muito interessante que usa de forma combinada várias tecnologias e alguns produtos é o projeto GriPhyN (*Grid Physics Network*) [45]. GriPhyN é focado no suporte a aplicações de larga escala, cuja idéia central é a virtualização dos dados, onde estes podem estar disponíveis em algum sistema de armazenamento, ou podem "existir" na forma de recipiente (ou *workfow* a ser executado) [26]. Quando um usuário requisita um conjunto de dados, o sistema (composto por vários serviços) avalia a requisição e gera um "*workflow abstrato*". Esse "*workflow abstrato*" acessa os recursos necessários e executa a requisição.

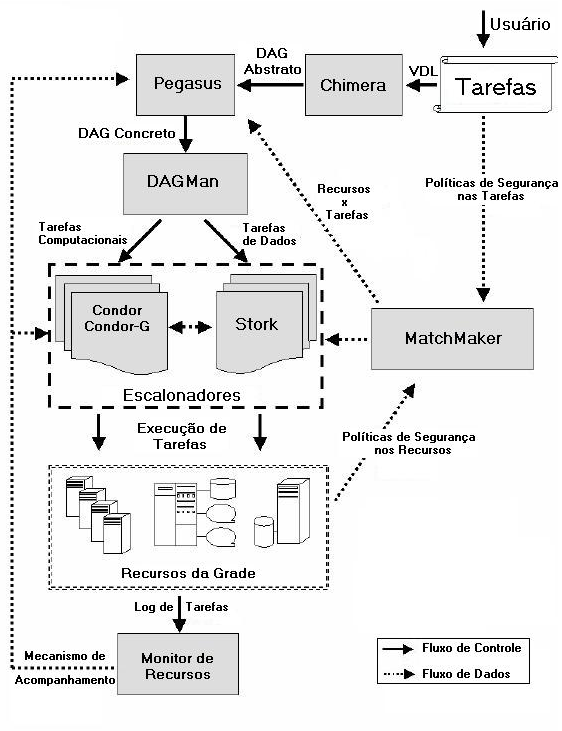

Figura 3.11: Componentes da gerência de *workflow* do projeto GriPhyN

A Figura 3.11 mostra os vários serviços que compõem a arquitetura do projeto. A definição da tarefa é feita em VDL (*Virtual Data Language*) e passada ao *Chimera* [28],

que é responsável pela criação do ADAG (*DAG abstrato*). O ADAG é passado ao *Pegasus* que faz as verificações e otimizações gerando o CDAG (*DAG concreto*). O CDAG é enviado ao *Condor DAGMan* que segue a execução padrão de submissão. As tarefas são executadas conforme a sua aplicação. Tarefas de computação são processadas pelo *Condor-G* enquanto que as tarefas de dados são processadas pelo *Stork* [51]. Esses escalonadores acessam os recursos da Grade.

Capítulo 4

Linguagem Proposta: GPOL

As Grades baseadas em OGSA com WSRF combinam Serviços *Web* e Serviços da Grade, tornando esse ambiente apropriado tanto aos processos de negócios como também ao processamento científico. Nesse ambiente, qualquer que seja o tipo de processo, é importante a forma como os Serviços são compostos.

Para fazer essa composição foi criada a GPOL (*Grid Process Orchestration Language*). GPOL é uma linguagem compacta baseada em XML, que contém primitivas que permitem ao usuário, descrever o relacionamento entre tarefas e serviços fortemente acoplados, formando um *workflow* para ser usado em Grades OGSA. GPOL une os conceitos de orquestração de serviços propostos em WS-BPEL às facilidade oferecidas pela *WS-GRAM Job Description*. Além disso, acrescenta diretivas para tratar requisitos específicos da Grade, como: manutenção de estado, serviços potencialmente transientes, notificação, serviços orientados a dados e serviços orientados a grupos.

A GPOL herda de WS-BPEL o forte suporte a Serviços *Web* e usa a notação *WS-Addressing* de pontos de referência (EPR), para simplificar a passagem de endereços entre os Serviços participantes do *workflow*. Essa mesma notação é usada nas Grades baseadas em OGSA com WSRF (GT4) para endereçar os Serviços da Grade.

No presente trabalho optou-se por criar uma linguagem simplificada e não usar a especificação WS-BPEL nos *workflows*. Essa escolha está relacionada ao objetivo central da WS-BPEL, que é um "*processo de negócio*" e pode envolver vários participantes. Para atingir esse objetivo WS-BPEL usa troca de mensagens (<*message*>) e estabelece ligações (<*partnerLink*>) entre os parceiros participantes.

O objetivo do usuário da Grade é executar tarefas e usar serviços. O usuário estabelece no *workflow* as regras para a execução das tarefas e dos serviços. Não existem usuários participantes e a necessidade de estabelecer ligações, ou troca de mensagens entre eles. Além disso, a GPOL atende requisitos específicos das Grades, como por exemplo, transiência, ciclo de vida, transferência de arquivos e execução de tarefas.

A GPOL oferece um conjunto completo de facilidades.

- Ela tem tratamento de variáveis, recurso importante na manutenção das informações entre os serviços (<*variables*>, <*assign*>, <*clear*>);
- Oferece as principais diretivas para controle de fluxo (<*if*>, <*while*>);
- Permite execuções seqüenciais e paralelas (<*sequence*>, <*flow*>). Isso possibilita o uso eficiente dos recursos disponíveis na Grade, inclusive com facilidades para a descoberta de recursos similares em Grades Oportunistas;
- Transferência de arquivos (<*transfer*>) e execução de tarefas (<*execute*>);
- Permite invocar serviços da Grade (<*invoke*>) diretamente no *workflow*, não sendo obrigatório o uso de programas clientes para fazê-lo; e
- Aceita código *Java* no *workflow* (<*javacode*>). Esse recurso usado em conjunto com o elemento <*import*>, permite ao usuário criar e agregar novas facilidades ao *middleware* proposto.

A GPOL é compacta reunindo as principais facilidades apresentadas pelas outras linguagens. Como diferenciais a GPOL oferece:

- Tratamento para variáveis e recursos para execução de tarefas e invocar serviços (JSDL não trata serviços);
- Estruturas para controle de fluxo (JSDL e *XML-based Description Language* não oferecem);
- Diretivas para que o usuário possa usar o conceito de fábrica/instância dos serviços da Grade (WS-BPEL não oferece esse recurso);
- Endereçamento dos serviços (*Endpoint Reference - WS-Addressing*) conforme a especificação WSRF (JSDL e *XML-based Description Language* não oferecem); e
- Aceita código Java como parte do *workflow* (WS-BPEL não oferece esse recurso).

Além disso, a GPOL é direcionada ao usuário da Grade (mesmo iniciantes), usando diretivas e expressões (dialeto) mais coerentes com esse ambiente, como por exemplo, *provider*, *transfer* e *host*. WS-BPEL e GSFL são direcionadas a participantes em processos de negócios, e usam expressões que não são comumente usadas nas Grades (*partner*, *controlLink*, etc).

4.1 A Estrutura de um *Workflow* GPOL

Em uma visão geral, um *workflow* GPOL é similar a um *processo de negócio* WS-BPEL. Ele é composto por dois elementos: definições (<*definitions*>) e processo (<*process*>). Nas *definições* são feitas as declarações das variáveis e identificados os serviços que serão usados na execução do *workflow*. No *processo* é definida a ordem na qual os serviços são acionados, como tratar seus retornos, o que fazer em caso de falha, indicar a destruição de instâncias e indicar o valor a ser retornado ao GPO *Run* após a execução. Ou seja, as definições indicam o que será usado, e o processo indica como será usado.

A Figura 4.1 mostra um *workflow* descrito em GPOL que apresenta uma mensagem na saída padrão de vídeo.

```
<workflow name="Hello">
 <definitions name="HelloDefs">
  <variables>
   <variable name="sMsg" type="string"
          value="GPOL - Hello!"/>
  </variables>
 </definitions>
```

```
<process name="HelloProcess">
 <execute>
   executable="/bin/echo"
   <argument variable="sMsg"/>
 </execute>
</process>
</workflow>
```

Figura 4.1: Exemplo de *workflow* em GPOL

Uma submissão GPOL usa como *raiz* do documento (XML Seção 2.1), o elemento <*workflow*>. No exemplo da Figura 4.1, o *workflow* definido é chamando de "*Hello*" (*name*). Ele contém a variável de nome "*sMsg*", do tipo "*string*", cujo valor inicial é a mensagem "GPOL - Hello!". Essa é a única definição feita no elemento <*variables*> inserido nas definições (<*definitions*>).

Seu processo (<*process*>) tem um único elemento <*execute*>. No elemento <*execute*> está especificado que deve ser executado o utilitário *echo* (*/bin/echo*), cuja *string* argumento é o conteúdo da variável *sMsg*, a mensagem "*GPOL - Hello!*".

Em um *workflow* GPOL a área de definições pode conter os elementos <*variables*> e <*services*>, e o processo pode conter qualquer combinação das atividades GPOL (<*invoke*>, <*assign*>, <*if*>, <*while*>, <*clear*>, <*sequence*>, <*flow*>, <*return*>, <*exit*>, <*echo*>, <*execute*>, <*transfer*> e <*javacode*>). As seções a seguir mostram a notação usada, fazem algumas considerações de caráter geral e detalham cada um dos elementos da linguagem.

4.2 Notação e Considerações Gerais

A linguagem GPOL é baseada na linguagem XML e usa elementos XML (marcadores - *tags*) como seu bloco básico construtivo. Na especificação da linguagem GPOL são usadas as seguintes convenções:

- Alguns caracteres podem ser adicionados no final de elementos, atributos e descrições e indicam: "?" (0 ou 1), "*" (0 ou mais) e "+" (1 ou mais) repetições. Os caracteres "[" e "]" indicam um grupo de elementos ou atributos, e "|" indica que uma das opções deve ser escolhida.
- Prefixos *namespace* XML são usados para indicar o *namespace* do elemento que está sendo definido.
- GPOL depende das especificações WSDL 2.0 [19], XML Schema 1.0 [89] e WS-Addressing. Esquemas XSD [90, 91] e definições WSDL são usadas como prefixos em definições formais de alguns itens da gramática. Esses prefixos indicam *namespaces* associados com as URIs descritas abaixo:

  xsd - *"http://www.w3.org/2001/XMLSchema"*;

  wsdl - *"http://schemas.xmlsoap.org/wsdl/"*;

  xsi - *"http://www.w3.org/2001/XMLSchema-instance"*;

  wsa - *"http://schemas.xmlsoap.org/ws/2004/03/addressing/"*;

  gjd - *"http://www.globus.org/namespaces/2004/10/gram/job/description/"* e

  gjt - *"http://www.globus.org/namespaces/2004/10/gram/job/types/"*

4.2.1 *Atividade*

A linguagem GPOL define como *Atividade* um grupo de elementos que podem ser usados de forma individual ou combinada dentro do processo. Uma *Atividade* é um elemento que esteja de alguma forma relacionado com processamento. As atividades podem ser básicas (um processamento atômico) ou estruturadas (várias atividades associadas por uma estrutura). As atividades básicas são: *invoke*, *assign*, *clear*, *return*, *exit*, *echo* e *transfer*. As estruturadas são: *if*, *while*, *sequence*, *flow*, *execute* e *javacode*. Todas esses elementos estão descritos em seus respectivos itens.

4.2.2 *Expressão*

Expressão é a forma de fazer comparações simples em GPOL. As expressões são usadas em alguns elementos GPOL (*if, while*), que podem fazer parte de um processo. Sua forma

geral é:

variable operator sExpression

O atributo *variable* é comparado com *sExpression* conforme indicado por *operator*. O resultado é um valor booleano (verdadeiro ou falso) que é usado nos locais onde estão as expressões. Como *variable* deve ser usada uma variável declarada em *definitions*. O atributo *operator* deve ser uma das seguintes opções:

- == (*eq*) verdadeiro se iguais;
- != (*ne*) verdadeiro se diferentes;
- <= (*le*) verdadeiro se menor ou igual que;
- >= (*ge*) verdadeiro se maior ou igual que;
- < (*lt*) verdadeiro se menor que e
- > (*gt*) verdadeiro se maior que.

Os parênteses indicam a notação alternativa. Por exemplo, "*flag == 1*" e "*flag eq 1*" são equivalentes. O atributo *sExpression* pode ser qualquer combinação válida envolvendo operadores, variáveis e valores. Os operadores aceitos são ( ), *, /, +, -, !, | e &, com a seguinte ordem de precedência: (e) precedem !, que precede * e /, que precedem + e -, que precedem | e &. No protótipo implementado o atributo *sExpression* pode ser preenchido apenas com *variable* ou *value*. A coerência entre os tipos comparados é de responsabilidade do usuário.

4.3 Elementos da Linguagem

Nessa seção são detalhados todos os elementos da linguagem. Para cada um deles são apresentadas as funcionalidades, a sintaxe para uso, a definição dos atributos e quais elementos podem ser usados de forma combinada internamente. As descrições começam com o elemento <*workflow*>.

4.3.1 *workflow*

Um documento GPOL é definido basicamente pelo elemento <*workflow*>, a raiz do documento. Todos os outros elementos são *filhos* de <*workflow*>. Sua sintaxe é:

```
<workflow
    name="xsd:NCName"
    targetNamespace="xsd:anyURI"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl"
    xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable">

    <import>?
    <definitions>
    <process>
    <workflow>*
</workflow>
```

O elemento *<workflow>* tem como atributos *name* e *targetNamespace*, e aceita outras indicações de esquemas (*XML Schema*) que podem ser usados no *workflow*. O atributo *name* indica um nome para o *workflow*. Esse nome é usado no monitoramento da execução, facilitando a identificação por parte do usuário. O atributo *targetNamespace* indica o espaço local que será associado a todas as definições usadas no *workflow*.

Além dos atributos, o elemento *workflow* é constituído pelos elementos *definitions* (Seção 4.3.3) e *process* (Seção 4.3.4). Esses dois elementos combinados é que realmente formam o *workflow* GPOL. O elemento *import* é opcional (Seção 4.3.2). Outros detalhes sobre esses elementos são encontrados nos itens correspondentes.

