MATHIAS SANTOS DE BRITO

Arquitetura de um sistema para Integração de Bancos de Dados
com Suporte a Replicação Utilizando Tecnologia de Grades
Computacionais

São Paulo
2009

MATHIAS SANTOS DE BRITO

Arquitetura de um sistema para Integração de Bancos de Dados
com Suporte a Replicação Utilizando Tecnologia de Grades
Computacionais

Dissertação apresentada à Escola
Politécnica da Universidade de São Paulo
para obtenção do título de Mestre em
Engenharia.

São Paulo
2009

MATHIAS SANTOS DE BRITO

Arquitetura de um sistema para Integração de Bancos de Dados
com Suporte a Replicação Utilizando Tecnologia de Grades
Computacionais

Dissertação apresentada à Escola
Politécnica da Universidade de São Paulo
para obtenção do título de Mestre em
Engenharia.

Área de Concentração: Engenharia de
Computação e Sistemas Digitais

Orientadora: Prof. Livre-Docente Líria
Matsumoto Sato

São Paulo
2009

DEDICATÓRIA

Dedico este trabalho aos meus pais.

AGRADECIMENTOS

Gostaria de agradecer primeiramente a Deus, à minha namorada Larissa, por ter compreendido as minhas necessidades e ter suportado todo este tempo longe de mim e também pelo seu apoio nos momentos em que precisei.

Agradeço aos meus pais por terem me dado a criação e condições necessárias para ter chegado aqui. Aos meus avós, os presentes e ausentes pelos ensinamentos e palavras de sabedoria, ao meu tio Jorge minha tia e primas aos quais tenho imensa admiração. Aos meus irmãos, pelo apoio e pela excelente companhia. Gostaria de agradecer também aos amigos que deixei em Itabuna aos quais me afastei, desde já agradeço a compreensão. Aos novos amigos feitos em São Paulo. A todos e a todas que direta ou indiretamente me fizeram chegar até aqui.

Gostaria de agradecer a todos os integrantes do LAHPC pelo apoio, em especial à Prof. Líria Matsumoto Sato pelos conselhos valiosos e pela orientação a qual me foi dada.

Agradeço também ao CNPq por ter financiado esta pesquisa fornecendo-me uma bolsa de doutorado pelo período de 2 anos.

A verdadeira glória consiste em fazer o
que merece ser escrito, em escrever o
que merece ser lido.

(Plínio, o Velho)

RESUMO

Grades computacionais tem a finalidade de oferecer meios para o compartilhamento de recursos distribuídos geograficamente para o uso em aplicações que necessitem. A integração de Bancos de Dados distribuídos geograficamente pode ser obtida através do compartilhamento de recursos provido pela tecnologia de *grades*. Este trabalho utiliza *middlewares* de *grade* computacional, especificamente *Globus* e OGSA-DAI bem como outras tecnologias como LDAP e JDBC, especificando uma arquitetura distribuída para possibilitar a integração de Bancos de Dados, oferecendo também suporte a replicação. A arquitetura proposta possibilita a configuração de níveis hierárquicos de replicação. O presente trabalho apresenta a implementação de um protótipo desta arquitetura. Um *driver* JDBC é apresentado para possibilitar o uso dos bancos de dados expostos na *grade*. Tem-se como resultado deste trabalho, a definição da arquitetura e o desenvolvimento de ferramentas culminando em um protótipo funcional, bem como uma versão estável do *driver* JDBC. A possibilidade de integração de bancos de dados na *grade* com suporte a replicação, torna possível aplicações de alto desempenho e alta disponibilidade sem comprometer a autonomia local dos bancos de dados integrados.

Palavras-chave: Bancos de Dados Distribuídos, *Grades* Computacionais, OGSA-DAI, *Globus Toolkit*

ABSTRACT

Grid Computing have the goal of providing means to share resources to be used by applications that need these resources. The integration of geographically distributed databases can be obtained by sharing resources using grid computing technology. This work uses grid middlewares, specifically Globus Toolkit and OGSA-DAI, as well as other technologies as LDAP and Java JDBC, specifying a distributed architecture to allow database integration, also offering support for replication. The proposed architecture offers also its configuration in hierarchical levels of replication. The work also presents the implementation of a prototype based on the proposed architecture. A JDBC *driver* is presented, it allows the ease access to databases exposed in the grid. The results of this work are the definition of the architecture, the development of a functional prototype and a stable version of the JDBC Driver. The possibility of integrating databases with replication support allows applications of high performance and high availability without interfering in the local autonomy of the databases.

Keywords: Distributed Databases, grade Computing, OGSA-DAI, Globus Toolkit

LISTA DE ILUSTRAÇÕES

Figura 1: Divisão em camadas de uma infra-estrutura de grades.....	21
Figura 2: Dois cenários de uso do OGSA-DAI. Exposição simples de Banco de Dados (a), exposição de vários bancos de dados com integração utilizando ResourceGroup (b).....	31
Figura 3: Componentes do sistema, suas dependências e interfaces.....	45
Figura 4: Componentes internos ao componente DataHandler.....	50
Figura 5: As etapas para acesso e modificação dos dados. O acesso ao diretório de dados está condicionado à identificação do tipo de operação.....	51
Figura 6: Replicando dados de forma hierárquica. Registros do nível mais inferior são replicados em níveis superiores. Atualizações se propagam da mesma forma..	52
Figura 7: Diagrama de Classes simplificado do driver JDBC.....	55
Figura 8: Diagrama de sequência simplificado exemplificando o funcionamento da activity DatabaseMetaDataMethod.....	57
Figura 9: Fluxo de dados dentro da estrutura dos sistemas de saúde no Brasil.....	70
Figura 10: Visão de uma possível solução para a problemática dos Hemocentros utilizando a arquitetura proposta.....	74
Figura 11: Infra-estrutura de testes para o sistema implementado.....	76
Figura 12: Estrutura dos testes realizados com o DictionaryHandler.....	79
Figura 13: Detalhes do funcionamento dos testes executados.....	80
Figura 14: Teste realizado para verificar o comportamento do protótipo quando configurado hierarquicamente.....	83
Figura 15: Testes realizados para validar a funcionalidade das activities de suporte ao driver JDBC desenvolvido.....	85
Figura 16: Testes realizados para validar a funcionalidade do driver JDBC para OGSA-DAI.....	86

LISTA DE TABELAS

Tabela 1: Especificações recomendadas pela OGF.....	23
Tabela 2: Especificações utilizadas na definição do WSRF.....	26
Tabela 3: Bancos de dados compatíveis pelo OGSA-DAI, e consequentemente pelo protótipo apresentado. A tabela lista as versões testadas, porém outras versões podem vir a funcionar. Fonte: OGSA-DAI Documentation (http://www.ogsadai.org.uk/documentation).....	49
Tabela 4: Propriedades obrigatórias para a configuração do protótipo.....	50
Tabela 5: Activities do componente DataHandler e a qual subcomponente pertence.	63
Tabela 6: Linhas de Código e quantidade de Classes implementadas durante a execução do projeto.....	68
Tabela 7: Máquinas utilizadas para teste dos componentes.....	78
Tabela 8: Lista dos testes realizados para validar o componente DataHandler, suas descrições e resultados esperados.....	80
Tabela 9: Configurações das máquinas para a definição de uma hierarquia de replicação.....	83

LISTA DE ABREVIATURAS E SIGLAS

DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
ETL	Extract Transform Load
GT	Globus Toolkit
GT4	Globus Toolkit 4
HTTP	HiperText Transfer Protocol
JDBC	Java Database Connectivity
LDAP	Lightweight Directory Access Protocol
NFS	Network File System
ODBC	Open Data Base Connectivity
OGF	Open Grid Forum
OGSA	Open Grid Services Architecture
OGSA-DAI	Open Grid Services Architecture – Database Access Integraion
OGSI	Open Grid Service Infrastructure
OSI	Open Systems Interconnection
SGBD	Sistema Gerenciador de Banco de Dados
SOAP	Simple Object Access Protocol
SOAP-RPC	Simple Object Access Protocol - Remote Procedure Call
SSH	Secure Shell
TCP/IP	Transmission Control Protocol/Internet Protocol
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WS	Web Service
WSDL	Web Services Description Language
WSRF	Web Services Resource Framework
XML	eXtensible Markup Language

SUMÁRIO

LISTA DE ILUSTRAÇÕES.....	9
LISTA DE TABELAS.....	10
LISTA DE ABREVIATURAS E SIGLAS.....	11
1 INTRODUÇÃO.....	15
1.1 Objetivos.....	16
1.2 Motivação.....	16
1.3 Metodologia Utilizada.....	17
1.4 Nota sobre Termos Utilizados Neste Documento.....	18
1.5 Estrutura do Documento.....	19
2 CONCEITOS E TECNOLOGIAS UTILIZADOS.....	20
2.1 Middlewares.....	20
2.2 Web Services.....	20
2.3 Grades.....	21
2.3.1 Grades e a Necessidade de Padrões Abertos.....	23
2.3.2 Open Grid Services Architecture.....	24
2.3.3 Open Grid Services Infrastructure.....	25
2.3.4 Web Services Resource Framework.....	26
2.3.5 Middlewares para Grades.....	27
2.4 Globus Toolkit 4 (GT4).....	28
2.5 Bancos de Dados Distribuídos.....	29
2.6 OGSA-DAI.....	30
2.6.1 Estendendo as Funcionalidades do OGSA-DAI.....	33
Atividades (Activities).....	33
Recursos (Resources).....	34
Provedores de Autenticação (Login Providers).....	35
Camadas de Apresentação (Presentation Layers).....	35
2.7 Replicação de Dados.....	36
2.8 Java Database Connectivity (JDBC).....	38
2.9 Uso de Padrões.....	38
2.9.1 Motivação para o uso de Padrões.....	38
2.9.2 Protocolos Padrões na Comunicação e Troca de Mensagens.....	39
3 TRABALHOS RELACIONADOS.....	41
3.1 IBM OGSA-DAI Replication.....	41
3.2 Sybase Data Integration Suite.....	42
3.3 WS-JDBC e JDBC-WS, Tentativas do Uso de WS para Acesso a Bancos de Dados.....	42
3.4 OGSA-DAI ODBC Driver.....	42
4 ARQUITETURA PARA INTEGRAÇÃO DE BANCOS DE DADOS EM GRADE COM SUPORTE A REPLICAÇÃO.....	44
4.1 Objetivos da Arquitetura.....	44
4.2 Descrição da Arquitetura.....	45
4.2.1 Componentes do Sistema.....	46

4.2.2	Componente Client.....	46
	<i>Mapeamento para a Tecnologia Escolhida.....</i>	<i>47</i>
4.2.3	Componente Data Dictionary.....	47
	<i>Mapeamento para a Tecnologia Escolhida.....</i>	<i>48</i>
4.2.4	Componente DataHandler.....	48
	<i>Mapeamento para a Tecnologia Escolhida.....</i>	<i>48</i>
4.2.5	Componentes Databases.....	49
4.2.6	Componente Configurator.....	49
4.3	Detalhes do Componente DataHandler.....	50
4.4	Executando uma Operação.....	52
4.5	Utilizando a Arquitetura de Forma Hierárquica.....	53
4.5.1	Benefícios no Uso de Hierarquias de Replicação.....	53
5	Java Database Connectivity for OGSA-DAI.....	55
5.1	Suporte ao Globus Toolkit e ao Apache Axis.....	55
5.2	Implementação.....	55
5.2.1	Segurança.....	56
5.2.2	Pacote Utilitário.....	57
5.2.3	Extensões ao OGSA-DAI.....	57
5.3	Utilização do Driver.....	59
5.4	Benefícios no Uso do Driver.....	60
6	Implementação de um Protótipo Baseado na Arquitetura	
	Proposta.....	61
6.1	Componente Client.....	61
6.2	Componente DataHadler.....	62
6.2.1	Utilizando o OGSA-DAI para Implementação do DataHandler.....	62
6.2.2	ReplicaManager.....	63
6.2.3	DictionaryHandler.....	64
6.3	Componente Configurator.....	64
6.4	Criando uma Infra-Estrutura de Integração Baseado no Protótipo	
	Desenvolvido.....	65
6.5	Utilizando o Protótipo a Partir de Programas Java.....	66
6.6	Métricas de Código.....	67
7	Estudo de Caso: Aplicação da Arquitetura em um Sistema de	
	Saúde para Hemofílicos.....	69
7.1	Motivação para o Uso da Arquitetura.....	69
7.1.1	Sistemas de Informação em Saúde Pública no Brasil.....	69
7.1.2	Estrutura Hierárquica nos Hemocentros.....	73
7.1.3	As Especificidades dos Hemocentros e dos Pacientes Hemofílicos	73
7.1.4	Adaptando a Arquitetura ao Problema dos Hemocentros.....	74
7.2	Considerações Finais.....	76
8	Testes e Análises.....	77
8.1	Ambiente de Testes.....	77
8.2	Testes Funcionais dos Componentes	78
8.2.1	Teste do Componente Configurator.....	79
8.2.2	Teste do Componente DataHandler.....	79
	<i>Testes do DictionaryHandler.....</i>	<i>79</i>
	<i>Testes do ReplicaManager.....</i>	<i>80</i>
8.3	Testes utilizando Hierarquias.....	83