Como mostrado na definição da sintaxe, o próprio elemento *workflow* pode ser usado em sua definição. Isso permite ao usuário criar um *workflow* que organize a execução sequencial de vários *workflows*. Para cada elemento *workflow* usado, é criada uma instância do GPO Maestro responsável pela sua execução (Seção 5.3). Dessa forma cada execução é totalmente independente, não havendo qualquer tipo de relacionamento entre os processamentos, inclusive não havendo a garantia na ordem das execuções.

4.3.2 *import*

O elemento *import* é usado para declarar uma dependência que está em um arquivo *XML Schema* ou *WSDL* externo. Ele pode também ser usado para "incluir" um arquivo GPOL, como parte do *workflow* (similar ao *include* usado em linguagens como *C* ou *assembler*). Sua sintaxe é:

```
<import
```

namespace="xsd:anyURI"?
location="*xsd:anyURI*"?
importType="*xsd:anyURI*"?
file="*xsd:string*"*/>

O elemento *import* tem quatro atributos: *namespace*, *location*, *importType* e *file*. O atributo *namespace* especifica a URI que identifica as definições importadas. O atributo *namespace* é opcional, mas se usado especifica que as definições importadas devem estar nesse *namespace*.

O atributo *location* contém a URI que indica a localização do documento com as definições. Obrigatório se o atributo *namespace* for usado.

O atributo *importType* identifica o tipo de documento, e o seu valor deve constar das especificações *xsd* e *wsdl*. Obrigatório se o atributo *namespace* for usado.

O atributo *file* indica a localização de um arquivo que será incluído como parte do *workflow*. Ele é opcional e também aceita repetições. Esse atributo permite a modularização do *workflow*. O usuário pode criar uma "*biblioteca*" com especificações e combiná-las formando um *workflow*, permitindo a reutilização de fragmentos de código de grande uso.

4.3.3 *definitions*

Um *workflow* GPOL é formado pelos elementos *definitions* e *process*. O elemento *definitions* contém a definição das variáveis e serviços que serão usados no processo (*process*). A sintaxe para *definitions* é:

<*definitions name*="*xsd:NCName*">
  <*variables*>*
  <*services*>*
</*definitions*>

O elemento *definitions* tem em *name* o seu único atributo. Esse atributo é opcional, e quando usado, deve ser uma identificação única para as definições do *workflow*. Essa identificação poderá ser útil no monitoramento da execução.

Os elementos *variables* e *services* podem fazer parte de *definitions*. O elemento *variables* contém a definição das variáveis que serão usadas durante a execução do *workflow*. *Variables* (Seção 4.3.5) é formado por um ou mais elementos *variable* (Seção 4.3.5), onde são descritas as propriedades de cada variável a ser usada no *workflow*. O elemento não é necessário se não forem usadas variáveis no processo.

O elemento *services* (Seção 4.3.6) contém um ou mais elementos *service*. Cada elemento *service* descreve um dos serviço da Grade que será invocado durante a execução do *workflow*. Se nenhum serviço for usado no *workflow*, não é necessário incluir o elemento *services* em *definitions*.

4.3.4 *process*

O elemento *process* define os passos a serem executados pelo *workflow*. Em *process* são executadas tarefas e invocados os serviços definidos em *definitions*. *Process* usa as variáveis como argumento em chamadas, para salvar valores retornados por serviços ou para fazer pequenos cálculos. Em *process* o usuário pode combinar livremente as *atividades* e estabelecer um tratamento de falhas global para todos os serviços e tarefas. A sua sintaxe é:

```
<process
   name="xsd:NCName"
   targetNamespace="xsd:anyURI"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <faultHandlers>?
   Atividade+
</process>
```

Três atributos podem ser usados em *process*: *name*, *targetNamespace* e *namespace*. O atributo opcional *name* deve ser uma identificação única do processo. Essa identificação poderá ser útil no monitoramento da execução. O atributo *targetNamespace* é o *namespace* local do processo. Ele é semelhante ao *namespace* global especificado no elemento *workflow*. *Process* aceita a indicação de outros espaços de nomes que devem ser incorporadas ao processo.

A GPOL permite qualquer combinação de *atividades* como elementos em *process*.

O elemento *faultHandlers* aciona o tratamento de falhas geral para o processo (Seção 4.3.7). Se não houver uma indicação de local no serviço invocado, será usada o tratamento padrão aqui indicado caso uma falha ocorra.

4.3.5 *variables* e *variable*

Variáveis tem importância significativa em *workflows* nas Grades, mesmo contando com o fato de que os serviços das Grades tem facilidades para a manutenção do estado. Em Grades OGSA com WSRF, os dados tratados nos serviços são guardados em áreas definidas como recursos.

Mas um *workflow* pode executar tarefas que não tem essa capacidade. Além disso, cada serviço pode manter o seu estado, mas são necessárias variáveis para apoiar a passagem de informações entre os serviços.

A GPOL permite ao usuário criar variáveis através dos elementos *variables* e *variable*. A sintaxe desses elementos é:

```
<variables>
   <variable>
       name="xsd:Name"
       type="tns:VariableType"
       value="xsd:string"?
   < /variable>+
</variables>
```

O elemento *variables* agrupa todas as definições de variáveis do *workflow*. Ele não tem atributos e deve conter pelo menos um elemento *variable* em sua definição.

Não é o objetivo das variáveis conter um grande volume de dados a serem trocados entre os serviços ou tarefas. A troca de grandes volumes deve ser feita através de arquivos. Nesse caso as variáveis podem ser usadas para passar a localização de onde estão os dados.

Cada variável deve ser definida através do elemento *variable*. O atributo *name* indica o nome de cada variável e deve ser único dentro do *workflow*. O atributo *VariableType*, referenciado como um tipo simples do *XML Schema*, define o tipo de cada variável. Os tipos válidos são: *string*, *boolean*, *decimal*, *integer*, *float* e *double*. Não é permitido declarar tipos diferentes para a mesma variável. O atributo opcional *value* deve conter uma *string* que represente um valor válido conforme o tipo declarado. Esse é o valor inicial atribuído à variável declarada. O valor padrão é zero para valores numéricos e *string nula* para variáveis do tipo *string*. Por exemplo:

```
<variable name="Quantidade" type="integer" value="10"/>
```

declara a variável *Quantidade*, tipo *inteiro* e valor inicial *10*.

A responsabilidade pela consistência de conteúdo nas variáveis é responsabilidade do usuário.

4.3.6 *services* e *service*

Uma característica importante do GPO é permitir ao usuário invocar serviços da Grade diretamente no *workflow*. Para invocar um serviço, o usuário deve declará-lo em *defini-*

tions para usá-lo em *process*. A definição do serviço informa sua localização, seu tipo e fornece um nome único para referenciá-lo no processo. Essa definição é feita através do elemento *service*, e todas as definições devem estar agrupadas no elemento *services*. A sintaxe para os elementos *services* e *service* é:

```
<services>
   Atributos-gerais
   <service>
      name="xsd:Name"
      host=["xsd:anyURI" | "local" | "anyHost"]?
      uri="xsa:EndpointReferenceType"?
      fal="xsd:anyURI"?
      sal="xsd:anyURI"?
      type="[Factory | Instance]"
      gsh="xsd:anyURI"?
      provider="tns:ProviderType"?
      ret-type="tns:VariableType"?
   < /service>+
</services>
```

O elemento *service* tem como atributos: *name*, *host*, *uri*, *fal*, *sal*, *type*, *gsh*, *provider* e *ret-type*. O atributo *name* indica o nome (apelido) de cada serviço, que deve ser único dentro do *workflow*, e que será usado para invocação no processo.

O atributo *ret-type*, referenciado como um tipo simples do *XML Schema*, define o tipo para o valor de retorno após a execução do serviço. São os mesmos tipos aceitos na definição de variáveis.

O atributo *type* indica qual o tipo de serviço a ser invocado. Pode ser *Factory* ou *Instance*. Esse atributo está relacionado com o conceito de *"Fábrica de Serviços"* usado nas Grades baseadas em OGSA. Se o tipo indicado for *Factory*, o GPO criará uma nova instância desse serviço para usá-la durante a execução do processo. Esse recurso é muito flexível permitindo ao usuário criar várias instâncias de mesmo serviço da Grade. O GPO também controla o *ciclo de vida* dessa instância, ou seja, ao final da execução do *workflow* o GPO destruirá todas as instâncias por ele criadas durante a execução. O valor padrão para *type* é *Factory*.

Os atributos *uri* e *gsh* indicam a localização do serviço da Grade a ser invocado. O uso de uma ou outra opção está relacionado com o provedor.

O atributo *host* pode ser usado em conjunto com *uri* ou *gsh*, *host* indica a localização dentro da Grade e *uri/gsh* o serviço alvo. Isso facilita a definição para uso *local*, pois não

será necessário indicar o caminho completo. A opção *anyHost* é destinada principalmente às "*Grades Oportunista*", informando que qualquer *host* que tenha o serviço pode ser usado.

O atributo *provider* indica qual é o tipo de provedor onde está o serviço alvo. Ele pode ser GT4 ou GT3, indicando ser o Globus Toolkit versão 4 ou 3 respectivamente. Quando o provedor for GT4 será usado o atributo *uri* para localização do serviço. Se o provedor indicado for GT3 a localização será feita por *gsh*. O valor padrão para *provider* é GT4.

Os atributos *fal* (*Factory Addressing Locator*) e *sal* (*Service Addresing Locator*) devem indicar a localização dos *stubs* do serviço alvo. Se o serviço for do tipo *Instance* é necessário indicar apenas o atributo *sal*.

O elemento *services* agrupa todas as definições de serviços que são usados no *workflow*. Em *Atributos-gerais* é possível incluir todos os atributos aceitos em <*service*> exceto o nome (*name*). Todo atributo definido nessa área será usado nas definições *service*, podendo ser alterado dentro de cada definição individual. A possibilidade de invocar serviços da Grade diretamente no *workflow* é talvez a maior facilidade oferecida pelo GPO, evitando que o usuário tenha que criar programas cliente para poder acessar os serviços.

4.3.7 *faultHandlers*

O GPO Maestro e o GPO Run mantêm um conjunto de variáveis para acompanhar a execução do *workflow* (Seção 5.3). Essas variáveis podem ser acessadas pelo GPO Run através de métodos exclusivos, e podem ser consultadas pelo usuário através da opção *-stat*. Parte desse conjunto de variáveis está relacionada com o controle de falhas.

Através do elemento opcional *faultHandlers* é possível definir o tratamento personalizado para falhas que ocorrerem no *workflow*. A sintaxe de *faultHandlers* é:

```
<faultHandlers>
   <catch
      faultName="xsd:NCName"
      faultVariable="xsd:Name"?>
      Atividade*
   < /catch>*
   <catchAll>
      Atividade*
   < /catchAll>?
</faultHandlers>
```

O elemento *faultHandlers* não tem atributos e pode ser constituído por um elemento *catchAll* ou por um ou mais elemento *catch*. Esses elementos oferecem facilidades para acessar o conjunto de variáveis que são usadas para indicar falhas que possam ocorrer.

Podem ser usados um ou vários elementos *catch*. Em cada um deles é possível sinalizar com o atributo *faultName* a falha a ser rastreada, e com o atributo *faultVariable* a variável associada a essa falha. O elemento *catch* permite indicar uma ou mais atividades que são úteis no tratamento da falha.

O elemento *catchAll* aciona a captura de todas as falhas que possam ocorrer no *workflow* e também permite indicar atividades.

O tratamento de falhas oferecido pelo GPO visa as falhas do sistema. O tratamento de falhas envolvendo atividades do *workflow* (a inclusão do elemento *throw*, por exemplo) será considerada em trabalhos futuros.

4.3.8 *invoke*

O elemento *invoke* é usado para invocar uma operação de um serviço da Grade. *Invoke* funciona de forma síncrona, ou seja, ao ser executado, o processamento do *workflow* fica aguardando o término da operação para então continuar o processamento. A sintaxe desse elemento é:

```
<invoke>
   service="xsd:Name"
   operation="xsd:NCName"
   <argument>
      [variable="xsd:Name" |
      value="xsd:string"]
      type="tns:VariableType"?
   </argument>*
   <return>
      [variable="xsd:Name" |
      value="xsd:string"]
      type="tns:VariableType"?
   </return>?
   [<faultHandlers?> | <catch*> | <catchAll?>]
</invoke>
```

O elemento *invoke* tem os atributos *service* e *operation*. O atributo *service* indica o apelido do serviço alvo e deve ser o mesmo usado nas definições (*service* em *definitions*). O

atributo *operation* é o nome de uma operação válida para o serviço (*métodos da classe*). O usuário deve conhecer previamente as operações disponíveis nos serviços que serão usados.

Os elementos *argument* e *return* estão associados logicamente ao atributo *operation*. O elemento *argument* deve ser usado para definir argumentos a serem passados para a operação do serviço. Para cada argumento requisitado pela operação desejada, um elemento *argument* deve ser declarado. Os seus atributos são iguais aos já apresentados no elemento *variable* descrito anteriormente. O usuário pode indicar um valor como argumento (atributo *value*), ou indicar uma variável (atributo *variable*), cujo conteúdo será passado como argumento. Ou seja, o GPO usa passagem de argumentos por valor e não por referência. O tipo do argumento (atributo *type*) deve ser coerente com as definições da operação escolhida.

O elemento *return* pode ser usado para salvar o retorno da operação executada em uma das variáveis definidas no *workflow*. Seus atributos são semelhantes aos encontrados em *argument*.

Os elementos *faultHandlers*, *catch* e *catchAll* podem ser usados para tratar falhas no *invoke*. Por exemplo, se ao tentar invocar um serviço for sinalizado que "as credenciais do usuário" estão vencidas, é possível renová-las usando a atividade *execute* com o utilitário *grid-proxy-init* (utilitário GT), ou informá-lo que essa ação deve ser feita (*Atividade echo*). A declaração local de *faultHandlers* tem prioridade sobre a declaração geral feita em *process*.

4.3.9 *assign*

A GPOL permite ao usuário criar variáveis de *workflow*. Essas variáveis servem para manter o estado do *workflow*, armazenando valores e resultados intermediários que são usados em cálculos ou como argumentos em serviços e tarefas. As variáveis de *workflow* são definidas pelo usuário através dos elementos *variables* e *variable* já mostrado na Seção 4.3.5.