8.4	Teste do Driver JDBC para OGSA-DAI.....	85
8.4.1	Testes das activities de suporte.....	85
8.4.2	Demais Testes.....	86
8.4.3	Considerações Finais.....	87
9	CONCLUSÃO.....	88
9.1	Publicações.....	89
9.2	Contribuições.....	89
9.3	Trabalhos Futuros.....	90
	REFERÊNCIAS.....	91
	APÊNDICE A - Activities dos Componentes Configurator, DataHandler e seus Subcomponentes.....	96
	APÊNDICE B – Exemplo de Implantação do Protótipo.....	103
	APÊNDICE C - Exemplos de Testes Executados.....	112

1 INTRODUÇÃO

A tecnologia de *Grades*(FOSTER ET AL., 2001) propõe-se resolver a problemática envolvida no compartilhamento de recursos computacionais pertencentes a diferentes entidades/instituições. No meio acadêmico, esta tecnologia tem sido utilizada para aumentar a produtividade em atividades de pesquisa que envolvem colaboração entre instituições. Na indústria, a tecnologia tem sido aplicada principalmente no processamento de tarefas que demandam alto poder computacional, bem como na integração de dados(FOSTER; KESSELMAN, 2003).

Os recursos necessários para a implantação de *grades* têm sido oferecidos na forma de *middlewares*. Estes por sua vez, são camadas de *software* que fazem a intermediação do acesso a determinados recursos, manipulando e tratando as requisições antes de serem repassadas para as partes comunicantes. Nos últimos anos surgiram diversos *middlewares* que oferecem as funcionalidades básicas da tecnologia de *grades*, alguns exemplos são: *Globus Toolkit* (GT)(FOSTER; KESSELMAN, 1997), *Unicore*(ERWIN; SNELLING, 2001), *gLite*(LAURE ET AL., 2006), e outros.

A principal problemática a ser resolvida pelos *middlewares* é a manutenção da autonomia dos recursos e o provimento de ferramentas para realizar transferência de arquivos, execução de tarefas remotamente, indexação de serviços, autenticação e autorização. Este último aspecto possibilita a inserção de recursos na *grade* seguindo as regras e as políticas da instituição que disponibiliza tal recurso. Não menos importante, os *middlewares* têm se preocupado com a utilização de protocolos abertos, para conseguir alcançar um alto nível de interoperabilidade.

Pesquisas sobre a integração de dados mantidos por Sistemas Gerenciadores de Bancos de Dados (SGBD) também têm sido alvo de pesquisas recentes(ANTONIOLETTI ET AL., 2005).

Bancos de Dados distribuídos(OZSU; VALDURIEZ, 1999) visam descentralizar os dados mantidos por um SGBD afim de se obter maior desempenho e confiabilidade

do sistema. Com o advento da tecnologia de *grades*, novas abordagens estão sendo propostas para o desenvolvimento de Bancos de Dados Distribuídos (ANTONIOLETTI ET AL., 2005)(CHEN ET AL., 2007).

1.1 Objetivos

O objetivo deste trabalho é aliar técnicas de Bancos de Dados Distribuídos à tecnologia de *grades*. O principal foco do trabalho é a proposta de uma arquitetura que possibilite a integração de dados armazenados em SGBDs de diferentes fornecedores e espalhados geograficamente. Tal arquitetura deverá também, prever a replicação de dados entre os SGBDs integrados. A implementação de um protótipo da mesma é apresentada.

Este trabalho resulta em uma arquitetura distribuída para a integração e replicação de dados que seja configurável e que possibilite a configuração de hierarquias de replicação. Esta hierarquia diz respeito ao fluxo das replicações, sendo que operações feitas em um banco de dados devem ser propagadas para os níveis mais altos da hierarquia. Apesar da arquitetura ser totalmente distribuída, ela deverá ser flexível o suficiente para possibilitar a sua configuração de uma forma hierárquica.

É também objetivo deste trabalho o desenvolvimento de ferramentas para facilitar a criação de aplicações que façam uso da infra-estrutura, dentre elas a de um *driver Java Database Connectivity* (JDBC) e uma extensão deste para possibilitar o uso dos recursos de replicação.

1.2 Motivação

Middlewares para integração de diferentes SGBDs em uma *grade computacional* são bastante escassos. Soluções proprietárias existem, porém sem um foco na interoperabilidade entre SGBDs de outros fabricantes ou sem compromisso no uso de padrões abertos.

A motivação para este trabalho surgiu da necessidade de se desenvolver um sis-

tema de informação para pacientes com hemofilia dos hospitais das clínicas de Campinas e de São Paulo. Com o intuito de se criar um sistema distribuído e heterogêneo, padrões abertos foram adotados bem como um *middleware* de integração dos dados, oferecendo desta forma uma visão centralizada do sistema, apesar deste ser totalmente distribuído.

A partir da escolha dos *middlewares*, percebeu-se que a replicação de dados armazenados em SGBDs não era possível com aqueles oferecidos pela comunidade acadêmica, surgindo então a necessidade de adaptar um destes para suportar replicação.

Também foi motivação para este trabalho, a percepção da dificuldade na escrita de código para manipular os recursos disponíveis pelos *middlewares*, desenvolvendo-se, então, ferramentas que facilitassem esta tarefa.

1.3 Metodologia Utilizada

A metodologia utilizada seguiu a seguinte estrutura:

- ✓ Identificação dos obstáculos para o desenvolvimento de um sistema com bancos de dados distribuídos utilizando tecnologia de *grids*. Foi realizado um levantamento bibliográfico e o estudo do estado da arte na área em questão.
- ✓ Levantamento dos *middlewares* que se propõem resolver o problema da integração de dados de SGBDs e *middlewares* de *Grids* para propósito geral, seguido de levantamento bibliográfico, e a identificação das vantagens e desvantagens de cada um deles;
- ✓ Desenvolvimento da arquitetura para resolução do problema proposto.
- ✓ A escolha de um dos *middlewares* e o levantamento das deficiências do mesmo;
- ✓ Aperfeiçoamento do *middleware* e desenvolvimento de ferramentas para dar suporte ao problema motivador do trabalho em questão.
- ✓ Validação da Arquitetura e das ferramentas e protótipos desenvolvidos.

O levantamento bibliográfico se deu de forma continua até o fechamento do trabalho. Foi definida a situação corrente na área de pesquisa em questão e a partir daí definiu-se os principais focos de trabalho.

A partir do levantamento bibliográfico, pôde-se então definir as frentes de trabalho para a realização do objetivo proposto. Como primeira etapa, definiu-se a arquitetura do sistema utilizando-a como base para a infra-estrutura, baseada em uma coleção de tecnologias e ferramentas.

Para o desenvolvimento das ferramentas, utilizou-se a metodologia de desenvolvimento incremental, onde etapas de levantamento de requisitos, projeto, implementação e testes são realizados em etapas incrementais. Foram utilizadas algumas características e técnicas de desenvolvimento ágil.

Uma etapa de integração foi realizada, onde se aplicou as ferramentas distribuídas no sistema de saúde citado. Para que esta etapa se desse de uma forma suave, o desenvolvimento do sistema seguiu uma estrutura modularizada, a fim de se obter um sistema fracamente acoplado e de fácil manutenção.

1.4 Nota sobre Termos Utilizados Neste Documento

O termo *expor um recurso* é bastante utilizado neste documento, sendo que este termo se refere ao fato de um recurso poder ser localizado e utilizado dentro de uma infra-estrutura de *grade*. O termo *recurso*, se refere a recursos computacionais que recebem algum tipo de informação as processam e retornam informações através de interfaces de comunicação homem-máquina ou máquina-máquina.

Optou-se por utilizar a palavra *grade*, uma tradução livre do termo do inglês *grid*. O termo *grade* será apresentado em itálico para destacá-la no texto.

Será utilizada também a palavra *resource* (recurso em português) quando o texto discorrer sobre o OGSA-DAI. Um *resource* no OGSA-DAI é uma abstração dentro do *middleware* para um recurso computacional, especialmente bancos de dados. A palavra *middleware* será utilizada para se referir a um software de intermediação entre dois sistemas.

1.5 Estrutura do Documento

O restante deste documento segue a seguinte estrutura:

No capítulo dois são aprofundados os conceitos sobre tecnologias utilizadas no desenvolvimento do trabalho. Apresenta-se também uma discussão das desvantagens e vantagens de cada uma delas e o porquê da escolha destas tecnologias para o desenvolvimento do presente trabalho. No capítulo três são apresentadas as propostas de trabalhos relacionados que visam solucionar o mesmo problema abordado nesta dissertação. Fez-se o levantamento das tecnologias, uma análise dos seus pontos fracos e fortes, bem como um comparativo com a solução proposta. No capítulo quatro discorre-se acerca da proposta para a solução do problema em questão e detalha-se a arquitetura para integração de dados com suporte a replicação. No capítulo cinco detalha-se o desenvolvimento do componente cliente da arquitetura. No capítulo seis são apresentados os demais componentes do sistema e detalhes de implementação destes. O capítulo sete apresenta um estudo de caso, onde se aplicou a arquitetura desenvolvida em um sistema de saúde. No capítulo oito são apresentados as análises realizadas e os testes realizados. Por fim, no capítulo nove são apresentadas as conclusões observadas a partir do desenvolvimento do trabalho.

O Apêndice A lista serviços desenvolvidos e suas características, como entrada, saída e descrição dos seus propósitos.

2 CONCEITOS E TECNOLOGIAS UTILIZADOS

Uma visão geral dos conceitos e tecnologias utilizados é necessária para o entendimento do trabalho proposto. Nesta seção os principais conceitos serão apresentados de forma aprofundada bem como as tecnologias em questão.

Conceitos de *middlewarees*, *grades*, Bancos de Dados Distribuídos, Replicação de Dados, bem como o aprofundamento nos *middlewarees Globus Toolkit 4 (GT4)* (FOSTER, 2005) e OGSA-DAI(ANTONIOLETTI ET AL., 2005) são apresentados.

2.1 Middlewares

Middleware, também referido como mediador, é uma peça de *software* que se coloca entre dois outros *softwares*. A intenção de tal ferramenta é permitir que os *softwares* nos pontos terminais possam interoperar entre si. O *middleware* então funciona como um habilitador, possibilitando a comunicação entre sistemas sem o qual não poderiam se comunicar.

O termo *middleware* tem sua aparição na década de 60. O termo passou a ser extensamente utilizado nos anos 80 como forma de interligar sistemas legados e também na criação de sistemas desacoplados e distribuídos.

2.2 Web Services

Web Service é descrito como sendo(BOOTH ET AL., 2004):

“Um *Web Service* é um sistema de software projetado para permitir interoperabilidade entre máquinas que interagem em uma rede. Ele tem uma interface descrita em uma linguagem que é entendida por máquinas (especificamente WSDL). Outros sistemas interagem com o *Web Service* da maneira definida em sua

descrição usando mensagens SOAP (Simple Object Access Protocol), tipicamente encapsuladas usando HTTP (HiperText Transfer Protocol) com uma serialização XML (eXtensible Markup Language) em conjunto com outros padrões relacionados com a *Web*.”

Tal definição é completa e bastante objetiva, atribuindo para cada função um padrão, definido pela *World Wide Web Consortium* (W3C). A definição não estabelece um protocolo para transporte apesar de mencionar o HTTP.

Por ter o seu foco principal na interoperabilidade entre sistemas utilizando padrões estabelecidos pelo W3C, *Web Services* tem sido extensamente utilizado para a integração de sistemas.

Como se pode perceber na descrição, *Web Services* se fundamenta basicamente em dois padrões, um específico para descrever as funcionalidades do *Web Services*, o *Web Services Description Language* (WSDL)(CHINNICI ET AL., 2004), e outro para submeter requisições e receber resultados (SOAP)(GUDGIN ET AL., 2002).