Para atribuir valores a essas variáveis é usado o elemento *assign*. A sua sintaxe de uso é:

```
<assign>
   name="xsd:Name"
   operator="[ = | += | -= | *= | /= ]"
   [value | variable]="xsd:string"
</assign>
```

O elemento *assign* usa quatro atributos para fazer cada atribuição: *name*, *operator* e *value* ou *variable*. O atributo *name* indica o nome da variável do *workflow* que recebe o

valor resultante da operação.

O terceiro atributo pode indicar um valor (*value*) a ser usado na operação, ou uma variável (*variable*) cujo conteúdo será usado na operação.

O atributo *operator* indica as possibilidades de atribuição. O símbolo **"="** indica que a variável (*name*) deve receber o valor indicado pelo terceiro atributo (*value* ou *variable*). Se em *operator*, o sinal de atribuição "=" for precedido por um operador (+, -, * ou /), essa operação será executa usando o valor da variável indicada em *name* e o valor definido pelo terceiro atributo. O valor resultante dessa operação será atribuído à variável indicada por *name*. A coerência de tipos envolvendo as variáveis e valores é responsabilidade do usuário.

4.3.10 *echo*

O elemento *echo* apresenta uma mensagem na saída padrão de vídeo (*stdout*). Sua sintaxe é:

<echo
*message="xsd:string"**
file="xsd:string"/>*

O elemento *echo* tem dois atributos, *message* e *file*. O atributo *message* indica uma *string* cujo conteúdo será apresentado na saída padrão.

O atributo *file* indica o arquivo cujo conteúdo é apresentado na saída padrão (*stdout*). O conteúdo do arquivo deve ser exclusivamente texto.

Se o processo for remoto, isto é, a instância do GPO Maestro criada para executar o *workflow* não estiver no *host* local, as mensagens são automaticamente redirecionadas para arquivo pela infra-estrutura da Grade. Nesse caso o usuário poderá usar *transfer* (Seção 4.3.18) para ter acesso às mensagens apresentadas.

4.3.11 *return*

Ao final do processamento é possível usar o elemento *return* para retornar um valor ao GPO Run. Sua sintaxe é:

<return>
[variable="xsd:Name" |
value="xsd:string"]?
</return>

Pode ser indicado um valor, por exemplo, *<return value="-1" />*, ou usar o valor armazenado em uma das variáveis do *workflow*. Por exemplo,

<assign name="var" operator="=" value="0" /> ...
<return variable="var" />

retorna o valor 0 para o GPO Run.

Em ambos os casos, somente um valor do tipo inteiro (*xsd:integer*) pode ser retornado.

4.3.12 *exit*

O elemento *exit* encerra imediatamente a execução do *workflow*. A sua sintaxe é:

<exit>
[*variable="xsd:Name"* |
value="xsd:string"]?
</exit>

O elemento *exit* permite ao usuário indicar o código de encerramento usando um dos atributos *variable* ou *value*. Todas as instâncias dos serviços que foram criadas são destruídas antes do encerramento.

4.3.13 *clear*

O elemento *clear* é usado para *destruir* instâncias de serviços ou limpar valores das variáveis do processo. A sua sintaxe é:

<clear name="xsd:Name" />

O atributo *name* pode indicar o nome de uma variável ou o *apelido* de um serviço. Não aceita repetições do atributo *name*, mas não tem limite no número de utilizações.

4.3.14 *if*

O elemento *if* provê desvios condicionais na execução do *workflow*. A sua sintaxe é:

<if condition=Expressão>
*Atividade**
<elseif condition=Expressão>

```
    Atividade*
  </elseif>*
  <else>
    Atividade*
  </else>?
</if>
```

Se a *Expressão* indicada pelo atributo *condition* for verdadeira, as *atividades* relacionadas ao *if* serão executadas. Se a condição (*condition*) associada ao elemento *if* for falsa, o próximo elemento *elseif* será avaliado e assim sucessivamente. Se nenhuma condição verdadeira for encontrada, e houver o elemento *else*, suas atividades serão executadas incondicionalmente. Todas as *atividades* oferecidas pela GPOL podem ser usadas no elemento *if*, inclusive o próprio *if*.

4.3.15 *while*

O elemento *while* é responsável pelos ciclos dentro dos processos GPOL. A sua sintaxe é:

```
<while condition=Expressão>
  Atividade+
</while>
```

Enquanto a *Expressão* indicada pelo atributo *condition* for verdadeira, as atividades relacionadas ao *while* serão executadas. Se a condição for falsa o processo continuará normalmente.

4.3.16 *sequence*

O elemento *sequence* permite ao usuário definir a ordem na qual devem ser executadas as *atividades* (tarefas, serviços, etc) no *workflow*. Na execução do *workflow* GPOL as atividade são processadas sequencialmente conforme a sua ordem no arquivo. Existem situações onde o usuário poderá formar um grupo de atividades, e informar que as atividades desse grupo devem ser executadas sequencialmente. Isso permite ao usuário executar grupos de atividade em paralelo, usando um *flow* com vários elementos *sequence*. A sintaxe para *sequence* é:

```
<sequence>
  Atividade*
```

</sequence>

As *atividades* relacionadas no elemento *sequence* serão executadas na ordem em que são declaradas. Uma atividade só é processada após o término completo da atividade que a antecede. Qualquer uma das *atividades* GPOL pode fazer parte de uma seqüência (*sequence*), inclusive outra seqüência.

4.3.17 *flow*

De forma complementar à seqüência, a GPOL permite a execução de atividades em paralelo. Isso é feito através do elemento *flow*. Esse elemento permite ao usuário usar os recursos computacionais da Grade simultaneamente, disparando a execução de tarefas ou executando serviços em cada um deles. A sua sintaxe é:

<flow>
*Atividade**
</flow>

Todas as *atividades* relacionadas em *flow* são executadas em paralelo (simultaneamente). Para cada atividade é criada uma *thread* responsável pelo seu processamento. Dessa forma o GPO garante a execução independente mesmo quando for invocado um serviço da Grade, pois o elemento *invoke* tem processamento síncrono quando usado fora de um *flow*. Após disparar a execução de todas as atividades relacionadas, o *flow* entra em estado de espera aguardando o término de todas as atividades, para só então, permitir a continuidade do processo. Ou seja, cada atividade é executada de forma assíncrona e o final do *flow* é usado como ponto de sincronização do processo. Qualquer atividade pode fazer parte de um *flow*, inclusive o próprio *flow*.

4.3.18 *transfer*

Aplicações que usam grandes volumes de dados são comuns nas Grades. Antes de iniciar um processamento em um recurso remoto, é necessário garantir o acesso aos dados usados nesse processamento. Uma das formas de disponibilizar os dados é transferir o(s) arquivo(s) que os contém para o *host* onde será feito o processamento. Uma das maneiras de transferir arquivos em GPOL é através do elemento *transfer*. A sua sintaxe é:

<transfer>
srcHost="xsd:anyURI"
srcFile="xsd:string"

```
    srcProvider="tns:ProviderType"
    dstHost="xsd:anyURI"
    dstFile="xsd:string"
    dstProvider="tns:ProviderType"
</transfer>
```

O elemento *transfer* usa dois conjuntos de atributos. Um conjunto formado pelos atributos *srcHost*, *srcFile* e *srcProvider*, que estão relacionados ao arquivo origem. Um segundo conjunto, formado pelos atributos *dstHost*, *dstFile* e *dstProvider*, indica o arquivo destino da transferência.

Transfer faz uma cópia do arquivo *srcFile*, que está no *host srcHost*, para o arquivo *dstFile* no *host dstHost*. A operação deve respeitar as indicações dos provedores envolvidos (*srcProvider* e *dstProvider*).

4.3.19 *execute*

Uma parte significativa do processamento feito nas Grades envolve aplicações BoT (saco de tarefas), que são caracterizadas pela independência das tarefas relacionadas na aplicação. A linguagem GPOL oferece o elemento *execute* para a execução de tarefas. A sua sintaxe é:

```
<execute>
    executable="xsd:string"
    host=["xsd:anyURI" | "local"]?
    provider="tns:ProviderType"?
    <argument>
        [variable="xsd:Name" |
        value="xsd:string"]
    </argument>*
    <return>
        [variable="xsd:Name" |
        value="xsd:string"]
    </return>?
</execute>
```

Execute possui os atributos *executable*, *host* e *provider*, e permite o uso dos elementos *argument* e *return*. O atributo *executable* indica o programa a ser executado em *host* e *provider* informa o provedor. A expressão "*local*" no atributo *host* indica que a execução deve ser no recurso computacional local.

O elemento *argument* está relacionado ao executável indicado. *Argument* deve conter uma *string* com os argumentos (linha de comando) para o programa alvo. O elemento *return* pode ser usado para obter o código de encerramento do programa executado (sempre um valor *inteiro*).

O exemplo abaixo:

<execute executable="bin/sleep" arguments="10" host="local"/>

executa o utilitário *sleep* por 10 segundos no *host* local.

Outro exemplo é uma submissão WS-GRAM usando o utilitário *globusrun-ws*, como mostrado abaixo:

<execute executable="globusrun-ws" arguments="-submit -f echo-job.xml"/>

onde o *globusrun-ws* executa os comandos contidos no arquivo *echo-job.xml* (mostrado na Figura 4.2). O código aciona o utilitário *echo*, mostrando a frase "*Tarefa WS-GRAM com Job Description*", e redirecionando as saídas de vídeo (*stdout*) e erro (*stderr*) para arquivos.

Uma falha será gerada se o executável não existir no *host* remoto. O provedor padrão é GT4.

```
<?xml version="1.0" encoding="UFT-8"?>
<job>
 <executable /bin/echo /executable />
 <argument "Tarefa WS GRAM com Job Description" />
 <sdtout ${GLOBUS-USER-HOME}/stdout />
 <stderr ${GLOBUS-USER-HOME}/stderr />
</job>
```

Figura 4.2: Submissão de tarefa definida com *WS-GRAM Job Description*

4.3.20 *javacode*

O elemento *javacode* contém código fonte *Java*. Esse recurso pode ser usado para pequenos cálculos, extração de *strings* ou acessar fragmentos de arquivos, por exemplo. A sua sintaxe é:

<javacode file="xsd:string" />

Código Java
</javacode>

O atributo *file* indica um arquivo contendo um programa em código *Java* que deve ser executado pelo GPO. O usuário poderá usar variáveis definidas na parte GPOL do *workflow*, para flexibilizar a execução do código *Java*, mas somente seu valor será usado. Isto é, as variáveis GPOL são tratadas como parâmetros de entrada (passagem de parâmetros por valor). Dessa forma, o GPO gera o código *Java* equivalente e usa a máquina virtual para executar esse código resultante.

4.4 *Um exemplo de workflow GPOL*

Para ilustrar o uso da GPOL será mostrado um *workflow* exemplo, onde é usado o serviço *MathFactoryService*. Esse serviço faz cálculos matemáticos, através das operações *add*, *sub*, *mul* e *div*, mostradas no lado esquerda da Figura 4.3. O serviço é usado para calcular a expressão numérica simples *((a+b)*(c+d))/(e-f)*, cuja seqüência de operações é representada pelo grafo orientado mostrado no lado direito da Figura 4.3.

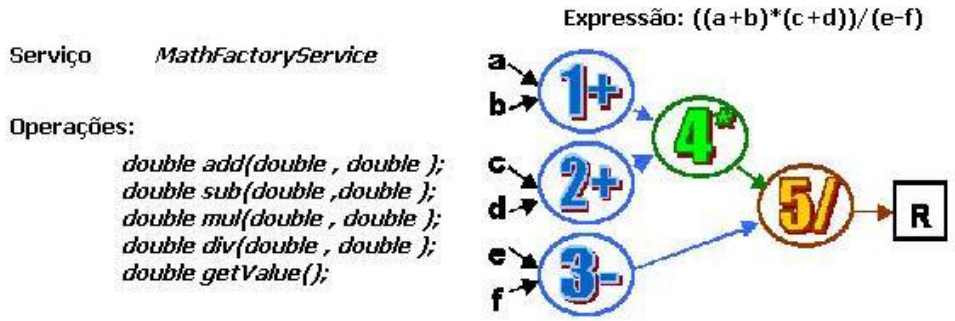

Figura 4.3: Operações do Serviço *MathFactoryService* e DAG da Expressão

No grafo vemos que as operações 1, 2 e 3 são independentes e podem ser executadas em paralelo, e as operações 4 e 5 são seqüenciais. Se o usuário fosse usar os recursos do WS-GRAM teria que executar uma operação de cada vez, ou criar clientes (em *Java* por exemplo) para executar as operações. Mas para cada nova combinação de operações um novo cliente seria necessário.

Usando o GPO, o usuário invoca diretamente o serviço matemático, indicando as operações que devem ser feitas em paralelo e quais serão sequenciais, diretamente no *workflow*, sem necessitar de suporte adicional.

O *workflow* é detalhado em duas partes. A primeira parte, onde estão as suas

definições, é mostrado na Figura 4.4. A segunda parte, onde está o processo é apresentado na Figura 4.5.

```
<definitions name="Math-Defs">
 <variables>
   <variable name="a" type="double" value="1"/>
   <variable name="b" type="double" value="2"/>
   <variable name="c" type="double" value="3"/>
   <variable name="d" type="double" value="4"/>
   <variable name="e" type="double" value="5"/>
   <variable name="f" type="double"  value="6"/>
   <variable name="R1" type="double"/>
   <variable name="R2" type="double"/>
   <variable name="R3" type="double"/>
   <variable name="TOT" type="double"/>
 </variables>

 <services>
   <service name="Inst1">
    uri=hostname:8080/.../MathFactoryService"
    type="Factory"
    ret-type="double"
   </ service >
   <service name="Inst2">...</service>
   <service name="Inst3">...</service>
 </services>
</definitions>
```

Figura 4.4: Definições do *workflow* para a expressão matemática

Dentro do elemento *variables* são definidas as variáveis *a, b, c, d, e,* e *f*, que são iniciadas com os valores a serem usados no cálculo. São definidas também quatro variáveis (*R1, R2, R3* e *TOT*) que guardam os valores intermediários gerados. No elemento *services* são definidos três serviços (*Inst1, Inst2 e Inst3*), que são usadas no processo. Esses serviços são instâncias de *MathFactoryService*, pois o serviço matemático é uma fábrica e permite esse tipo de uso. No *workflow* exemplo as instâncias de *MathFactoryService* executam as operações 1, 2 e 3 em paralelo conforme o grafo da Figura 4.3. Cada instância criada recebe um nome único (*Inst1*, *Inst2* e *Inst3*), tem seu tipo indicado (*Factory*) e o tipo de retorno esperado.

No processo mostrado na Figura 4.5, as instâncias *Inst1*, *Inst2* e *Inst3* são executadas em paralelo dentro de um *flow*. *Inst1* calcula a operação *(a + b)*, *Inst2* calcula *(c + d)* e *Inst3* calcula *(e - f)*. As variáveis *R1*, *R2* e *R3* guardam esses cálculos intermediários. A instância *Inst1* é usada nos cálculos finais, envolvendo *R1*, *R2* e *R3*, colocando o resultado final em *TOT*.

```
<process name="Math-Process">
 <flow>
   <!-- (a + b) -->
   <invoke>
    <service name=Inst1>
      <operation name="add"/>
      <argument>
       <variable="a" type="double"/>
       <variable="b" type="double"/>
      </argument>
      <return variable=R1/>
    </service>
   </invoke>
   <!-- (c + d) -->
   <invoke>
    <service name=Inst2>
      <operation name="add"/>
      <argument>
       <variable="c" type="double"/>
       <variable="d" type="double"/>
      </argument>
      <return variable=R2/>
    </service>
   </invoke>
   <!-- (e - f) -->
   <invoke>
    <service name=Inst3>
      <operation name="subtract"/>
      <argument>
       <variable="e" type="double"/>
       <variable="f" type="double"/>
      </argument>
      <return variable=R3/>
    </service>
   </invoke>
 </flow>
```

```
<sequence>

  <!-- (a + b) * (c + d) -->
   <invoke>
    <service name=Inst1>
      <operation name="mul"/>
      <argument>
       < variable=R1/>
       < variable=R2/>
      </argument>
      <return variable=TOT/>
    </service>
   </invoke>
   <!-- ((a + b) * (c + d)) / (e - f) -->
   <invoke>
    <service name=Inst1>
      <operation name="div"/>
      <argument>
       < variable=TOT/>
       < variable=R3/>
      </argument>
      <return variable=TOT/>
    </service>
   </invoke>
  </sequence>

  <!-- Display results (test only) -->
  <execute>
    executable="/bin/echo"
    <argument value="End MathExpress"/>
  </execute>

  <!-- Return value to GPO_Run -->
   <return variable="TOT"/>

</process>
```

Figura 4.5: O processo para executar a expressão matemática

O elemento *execute* é usado para avisar ao usuário do término do cálculo e o elemento *return* devolve o valor obtido para o GPO Run, encerrando a execução do *workflow*.

O elemento *sequence* usado após o *flow* não é necessário, pois a execução das atividades do *workflow* GPOL é naturalmente sequencial. A inclusão de *sequence* foi por motivos didáticos.

4.5 *Esquemas XML GPOL*

A seguir estão os esquemas XML para alguns elementos GPOL, cuja descrição é importante para o completo entendimento de alguns elementos da linguagem.

- *targetNamespace* e *Namespace*

  A seguir estão as definições para *targetNamespace* usado pelo GPOL, e outros esquemas que são usados nas definições da linguagem.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.ic.unicamp.br/gpo/language"
    xmlns:tns="http://www.ic.unicamp.br/gpo/language"
    xmlns:gjd="http://www.globus.org/namespaces/2004/10/gram/job/description/"
    xmlns:gjt="http://www.globus.org/namespaces/2004/10/gram/job/types/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    ...
</schema>
```

- *VariableType*

  Esse esquema mostra as opções para tipos de variáveis da GPOL.

```
<!– Types declaration –>
<simpleType name="VariableType">
    <restriction base="string">
        <enumeration value="string"/>
        <enumeration value="boolean"/>
        <enumeration value="decimal"/>
        <enumeration value="float"/>
        <enumeration value="double"/>
        <enumeration value="integer"/>
    </restriction>
</simpleType>
```

- *ProviderType*

  Esse esquema mostra os tipos de provedores aceitos pela GPOL.

```
<!– Types declaration –>
<simpleType name="ProviderType">
    <restriction base="string">
        <enumeration value="GT4"/>
        <enumeration value="GT3"/>
    </restriction>
</simpleType>
```

Capítulo 5

Middleware Proposto: GPO

O GPO (*Grid Process Orchestration*) é um *middleware* para interoperabilidade de aplicações distribuídas que requerem composição de serviços em uma Grade Computacional [77]. O GPO permite a criação e gerência de fluxos de aplicações, tarefas e principalmente serviços das grades computacionais. No projeto do GPO foram consideradas aplicações fortemente acopladas que necessitam interagir na Grade, e que podem ser facilmente controladas pelo usuário. A arquitetura GPO é baseada nos conceitos apresentados em OGSA, e usa a linguagem GPOL (Capítulo 4) na descrição dos *workflows* a serem executados.

5.1 Arquitetura GPO

A arquitetura GPO consiste de três componentes principais, *GPO Run*, *GPO Maestro Factory Service* e *GPO Maestro Service Instance*, além de arquivos especificados com GPOL. A Figura 5.1 mostra esses componentes.

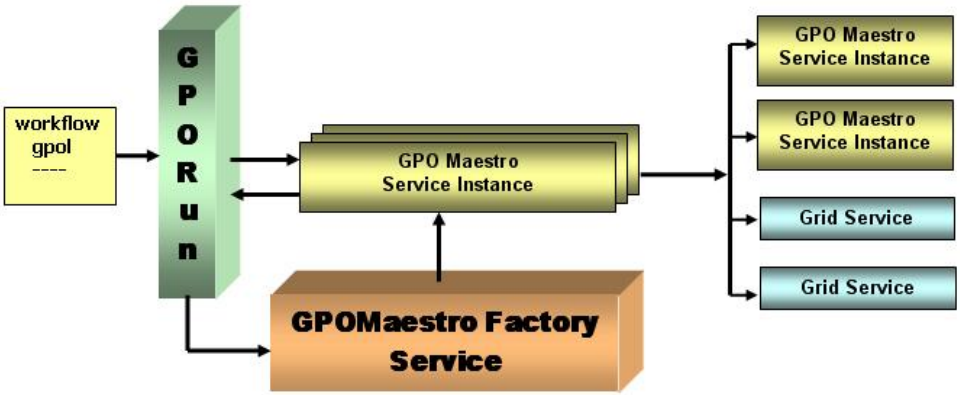

Figura 5.1: A arquitetura GPO e seus componentes

O *GPO Run* é um utilitário independente que recebe arquivos escritos em GPOL, verifica sua correção sintática, e inicia a execução do *workflow* recebido. Para executar o *workflow*, o *GPO Run* usa a "*fábrica de serviços*" *GPO Maestro Factory Service* para criar uma nova instância do *GPO Maestro* que fica responsável pela execução do *workflow*. A instância do *GPO Maestro* processa o *workflow* retornando um código indicando o resultado da execução. O valor zero indica que o *workflow* foi executado corretamente e qualquer outro valor indica o erro ocorrido. O *GPO Run* destrói a instância do *GPO Maestro*, entregando o resultado ao usuário e finalizando o processo.

Essa arquitetura é similar a do WS-GRAM mantendo todas as suas facilidades, porém, incorporando o conceito de *workflow* e o uso direto dos serviços da grade.

A seguir são apresentados detalhes sobre os componentes *GPO Run* e do*GPO Maestro*.

5.2 *GPO Run*

O *GPO Run* é o utilitário para o usuário submeter *workflows* ao GPO. Sua operação é similar a usada pelo WS-GRAM, o que facilita o aprendizado para os novos usuários.

Para ilustrar o uso do *GPO Run*, a Figura 5.2 mostra a execução do *workflow Math-Express.gpol* (Seção 4.4), através da fábrica *GPOMFactoryService*.

```
$java GPORun \
-submit
-factory http://143.106.24.147:8080/wsrf/service/GPOMFactoryService \
-file MathExpress.gpol