Cada vez mais os *middlewares* estão utilizando *Web Services* para troca de informação com os clientes, oferecendo uma interface e implementando-a como *Web Services*. Os *middlewares* para *grade* têm seguido esta tendência como descrito neste trabalho.

2.3 Grades

A idéia de grades(FOSTER ET AL., 2001) surgiu na década de 90, tendo como principal foco a integração de recursos computadorizados distribuídos geograficamente, a fim de se formar um sistema que promova uma visão de um grande sistema centralizado, para seus usuários. O cumprimento desta tarefa esbarra na complexidade que envolve o desenvolvimento de sistemas distribuídos.

Para melhor exemplificar o funcionamento de uma *grade*, a figura 1 divide os componentes participantes em camadas. A Camada de Rede é responsável por interconectar os recursos da camada superior a ela e é através desta que irão trafegar todos os dados da *grade*. Esta camada é composta por equipamentos de redes, bem como seus protocolos de comunicação.

Figura 1: Divisão em camadas de uma infraestrutura de grades.

A Camada de Recursos contemplam todos os recursos que podem se comunicar direta ou indiretamente através da rede oferecida pela camada inferior. Tais recursos podem variar desde supercomputadores até sensores. Comumente, fazem parte desta camada: supercomputadores e sistemas de armazenamento de dados.

A Camada de Middleware realiza a tarefa de coordenar e gerenciar os recursos para que os mesmo façam parte da *grade*. É função de um middleware para grade, divulgar os recursos da camada inferior para os usuários, possibilitar autenticação de usuário e garantir que o mesmo possua autorização para utilizar os recursos. Além disso deve-se oferecer recursos para transferência de arquivos, padronizar acesso a recursos com o mesmo propósito, dentre outras funcionalidades.

A última camada, por fim, é constituída por aplicações que fazem uso dos recursos através do middleware utilizado na camada inferior. As aplicações em geral fazem uso coordenado de mais de um recurso para oferecer aos seus usuários a funcionalidade desejada, além de oferecer para eles uma visão unificada do sistema.

Estas quatro camadas são a base para a construção de uma *grade*, sendo a camada de aplicação a interface de comunicação entre a *grade* e seus usuários.

2.3.1 Grades e a Necessidade de Padrões Abertos

A necessidade por padrões abertos no desenvolvimento de *middlewares* para *grades*, se dá pelo fato de que tal software deve promover a interoperabilidade entre os diferentes recursos participantes da *grade*, bem como para as aplicações que fazem uso destes recursos. Devido à heterogeneidade dos recursos que podem ser utilizados em uma *grade* computacional o uso de padrões abertos se torna uma necessidade ainda maior, pois permitirá desta forma que componentes da *grade* possam se comunicar entre si, mesmo pertencendo a fabricantes diferentes. Por não se ter essa homogeneidade de protocolos, recursos de mesmo propósito precisam de intermédio para oferecerem seus serviços sob um protocolo comum. Este é um dos propósitos dos *middlewares* de *grade*.

Com este propósito foi criada a *Open Grid Foundation* (OGF). A principal função da OGF é discutir e criar padrões abertos para dar suporte às necessidades de uma *grade* computacional. Diversos protocolos já foram aprovados, na tabela 1 está listada uma série de especificações já aprovadas e recomendadas pela OGF bem como a sua função dentro da *grade*.

Tabela 1: Especificações recomendadas pela OGF.

<i>Título</i>	<i>Data Da Publicação</i>
Job Submission Description Language (JSDL) Specification, Version 1.0	28/07/08
Distributed Resource Management Application API Specification 1.0	13/06/08
HPC Basic Profile, Version 1.0	28/08/07
JSDL HPC Profile Application Extension, Version 1.0	28/08/07
OGSA® Basic Execution Service Version 1.0	07/08/07
A GridRPC Model and API for End-User Applications	26/09/05
Distributed Resource Management Application API Specification 1.0	29/06/07

Além das especificações recomendadas, a OGF fornece documentos informativos, especificações experimentais, documentos contendo práticas comuns da comunidade e recomendações propostas. Este esforço da comunidade reforça a necessidade do uso de padrões no desenvolvimento de recursos e *middlewares* para *grades*.

2.3.2 Open Grid Services Architecture

O *Open Grid Services Architecture* (OGSA) é a descrição de uma arquitetura recomendada pelo *Open Grid Forum* (OGF) para implementação de *grades*. O principal objetivo do OGSA é suprir as necessidades de padronização na implementação e desenvolvimentos de ferramentas e *softwares* para *grades*, baseando-se por sua vez na tecnologia de *Web Services* discutida anteriormente.

O OGSA, especifica uma arquitetura padrão para as tarefas comuns em uma *grade*, como: autenticação, definição de políticas de acesso, descoberta de serviços, monitoramento dos serviços, integração de recursos computacionais, dentre outros.

Tal arquitetura define suas bases na Arquitetura de *Web Services* (*Web Services Architecture – WS-Architecture*), tal escolha baseou-se no fato deste ser um padrão bem estabelecido no mercado e pelo fato de o OGF acreditar que esta adoção é o melhor caminho para uma *grade* padronizada e aceita pela indústria, já que *Web Service* já se encontram bem estabelecido.

Com isso a OGF ao liberar a versão 1.0 do OGSA, define o *WS-Architecture* como sendo o *framework* para se formar as bases de uma *grade* padronizada. Para isso define-se que serviços, para serem utilizados na *grade*, devem ser descritos conforme o padrão *Web Services Description Language* (WSDL) e manipulados através do protocolo *Simple Object Access Protocol* (SOAP). O OGSA está comprometido no desenvolvimento de serviços que estejam em conformidade com a especificação do *WS-Interoperability*.

O grupo do OGSA também se encontra fortemente envolvido no desenvolvimento de padrões como o *WS-Security* e outros padrões que estão sendo estabelecidos no

contexto do *WS-Architecture*. Esse comprometimento torna o OGSA uma arquitetura que de fato está promovendo a interoperabilidade entre sistemas dentro de um grid, independentemente do distribuidor e desenvolvedor do sistema.

2.3.3 Open Grid Services Infrastructure

O *Open Grid Services Infrastructure (OGSI)* (TUECKE ET AL., 2003), descreve a arquitetura proposta pelo OGSA em um nível mais detalhado. O OGSI descreve o conceito de *Grid Services*, que são serviços compatíveis com *Web Services*, e de fato *Grid Services* são *Web Services*, porém a recíproca não é verdadeira.

O OGSI define extensões para o padrão WSDL, para que o mesmo possa descrever as especificidades de um *Grid Service*, definindo para isso um conjunto de esquemas XML para possibilitar o uso de serviços que mantêm o seu estado ao longo de diferentes chamadas (*stateful*). Serviços baseados puramente no padrão *Web Services* não possuem estado, sendo cada chamada tratada independentemente.

As extensões propostas visam acrescentar funcionalidades aos padrões relacionados a *Web Services* para suprir as necessidades e os requisitos de uma *grade*. As extensões se concentram em fornecer meios de criar, nomear e gerenciar o tempo de vida de um serviço, verificar o estado de um determinado serviço, adicionar notificações assíncronas sobre mudanças de estado de um serviço, gerenciar erros e gerenciar coleções de serviços. Baseado na OGSA, o OGSI detalha estas características descrevendo soluções e propondo padrões para dar sustentação às propostas.

Em um nível detalhado, o OGSI descreve em detalhes as propostas de extensão dos *Web Service*, fornecendo códigos XML contendo esquemas que devem ser seguidos ao se criar *Grid Services* que sejam compatíveis com o OGSI.

O OGSI passou a ser criticado devido a alguns fatos que surgiram após a implementação de sistemas baseado na proposta e devido à confrontação de ideias com alguns aspectos inerentes aos *Web Services*. O OGSI se tornou uma

especificação densa, contendo muitos conceitos em uma só especificação, o que o tornou complexo. As extensões das especificações de *Web Services* levou os serviços baseados no OGSi a terem incompatibilidades com muitas ferramentas desenvolvidas para suportar *Web Services*, além disso, e segundo alguns críticos o OGSi se tornou muito orientado a objetos ao utilizar *Web Services* para encapsular um recurso que deve manter um estado. Com isso foi observado uma divergência entre as tecnologias de *Grades* e de *Web Services*.

2.3.4 Web Services Resource Framework

O Web Services Resource Framework (WSRF)(CZAJKOWSKI ET AL., 2004), foi uma evolução natural do OGSi levando-se em consideração as deficiências discutidas. O WSRF divide-se em seis especificações que estão sendo desenvolvidas em conjunto com a comunidade desenvolvedora do WS-Architecture.

O WSRF é uma refatoração do OGSi, o WSRF fornece as mesmas funcionalidades, porém tende a se alinhar melhor com as evoluções do *WS-Architecture*. Para isso foram modificadas algumas sintaxes e nomenclaturas e novas especificações propostas como, por exemplo, o *WS-Notification*. As especificações definidas e utilizadas pelo WSRF são mostradas na tabela .

A partir da tabela 1 nota-se que a nomenclatura das especificações agora se alinham com a nomenclatura utilizada pelos mantenedores das especificações do *WS-Architecture*. Isso se dá devido a uma maior integração das duas equipes de padronização e o desenvolvimento mútuo dos padrões.

Tabela 2: Especificações utilizadas na definição do WSRF.

Nome	Descrição
WS-ResourceProperties	Descreve a associação de recursos que mantém estado com Web Services para produção do WS-Resources, e como elementos de propriedades publicamente visíveis de um WS-Resource são lidos, modificados e removidos.
WS-ResourceLifeTime	Permite que um requisitante destrua um WS-Resource imediatamente ou em um tempo agendado para o futuro.
WS-RenewableReferences	Toma nota de informações necessárias de uma referência WS-Addressing, para poder requisitar uma referência para um novo ponto quando a referência para o atual se tornar inválida.

WS-ServiceGroup	Criação e uso de coleções heterogêneas de Web Services por referência.
WS-BaseFault	Descreve um tipo básico de falta para reportar erros.
Família de especificações WS-Notification	Abordagem definida para notificação usando um padrão (pattern) de inscrição e publicação baseada em tópicos.

Fazendo uma breve análise da tabela 1, pode-se observar que, tais especificações, preenchem lacunas na tecnologia de *Web Services* para que seja possível o seu uso para aplicações de *grades*, por exemplo, a especificação *WS-ResourceProperties* adiciona aos Web Services a capacidade de oferecer interfaces para recursos expostos em uma *grade*, possibilitando a modificação de parâmetros deste recurso, enquanto o mesmo mantém o seu estado ao longo de várias chamadas às interfaces *Web Services*. Outra especificação importante é o *WS-LifeTime*, que possibilita a destruição ou término de um *WS-Resource* baseando-se em solicitações de aplicações clientes ou na especificação de um tempo de vida para o recurso.

De fato podemos considerar o OGSi obsoleto. O WSRF é uma evolução do OGSi e vêm sendo adotada por diversos *middlewares* para *grade*. Além disso o WSRF foi definido para que se obtivesse uma convergência das especificações para a *grade* e as especificações para *Web Services*, evitando assim a duplicação de esforços para atividades comuns.

2.3.5 Middlewares para Grades

O propósito dos *middlewares* destinados à tecnologia de *grades*, é prover acesso aos recursos, de uma maneira padronizada aos desenvolvedores e aplicações. Primeiramente, um *middleware* para *grade* deve prover meios eficazes de controle de acesso e uso dos recursos, deve oferecer também uma maneira de consultar quais são os recursos disponíveis no momento, executar tarefas e gerenciar transferência de arquivos.

Desde a criação do conceito de *Grids*, observou-se o surgimento de dezenas de *middlewares* com o propósito de oferecer as funcionalidades citadas. Dentre os

middlewares, o *Globus Toolkit 4* (GT4)(FOSTER, 2005) vem sendo o mais utilizado. O GT4 se baseia em padrões abertos e está engajado nas discussões e definições de padrões para *grades* através do OGF. O OGF por sua vez, tem como principal função discutir novos padrões para o desenvolvimento de novas funcionalidades, ou para dar suporte a funcionalidades existentes no que diz respeito a *grades*. O GT4 possui as funcionalidades básicas de um *middleware* para *grade*, autenticação/autorização, transferência de arquivos, índice de serviços, execução de jobs, e outros.

Alguns outros *middlewares* surgiram, mas nenhum deles ganhou a visibilidade do GT4. Dentre estes estão o *gLite*, o *Unicore*(ERWIN; SNELLING, 2001), o *OMII*(BRADLEY ET AL., 2006), o *Integrate* e o *GridBus*(BUYA; VENUGOPAL, 2004). Tem-se notado uma tendência, nos *middlewares* para *grades*, do uso de *Web Services* (WS)(BOOTH ET AL., 2004) como a forma de comunicação padrão(FOSTER ET AL., 2002).

2.4 Globus Toolkit 4 (GT4)

O *Globus Toolkit* (GT)(FOSTER; KESSELMAN, 1997) foi o responsável pela popularização e utilização na prática das teorias de *grids*. Iniciado em 1996 o projeto contava com poucos colaboradores, sendo que atualmente o GT possui diversos colaboradores espalhados pelo mundo, bem como dezenas de aplicações em problemas científicos e comerciais(FOSTER; KESSELMAN, 2003).

Em sua versão 1.0 e 2.0 o *middleware* ainda não contava com o suporte da tecnologia de *Web Services*, cujos componentes mais tarde foram denominados de *pré-Web Services*. Alguns deles como o *GridFTP* continuam sendo oferecidos nas versões mais novas, enquanto outros se tornaram obsoletos. O GT na sua versão 3.0 foi baseado no *Open Grid Services Infrastructure* (OGSI)(TUECKE ET AL., 2003), que trouxe a definição de serviços que mantém o seu estado ao longo de diversas chamadas, definindo operações para criar, nomear e definir o tempo de vida dos serviços utilizando os chamados *Grid Services*. O GT segue o *Open Grid Services Architecture* (OGSA)(FOSTER ET AL., 2004)(FOSTER ET AL., 2002), que define a arquitetura de *grades* baseada em *Web Services* discutida e publicada pela

OGF.

Devido as divergências entre a comunidade de *grades* e a comunidade de *Web Services*, O OGSi evoluiu para o que foi denominado *Web Services Resource Framework* (WSRF)(CZAJKOWSKI ET AL., 2004), as mudanças estão na revisão de terminologias, introdução de padrões aceitos e discutidos por ambas as comunidades, em conformidade com as especificações dos *Web Services*.

O GT tem por objetivo ser uma implementação de referência do OGSA, bem como oferecer através de um *middleware*, os recursos e funcionalidades requeridos para o desenvolvimento de aplicações baseadas em *grades*.

O GT4(FOSTER, 2005) oferece infra-estrutura de segurança, autenticação e autorização, bem como para gerenciamento de dados, execução de tarefas, e serviços de informação para acompanhamento de tarefas e indexação dos serviços oferecidos. O GT4 oferece também várias opções para execução dos serviços como núcleos para Java e Python e C, bem como bibliotecas C para o desenvolvimento de aplicações que utilizem a antiga arquitetura pré-*Web Services* do *Globus*, e ainda o *Globus XIO*.

2.5 Bancos de Dados Distribuídos

Bancos de Dados distribuídos é um antigo tópico de pesquisas na área de computação e com um alto nível de maturidade(OZSU; VALDURIEZ, 1999)(ELMAGARMID ET AL., 1999). Tal matéria aborda a distribuição e integração de dados distribuídos entre computadores interligados por algum tipo de tecnologia de interconexão.

Bancos de Dados distribuídos, devem exibir algumas características, porém nem todas elas são obrigatórias.

- ✓ Distribuição – Os dados estão distribuídos entre vários Bancos de Dados. A distribuição pode ser considerada, horizontal, onde as linhas de uma tabela são distribuídas, ou vertical, onde as colunas de uma tabela são distribuídas. Ainda sim existem propostas de distribuição mista, utilizando ambos os méto-

dos(OZSU; VALDURIEZ, 1999).

- ✓ Heterogeneidade – Os softwares que gerenciam os bancos de dados são diversos, os esquemas variam entre eles, ou seja, qualquer diferença na implementação e no projeto dos dados pode incorrer em heterogeneidade.
- ✓ Autonomia – Autonomia diz respeito à independência de cada parte integrante do sistema. Componentes podem ter seus próprios modelos, linguagens e semântica (Autonomia de Projeto). Um componente pode decidir quando deve responder requisições externas (Autonomia de Comunicação) e execuções de transações locais podem ter prioridades sobre as externas (Autonomia de execução). Além disso, cada componente pode decidir quais operações, e quais informações devem estar disponíveis (Autonomia de associação).
- ✓ Interoperabilidade – Os sistemas devem ser capazes de enviar requisições e receber repostas uns dos outros.

Várias abordagens para prover a integração de bancos de dados, considerando as características citadas foram propostas, as principais delas são: integração utilizando esquema global(BATINI ET AL., 1986), bancos de dados federados(SHETH; LARSON, 1990), linguagens para sistemas de vários bancos de dados (*Multidatabase Language*)(ELMAGARMID ET AL., 1999).

Este trabalho baseia-se na distribuição horizontal e da replicação, através do qual, linhas de uma tabela em um banco de dados relacional são distribuídas e replicadas, promovendo desta forma maior disponibilidade dos dados e a possibilidade de um maior desempenho na consultas a dados. Dentre as características a serem alcançadas estão: distribuição, autonomia e interoperabilidade entre os bancos de dados envolvidos no sistema.

2.6 OGSA-DAI

O OGSA-DAI(ANTONIOLETTI ET AL., 2005), é um middleware para integração de bancos de dados em uma infra-estrutura de grades. O OGSA-DAI define uma

interface padrão orientada a serviços para expor um banco de dados em uma grade. Hoje o OGSA-DAI está em conformidade com as seguintes tecnologias: (a) Axis, baseada no WS-Interoperability, (2) com a infra-estrutura OMII_UK, baseada no WS-Interoperability+ e (3) no Globus Toolkit 4, baseado na especificação WSRF(CZAJKOWSKI ET AL., 2004). Recentemente o OGSA-DAI foi adaptado para poder ser utilizado com os middlewares gLite(LAURE ET AL., 2006) e Unicore(ERWIN; SNELLING, 2001).

O OGSA-DAI é também desenvolvido sob a influência de diversas especificações desenvolvidas pela OGF, vários desenvolvedores e coordenadores do projeto OGSA-DAI fazem parte do desenvolvimento destas especificações, sendo as principais pertencentes à família de especificações *Web Services Data Access and Integration* (WS-DAI), desenvolvida pelo *Data Access Integration Services – Work Group* (DAIS-WG).

O OGSA-DAI, possibilita que um usuário da *grade*, realize as seguintes operações:

- ✓ Descubra os Bancos de Dados expostos;
- ✓ Consulte bancos de dados expostos usando uma interface única, independentemente de qual SGBD o banco de dados foi desenvolvido.
- ✓ Permite consulta simultânea a vários bancos de dados, unificando os resultados, porém a *query* enviada pelo cliente deve poder ser executada sem erros em diferentes SGBDs.
- ✓ Requisitar meta informações acerca dos bancos de dados expostos, como por exemplo, esquemas lógicos de bancos de dados relacionais, informações acerca das funcionalidades de um determinado banco de dados exposto, além de permitir que informações adicionais sejam embutidas pelo administrador.
- ✓ Permite definição de destinos para os resultados da consultas.
- ✓ Operações podem ser feitas de forma assíncrona, podendo o usuário ler as informações quando necessário.
- ✓ O OGSA-DAI permite que os dados obtidos sejam enviados para outro serviço, ou armazenados em um outro recurso. Os dados podem, ao invés de

retornar para o cliente que os solicitou, ir diretamente para um outro serviço de forma passiva, ficando os dados disponíveis para que o serviço de destino use-os quando necessários. Outra opção é o uso da forma ativa, onde os dados são imediatamente enviados para o destino e devem ser consumidos imediatamente.

- ✓ Que o usuário da *grade* seja mapeado para usuários dos SGBDs.

Figura 2: Dois cenários de uso do OGSA-DAI. Exposição simples de Banco de Dados (a), exposição de vários bancos de dados com integração utilizando *ResourceGroup* (b).

Com essas características, o OGSA-DAI possibilita a construção de aplicações que necessitem de integração de Bancos de Dados a uma infra-estrutura de grades. A figura 2 mostra a arquitetura por trás do OGSA-DAI, em sua versão 3.0 (HONG ET AL., 2007), e as tecnologias utilizadas pelo mesmo. Pode-se, a partir da Figura 2, notar as diferentes camadas e como elas se relacionam até a aplicação cliente. O OGSA-DAI faz o controle de acesso ao banco de dados exposto e apresenta estes dados sob a forma de um Data Resource. Estes dados podem ser consumidos pelos

usuários do sistema utilizando o `DataRequestExecutionResource` (DRER).

A partir da infra-estrutura oferecida, pode-se concretizar cenários de integração de diversos bancos de dados, permitindo consultas sob múltiplos bancos de dados autônomos, ou a construção de bancos de dados paralelos habilitados para a *grade*. Para que se possa utilizá-lo no presente trabalho, é necessária a adição de novas funcionalidades, através da escrita de código sobre a infra-estrutura oferecida pelo OGSA-DAI.

O OGSA-DAI oferece diversos pontos de extensibilidade (“OGSA-DAI Documentation. <http://www.ogsadai.org.uk/documentation/>”). Por pontos de extensibilidade, são referidos os recursos oferecidos pelo middleware para a personalização do mesmo, tanto do ponto de vista de configurabilidade, como também da personalização através da programação de novas funcionalidades.

A integração de dados, deve portanto, implementar os recursos necessários, estendendo as funcionalidades oferecidas pelo OGSA-DAI. Dessa forma podemos criar soluções para Bancos de Dados paralelos e distribuídos a partir desta tecnologia.

Abaixo discutir-se-á os pontos de extensibilidade mais importantes e os utilizados neste trabalho.

2.6.1 Estendendo as Funcionalidades do OGSA-DAI

A adoção do *middleware* para o trabalho proposto foi decidida com base no seu comprometimento com os padrões, sua popularidade e por fim a oferta de pontos de extensão. Estender as funcionalidades para adaptar o *middleware* às necessidades é um dos pontos chaves da tecnologia por trás do OGSA-DAI. O OGSA-DAI possui diversos pontos que permitem estender as suas funcionalidades. São apresentados em detalhes, os mais importantes, dentre eles: *activities*, *resources*, *Login Providers* e *Presentation Layers*.

Atividades (Activities)

Activities são interfaces oferecidas para manipulação de recursos expostos na *grade*, as quais aplicações clientes podem utilizar. As *activities* em geral, recebem dados e retornam os resultados para o cliente. A *activity* executa uma série de instruções afim de obter o resultado que deve ser retornado. Para isso ele manipula um recurso que é especificado pelo cliente ao definir a requisição.

A sua principal função dentro do *middleware* OGSA-DAI é oferecer aos clientes uma interface para manipular recursos expostos pelo OGSA-DAI, como bancos de dados relacionais ou sistemas de arquivos. No OGSA-DAI, toda *activity* está ligada a um recurso específico.

Activities a serem executadas no OGSA-DAI são explicitadas na requisição e cada *activity* é mapeada para uma classe Java, que faz o processamento de seu conteúdo. Portanto o desenvolvimento de tarefas específicas, por exemplo, a replicação, pode ser feito diretamente na infra-estrutura oferecida pelo OGSA-DAI criando-se novas *activities* e relacionando-a a um recurso.

O desenvolvimento de novas *activities* é bastante útil quando se deseja controlar melhor o processo de manipulação de um determinado recurso, manipular os dados de entrada antes de utilizá-los, ou por exemplo, formatar a saída antes de enviá-las de volta para o cliente que realizou a solicitação. Neste trabalho fez-se uso intenso desta ferramenta.

Resources

Resources no OGSA-DAI são abstrações para representar um determinado recurso oferecido pelo mesmo. Um exemplo concreto de um *resource* são os bancos de dados relacionais.

Ao desenvolver um novo *resource*, define-se mecanismos para a instanciação destes, como por exemplo, a instanciação de uma conexão com um banco de dados, que pode ser considerado um *resource*. No OGSA-DAI quando um recurso é exposto, informa-se alguns dados sobre ele, sendo então, o novo *resource* criado sob os moldes de base pré-definida. Quando se deseja expor um banco de dados relacional, define-se este novo recurso como sendo do tipo *RelationalResource*.

Um *resource* geralmente é exposto através das *activities*, que são as interfaces para manipulação do *resource* por parte do usuário. Um exemplo disso são as *activities* *SQLQuery* ou *SQLUpdate* que manipulam um *resource* que expõe um banco de dados relacional.