GPORun Vr. 1.01
  Endpoint reference: GPORun1015532839068.gpo
  GPO Instance for: submit MathExpress.gpol
  GPO submit retCode: 21
GPO End
```

Figura 5.2: Exemplo de execução com *GPO Run*

Ao iniciar a execução o *GPO Run* se identifica (*GPORun Vr. 1.01*), mostra o código único da instância do *GPO Maestro* criada e o comando submetido (*submit*). Esse código único gerado para cada instância criada indica internamente ao GPO o *EPR* da instância, sendo útil em outros comandos. Ao final do processamento mostra o valor de retorno (*21*) e avisa o encerramento do processo (*GPO End*).

O GPO Run oferece as seguintes opões ao usuário:

- ***submit***: executa o *workflow* cujo arquivo é indicado pela opção "*−file*". Usa a fábrica indicada na opção "*−factory*" para criar a instância do *GPO Maestro*

responsável pela execução.

- ***stat***: mostra o estado da execução que está sendo feita pela instância indicada por "$-epr$". A opção "$-epr$" deve ser seguida pelo código único apresentado no início do processamento, conforme mostrado na Figura 5.2 em "*Endpoint reference: GPORun1015532839068.gpo*".

- ***kill***: destrói a execução que está em andamento na instância "$-epr$" indicada.

5.3 GPO Maestro

O *GPO Maestro* foi construído usando o conceito de fábrica/instância oferecido pelas grades computacionais. Ele é formado por um conjunto de serviços, cujos principais são, o *GPO Maestro Factory Service* e pelo *GPO Maestro Service Instance*. O *GPO Run* usa o serviço *GPO Maestro Factory Service* para criar instâncias do *GPO Maestro Service Instance* que ficam responsáveis pela execução do *workflow* que lhes é passado como argumento. Ao ser criada, a instância passa a interagir diretamente com o *GPO Run*, permitindo que outros programas possam usar as facilidades da fábrica *GPO Maestro*.

O desenho arquitetônico do *GPO Maestro* permite reentrância nos *workflows*. Uma vez que o *GPO Maestro* é um serviço da grade, ele pode ser usado dentro de um *workflow* como qualquer outro serviço. Isso dá grande versatilidade no uso do GPO permitindo ao usuário distribuir tarefas de acordo com os recursos disponíveis.

A seguir são apresentados detalhes sobre os serviços que compõem o GPO Maestro.

5.3.1 GPO Maestro Factory Service

Na construção do *GPO Maestro* foi usado o padrão de fábrica da especificação *WS-Resource* implementado no GT4. Esse padrão separa serviço e recurso, onde recurso faz a persistência de estado (valores e variáveis), não estando associado ao conceito tradicional de recurso da Grade (recurso computacional, por exemplo). O padrão propõe um tratamento de fábrica/instância também para recursos, com operação similar ao conceito fábrica/instância já usado pelos serviços da Grade.

Seguindo esse padrão o GPO Maestro tem mais dois componentes além dos *GPO Maestro Factory Service* e do *GPO Maestro Service Instance*. Os dois componentes são o *GPO Maestro Resource Home* e o *GPO Maestro Resource*. Eles são de uso interno não oferendo interfaces para outros clientes. A Figura 5.3 mostra o relacionamento entre os quatro componentes.

Ao receber a requisição para criar uma nova instância do serviço, o *GPO Maestro Factory Service* aciona a *Resource Home* (fábrica de recursos), que cria uma nova instância

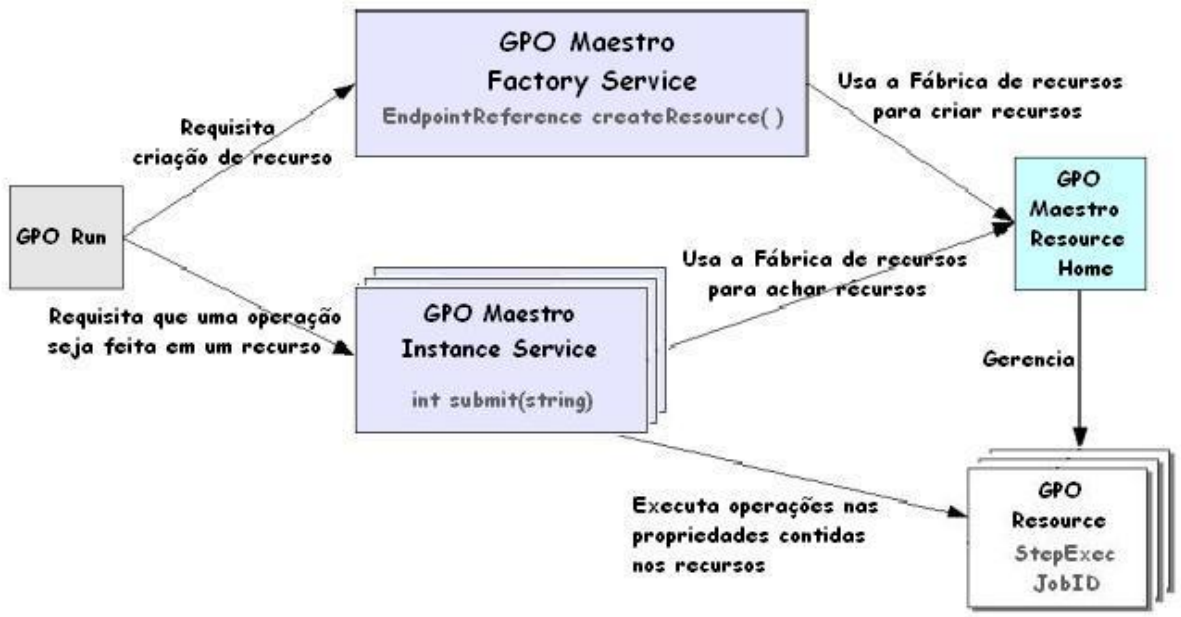

Figura 5.3: Relacionamento entre os componentes do GPO Maestro

dos recursos. *Resource Home* coloca a nova instância em sua lista de recursos e devolve uma chave para acesso a essa instância. O *GPO Maestro Factory* usa essa chave em conjunto com a referência à instância do serviço (*Instance Service*) por ele criada, gerando o *EPR* de acesso. Finalmente esse *EPR* gerado é retornado ao GPO Run que o usará para acessar a instância do serviço responsável pela execução do *workflow*.

Dessa forma, cada instância do serviço tem um recurso a ela associado. Nesse recurso são mantidas as variáveis que acompanham a execução passo a passo do *workflow*. Essas variáveis são denominadas "*variáveis do processo*", e são:

- RP_VALUE: contém o último valor retornado ao GPO Run;
- RP_LASTOP: contém a descrição da última operação executada;
- RP_EPRFILE: é o EPR da instância criada;
- RP_WRFFILE: é o nome da arquivo GPOL em execução;
- RP_WRFJOBID: é a identificação única criada pelo GPO para o *workflow*;
- RP_STATUS: contém o estado da execução (uso interno);
- RP_STEPEXEC: indica a atividade em execução;

- RP_FAULTVALUE: código da falha ocorrida; e
- RP_FAULTDESC: descrição da falha ocorrida.

É através dessas variáveis que o GPO possibilita o monitoramento na execução do *workflow*. Ao usar a opção "*−stat*" no *GPO Run*, o usuário está acessando indiretamente essas variáveis do processo.

5.3.2 GPO Maestro Service Instance

Para cada *workflow* submetido, uma instância do *GPO Maestro Service Instance* é criada. A instância criada é que realmente executa o *workflow*.

O *workflow* é executado de forma sequencial. A execução começa preenchendo as classes internas com as variáveis e os serviços declarados em *definitions*. Para cada variável declarada, um novo objeto da classe *GPOJobVariables* é criado. Esses objetos são iniciados com o nome, o tipo e valor inicial para a variável. De forma semelhante, para cada serviço declarado, um novo objeto da classe *GPOJobServices* é criado. Esses objetos são iniciados por informações como o apelido dado ao serviço, sua URI, seu tipo, tipo de retorno, e também são preparadas informações de controle do GPO, como por exemplo, o objeto *siPortType*, uma referência para a instância do serviço, ou o *flag ServiceOn*, que indica se a instância está ativa ou não.

Os objetos *GPOJobVariables* e *GPOJobServices* são atualizados durante a execução do *workflow*. Por exemplo, ao executar a atividade *assign*, o objeto corresponde à variável envolvida é atualizado com o valor resultante da operação.

Após preparar as definições é iniciada a execução do processo. Cada atividade é executada conforme a sua ordem na sequência do processo. A exceção é a atividade *flow*. Para cada atividade participante do *flow*, é criada uma *thread* exclusiva para a sua execução. Essa técnica garante a execução em paralelo das atividades sem bloquear a execução do *workflow*. Ao reconhecer o final do *flow* (*</flow>*), a execução entra em estado de espera acompanhando as *threads* disparadas até que a última esteja encerrada.

Ao final da execução do *workflow*, a instância do GPO Maestro destrói todas as classes de controle criadas, destrói as instâncias dos serviços, e retorna o valor indicado em *<return>*, ou o código de final de execução, ao GPO Run.

Se uma falha ocorrer durante a execução do *workflow*, a instância preenche as variáveis de controle *RP_FAULTVALUE* e *RP_FAULTDESC*, com o código e com a descrição da falha detectada. Essas informações ficam disponíveis ao GPO Run, mesmo depois de encerrada a execução do *workflow*. O GPO Run é o responsável por requisitar a criação e a destruição da instância do GPO Maestro executora do *workflow*. Como essas informações estão no recurso associado à instância, elas são mantidas até que a instância seja destruída pelo GPO Run.

5.4 A integração do GPO com o ambiente da Grade

O GPO pode ser usado de forma totalmente distribuída permitindo ao usuário acioná-lo de qualquer ponto da grade, indicando onde devem ser criadas instâncias (*-factory*) de forma a extrair o máximo dos recursos disponíveis. Isso é possível devido ao GPO Maestro ser um serviço e usar justamente o conceito de transiência oferecido pela OGSA.

5.4.1 O GPO integrado ao sistema de segurança da Grade

Segurança é um dos aspectos mais importantes nas grades, devido à sua natureza abrangente que pode envolver recursos em várias organizações. É fundamental garantir que os participantes da grade, de usuários a aplicações, estejam apropriadamente *autorizados* a fazer uso dos recursos. Nesse ambiente multi-institucional alguns termos relacionados com a segurança de sistemas e serviços requerem maior atenção. Esses termos são:

- Autenticação: é o processo de verificação da validade de um usuário, seja um indivíduo, um serviço ou uma aplicação.
- Controle de acesso: visa garantir que cada usuário ou recurso que use um serviço tenha a permissão necessária para fazê-lo.
- Integridade de dados: assegura que os dados não são alterados ou destruídos de uma maneira não autorizada.
- Confidencialidade de dados: informações especiais não podem ser reveladas sem a devida autorização ao acesso.
- Gerência de chaves: chaves são usadas para codificar informações. A gerência apropriada dessas chaves de acesso é parte fundamental da segurança.

No *Globus Toolkit* os módulos GSI (*Grid Security Infrastructure*) e PKI (*Public Key Infrastructure*) formam um *framework* que inclui protocolos, serviços e padrões, que suporta autenticação dos usuários (certificados), confidencialidade e integridade dos dados e gerência de chaves de codificação.

Os módulos do GPO estão integrados aos aspectos de segurança providos GSI/PKI do GT. Para ter acesso ao GPORun o usuário deve ter um certificado que permita a sua autenticação na Grade. Para a execução de um *workflow*, o usuário deve ter as devidas autorizações de acesso para todos os serviços e tarefas envolvidas no *workflow*, incluindo as autorizações referentes ao GPO Maestro, que para os aspectos de segurança é um serviço como qualquer outro da Grade.

Toda a segurança oferecida pelo GSI pode ser usada em conjunto com o GPO. Dessa forma são mantidos todos os aspectos relativos à segurança na operação da grade, na localização e no acesso aos recursos, nos diversos domínios suportados pelo *GT*.

5.4.2 A integração do GPO com os escalonadores da Grade

A integração entre o GPO e o *GT* permite que a infra-estrutura da grade mantenha inalterado o seu relacionamento com outros produtos, como escalonadores, gerenciadores de serviços e outros, garantindo suas funcionalidades individuais e conjuntas.

Um exemplo dessa integração é a execução de tarefas (elemento *execute*) de um *workflow* GPOL. Para fazer a execução de tarefas, o GPO cria instâncias dos serviços *ManagedJobFactory* e *ManagedJob* (WS-GRAM Seção 3.1.3), que são responsáveis pela execução. Essa integração permite naturalmente o uso dos escalonadores disponíveis na Grade. A execução de tarefas do *workflow* GPOL tem funcionamento similar a provida pelos utilitários do GT, mantendo a total integração com os escalonadores disponíveis no ambiente.

5.5 Aspectos Sobre a Implementação Feita

Um protótipo do *middleware* proposto foi implementado em *Java*. A escolha por *Java* é devida à sua integração com os recursos para o desenvolvimento oferecidos pelo *Java WS Core do Globus Toolkit 4.1* (Seção 3.1.3).

Para implementar a análise sintática dos arquivos GPOL é usado a API DOM (*Document Object Model*) que acompanha o *JDK* 1.4 da *Sun*. O analisador sintático DOM constrói uma estrutura de árvore hierárquica com os elementos do *workflow* GPOL na memória.

Cada elemento GPOL representa um nó na árvore, e pode conter outros nós (chamados de nós filhos). Cada nó é um objeto que tem propriedades, métodos e eventos. As propriedades associadas a um nó incluem nomes, valores, nós filhos, etc. Os métodos permitem criar, excluir, acrescentar nós, carregar documentos XML, etc. Por exemplo, o trecho de código a seguir cria um objeto *Document* chamado *doc*, já contendo a árvore hierárquica gerada com o *workflow* GPOL indicado pelo arquivo "*MathExpress.gpol*".

```
Document doc = builder.parse("MathExpress.gpol");
Element root = doc.getDocumentElement();
```

Usando o método *getDocumentElement* é criado o elemento *root* que passa a ser a referência para o elemento raiz do *workflow* "*MathExpress.gpol*" (o elemento *workflow*).

Para obter o nome dado ao *workflow* basta usar o método *root.getAttribute("name")*.

Na implementação dos serviços do GPO Maestro (*Factory Service, Service Instance, ResourceHome e Resource*) foram usadas as facilidades do *Java WS Core* do *GT4*. O *GT* fornece pacotes como:

org.globus.wsrf.ResourceContext;
org.globus.wsrf.ResourceKey;
org.apache.axis.message.addressing.EndpointReferenceType;
org.apache.axis.MessageContext;
org.globus.wsrf.utils.AddressingUtils; e
org.globus.wsrf.container.ServiceHost;

que oferecem os métodos *createResource*, *getResourceContext*, *getResourceHome* e *createEndpointReference*, etc.

A implementação do *GPO Maestro Service Instance*, responsável pela execução dos *workflows* GPOL, é modular. Cada elemento da linguagem foi implementado como uma classe, e todo acesso aos recursos é feito através de métodos. Por exemplo, *getEprFile*, *getWrfFile* e *getWrfStatus* são métodos para acessar algumas das variáveis de processo.

O elemento *flow* foi implementado com o apoio de *threads* para garantir a execução paralela e assíncrona do *workflow*.

Na implementação do elemento *invoke* também foram usadas as facilidades para manipulação de serviços oferecida pelo *Java WS Core*.

Já o elemento *execute* usa os serviços *ManagedJobFactory* e *ManagedJob* do WS-GRAM (Seção 3.1.3) para executar as tarefas.

Capítulo 6

Uma Aplicação da Grade: Fotoionização de Átomos

Com o objetivo de comprovar as facilidades de uso da GPOL e a versatilidade do protótipo implementado, esta seção mostra uma aplicação envolvendo um fragmento dos cálculos feitos em uma seção de choque de fotoionização relativística por luz linearmente polarizada, aplicada em átomos alcalinos (sódio, potássio, rubídio e césio), cuja teoria está formalizada em [70].

$$\frac{d\sigma}{d\Omega} = \sum_{L} (-)^{j(k)+1/2} \sum_{k'k''} \overset{\boxed{A}}{\boxed{\frac{[j(k'), j(k''), L]}{4\pi}}} \overset{\boxed{B}}{\boxed{\begin{pmatrix} 1 & 1 & L \\ -q & q & 0 \end{pmatrix}}} \overset{\boxed{C}}{\boxed{\begin{pmatrix} j(k') & j(k'') & L \\ 1/2 & -1/2 & 0 \end{pmatrix}}}$$

$$\times \overset{\boxed{D}}{\boxed{\begin{pmatrix} j(k') & 1 & j(k) \\ 1/2 & 0 & -1/2 \end{pmatrix}}} \overset{\boxed{E}}{\boxed{\begin{pmatrix} j(k'') & 1 & j(k) \\ 1/2 & 0 & -1/2 \end{pmatrix}}} \overset{\boxed{F}}{\boxed{\begin{Bmatrix} 1 & 1 & L \\ j(k'') & j(k') & j(k) \end{Bmatrix}}}$$

$$\times \boxed{P_L(\cos\theta) R_{k'} R^*_{k''} \delta_{\Pi(k),-\Pi(k')} \delta_{\Pi(k),-\Pi(k'')} \delta_{\Pi(L),\Pi(k')\Pi(k'')}} \boxed{G}$$

Figura 6.1: Fórmula da seção de choque de fotoionização de átomos relativísticos

A fórmula da seção de choque, mostrada na Figura 6.1, usa duas somatórias. A somatória interna é formada pela multiplicação dos cálculos feitos pelos termos dos retângulos nomeados de A a G. Cada iteração depende dos valores associados aos parâmetros L, k, j e q, conforme mostrado na Tabela 6.1.

Orbital	$s_{1/2}$	$p_{1/2}$	$p_{3/2}$	$d_{3/2}$	$d_{5/2}$	$f_{5/2}$	$f_{7/2}$	...
	s	$\overline{p}$	p	$\overline{d}$	d	$\overline{f}$	f	...
k	-1	$+1$	-2	$+2$	-3	$+3$	-4	...
l	0	1	1	2	2	3	3	...
a	$+1$	-1	$+1$	-1	$+1$	-1	$+1$	...
j	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{5}{2}$	$\frac{5}{2}$	$\frac{7}{2}$	...
Paridade	$+$	$-$	$-$	$+$	$+$	$-$	$-$	...

Tabela 6.1: Relação entre os números quânticos

As seções de choque são usadas de maneira ampla em estudos de Física, como por exemplo, nas áreas de *Eletrônica Quântica* e *Física da Matéria Condensada*. Muitas são as combinações dos parâmetros e dos termos envolvidos. Na aplicação mostrada nesta dissertação, o objetivo é a fotoionização relativística por luz linearmente polarizada. No processo de cálculo essa restrição é representada pelo parâmetro *q=0*. A medida da *"seção de choque total"* é traduzida no cálculo pela integração dos seus parâmetros, o que conduz a que somente *L=0* contribua para essa medida. Os valores para o parâmetro *j* dependem das possibilidades do parâmetro *k*. São três possibilidades para *k (k, k' e k")* que implicam em três possibilidades para *j (j(k), j(k') e j(k")*. As combinações de valores para esses parâmetros são encontrados na Tabela 6.1, de acordo com os parâmetros permitidos para a seção de choque.

6.1 A Modularização da Fórmula

Cada um dos termos da fórmula (de *A* a *G*) envolve uma teoria que gera uma seqüência específica de cálculos. Essas teorias são usadas em outros trabalhos e em outras áreas do estudo da Física, e portanto são de aplicação geral. Cada termo está relacionado a:

- Termo A: é o termo condicional de acordo com os valores da Tabela 6.1. Sua fórmula de cálculo é: *[j(k'), j(k"), L] = (2j(k')+1)(2j(k")+1)(2L+1)*;

- Termos B, C, D e E: esses quatro termos fazem o mesmo cálculo. A variação está na matriz de parâmetros fornecida como argumento. Em cada um dos termos são calculados os *"coeficientes 3j de Clebsch-Gordon"*;

- Termo F: esse termo é similar aos anteriores, mas usa uma outra fórmula para cálculo. Esse termo calcula o *"coeficiente 6j de Clebsch-Gordon"*; e

- Termo G: esse termo está relacionado ao *"Polinômio de Legendre"*, às *funções radiais dos átomos* e aos *"deltas de Kronecker"*.

6.2 Visão Geral da Implementação do Workflow em GPOL

Na construção da aplicação GPOL exemplo, optou-se por implementar apenas o cálculo de parte da somatória interna da fórmula. O *workflow* GPOL implementado faz a somatória das multiplicações dos termos de A a F, e o diagrama da Figura 6.2 mostra as dependências entre eles. O código completo está no Apêndice A.

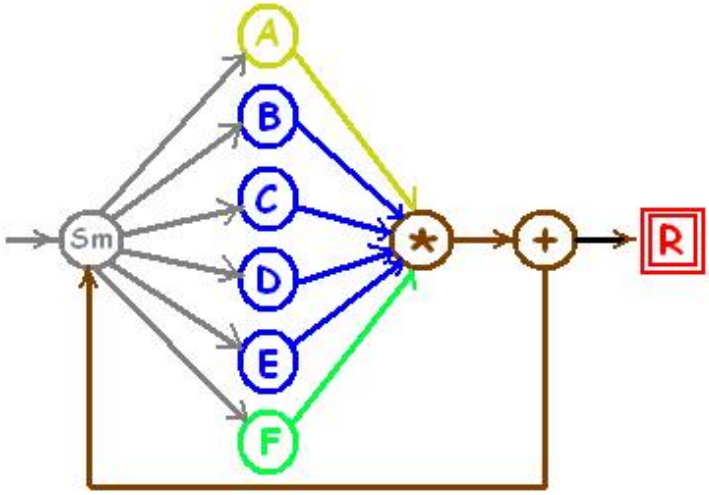

Figura 6.2: Diagrama com as dependências entre os termos da fórmula

A Figura 6.3 mostra uma visão geral do *workflow Fotoion*. O *workflow* começa definindo as variáveis e os serviços que serão usados (*FotoionDefs*) e seu processo (*FotoionProcess*) é um ciclo de leitura de tuplas, cujas informações são usadas nos cálculos dos termos de A a F. Cada linha do arquivo de tuplas contém valores para os parâmetros da fórmula, conforme a Tabela 6.1, e estão na seguinte ordem: *L, q, k, k', k", j(k), j(k') e j(k")*.

No *workflow* são usados quatro Serviços da Grade: *accessFile*, *strFunctions*, *CG3j* e *CG6j*. Esses serviços são descritos a seguir.

- ***accessFile***

  A definição do serviço feita em *FotoionDefs* é:

  <service name="accessFile"