O OGSA-DAI permite definir novos *resources* e pode também expor outros tipos de recurso além dos já oferecidos, por exemplo, poder-se-ia expor um telescópio, para isso deve-se programar um *resource* que representaria o telescópio, e depois definir *activities*, que seriam as operações que se poderiam realizar sobre aquele *resource*.

Provedores de Autenticação (Login Providers)

Os *Login Providers* permitem que se crie formas personalizadas de autenticação. O OGSA-DAI fornece um *login provider* padrão. O *login provider* padrão para bancos de dados relacionais expostos com o OGSA-DAI, mapeia um usuário da *grade* para um usuário do banco de dados exposto. Neste trabalho utilizamos o *login provider* padrão.

Camadas de Apresentação (Presentation Layers)

Esta opção de extensão foi desenvolvida para desacoplar o OGSA-DAI da tecnologia utilizada para a comunicação entre clientes e softwares. O OGSA-DAI possui cinco *presentation layers* implementados, possibilitando a utilização do OGSA-DAI com os *middlewares* Globus Toolkit, Apache Tomcat (com Axis 1.4 ou 1.6), OMII-UK, gLite e Unicore.

Dado que os *middlewares* listados possuem diferenças na forma com que se comunicam com os clientes, é necessário o desenvolvimento de diferentes *presentation layers*.

Presentation layers, são especialmente úteis para adaptar o OGSA-DAI para novas plataformas ou infra-estruturas de *grade*. Neste trabalho utilizamos o OGSA-DAI implantado dentro da infra-estrutura do *Globus Toolkit*, cujo *presentation layer* é utilizado.

2.7 Replicação de Dados

Replicação de dados é um tópico de pesquisa maduro e necessário em diversas áreas da computação, especialmente Sistemas Distribuídos e Bancos de Dados Distribuídos.

Os principais aspectos da replicação de informações é a manutenção da sincronia entre as várias réplicas, o controle de concorrência e o aspecto transacional das operações, especialmente durante operações de escrita.

Controle de concorrência diz respeito ao controle de acesso aos dados, quando dois ou mais usuários tentam escrever ou modificar e ler dados no sistema. Em (CAREY; LIVNY, 1988), realizou-se um estudo sobre o estado da arte na área de controle de concorrência para bancos de dados distribuídos debatendo as principais técnicas e suas variações.

Abordagens para controle de concorrência baseiam-se ou se derivam de dois principais algoritmos, são eles: bloqueio em duas fases (two phase locking) e ordenação baseada em carimbos de tempo (timestamp ordering)(GRAY; REUTER, 1993). Esses tipos de controle estão fortemente relacionados com as transações, e são elas quem garantem o caráter atômico às operações.

Componentes conhecidos como Gerenciador de Transações, são responsáveis por obter os recursos necessários para a execução da tarefa que está prestes a ocorrer, e conseqüentemente bloqueá-los para acesso exclusivo em caso de conflitos durante o acesso, para isso ele utiliza os algoritmos de controle de concorrência.

Um problema inerente a sistemas de replicação é a difícil tarefa de manter as réplicas criadas em sincronia. Modificações em uma das réplicas devem ser refletidas nas outras.

Em estudos realizados(WIESMANN ET AL., 2000), definiu-se um arcabouço para estudos de estratégias e protocolos de replicação. Devido a diferente natureza de problemas que envolvem replicação, tal arcabouço define um problema abstrato no que diz respeito a operações de replicação. Um exemplo, dessas diferenças é a aplicação de replicação em Sistemas Distribuídos, e em Bancos de Dados. O

primeiro faz uso extensivo de dois conceitos, replicação síncrona e assíncrona enquanto que o segundo se baseia em sua maioria no uso de travas, o que não é muito utilizado em Sistemas Distribuídos.

Tal arcabouço define cinco fases que podem ocorrer em um protocolo de replicação, são eles:

- ✓ Requisição – ato de submeter uma operação para uma ou mais réplicas.
- ✓ Coordenação de Servidores – Os servidores contendo as réplicas devem coordenarem-se afim de executarem a operação.
- ✓ Execução – A operação é executada nos servidores contendo as réplicas.
- ✓ Resposta – O resultado da operação é retornada ao cliente.

Pode-se definir os protocolos de replicação para bancos de dados tomando alguns parâmetros (GRAY ET AL., 1996). Propõe-se dois parâmetros, um em relação à propagação das réplicas e outro em relação a quem pode realizar as operações de atualização. A tabela Error: Reference source not found exemplifica a relação entre estes dois parâmetros

Abordagens *gulosas* são executadas em um contexto de transação, usuários não recebem a resposta até que o sistema tenha certeza de um determinado número ou todas as réplicas foram atualizadas. Abordagens *preguiçosas* permitem um certo nível de inconsistência no sistema em troca de tempo de resposta, permitindo que atualizações sejam feitas localmente e propagadas depois.

A abordagem *Atualização na Cópia Primária* permite inicialmente a atualização em uma cópia (a cópia primária ou mestra) e só então a modificação é propagada. *Atualização em Qualquer Lugar* por outro lado permite a atualização de qualquer cópia, aumentando o desempenho no acesso em detrimento da simplicidade de coordenação e sincronia das cópias.

2.8 Java Database Connectivity (JDBC)

O padrão JDBC (ELLIS ET AL., 2006) foi estabelecido pela *Sun Microsystems* com a

finalidade de estabelecer uma interface padrão para acesso a bancos de dados a partir da linguagem Java. Para que um SGBD possa ser acessado através da linguagem Java utilizando a tecnologia JDBC, cada distribuidor de SGBD deve implementar as interfaces, criando assim o que o padrão define como um *Driver* JDBC.

A Máquina Virtual Java (Java Virtual Machine – JVM), oferece um gerenciador de *drivers*, chamado *DriverManager*, que é responsável por verificar quais *Drivers* estão instalados, e baseado na requisição feita no código, retornar uma conexão com o SGBD desejado utilizando o *Driver* correto.

2.9 Uso de Padrões

Neste trabalho é feito o uso sistemático de padrões. As principais razões para isso são descritas nas subseções que seguem.

2.9.1 Motivação para o uso de Padrões

A utilização de padrões é recomendada pelos conceitos de Sistemas Distribuídos e eles garantem um maior nível de interoperabilidade e facilidade de manutenção, proporcionando assim um tempo maior de vida do software.

Ficou estabelecido após os estudos realizados neste trabalho, que a utilização de padrões é uma necessidade para a integração de recursos em uma *grade* computacional e Bancos de Dados não é uma exceção a essa regra.

O motivo para tal escolha é que, além das vantagens já citadas, a utilização de padrões permitem que novos recursos e serviços sejam inseridos na *grade* de uma forma mais suave, com pequenas ou nenhuma modificações do *middleware* utilizado.

Se um novo produto é desenvolvido e segue os padrões estabelecidos, ele conseguirá se comunicar com os componentes que seguem o padrão da *grade* sem que modificações sejam realizadas em seu código.

Os problemas que ocorrem envolvendo diferentes *middlewares* desenvolvidos com o mesmo propósito, se dá devido ao fato de cada empresa, ao desenvolver o *middleware*, definir o seu próprio conjunto de mensagens e protocolos. Este fato se dá a partir da necessidade de se obter um determinado nível de desempenho e/ou funcionalidade que determinado protocolo padrão não contempla. Por essa razão, em termos de interoperabilidade e padronização, tais ferramentas não são satisfatórias.

Para o presente trabalho, fez-se a escolha do *middleware* de integração baseado no compromisso do mesmo em especificar e seguir padrões completamente abertos ao público. Por estas razões escolhemos o OGSA-DAI e o GT4.

2.9.2 Protocolos Padrões na Comunicação e Troca de Mensagens

Os padrões utilizados para a troca de mensagens no GT4 é o *Simple Object Access Protocol* (SOAP). A especificação define um formato padrão para a execução de procedimentos remotos, SOAP-RPC, ou a requisição de documentos, *Document-centric* SOAP. O protocolo é baseado em XML.

O OGSA-DAI por sua vez define quais documentos devem ser trocados dentro de uma mensagem SOAP, padronizando assim a representação dos dados consultados, bem como das operações que podem ser realizadas no recurso exposto, no caso um Banco de Dados relacional.

O OGSA-DAI apesar de trabalhar em conjunto com o *Open Grid Forum* (OGF) no grupo de trabalho para especificação de padrões para a integração de dados em *Grid*, grupo este chamado de *Data Access and Integration Services - Working Group* (DAIS-WG), o *middleware* ainda não está em conformidade com os padrões e protocolos especificados pelo DAIS-WG, porém o OGSA-DAI pretende estar em conformidade com os padrões ainda este ano.

Além dos protocolos citados, fez-se uso do protocolo *Lightweight Directory Access Protocol* (LDAP)(YEONG ET AL., 1995) para acesso a Diretórios de Dados. O LDAP é um padrão bem aceito e em uso pela indústria e academia.

3 TRABALHOS RELACIONADOS

A integração de dados de fontes homogêneas já é uma realidade na indústria, grandes empresas como, por exemplo, a IBM, Oracle e Microsoft oferecem soluções para este tipo de problema acoplados aos seus produtos. Já a integração de dados distribuídos por fontes de dados heterogêneas não é tão comum em grandes empresas na área, algumas delas que oferecem este tipo de serviço são a IBM e a Sybase, sendo que estes são vendidos como produtos separados, e não pertencem ao pacote inicial do SGBD.

Para exemplificar melhor e discorrer sobre as funcionalidades oferecidas por estas ferramentas que se propõem a realizar integração em fontes heterogêneas, foram escolhidos dois produtos comerciais são eles: *IBM OGSA-DAI Replication* e o *Sybase Data Integration Suite*.

3.1 IBM OGSA-DAI Replication

Pesquisas vêm sendo realizadas pela IBM a fim de habilitar os seus *softwares* de replicação de Bancos de Dados para a *grade*(CHEN, 2008). Estudos recentes culminaram em um protótipo que habilita o uso do DB2 Data Replication no *Grid*. Isso corrobora com a tese de que é necessária a utilização de padrões de interoperabilidade entre recursos computacionais, neste caso os Bancos de Dados.

Os autores da proposta oferecem um arcabouço para a replicação de Bancos de Dados em *grades*(CHEN ET AL., 2007), tal trabalho foi realizado paralelo ao trabalho apresentado e os resultados mostram uma abordagem ligeiramente diferente, sendo que o primeiro oferece um arcabouço para replicação em *grids* e o segundo uma *arquitetura* de integração. Os trabalhos diferem no foco dado à replicação dos dados.

3.2 Sybase Data Integration Suite

Esta ferramenta é bastante completa, oferecendo integração através de uma federação de bancos de dados, bem como permitindo replicação dos dados, análise, *Extraction-Transform and Load* (ETL), processo utilizado para povoamento de *Data Warehouses* e suporte a eventos de tempo-real entre as bases de dados.

No entanto, a ferramenta é fechada, assim como a solução da IBM. As duas ferramentas possuem preços elevados, e não utilizam padrões de interoperabilidade com padrões de mensagens abertos, o que torna difícil a interoperabilidade entre os sistemas integradores dos dados.

3.3 WS-JDBC e JDBC-WS, Tentativas do Uso de WS para Acesso a Bancos de Dados

JDBC e *Web Services* são dois padrões bem estabelecidos, o primeiro dentro da comunidade Java, e o segundo estabelecido como padrão de interoperabilidade entre máquinas. Duas tentativas de unir esses padrões foram o WS-JDBC, projeto observado no Sourceforge.net, e o JDBC-WS(DOGDU ET AL., 2006). Ambos propõem arquiteturas semelhantes, colocando um novo componente entre o cliente e o banco de dados utilizando *Web Services* para realizar a comunicação. Tais trabalhos propõem seus próprios *middlewares* de integração, sendo este desenvolvido pelos próprios autores. Isso faz com que cada um defina o seu próprio conjunto de mensagens e protocolos a serem encapsulados pelo protocolo SOAP, base dos *Web Services*.

3.4 OGSA-DAI ODBC Driver

Outra tentativa no sentido de unir padrões de programação e comunicação, foi a de unir o OGSA-DAI, que por sua vez utiliza *Web Services*, com o padrão *Open Database Connectivity* (ODBC)(JACKSON ET AL., 2005). Nesta solução usou-se o OGSA-

DAI como suporte, sendo ele o *middleware* responsável por transportar requisições do usuário para o banco de dados a ser acessado.

Este trabalho hoje está inativo e o desenvolvimento da ferramenta estagnado. Sendo que não existe mais previsão para retomada dos trabalhos.

4 ARQUITETURA PARA INTEGRAÇÃO DE BANCOS DE DADOS EM GRADE COM SUPORTE A REPLICAÇÃO

Nesta seção serão apresentadas as propostas deste trabalho, para a problemática da integração de dados utilizando a tecnologia de *Grids*. A proposta principal é o desenvolvimento de uma arquitetura para integração de Bancos de Dados autônomos e distribuídos, o sistema também deverá suportar replicação.

Além da proposta de desenvolver a arquitetura, também foi proposto o desenvolvimento de um protótipo a partir da extensão do *middleware* OGSA-DAI.

O sistema proposto é completamente distribuído, os seus componentes podem ser instalados em diferentes máquinas dispersas através de uma rede de computadores global ou local. Apesar de completamente distribuída a arquitetura tem por objetivo, também, ser utilizada de uma forma hierárquica.

A arquitetura é flexível e configurável no que diz respeito à criação das réplicas e acesso aos dados, pode-se definir preferências por determinados computadores no momento de se replicar os dados, bem como obrigar que o sistema crie as réplicas em determinados computadores, desta forma tem-se a possibilidade de definir uma infra-estrutura hierárquica como proposto.

Uma vez definida a arquitetura, iniciou-se a fase de implementação dos componentes adicionais, que permitem o uso das funcionalidades e vantagens da solução proposta.

4.1 Objetivos da Arquitetura

Manter um ambiente completamente distribuído também é um dos objetivos deste trabalho. Cada componente pode estar em uma máquina diferente, para realizar as tarefas de recebimento e envio de dados, de e para os clientes. Porém a utilização do mesmo na aplicação discutida nesta dissertação se dará de forma hierárquica,

cujos benefícios são discutidos neste trabalho.

O propósito de tornar o sistema configurável é permitir a definição de regras na criação das réplicas, seja por motivos de desempenho, como pelo fato de uma máquina estar mais próxima, ou motivos de segurança, por exemplo, dados são replicados em máquinas mais confiáveis e com regras de segurança mais rígidas.

A arquitetura proposta foi desenvolvida para oferecer suporte a replicação no *middleware* OGSA-DAI, a fim de se obter um maior nível de disponibilidade, confiabilidade e segurança dos dados.

Como uma forma de oferecer um ambiente de programação familiar para desenvolvedores, decidiu-se por implementar um *Driver* JDBC, para fornecer aos usuários a capacidade de utilizarem a infra-estrutura proposta neste trabalho. O *Driver* mencionado é o componente denominado *Client*.

4.2 Descrição da Arquitetura

A partir da problemática discutida, este trabalho consistiu em desenvolver uma arquitetura para a integração de dados, visando a interoperabilidade através do uso dos padrões discutidos.

A arquitetura proposta é flexível o bastante para possibilitar a distribuição completa dos seus componentes, bem como permitir outros tipos de arquitetura através de configurações, como é o caso da configuração para uma arquitetura hierárquica.

Figura 3: Componentes do sistema, suas dependências e interfaces.

4.2.1 Componentes do Sistema

São cinco os componentes básicos do sistema proposto, a figura 3 mostra quais são eles e quais são as suas interfaces e dependências, tais componentes são:

- ✓ *Client*
- ✓ *Data Dictionary/Directory*
- ✓ *Data Handler*
- ✓ *DataBase*
- ✓ *Configurator*

Nas subseções abaixo detalhar-se-á cada um destes componentes.

4.2.2 Componente Client

É o componente responsável por fazer com que o cliente possa utilizar o sistema, o usuário do sistema deve montar um conjunto de operações, na forma de um fluxo de

trabalho (*Workflow*), e submetê-las para execução. Além disso o cliente deve saber lidar com as respostas enviadas de volta como resultado das solicitações enviadas para o sistema.

Baseado nas informações fornecidas pelo usuário, o componente *Client* deve criar um fluxo de atividades a ser executado no recurso definido, tal fluxo deve terminar com o retorno de dados solicitados, nos casos em que os mesmos existam, caso a operação não receba dados do sistema para serem manipulados, o sistema deverá retornar um valor especificando o estado final da execução (falha ou sucesso), o qual deverá ser usado pelo componente *Client* para tomar as decisões sobre como proceder após a operação.

Mapeamento para a Tecnologia Escolhida

O componente *Client* é uma implementação do *Java Database Connectivity* (JDBC), versão 3.0, utilizando a biblioteca de desenvolvimento do OGSA-DAI, também versão 3.0.

A escolha por esta tecnologia se baseia na familiaridade de programadores Java com a tecnologia JDBC, sua facilidade de uso e a possibilidade de habilitar facilmente programas legados para o *Grid*. Outro fato determinante para esta escolha é o fato dos *middlewares* para *grades* oferecerem bibliotecas de programação escritas na linguagem Java.

4.2.3 Componente Data Dictionary

Este componente é responsável por manter as informações sobre a localização das réplicas geradas pelo sistema. Sempre que o usuário executar uma operação com suporte a réplicas, em caso da não existência de uma réplica para os dados sendo manipulados, o sistema deve criá-las. Caso existam réplicas, deve-se atualizar as informações acerca da localização das mesmas. Em caso de informações removidas, as informações no *Data Dictionary* devem também ser removidas.

As informações contidas neste componente, como também aquelas sobre a

localização das réplicas, devem poder ser replicadas e/ou distribuídas. Dessa forma o sistema se mantém sem pontos de centralização.

Mapeamento para a Tecnologia Escolhida

Para esta tarefa escolheu-se o OpenLDAP, um sistema de diretórios que utiliza o padrão LDAP para manipulação dos dados nele contido.

O OpenLDAP oferece suporte à replicação dos dados armazenados por ele, bem como suporte à distribuição dos dados contidos nos diretórios. Essas características estão de acordo com os objetivos do projeto, de ser um sistema totalmente distribuído.

4.2.4 Componente DataHandler

O componente *DataHandler* é o mais complexo dentre os componentes do sistema, sua principal função é a de receber as requisições submetidas pelo componente *Client* e coordenar a sua execução. Uma vez recebida a requisição, ele deve executar as operações especificadas na mesma, contando com o auxílio dos outros componentes do sistema. Uma vez executada todas as operações desejadas o componente envia ao cliente os resultados obtidos.

Mapeamento para a Tecnologia Escolhida

Para o papel do processador de requisições utilizou-se o GT4 acrescido do OGSA-DAI 3.0. O GT4 além de ter a funcionalidade de encaminhar as requisições para os devidos componentes, ele também será responsável por toda a segurança do sistema.

As requisições para execução das operações chegam a este componente como mensagens HTTP contendo em seu corpo uma requisição SOAP, o papel do GT4 é o de averiguar o serviço responsável pelo processamento da requisição e repassá-la.

Neste caso as requisições do componente *Client* são repassadas para o OGSA-DAI, que reside dentro do GT4. O OGSA-DAI foi o *middleware* escolhido para lidar com a distribuição dos bancos de dados, o mesmo executa o fluxo de trabalho e se

comunica com os bancos de dados e o componente *DataDictionary*. A comunicação com o dicionário de dados está condicionada ao desenvolvimento de uma nova operação a ser inserida no OGSA-DAI.

4.2.5 Componentes Databases

Os *Databases* ou Bancos de Dados, são na realidade sistemas independentes que são integrados ao sistema em questão. Pode haver mais de um banco de dados, e os mesmos podem estar dispersos em diferentes localidades. O papel deste componente é ser um recurso onde dados são escritos e consultados. Integrá-los de maneira que o usuário manipule-os como se fossem um único banco é tarefa do *DataHandler*. Diversos Gerenciadores de Bancos de Dados podem ser utilizados, os quais são listados na tabela 3 a seguir:

Tabela 3: Bancos de dados compatíveis pelo OGSA-DAI, e consequentemente pelo protótipo apresentado. A tabela lista as versões testadas, porém outras versões podem vir a funcionar. Fonte: OGSA-DAI Documentation (<http://www.ogsadai.org.uk/documentation>)

SGBD	Versão
MySQL	5.0.15
IBM DB2	8.1 FixPack 5 (Express-C)
Microsoft SQL Server	2000
Oracle	10g Enterprise Edition Release 10.2.0.1.0 - Production
PostgreSQL	8.1.4
OGSA-DAI resource group	Bancos de dados suportados agrupados como um único recurso.

4.2.6 Componente Configurator

Este componente tem por objetivo manter as configurações dos componentes da arquitetura. Ao implementar uma infra-estrutura baseada na arquitetura proposta, esta deve ser configurável, e essas configurações devem ser disponibilizadas e compartilhadas com os componentes do sistema.

Alguns exemplos de configuração são: quais bancos de dados devem receber réplicas e de quem, quais computadores podem ser consultados e quais são os

endereços dos dicionários de dados.

O *Configurator* pode manter mais de uma configuração, diferentes configurações recebem identificadores únicos. Ao consultar o *Configurator* informando a identificação da configuração solicitada, as informações correspondentes são retornadas. Desta forma podemos manter diferentes configurações, o que torna a arquitetura ainda mais flexível.

Na tabela abaixo estão definidas as propriedades que podem ser especificadas em uma configuração.

Tabela 4: Propriedades obrigatórias para a configuração do protótipo.

Chave	Valor
configurator.url	URL do componente configurator que irá enviar as configurações, caso o valor seja a string "local", os valores da configuração local serão levadas em consideração.
configuration.id	Identificador único da configuração.
replication.site	Recurso onde irá replicar as operações realizadas localmente.
dictionary.url	URL do dicionário de dados onde as informações sobre as réplicas serão adicionadas.
dictionary.handler.class	Classe que irá gerenciar o dicionário de dados.
replica.manager.class	Classe que irá gerenciar a criação de réplicas.

4.3 Detalhes do Componente *DataHandler*

Por ser um componente complexo, os detalhes do *DataHandler* serão discutidos com mais ênfase, a figura 4 mostra os detalhes internos deste componente e como suas interfaces se relacionam com os subcomponentes internos ao *Datahandler*.

O componente possui uma interface de comunicação com o cliente chamada *OperationSet*, que corresponde a um conjunto de operações escolhidas pelo cliente

para o componente responsável (*DataHandler*), que processa o fluxo de trabalho, verificando se há uma necessidade de replicação ou não e toma as devidas providências para replicar ou não os dados enviados na requisição.

Continuando a análise da figura 5, na etapa 1, a requisição é feita através do protocolo SOAP contendo as informações das operações a serem executadas, tais informações são geradas no componente *Client*. O componente *DataHandler* verifica o destino da mensagem e a repassa para o serviço responsável, neste caso, o subcomponente *OperationSetExecutor*. Na etapa 2 o *OperationSetExecutor* processa a requisição verificando quais tarefas deve executar, e dependendo do tipo de requisição, ele irá solicitar a replicação (etapa 3). É então feito o acesso ao Dicionário de Dados (etapa 4) e aos Bancos de Dados através dos subcomponentes *ReplicaManager* e *DictionaryHandler*, que então irá retornar o resultado da operação para o *OperationExecutor*, que por sua vez repassará para o cliente através das etapas 6 e 7.

Figura 5: As etapas para acesso e modificação dos dados. O acesso ao diretório de dados está condicionado à identificação do tipo de operação.

4.5 Utilizando a Arquitetura de Forma Hierárquica

Quando se menciona o termo Hierarquia no sistema em questão, o mesmo refere-se a uma hierarquia no que diz respeito à criação de réplicas. Na arquitetura

proposta isso é conseguido por meio da configuração do sistema.

Solicitações de replicações advindas de determinados computadores devem ser feitas em recursos

previamente determinados

e estes por sua vez podem

também solicitar que uma

outra réplica seja feita, se

configurado para isso. A

figura 6 ilustra esse

conceito. A seta em negrito

indica uma replicação

sendo solicitada pelo *host4*. Note que o sistema realiza a réplica obrigatoriamente no *host1*, que por sua vez está configurado para replicar as informações no *host0*.

O sistema está sendo desenvolvido de forma que, se possa especificar os recursos onde se devem escrever as réplicas.

Figura 6: Replicando dados de forma hierárquica. Registros do nível mais inferior são replicados em níveis superiores. Atualizações se propagam da mesma forma.

4.5.1 Benefícios no Uso de Hierarquias de Replicação

O uso de hierarquias de replicação se torna bastante interessante quando se tem nós contendo informações não redundantes nos níveis mais baixo da hierarquia. Por exemplo, as informações operacionais de um sistema de saúde estariam armazenados nos níveis mais baixos da hierarquia e só pode replicar os seus dados em recursos que estejam um nível acima na hierarquia. Os dados operacionais, dados de pacientes por exemplo, estão localizados na unidade onde ele foi atendido pela primeira vez, os dados nunca estariam em outras unidades no mesmo nível hierárquico.