```
- <workflow name="Fotoion">
  - <definitions name="FotoionDefs">
    + <variables></variables>
    + <services></services>
    </definitions>
  - <process name="FotoionProcess">
      <!-- Abre o arquivo com as tuplas -->
    + <invoke service="accessFile" operation="openFile"></invoke>
      <!-- Le a primeira tupla -->
    + <invoke service="accessFile" operation="readLine"></invoke>
      <!-- Verifica a leitura feita -->
    + <invoke service="strFunctions" operation="isNull"></invoke>
    - <!--
        Enquanto ler tuplas faz o calculo
              acumulando valor resultante
      -->
    + <while condition="iEoF eg 0"></while>
      <!-- Return value to GPO_Client -->
      <return variable="dSomatoria"/>
    </process>
  </workflow>
```

Figura 6.3: Visão geral do *workflow Fotoion*

uri="http://143.106.24.147:8080/wsrf/services/.../FactAFService"

fal="org.globus...FactAFServiceAddressingLocator"

sal="org.globus...AFServiceAddressingLocator"

type="Factory"/>

Todo o acesso ao arquivo de tuplas é feito através do *Serviço accessFile*. Esse serviço oferece as operações:

- *openFile*: abre o arquivo indicado pelo argumento *string FileName* e retorna um valor inteiro. Retorna o valor zero para arquivo aberto e pronto para ser acessado. Qualquer outro valor indica o código do erro ocorrido;
- *readLine*: retorna uma *string* com a próxima linha do arquivo. Tem um argumento inteiro que indica o deslocamento do ponteiro de leitura. Se o argumento for 1, lê e desloca para a próxima linha. Se o argumento for 0, lê e mantém a posição. Retorna uma *string* vazia quando chega ao final do arquivo; e
- *closeFile*: fecha o arquivo indicado pelo argumento *string FileName*. Retorna o valor quando o arquivo foi fechado corretamente. Qualquer outro valor é o código de erro ocorrido na operação.

- ***strFunctions***

  A definição do serviço feita em *FotoionDefs* é:

  <service name="strFunctions"

  uri="http://143.106.24.147:8080/wsrf/services/.../FactSFService"

  fal="org.globus...FactSFServiceAddressingLocator"

  sal="org.globus...SFServiceAddressingLocator"

  type="Factory"/>

  O *Serviço strFunctions* oferece um conjunto de operações para tratamento de *strings*. As operações usadas no *workflow Fotoion* são:

  - *isNull*: verifica se a *string* recebida como argumento está vazia (nula). Retorna 1 se a *string* é vazia e 0 se a *string* tem algum símbolo.
  - *getElement*: recebe dois argumentos, uma *string* e um *inteiro*. O *inteiro* indica qual elemento deve ser retornado como *substring*. A operação entende como elemento qualquer seqüência de símbolos separados por espaços. Por exemplo, na *string* "um dois três quatro", a palavra "dois" é o segundo elemento. Retorna uma *string* vazia (nula) se não encontrar o elemento desejado.
  - *strToDouble*: Retorna um valor *double* equivalente aos símbolos encontrados na *string* recebida como argumento.

- ***CG3j***

  A definição do serviço feita em *FotoionDefs* é:

  <service name="CG3j"

  uri="http://143.106.24.147:8080/wsrf/services/.../FCG3jService"

  fal="org.globus...FCG3jServiceAddressingLocator"

  sal="org.globus...CG3jServiceAddressingLocator"

  type="Factory"/>

  O *Serviço CG3j* tem por objetivo principal calcular os *"coeficientes 3j de Clebsch-Gordon"*. Ele oferece as operações:

  - *calc*: recebe seis argumentos tipo *double* como entrada e retorna o valor (*double*) dos coeficientes calculados.
  - *getValue*: o serviço *CG3j* mantém o último valor calculado. A operação *getValue* retorna esse valor (*double*). Não tem argumentos.

– *clearValue*: essa operação limpa o valor mantido pelo serviço. Não tem retorno.

- ***CG6j***

  A definição do serviço feita em *FotoionDefs* é:

  <service name="CG6j"

  uri="http://143.106.24.147:8080/wsrf/services/.../FCG6jService"

  fal="org.globus...FCG6jServiceAddressingLocator"

  sal="org.globus...CG6jServiceAddressingLocator"

  type="Factory"/>

  As operações e funcionalidades do serviço CG6j são as mesmas do serviço CG3j.

6.3 Detalhes sobre o Ciclo Somatório

O ciclo que implementa o somatório pode ser definido logicamente como:

```
<while condition="iEoF eg 0">
  <!- Prepara variaveis para o calculo
        Obtem os valores de j(k), j(k') e j(k")
        usando as posições 6, 7 e 8 da sTupla lida do arquivo ->
  <flow>
    <sequence>
      <!- Calculo do Termo A ->
      <!- ((2j(k') + 1)*(2j(k") + 1)*(2L + 1))/4Pi ->
    </sequence>
    <!- Calculo do Termo B ->
      <invoke service="CG3j" operation="calc" ... </invoke>
    <!- Calculo do Termo C ->
      <invoke service="CG3j" operation="calc" ... </invoke>
    <!- Calculo do Termo D ->
      <invoke service="CG3j" operation="calc" ... </invoke>
    <!- Calculo do Termo E ->
      <invoke service="CG3j" operation="calc" ... </invoke>
    <!- Calculo do Termo F ->
      <invoke service="CG6j" operation="calc" ... </invoke>
  </flow>
  <!- Faz multiplicacao dos Termos e Soma no Geral ->
```

```
        <assign name="dAux1" operator="=" variable="dResTA" />
        " ... "
</while>
```

Na seqüência lógica mostrada acima têm-se: a preparação das variáveis que compõem as matrizes que são argumentos dos termos de A a F, o cálculo individual de cada termo, a multiplicação dos cálculos e a soma acumulada. Os cálculos fazem parte de um *flow*, ou seja, são executados em paralelo. É importante destacar que o *"Termo A"* não usa um serviço da Grade. Os seus cálculos são feitos através de uma seqüência (*sequence*) de atribuições (*assign*). A seguir estão detalhados algumas partes do ciclo.

- **Obtendo valores a partir da tupla:** para copiar o valor de um elemento da tupla lida do arquivo, é usado o *Serviço strFunctions*. No código abaixo é obtido o valor de *j(k)* que é colocado na variável *sJK*. Em seguida o mesmo serviço é usado para converter a *string* obtida em um valor *double* a ser usado nos cálculos.

```
<invoke service="strFunctions" operation="getElement">
    <argument variable="sTupla" type="string"/>
    <argument value="6" type="integer"/>
    <return variable="sJK" type="string"/>
</invoke>
<invoke service="strFunctions" operation="strToDouble">
    <argument variable="sJK" type="string"/>
    <return variable="dJK" type="double"/>
</invoke>
```

- **Calculando *(2j(k') + 1)* do Termo A:** é usado *assign* por três vezes para calcular *(2j(k') + 1)* e colocar o valor resultante na variável *dAux1*.

```
<assign name="dAux1" operator="=" value="2" />
<assign name="dAux1" operator="*=" variable="dJK1l" />
<assign name="dAux1" operator="+=" value="1" />
```

- **Calculando o *Termo C*:** passando variáveis e valores como argumentos, o código abaixo usa o *Serviço CG3j* para calcular o *Termo C* e salva o valor do cálculo na variável *dResTC*.

```
<invoke service="CG3j" operation="calc">
   <!– Elementos da matriz (j(k') j(k") L 0.5 -0.5 0) –>
   <argument variable="dJK1l" type="double"/>
   <argument variable="dJK2l" type="double"/>
   <argument variable="dL" type="double"/>
   <argument value="0.5" type="double"/>
   <argument value="-0.5" type="double"/>
   <argument value="0" type="double"/>
   <return variable="dResTC" type="double"/>
</invoke>
```

6.4 Execução do *workflow* exemplo

Para iniciar a execução do *workflow*, o usuário usa o GPO Run com a sintaxe mostrada na Figura 6.4.