Uma vez cadastrado, o registro do paciente só pode ser replicado em um nível superior da hierarquia, onde o computador que está sendo utilizado para manter a réplica também é livre para replicar essa informação, porém apenas para um outro computador acima dele na hierarquia.

A vantagem da abordagem proposta é a de que eliminamos a concorrência. Os

dados são mantidos atualizados do nível mais baixo da hierarquia para o nível mais alto e as atualizações das réplicas se propagam na direção mostrada na figura 6.

Uma desvantagem desta abordagem é que dados não podem ser replicados em computadores que estão no mesmo nível ou em um nível inferior no qual o registro foi gerado.

Contudo a hierarquia de replicação se encaixa muito bem em algumas aplicações, por exemplo, no estudo de caso do sistema de saúde. A utilização de uma hierarquia também aumenta a segurança ao acesso aos dados em comparação a uma solução completamente distribuída.

5 Java Database Connectivity for OGSA-DAI

Como parte deste projeto, foi desenvolvido um *driver* JDBC(WHITE, S.; HAPNER, 1999) para fornecer uma forma fácil de acessar os bancos de dados expostos no *grid*, através da tecnologia Java. Esta seção apresenta aspectos técnicos do *driver* e os problemas que ele se propõe solucionar.

5.1 Suporte ao Globus Toolkit e ao Apache Axis

Como mencionado anteriormente, o OGSA-DAI pode ser utilizado tanto com o GT4 quanto com o *Apache Axis* utilizando o *Apache Tomcat*. Os pacotes de desenvolvimento do OGSA-DAI são separados com base na infra-estrutura utilizada para executar o OGSA-DAI, por exemplo GT4 ou *Apache Axis*. Este fato foi identificado como um problema, visto que para desenvolver um *driver* para um middleware de *grade* deve-se utilizar pacotes específicos, então, por exemplo, um *driver* desenvolvido para GT4 não funcionará com o *Apache Axis*. A partir deste problema foram criados diferentes processos para compilar e montar os *drivers* baseados na tecnologia utilizada pelo OGSA-DAI.

Ao compilar o *driver* duas versões são geradas: uma para o GT4 e outra para o *Apache Axis*. Desta forma permitimos que o usuário possa usar o *driver* JDBC em qualquer uma das situações.

5.2 Implementação

A *Sun Microsystems*, desenvolvedora do Java, publicou uma série de interfaces que devem ser implementadas por cada distribuidor de SGBDs a fim de que aplicações escritas nessa linguagem utilizassem uma sintaxe comum para acesso aos dados. A principal destas interfaces é a interface *Driver*. A classe que implementa a interface *Driver* fornece acesso ao banco de dados, funcionando como uma fábrica de

conexões. Por serem implementados sobre a mesma interface, o código-fonte escrito pelo cliente para acesso aos dados é sempre o mesmo, independente do SGBD utilizado.

Figura 7: Diagrama de Classes simplificado do driver JDBC.

A figura 7 mostra as classes desenvolvidas e os relacionamentos entre elas. As interfaces foram omitidas, porém se for removido o prefixo DAI dos nomes das classes mostradas no diagrama, têm-se os nomes das interfaces implementadas. Tais interfaces são todas fornecidas pela linguagem Java.

5.2.1 Segurança

Até o presente momento, o OGSA-DAI fornece suporte de segurança apenas quando é utilizado em conjunto com o GT4, porém novas soluções de segurança podem surgir, para que o OGSA-DAI possa ser utilizado seguramente com outros *middlewares* para *grade*.

Com o intuito de fornecer a possibilidade de se incluir novos sistemas de segurança, o driver JDBC implementa um *design pattern* (padrão de projeto, em uma tradução livre) conhecido como *Factory* (GAMMA ET AL., 1994). Criamos então uma fábrica (vide figura 7), *SecurityFactory*, que se responsabiliza por “fabricar” os objetos corretos com base no contexto. Cada um destes objetos devem estar em conformidade com uma interface, neste caso a interface *Security*. No momento apenas o *GlobusSecurity*, foi implementado, por ser o único a ser oferecido pelo OGSA-DAI, porém outros podem surgir como por exemplo o *AxisSecurity*.

5.2.2 Pacote Utilitário

Nota-se na figura 7 a presença do pacote ***br.usp.pcs.lahpc.ogsadai.jdbc.****, tal pacote destina-se a classes utilitárias que auxiliam no funcionamento das demais classes do *driver*. Nesta versão foi escrita a classe *DataResourceInfo*, que é responsável por analisar o recurso a ser acessado e retornar metadados a cerca do mesmo.

5.2.3 Extensões ao OGSA-DAI

O OGSA-DAI oferece metadados insuficientes para a implementação completa da interface *DatabaseMetaData* especificada no JDBC. Para que tal implementação fosse possível, foi necessária a implementação de uma *activity* para obter tais informações.

A *activity DatabaseMetaDataMethod* e *PackDatabaseMetada* foram desenvolvidas. Enquanto a primeira executa um método especificado, a segunda executa todos os métodos possíveis e retorna de uma só vez todos os dados.

DatabaseMetaDataMethod tem como entrada o nome de um método especificado pela interface *DatabaseMetaData* do JDBC bem como os seus atributos, tais informações são encapsuladas na classe *MethodInformation*, que é serializada utilizando a biblioteca Xstream(“XSTREAM <http://xstream.codehaus.org/>”) e enviada para execução no servidor que por sua vez, executa o método especificado e

retorna as informações para o cliente.

Figura 8: Diagrama de sequência simplificado exemplificando o funcionamento da activity DatabaseMetaDataMethod.

No diagrama simplificado da figura 8, pode-se verificar o funcionamento da *activity DatabaseMetaDataMethod*. Note que a classe de onde parte a requisição está sendo executada no cliente, enquanto que as outras duas são executadas no servidor. Quando recebida a requisição a *activity* solicita ao *JDBCResource* uma conexão com o banco de dados especificado, de posse da conexão ela solicita os metadados do banco de dados, tais metadados estão encapsulados em uma implementação da interface *DatabaseMetaData* do JDBC, então executa-se sobre essa classe o método requisitado pelo cliente retornando o resultado para o mesmo.

A *activity PackDatabaseMetadata* diferentemente da anterior, não possui dados de entrada, ao ser invocada a *activity* executa todos os métodos de *DatabaseMetaData* e retorna para o cliente. Esta *activity* é especialmente útil para aplicações que fazem muitas chamadas a métodos de *DatabaseMetaData*, podendo esta ter um ganho de desempenho fazendo apenas uma requisição ao invés de várias. O funcionamento desta *activity* é bastante semelhante com a *DatabaseMetaDataMethod*.

5.3 Utilização do Driver

O *driver* deve ser utilizado como qualquer outro *driver* JDBC (WHITE, S.; HAPNER, 1999), com a diferença apenas na *Uniform Resource Identifier* (URI) do banco de dados a ser utilizado, neste caso o sistema OGSA-DAI.

A versão atual se baseia na seguinte sintaxe para a URI para recursos expostos sem segurança.:

jdbc:ogsadai://caminho:porta/para/o/recurso

Primeiro especifica-se que está utilizando o *driver* OGSA-DAI, logo após o nome do computador no qual o OGSA-DAI está executando, na seqüência informamos onde estão os serviços OGSA-DAI dentro do sistema e finalmente o nome do recurso a ser utilizado.

O *driver* também oferece suporte a acesso seguro utilizando o *GSI*, infra-estrutura de segurança oferecido pelo *Globus Toolkit* 4, para isso tal informação deve ser passada na URI e a segurança deve estar configurada na infra-estrutura de *grade.*, como no exemplo abaixo:

jdbc:ogsadai:gsi//caminho:porta/para/o/recurso

Abaixo é apresentado um exemplo de utilização do driver a partir do Java.

```

1: /* ... código acima omitido... */
2: Class.forName("br.usp.pcs.lahpc.ogsadai.jdbc.Driver");
3: Connection conn = DriverManager.getConnection(
    "jdbc:ogsadai://localhost:8080/dai/services/LBB_WN00");
4: Statement stmt = conn.createStatement();
5: stmt.execute("SELECT * FROM littleblackbook WHERE id < 10;");
6: ResultSet rs = stmt.getResultSet ();
7: while(rs.next()) {
8:     System.out.println( rs.getInt("id") + "\t"
+     rs.getString("name") + "\t"
+     rs.getString("address") + "\t"
+     rs.getString("phone"));
9: }
10: /* ... código abaixo omitido ... */

```

Pode-se notar a partir do código acima, que o uso do *driver* se dá da mesma forma para os *drivers* oferecidos por fabricantes de SGBDs. Na linha 1 foi realizado o carregamento da classe do *driver*, então solicitou-se uma conexão utilizando a classe *DriverManager* que é oferecida pela plataforma Java e este irá retornar uma conexão para o banco de dados referenciado pela URI passada ao método *getConnection*. Na linha 4, um *Statement* é solicitado à conexão, este é responsável por executar uma *query* (linha 5). Os resultados da execução são retornadas na forma de um objeto *ResultSet* (linha 6). Nas linhas 7, 8 e 9, o programa faz uma leitura dos resultados e o imprime na tela.

Para fazer com que um programa existente, escrito na linguagem Java e que usa o JDBC, utilize um banco de dados exposto em uma *grade*, basta que se modifique as linhas que se referem às linhas 2 e 3 do exemplo citado.

5.4 Benefícios no Uso do Driver

A utilização do *driver* desenvolvido facilita o desenvolvimento de aplicações, para *grades* que manipulem bancos de dados. Esforços neste sentido levam a uma maior popularização da tecnologia.

Um segundo benefício, porém não menos importante, o *driver* deve possibilitar que aplicações já escritas na linguagem Java possam utilizar bancos de dados expostos na *grade*. Para isso o desenvolvedor fará poucas ou nenhuma modificação no código-fonte do seu aplicativo.

E por fim a escrita de um *driver* para esta tecnologia permite que *frameworks* implementados para trabalhar com a tecnologia java, por exemplo o *Hibernate*, poderão utilizar bancos de dados expostos na *grade*.

6 Implementação de um Protótipo Baseado na Arquitetura Proposta

A partir da arquitetura especificada foi desenvolvido um protótipo para validar a funcionalidade e a viabilidade da mesma. Esta seção apresenta os detalhes de como foi implementado o protótipo e o seu relacionamento com as tecnologias escolhidas. São descritos os detalhes de implementação dos componentes e seus subcomponentes.

6.1 Componente Client

A seção anterior mostrou detalhes do *driver* JDBC implementado dentro do escopo deste trabalho. Tal *driver* desempenha o papel do componente *Client* especificado na arquitetura. O *driver* provê uma forma simples e fácil de desenvolver aplicações que realizem acesso ao resto do sistema.

O *driver* é responsável por criar os conjuntos de operações e submetê-los para execução no sistema, tudo isso é feito sem a intervenção direta (utilizando bibliotecas do OGSA-DAI) do desenvolvedor que utiliza o *Driver*, caso ele utilize apenas as funcionalidades oferecidas na arquitetura aqui apresentada.

O JDBC permite que os desenvolvedores de *Drivers* adicionem funcionalidades ao sistema, quando em sua especificação, deixam em aberto quais informações devem ser passadas para o *Driver*, sendo que o mesmo recomenda a passagem de algumas informações comuns como senha e nome de usuário do banco de dados.

Fazendo uso deste ponto de extensibilidade do JDBC, o *driver* está sendo aperfeiçoado de forma que o cliente possa informar múltiplos bancos de dados a serem utilizados.

6.2 Componente DataHandler

O componente *DataHandler* é implementado utilizando o GT4 e o OGSA-DAI. Como mencionado, essas duas tecnologias não suportam as principais características da arquitetura proposta, que é a replicação dos dados e o gerenciamento da localização das réplicas. Para isso, foram desenvolvidas extensões para essas tecnologias de modo que o componente possa executar as tarefas atribuídas a ela.

Dentre os componentes do *DataHandler*, os subcomponentes *ReplicaManager* e *DictionaryHandler* serão criados, para isso, novas extensões ao OGSA-DAI serão implementadas para fornecer as funcionalidades destes dois subcomponentes. As funcionalidades do componente *OperationSetExecutor* é desempenhado pelo próprio OGSA-DAI através do seu mecanismo de workflow.