```
$java GPORun \
-submit \
-factory http://143.106.24.147:8080/wsrf/service/GPOMFactoryService \
-file Fotoion.gpol
```

Figura 6.4: Execução do *workflow Fotoion* usando o GPO Run

O GPO Run inicia a execução do *workflow* criando uma instância do *GPO Maestro Service Instance*, usando a fábrica *GPO Maestro Factory Service*. A instância é criada no mesmo recurso computacional onde está a fábrica. No exemplo, a fábrica está localizada no recurso identificado pelo endereço *143.106.24.147*.

A instância inicia a execução do *workflow* tratando as definições. Primeiro cria as variáveis usando objetos *GPOJobVariables*. Em seguida cria um objeto *GPOJobServices* para cada serviço identificado nas definições do *workflow*. Se o serviço for do tipo *Factory*, o GPO Maestro cria uma instância do serviço, colocando seu EPR no objeto *GPOJobServices* correspondente. Esse procedimento é executado para os serviços *accessFile*, *strFunctions*, *CG3j* e *CG6j*, que são os serviços usados no *workflow Fotoion*.

Após tratar as definições é iniciado o processo. A execução do processo começa com uma seqüência de três *invokes* (abrir o arquivo, ler uma tupla e verificar a leitura). Para invocar um serviço da Grade, o GPO usa as classes do GT sem nenhum tratamento extra.

Em seguida tem início o ciclo *while*. Ele começa com uma seqüência de *invokes* para preparar as matrizes, que são passadas como argumentos nos serviços de cálculo. Com as matrizes preparadas os cálculos são iniciados, que por serem independentes, são executados em paralelo por um *flow*.

No *workflow Fotoion*, o *flow* contém não só *invokes*, mas também um *sequence*. Nesse caso o GPO Maestro cria uma *thread* para a execução da seqüência da mesma forma como cria uma *thread* para cada *invoke*. Ou seja, qualquer atividade participante do *flow* é executada por uma *thread* exclusiva.

Ao encontrar o final do *flow* (*</flow>*), o GPO Maestro fica esperando que todas as *threads* terminem seu processamento, para então continuar a execução do *workflow*.

No exemplo *Fotoion*, a parte final do ciclo usa elementos *assign* para multiplicar os retornos e somar o valor obtido na variável *dSomatoria*.

Encerrado o ciclo, o GPO Maestro prepara o valor da variável *dSomatoria* para ser retornado ao GPO Run, encerrando o seu processamento. O GPO Run recebe o valor de retorno, destrói a instância do *GPO Maestro Service Instance* criada para a execução do *workflow*, e apresenta o resultado na saída padrão.

Capítulo 7

Conclusão

Grades computacionais são ambientes para computação distribuída de alto desempenho, que permitem o compartilhamento coordenado de recursos heterogêneos. O ambiente da Grade facilita a alocação e escalonamento de recursos distribuídos, abstraindo questões como localização. As Grades também atendem aos requisitos de segurança exigidos em organizações multi-institucionais, como autenticação, autorização e controle de acesso, fazendo da Grade um ambiente altamente colaborativo.

Além dessas características, a arquitetura da Grade baseada em serviços (OGSA) abre um largo espectro de aplicações para Grades dentro dos padrões mundialmente usados na Internet. No entanto esse ambiente ainda não oferece suporte adequado a *workflows*, principalmente para o usuário.

Esta dissertação contribui apresentando o *middleware* GPO (*Grid Process Orchestration)*, uma infra-estrutura que faz orquestração de serviços e processos em grades computacionais. O GPO é uma ferramenta destinada ao usuário, que permite criar e gerenciar fluxos complexos, com tarefas fortemente acopladas, sem suporte adicional. O GPO apresenta uma arquitetura flexível e extensível, ampliando as facilidades para o uso das Grades.

A linguagem GPOL criada para a construção de *workflows* é compacta, de fácil aprendizado, e compatível com os padrões atuais da *Web*. Ela tem recursos para execução de tarefas e permite a invocação de serviços da Grade. Além disso, pode ser continuamente expandida pelo usuário, na medida que aceita código *Java* como parte do *workflow*.

A arquitetura flexível garante escalabilidade ao GPO. A independência entre o GPO Run e o GPO Maestro permite ao usuário processar vários *workflows* simultaneamente. A implementação do GPO Maestro, feita através do conceito de fábrica/instância, possibilita o amplo uso dos recursos da Grade, pois é possível usar uma instância do GPO Maestro em cada recurso computacional disponível.

O GPO contribui de forma efetiva agregando suporte a *workflow* ao GT, pois o WS-

GRAM não oferece essa facilidade. Além disso, a orquestração de serviços feita através do GPO aumenta o potencial de aplicação da Grade, permitindo que um novo grupo de aplicações possa ser usado nesse ambiente. O GPO serve como ferramenta de integração, aproximando provedores e consumidores de serviços facilitando a colaboração entre os usuários da Grade.

Trabalhos Futuros

Possíveis extensões a este trabalho são:

- Linguagem GPOL

  Ampliar o tratamento de falhas, incluir tratamento para eventos e incluir recursos para construção e uso de funções.

- Infra-estrutura

  Incluir analisador sintático para WS-BPEL, permitindo o uso integrado de Serviços *Web* e Serviços da Grade, e desenvolver uma interface gráfica para facilitar a criação dos *workflows*; e

  Considerar outros mecanismos para fazer a transferência de arquivos, que ofereçam desempenho superior ao do *GridFTP* usado.

Referências Bibliográficas

[1] Altair Engineering. Altair PBS Professional 8.0. www.altair.com/software/pbspro.htm.

[2] K. Amin, G. von Laszewski, M. Hategan, N. J. Zaluzec, S. Hampton e A. Roddi. GridAnt: A Client-Controllable Grid Workflow System. In Proceedings of the 37th Hawaii International Conference on System Sciences (HICSS-37), Havaí, 5-8 Janeiro 2004.

[3] K. Amin, G. von Laszewski, R. Al Ali, O. Rana e D. Walker. An Abstraction Model for a Grid Execution Framework. Journal of Systems Architecture, Volume 52, Issue 2 , pp. 73-87, Fevereiro 2006.

[4] N. Andrade, W. Cirne, F. Brasileiro, P. Roisenberg. OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing. Proc. 9th Workshop on Job Scheduling Strategies for Parallel Processing, Junho 2003.

[5] Ant - a Java-based Build Tool. http://ant.apache.org.

[6] L. M. Applegate, C. W. Holsapple, R. Kalakota, F. J. Radermacher, e A. B. Whinston. Building Blocks of New Business Opportunity. Journal of Organization Computing and Electronic Commerce, 6(1):1-10, 1996.

[7] Arjuna Technologies, Ltd., BEA Systems, Hitachi, Ltd., International Business Machines Corporation, IONA Technologies, Microsoft Corporation, Inc. Web Services Transaction (WS-Transaction). ftp://www6.software.ibm.com/software/developer/library/ws-transpec.pdf, Agosto 2005.

[8] BEA Systems, Intalio, SAP e Sun Microsystems Web Service Choreography Interface 1.0. http://www.sun.com/software/xml/developers/wsci/wsci-spec-10.pdf, 2002.

[9] D. Bhatia, V. Burzevski, M. Camuseva, G. Fox, W. Furmanski e G. Premchandran. WebFlow - a visual programming paradigm for Web/Java based coarse grain distributed computing. Concurrency - Practice and Experience, volume 9, número 6, pp. 555-577, 1997.

[10] L. F. Bittencourt e E. R. M. Madeira. A Dynamic Approach for Scheduling Dependent Tasks on the Xavantes Grid Middleware. In 4th ACM International Workshop on Middleware for Grid Computing (MGC'06), ACM / IFIP / USENIX 7th International Middleware Conference, Melbourne, Australia, Dezembro 2006.

[11] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web Services Architecture. W3C Working Group Note, http://www.w3.org/TR/ws-arch/, Fevereiro 2004.

[12] T. Bray, J.Paoli, C. M. Sperberg-McQueen, E. Maler e F. Yergeau (Editores) Extensible Markup Language (XML) 1.0. W3C Recommendation, http://www.w3.org/TR/REC-xml/, Fourth Edition, Agosto 2006.

[13] P. Brittenham. An Overview if the Web Services Inspection Language. www.ibm.com/developerworks/webservices/library/ws-wsilover, 2001.

[14] R. Y. de Camargo, A. Goldchleger, F. Kon e A. Goldman. The Implementation of the BSP Parallel Computing Model over the InteGrade Grid Middleware. Concurrency and Computation: Practice & Experience, Volume 18, Issue 6, pp. 567-579, Maio 2006, John Wiley and Sons Ltd.

[15] L. M. Camarinha-Matos e H. Afsarmanesh. The Virtual Enterprise Concept. Em IFIP International Conference on Infrastructure for Virtual Enterprises (PRO-VE 99), pp. 3-14, Porto, Portugal, Outubro 1999

[16] J. Cao, S. A. Jarvis, S. Saini e G. R. Nudd. GridFlow: WorkFlow Management for Grid Computing. In Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'03), pp. 198-205, 2003.

[17] E. Cerami. Web Services Essentials Distributed Applications with XML-RPC, SOAP, UDDI. O'Reilly and Associates, Inc, 1a. Edição, 2002.

[18] K. Chao, M. Younas, N. Griffiths, I. Awan, R. Anane e C. Tsai Analysis of Grid Service Composition with BPEL4WS Proceeding of 18th International Conference on Advanced Information Networking and Application (AINA04), Volume 1, pp. 284-298, Fukuoka, Japão, Março 2004.

[19] R. Chinnici, J. Moreau, A. Ryman, and S. Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Candidate Recommendation, http://www.w3.org/TR/wsdl20/, Março 2006.

[20] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1.. W3C, Note 15, www.w3c.org/TR/wsdl, 2001.

[21] L. Clement, A. Hately, C. von Riegen, and T. Rogers. UDDI Version 3.0.2 Specification. OASIS UDDI Spec Technical Committee Draft, Outubro 2004.

[22] F. Curbera, R. Khalaf, N. Mukhi, S. Tai e S. Weerawarana. The Next Step in Web Services. Communications of the ACM, 46(10):29-34, Outubro 2003.

[23] D. Cybok A Grid Workflow Infrastructure GGF10 - The Tenth Global Grid Forum, Berlim, Alemanha, Março 2004.

[24] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling e S. Tuecke. From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring and Evolution, http://www.ibm.com/developerworks/library/ws-resource/ogsi_to_wsrf_1.0.pdf, Março 2004.

[25] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke e W. Vambenepe. The WS-Resource Framework. http://www.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf, Março 2004.

[26] E. Deelman, T. Kosar, C.Kesselman e M. Livny. What makes workflows work in an opportunistic environment. Concurrency and Computation: Practice and Experience, Volume 18, Issue 10, pp. 1187-1199, John Wiley and Sons Ltd., Agosto 2006.

[27] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob D. S. Katz. Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems. Scientific Programming Journal, Volume 13(3), pp. 219-237, 2005.

[28] I. Foster, J. Voeckler, M. Wilde e Y. Zhao Chimera: A virtual data system for representing, querying, and automating data derivation. In 14th International Conference on Scientific and Statistical Database Management (SSDBM'02), Edinburgo, Escócia, Julho 2002.

[29] I. Foster. Globus Toolkit version 4: Software for Service Oriented Systems. IFIP International Conference on Network and Parallel Computing, LNCS 3779, pp. 2-13, 2005.

[30] I. Foster and C. Kesselman and J. Nick and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Globus Project, 2002, www.globus.org/research/papers/ogsa.pdf.

[31] I. Foster, C. Kesselman e S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications, 15(3):200-222, 2001.

[32] G. Fox, W. Wu, A. Uyar, H. Bulit e S. Pollickara. Global Multimedia Collaboration System. Proceedings of Middleware 2003 Companion - Rio de Janeiro, Brasil, pp. 245-250, Junho 2003.

[33] J. Frey, T. Tannenbaum, I. Foster, M. Livny e S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. Cluster Computing, vol. 5, pp. 237-246, 2002.

[34] D. Gannon, R. Ananthakrishnan, S. Krishnan, M. Govindaraju, L. Ramakrishnan e A. Slominski. Grid Web Services and Application Factories www.extreme.indiana.edu/xcat/AppFactory.pdf, Junho 2002.

[35] D. Z. G. Garcia e M. B. F. de Toledo. A Web Service Architecture Providing QoS Management. Fourth Latin American Web Congress (LA-WEB'06) pp. 189-198, México, Outubro 2006.

[36] Globus Alliance. Globus Toolkit. http://www.globus.org/toolkit.

[37] Globus Alliance. Globus Toolkit. http://www.globus.org/toolkit/docs/3.0/.

[38] Globus Alliance. Globus Toolkit. http://www.globus.org/toolkit/docs/4.0/.

[39] Globus Alliance. GT 4 WS-GRAM www.globus.org/toolkit/docs/4.0/execution/wsgram.

[40] Globus Alliance GT 4.0 WS GRAM: Job Description Schema Doc http://www.globus.org/toolkit/docs/4.0/execution/wsgram/schemas/gram_job_description.html#SchemaProperties

[41] Globus Alliance. Commodity Grid (CoG) Kits. http://wiki.cogkit.org/index.php/Main_Page.

[42] A. Goldchleger, F. Kon, A. Goldman e M. Finger. InteGrade: Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines, Proceedings of Middleware 2003 Companion - Rio de Janeiro, Brasil, pp. 232-234, Junho 2003.

[43] A. Goldchleger, F. Kon, A. Goldman, M. Finger e G. C. Bezerra. InteGrade: Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines. Concurrency and Computation: Practice & Experience, Volume 16, pp. 449-459, Março 2004, John Wiley and Sons Ltd.

[44] A. Goldman e C. Queiroz A Model for Parallel Job Scheduling on Dynamic Computer Grids. Proceedings of Middleware 2003 Companion - Rio de Janeiro, Brasil, pp. 264-266, Junho 2003.

[45] Grid Physics Network The GriPhyN Project: Towards Petascale Virtual-Data Grids www.griphyn.org.

[46] M. Hategan, G. von Laszewski e K. Amin. Karajan: A Grid Orchestration Framework. Supercomputing 2004, Pittsburgh, 6-12 Novembro 2004.

[47] Portal InteGrade http://integrade.incubadora.fapesp.br/portal.

[48] ISO (International Organization for Standardization). Standard Generalized Markup Language (SGML). First Edition, Outubro 1986.

[49] B. Jacob, M. Brown, K. Fukui e N. Trivedi. Introduction to Grid Computing. IBM Corp., www.ibm.com/redbooks, Dezembro 2005.

[50] R. Kay. Markup Languages Quick Study. Computerworld Development, http://www.computerworld.com, Agosto, 2005.

[51] T. Kosar e M. Livny. Stork: Making Data Placement a First Class Citizen in the Grid. In the Proceedings of 24th IEEE International Conference on Distributed Computing Systems (ICDCS 2004), Tokio, Japão, Março 2004.

[52] S. Krishnan, P. Wagstrom e G. von Laszewski. GSFL: A Workflow Framework for Grid Services. In Preprint ANL/MCS-P980-0802, Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439, Estados Unidos, 2002.

[53] G. von Laszewski, I. Foster, J. Gawor e P. Lane. A Java Commodity Grid Kit. Concurrency and Computation: Practice and Experience, Volume 13, Issue 8-9, pp. 643-662, John Wiley and Sons Ltd., 2001.

[54] M. zur Muehlen Workflow-based Process Controlling. Foundation, Design and Application of Workflow-driven Process Information Systems. Logos Verlag Ed., Berlin 2004.

[55] J. Novotny, M. Russell e O. Wehrens. GridSphere: A Portal Framework for Building Collaborations. Proceedings of Middleware 2003 Companion - Rio de Janeiro, Brasil, pp. 178-185, Junho 2003.

[56] OASIS Organization for the Advancement of Structured Information Standards www.oasis-open.org.

[57] OASIS. Web Services Business Process Execution Language Version 2.0. Public Review Draft, http://docs.oasis-open.org/wsbpel/2.0/, Agosto 2006.

[58] OASIS Web Services Atomic Transaction (WS-AtomicTransaction). Version 1.1, Public Review Draft 01, http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-pr-01.pdf, Agosto 2006.

[59] OASIS. Web Services Business Activity (WS-BusinessActivity). Version 1.1, Public Review Draft 01, http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec-pr-01.pdf, Novembro 2006.

[60] OASIS.Web Services Coordination (WS-Coordination). Version 1.1, Public Review Draft 01, http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.1-spec-pr-01.pdf, Agosto 2006.

[61] OASIS. Web Services Resource 1.2 (WS-Resource). OASIS Standard, http://www.oasis-open.org/specs/index.php#wsrf1.2, Abril 2006.

[62] OASIS. Web Services Notification (WSN) TC. OASIS Standard, OASIS. http://www.oasis-open.org/specs/index.php#wsnv1.3, Version 1.3, Outubro 2006.

[63] OGF Open Grid Forum http://www.ogf.org/.

[64] OGF. The Open Grid Services Architecture, Version 1.5. www.ggf.org/documents/GFD.31.pdf, Julho 2006.

[65] OGF. Open Grid Service Infrastructure Primer. www.ogf.org/documents/GFD.31.pdf, Agosto 2004.

[66] OGF - Open Grid Forum. Job Submission Description Language (JSDL) Specification, Version 1.0. http://forge.gridforum.org/projects/jsdl-wg, Novembro 2005.

[67] OMC Object Management Group. CORBA (Common Object Request Broker Architecture) v3.0.3 Specification. http://www.omg.org/technology/documents/formal/corba_iiop.htm, Maio 2004.

[68] V. Ouzounis Electronic Commerce and New Ways of Work - An R&D Road-Map. European Commission - Directoral General III, 1998.

[69] S. Parastatidis, J. Webber, P. Watson e T. Rischbeck. WS-GAF: A Framework for Building Grid Applications Using Web Services. Journal of Concurrency and Computation: Practice and Experience, 17(2-4), p391-417, 2005.

[70] M. G. Pinho, R. F. Da Costa, M. A. P. Lima, e L. G. Ferreira. Calculation of Photoionization of Relativistic Atoms. Proceedings of 24th International Conference on Photonic Electronic and Atomic Collisions (XXIV ICPEAC), Rosário, Argentina, Julho, 2005.

[71] Platform Computing. Platform LSF. www.platform.com/Products/Platform.LSF.Family/.

[72] L. Reinehr e M. B. F. de Toledo. WorkToDo - um sistema de gerenciamento de workflows para ambientes de comunicação sem fio. XX Simpósio Brasileiro de Redes de Computadores - Brasil, Maio 2002.

[73] S. Reynaud e F. Hernandez. A XML-based Description Language and Execution Environment for Orchestration Grid Jobs. Proceedings of the 2005 IEEE International Conference on Services Computing (SCC'05), Volume 2, pp. 192-199, Orlando, Estados Unidos, Julho 2005.

[74] I. J. G. dos Santos VM-Flow - Centro de Negócios Virtual baseado em Políticas de Integração e em Orquestração e Coreografia de Serviços. Tese de Mestrado, UNICAMP - Universidade de Campinas, São Paulo, Brasil, Fevereiro 2004.

[75] B. Schulze e E. R. M. Madeira. Grid Computing with Active Services. Concurrency and Computation: Practice and Experience, Estados Unidos, Vol. 16, No. 5, pp. 535-542, Abril 2004.

[76] B. Schulze e R. Nandkumar. Special Issue: Middleware for Grid Computing. Concurrency and Computation: Practice and Experience 18(6):549-552 (2006).

[77] C. R. Senna e E. R. M. Madeira. Uma Infra-estrutura para Workflow de Serviços em Grades Computacionais. Simpósio Brasileiro de Redes de Computadores SBRC2005, Fortaleza, Brasil, Maio 2005.

[78] R. S. Silva Filho. Uma Arquitetura Baseada em CORBA para Workflow de Larga Escala. Dissertação de Mestrado, UNICAMP - Universidade de Campinas, São Paulo, Brasil, Agosto 2000.

[79] A. Slomiski. On Using BPEL Extensibility to Implement OGSI and WSRF Grid Workflows: Research Articles. Concurrency and Computation: Practice & Experience, Chichester, Reino Unido, 18(10):1229-1241, 2006.

[80] B. Sotomayor e L. Childers Globus Toolkit 4: Programming Java Services. Morgan Kaufmann, 2005.

[81] D. Thain, T. Tannenbaum e M. Livny. Condor and the Grid. Grid Computing: Making The Global Infrastructure a Reality, Fran Berman, Anthony J.G. Hey, Geoffrey Fox, editors. John Wiley, 2003.

[82] Triana Triana Workflow. http://triana.co.uk.

[83] UFCG OurGrid Home Page. www.ourgrid.org.

[84] UFCG. The OurGrid Project, http://dragao.lsd.ufcg.edu.br/twiki-public/bin/view/OurGrid/OurProject.

[85] J. Unger e M. Haynos. A visual tour of Open Grid Services Architecture. IBM Corp. http://www6.software.ibm.com/software/developer/library/gr-visual.pdf, Agosto 2003.

[86] University of Wisconsin-Madison (UW-Madison). Condor Research Project. www.cs.wisc.edu/condor/.

[87] W3C. SOAP Version 1.2 Part 0: Primer. W3C Recomendation, http://www.w3.org/TR/2003/REC-soap12-part0-20030624/, Junho 2003.

[88] W3C. SOAP Tutorial W3 Schools. World Wide Web Consortium W3C, http://www.w3schools.com/soap/default.asp.

[89] W3C. XML Schema Part 0: Primer. W3C Recommendation, http://www.w3.org/TR/xmlschema-0/, Second Edition, Outubro 2004.

[90] W3C. XML Schema Part 1: Structures Second Edition. W3C Recommendation, Outubro 2004, http://www.w3.org/TR/xmlschema-1/.

[91] W3C. XML Schema Part 2: Datatypes Second Edition. W3C Recommendation, Outubro 2004, http://www.w3.org/TR/xmlschema-2/.

[92] W3C. Web Services Addressing 1.0 - Core (WS-Addressing). W3C Recommendation, http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/, Maio 2006.

[93] S. Weerawarana e F. Curbera Business Processes: Understanding BPEL4WS, Part 1. IBM Developerworks, Agosto 2002.

[94] Workflow Management Coalition, http://www.wfmc.org.

[95] M. Zager. Business Process Orchestration with BPEL: BPEL supports time-critical decision making. SOA/Web Services, http://webservices.sys-con.com/read/155631_1.htm, Dezembro 2005.

[96] S. Zhang, Y. Zong, Z. Ding e J. Liu. Workflow-oriented Grid Service Composition and Scheduling. International Symposium on Information Technology: Coding and Computing, Estados Unidos, pp. 219-241, IEEE Computer Society, Abril 2005.

[97] S. Zhang, Y. Zong, Z. Ding e J. Liu. Workflow-oriented grid service composition and scheduling. In International Sysmposium on Information Technology: Coding and Computing, Estados Unidos, Volume 2, pp. 214-219, IEEE Computer Society, Abril 2005.

Apêndice A

Workflow GPOL para Fotoionização de Átomos


```
<?xml version="1.0"encoding="UTF-8"?>
  <workflow name="Fotoion">
    <definitions name="FotoionDefs">
      <variables>
        <variable name="sL"type="string"value="0.00"/>
        <variable name="dL"type="double"value="0.00"/>
        <variable name="sq"type="string"value="0.00"/>
        <variable name="dq"type="string"value="0.00"/>
        <variable name="sJK"type="string"value="0.00"/>
        <variable name="sJK1l"type="string"value="0.00"/>
        <variable name="sJk2l"type="string"value="0.00"/>
        <variable name="dJK"type="double"value="0.00"/>
        <variable name="dJK1l"type="double"value="0.00"/>
        <variable name="dJK2l"type="double"value="0.00"/>
        <variable name="dfPi"type="double"value="12.56"/>
        <variable name="iEoF"type="integer"value="1"/>
        <variable name="sTupla"type="string" />
        <variable name="iRetCode"type="integer" />
        <variable name="dAux1"type="double"value="0.00"/>
        <variable name="dAux2"type="double"value="0.00"/>
        <variable name="dAux3"type="double"value="0.00"/>
        <variable name="sAux1"type="string"/>
        <variable name="dResTA"type="double"value="0.00"/>
        <variable name="dResTB"type="double"value="0.00"/>
```

```
        <variable name="dResTC"type="double"value="0.00"/>
        <variable name="dResTD"type="double"value="0.00"/>
        <variable name="dResTE"type="double"value="0.00"/>
        <variable name="dResTF"type="double"value="0.00"/>
        <variable name="dSomatoria"type="double"value="0.00"/>
    </variables>
    <services>
        <service name="accessFile"
            uri="http://143.106.24.147:8080/wsrf/services/.../FactAFService"
            fal = "org.globus...FactAFServiceAddressingLocator"
            sal = "org.globus...AFServiceAddressingLocator"
            type="Factory"/>
        <service name="strFunctions"
            uri="http://143.106.24.147:8080/wsrf/services/.../FactSFService"
            fal = "org.globus...FactSFServiceAddressingLocator"
            sal = "org.globus...SFServiceAddressingLocator"
            type="Factory"/>
        <service name="CG3j"
            uri="http://143.106.24.147:8080/wsrf/services/.../FCG3jService"
            fal = "org.globus...FCG3jServiceAddressingLocator"
            sal = "org.globus...CG3jServiceAddressingLocator"
            type="Factory"
            ret-type="double"/>
        <service name="CG6j"
            uri="http://143.106.24.147:8080/wsrf/services/.../FCG6jService"
            fal = "org.globus...FCG6jServiceAddressingLocator"
            sal = "org.globus...CG6jServiceAddressingLocator"
            type="Factory"
            ret-type="double"/>
    </services>
</definitions>
<process name="FotoionProcess">
    <!– Abre o arquivo com as tuplas –>
    <invoke service="accessFile"operation="openFile">
        <argument value="FotoionTuplas.txt"type="string"/>
        <return variable="iRetCode"type="integer"/>
    </invoke>
    <!– Le a primeira tupla –>
```

```
<invoke service="accessFile"operation="readLine">
   <argument value="1"type="integer"/>
   <return variable="sTupla"type="string"/>
</invoke>
<!– Verifica a leitura feita –>
<invoke service="strFunctions"operation="isNull">
   <argument variable="sTupla"type="string"/>
   <return variable="iEoF"type="integer"/>
</invoke>
<!– Enquanto ler tuplas faz o calculo
      acumulando valor resultante –>
<while condition="iEoF eg 0">
   <!– Prepara variaveis para o calculo –>
   <!– Obtem o valor de j(k) string (posicao 6 da tupla) –>
   <invoke service="strFunctions"operation="getElement">
      <argument variable="sTupla"type="string"/>
      <argument value="6"type="integer"/>
      <return variable="sJK"type="string"/>
   </invoke>
   <!– j(k) string para double –>
   <invoke service="strFunctions"operation="strToDouble">
      <argument variable="sJK"type="string"/>
      <return variable="dJK"type="double"/>
   </invoke>
   <!– Obtem o valor de j(k') string (posicao 7 da tupla) –>
   <invoke service="strFunctions"operation="getElement">
      <argument variable="sTupla"type="string"/>
      <argument value="7"type="integer"/>
      <return variable="sJK1l"type="string"/>
   </invoke>
   <!– j(k') string para double –>
   <invoke service="strFunctions"operation="strToDouble">
      <argument variable="sJK1l"type="string"/>
      <return variable="dJK1l"type="double"/>
   </invoke>
   <!– Obtem o valor de j(k") string (posicao 8 da tupla) –>
   <invoke service="strFunctions"operation="getElement">
      <argument variable="sTupla"type="string"/>
```

```
    <argument value="8"type="integer"/>
    <return variable="sJK2l"type="string"/>
  </invoke>
  <!– j(k") string para double –>
  <invoke service="strFunctions"operation="strToDouble">
    <argument variable="sJK2l"type="string"/>
    <return variable="dJK2l"type="double"/>
  </invoke>
  <flow>
    <sequence>
      <!– Calculo do Termo A –>
      <!– (2j(k') + 1) –>
      <assign name="dAux1"operator="="value="2" />
      <assign name="dAux1"operator="*="variable="dJK1l" />
      <assign name="dAux1"operator="+="value="1" />
      <!– (2j(k") + 1) –>
      <assign name="dAux2"operator="="value="2" />
      <assign name="dAux2"operator="*="variable="dJK2l"/>
      <assign name="dAux2"operator="+="value="1" />
      <!– (2L + 1) –>
      <assign name="dAux3"operator="="value="2" />
      <assign name="dAux3"operator="*="variable="dL"/>
      <assign name="dAux3"operator="+="value="1" />
      <!– ((2j(k') + 1)*(2j(k") + 1)*(2L + 1))/4Pi –>
      <assign name="dResTA"operator="="variable="aux1" />
      <assign name="dResTA"operator="*="variable="aux2" />
      <assign name="dResTA"operator="*="variable="aux3" />
      <assign name="dResTA"operator="/="variable="dfPi" />
    </sequence>
    <!– Calculo do Termo B –>
    <invoke service="CG3j"operation="calc">
      <!– Elementos da matriz (1 1 L -q q 0) –>
      <argument value="1"type="double"/>
      <argument value="1"type="double"/>
      <argument variable="dL"type="double"/>
      <argument variable="dq"type="double"/>
      <argument variable="dq"type="double"/>
      <argument value="0"type="double"/>
```

```
    <return variable="dResTB"type="double"/>
</invoke>
<!– Calculo do Termo C –>
<invoke service="CG3j"operation="calc">
    <!– Elementos da matriz (j(k') j(k") L 0.5 -0.5 0) –>
    <argument variable="dJK1l"type="double"/>
    <argument variable="dJK2l"type="double"/>
    <argument variable="dL"type="double"/>
    <argument value="0.5"type="double"/>
    <argument value=-0.5"type="double"/>
    <argument value="0"type="double"/>
    <return variable="dResTC"type="double"/>
</invoke>
<!– Calculo do Termo D –>
<invoke service="CG3j"operation="calc">
    <!– Elementos da matriz (j(k') 1 j(k) 0.5 0 -0.5) –>
    <argument variable="dJK1l"type="double"/>
    <argument value="1"type="double"/>
    <argument variable="dJK"type="double"/>
    <argument value="0.5"type="double"/>
    <argument value="0"type="double"/>
    <argument value=-0.5"type="double"/>
    <return variable="dResTD"type="double"/>
</invoke>
<!– Calculo do Termo E –>
<invoke service="CG3j"operation="calc">
    <!– Elementos da matriz (j(k") 1 j(k) 0.5 0 -0.5) –>
    <argument variable="dJK2l"type="double"/>
    <argument value="1"type="double"/>
    <argument variable="dJK"type="double"/>
    <argument value="0.5"type="double"/>
    <argument value="0"type="double"/>
    <argument value=-0.5"type="double"/>
    <return variable="dResTD"type="double"/>
</invoke>
<!– Calculo do Termo F –>
<invoke service="CG6j"operation="calc">
    <!– Elementos da matriz (1 1 L j(k") j(k') j(k)) –>
```

```
                <argument value="1"type="double"/>
                <argument value="1"type="double"/>
                <argument variable="dL"type="double"/>
                <argument variable="dJK2l"type="double"/>
                <argument variable="dJK1l"type="double"/>
                <argument variable="dJK"type="double"/>
                <return variable="dResTF"type="double"/>
            </invoke>
        </flow>
        <!– Faz multiplicacao dos Termos e Soma no Geral –>
        <assign name="dAux1"operator="="variable="dResTA" />
        <assign name="dAux1"operator="*="variable="dResTB" />
        <assign name="dAux1"operator="*="variable="dResTC"/>
        <assign name="dAux1"operator="*="variable="dResTD" />
        <assign name="dAux1"operator="*="variable="dResTE" />
        <assign name="dAux1"operator="*="variable="dResTF" />
        <assign name="dSomatoria" operator="+=" variable="dAux1" />
    </while>
    <!– Return value to GPO-Client –>
    <return variable="dSomatoria"/>
  </process>
</workflow>
```