Para implementar os subcomponentes citados, está se utilizando o conceito de *Activities* do OGSA-DAI. Como vimos na figura 4, subseção 4.3, o *DataHandler* é composto pelos subcomponentes definidos. Cada subcomponente deste é representado como um conjunto de *activities* que foram desenvolvidos para estender as funcionalidades do OGSA-DAI.

6.2.1 Utilizando o OGSA-DAI para Implementação do DataHandler

O componente *DataHandler* recebe um conjunto de operações, o qual é denominado de *Workflow* no OGSA-DAI. Um *workflow* é um conjunto de operações, tais operações são conhecidas como *Activities* dentro do OGSA-DAI. Dentro de um *workflow*, as entradas e saídas das *Activities* ligadas umas as outras formam um fluxo de execução.

A abordagem utilizada é criar novas *activities*, fornecendo desta forma as funcionalidades que estão faltando ao OGSA-DAI para a completa implementação da arquitetura proposta.

As *Activities* desenvolvidas são descritas na figura 5 e 6 como sendo as interfaces oferecidas pelos componentes. A tabela 3 relaciona as *activities* e a qual subcomponente pertencem, detalhes destas podem ser encontradas no apêndice A.

Tabela 5: Activities do componente DataHandler e a qual subcomponente pertence.

Subcomponente	Noma da Activity
DictionaryHandler	DictionaryInsert
	DictionaryRemove
	DictionaryUpdate
	DictionaryLookup
ReplicaManager	InsertReplicate
	UpdateReplicate
	DeleteReplicate

6.2.2 ReplicaManager

O subcomponente *ReplicaManager* é representado pelas *activities* listadas na tabela 5. Essas *activities* foram implementadas para adicionar a funcionalidade de replicação a *activities* oferecidas pelo OGSA-DAI, as mesmas utilizam os bancos de dados expostos pelo OGSA-DAI como *resources*, para executar as operações solicitadas, uma peculiaridade dessas *activities* é o fato de que as mesmas verificam se é necessária a replicação da operação em outro banco de dados e o faz.

A *activity InsertReplicate* é utilizada para realizar uma inserção em um banco de dados. Se configurada para gerar réplicas, essa *activity* irá disparar a mesma operação nos bancos de dados onde deve manter as réplicas. Após feitas as réplicas, a *activity* insere novas entradas com informações de localidade das réplicas no dicionário de dados.

A *activity UpdateReplicate* é utilizada quando se quer modificar um registro, e solicitar a execução da operação nos bancos de dados que possuem réplicas do registro sendo alterado. Note que esta *activity* não escreve no dicionário de dados, porém faz uma busca para verificar os bancos de dados que possuem a informação com a finalidade de atualizar as réplicas mantidas em outros locais.

O OGSA-DAI utiliza uma *activity* denominada *SQLUpdate* para realizar tanto operações de atualização como operações de remoção e inserção de registros. No

entanto, no contexto da arquitetura proposta, isso não seria possível devido ao fato de que cada uma dessas operações invocarem de forma diferente o componente *DataDictionary*.

A *activity DeleteReplicate* faz a remoção de um registro e solicita a execução da mesma operação nos bancos de dados contendo réplicas do registro afetado. Ao remover um registro, esta *activity* deverá atualizar o dicionário de dados removendo as referências ao registro recém removido.

6.2.3 DictionaryHandler

O *DictionaryHandler* é um subcomponente do *DataHandler*, sua função é a de executar operações em um *DataDictionary*. As operações são de inserção, remoção, atualização e busca. Para oferecer estas funcionalidades foram implementadas quatro *activities*, como mostra a tabela 5.

As *activities* são utilizadas pelo *DataHandler* quando se cria ou remove uma réplica, e quando se precisa saber quem possui cópias de um determinado registro, por exemplo no caso de uma atualização, onde a mudança deve se propagar pelas réplicas existentes.

6.3 Componente Configurator

O componente *Configurator* possui duas operações que podem ser utilizadas pelos *DataHandlers*, são elas, *LoadConfiguration* e *StoreConfiguration*. Uma configuração pode ser carregada pelo sistema, tal configuração é mantida pelo administrador da *grade*, sendo que os *DataHandlers* então, são configurados para fazerem a leitura desta configuração a partir de um identificador único que é atribuído a cada configuração armazenada no componente *Configurator*.

As configurações armazenadas são descritas utilizando-se um par chave/valor, que é armazenado em um arquivo pelo componente *Configurator*. A cada configuração armazenada, atribuí-se um identificador único, o qual deverá ser informado pelos

DataHandlers para carregar as informações.

6.4 Criando uma Infra-Estrutura de Integração Baseado no Protótipo Desenvolvido

Para a implantação do protótipo desenvolvido, com o intuito de criar uma infra-estrutura de integração de dados com suporte a replicação, alguns passos devem ser seguidos. São eles:

1. Instalação do Globus Toolkit 4 ou Apache Axis 1.4
2. Instalação do OGSA-DAI no *middleware* escolhido.
3. Implantação das *activities* de suporte para o driver JDBC caso deseje usá-lo.
4. Exposição dos Bancos de Dados envolvidos na integração, através do OGSA-DAI.
5. Implantação das *activities* dos componentes do protótipo no OGSA-DAI.
6. Habilitar os Bancos de Dados expostos para a utilização das *activities* implantadas.

Todas as máquinas envolvidas na infra-estrutura devem seguir obrigatoriamente os passos de 1 a 2. O passo 3 é necessário caso se deseje usar o *driver*, porém o uso do *mesmo* é opcional, visto que cada um dos componentes possuem classes clientes que podem ser utilizadas diretamente para acessar o recurso exposto. O passo 4 também deve ser executado nos nós que terão instalados os bancos de dados. O passo 5 pode ser executado em etapas, sendo elas: A implantação do componente *Configurator*, a implantação do subcomponente *DataHandler* e a implantação do subcomponente *DictionaryHandler* ambos do componente *DataHandler*. No passo 6 devemos informar ao OGSA-DAI que um determinado usuário de um determinado recurso, no caso o banco de dados exposto, poderá fazer uso das *activities* implantadas.

Vale salientar que um nó pode ter apenas alguns componentes, por exemplo, um dos nós pode ser responsável apenas pela configuração, e portanto, ter somente o componente *Configurator* implantado. Passos detalhados juntamente com um

exemplo encontra-se disponível no Apêndice B.

6.5 Utilizando o Protótipo a Partir de Programas Java

Após a construção da infra-estrutura com base no protótipo em questão, existem duas maneiras de acessar os recursos oferecidos a partir de um programa de computador. Um requisito para ambos os métodos, é a utilização da linguagem Java.

O primeiro método para acessar a infra-estrutura é utilizando o *driver* desenvolvido, como é mostrado na seção 5, subseção 5.3. O segundo método é o acesso direto utilizando as classes clientes das *activities* desenvolvidas.

O segundo método de acesso também utiliza a biblioteca de desenvolvimento de aplicações clientes do OGSA-DAI. O método consiste em criar manualmente um *workflow* no código da aplicação que fará o acesso e submeter o mesmo para execução no OGSA-DAI. O código abaixo exemplifica o uso da *activity InsertReplicate* que faz parte do componente *DataHandler*, os blocos de captura de exceção (try e catch) são omitidos e somente o método *main* da classe de testes é apresentada.

```

1:  public static void main(String[] args) throws Exception {
2:      ServerProxy server = new ServerProxy();
3:      server.setDefaultBaseServicesURL(new
          URL("http://192.168.10.1:8080/wsrf/services/dai"));

4:      DataRequestExecutionResource drer =
          server.getDataRequestExecutionResource(
              new ResourceID("DataRequestExecutionResource"));

5:      InsertReplicate insert = new InsertReplicate();
6:      insert.setResourceID("lbb");
7:      insert.setQuery("insert into littleblackbook values(100013,
          'Jhon Doe'," +
          "'5th Avenue, 200', '333-4567')");
8:      insert.setDictInfo("dn=04556992 ou=tktktktk.com");
9:
10:     DeliverToRequestStatus delivery = new DeliverToRequestStatus();
11:     delivery.connectInput(insert.getDataOutput());

12:     PipelineWorkflow pipeline = new PipelineWorkflow();
13:     pipeline.add(insert);
14:     pipeline.add(delivery);
15:

```

```

16:         drer.execute(pipeline, RequestExecutionType.SYNCHRONOUS);
17:
18:         System.out.println(insert.getRequestStatus());
19:     }

```

A listagem acima exemplifica o uso da infra-estrutura a partir do uso da classe clientes da *activity InsertReplicate*. O primeiro passo é instanciar uma instância da classe *ServerProxy* a qual é delegada a tarefa de se comunicar com o servidor contendo o recurso a ser utilizado (linha 2). Na linha 3 informa-se o endereço do recurso a ser utilizado, neste caso o nome do recurso é *lbb*, o qual foi implantado no diretório */wsrf/services/dai* do OGSA-DAI implantado no *Globus Toolkit 4*. Na linha 4 solicita-se ao OGSA-DAI uma instância do executor de requisições. Nas linhas 5, 6 e 7, cria-se uma instância da classe cliente *activity* da *activity*, informa-se qual o recurso a ser utilizado e qual a operação a ser executada, respectivamente. Nas linhas 10 e 11, criamos uma instância de uma classe oferecida pelo OGSA-DAI que será responsável por informar sobre o status da requisição, caso necessário. Nas linhas de 12 a 14, define-se o *workflow*, informando quais *activities* deverão ser executadas, em seguida na linha 16 o *workflow* é enviado para execução.

A listagem acima representa um programa cliente OGSA-DAI comum, com o uso de uma nova *activity*, a *activity InsertReplicate*. Mais exemplos podem ser encontrados no Apêndice C.

6.6 Informações sobre o Código

Algumas informações do código desenvolvido foram extraídas, tais informações referem-se ao código das *activities* bem como do *Driver JDBC* para o OGSA-DAI, como mostra a tabela 6.

Tabela 6: Linhas de Código e quantidade de Classes implementadas durante a execução do projeto.

<i>Pacote</i>	<i>Número de Linhas de Código (LoC)</i>	<i>Número de Classes</i>
Activities	1017	26
Activities Test	153	12
Driver JDBC para OGSA-DAI	2788	21
Driver JDBC Para OGSA-DAI Testes	673	11
Activities de Suporte ao Driver	622	10
Total	5253	80

A tabela acima resume a quantidade de linhas de código escritas durante o desenvolvimento do projeto, bem como em quantas classes o sistema foi dividido. O código do *driver* encontra-se disponível publicamente na *Internet*. O código do protótipo pode ser solicitado pelo e-mail *mathiasbrito@gmail.com* e utilizados com a autorização expressa do autor deste documento.

7 Estudo de Caso: Aplicação da Arquitetura em um Sistema de Saúde para Hemofílicos

A arquitetura proposta foi aplicada no escopo de um sistema que está sendo desenvolvido na Escola Politécnica da Universidade de São Paulo. O sistema em questão visa gerenciar dados referentes ao pacientes hemofílicos dos Hemocentros de São Paulo e Campinas.

Foram realizadas, durante a definição do estudo de caso, entrevistas com médicos Hematologistas do Hemocentro do Hospital das Clínicas de São Paulo e do Hemocentro de Campinas, afim de se obter informações relevantes e pertinentes ao desenvolvimento do estudo de caso apresentado.

7.1 Motivação para o Uso da Arquitetura

Esta seção irá discutir os motivos que levaram aos estudos da arquitetura proposta em um sistema de saúde.

7.1.1 Sistemas de Informação em Saúde Pública no Brasil

Desde meados da década de setenta o governo brasileiro começou investir em sistemas de informação na saúde pública, nascendo o primeiro sistema denominado SIM (Sistema de Informação em Mortalidade) criado e desenvolvido pelo Núcleo de Informática do Ministério da Saúde (LEÃO ET AL., 2004). Desde então, as diversas esferas da administração pública vem adquirindo sistemas de informação para diversos tipos de gestão em saúde pública.

Em estudos realizados (PINTO, 2000) e (JORGE ET AL., 1993) relatam que os dados gerados pelo SIM e pelo SINASC (Sistema de Informação de Nascidos Vivo) são enviados periodicamente à Secretária de Saúde Municipal em um formato de arquivo já definido pelo sistema para que a consolidação regional seja realizada. Os órgãos locais são meros geradores de dados, salvo em algumas ocasiões onde a prefeitura gera uma cartilha com alguns dados estatísticos da região. Corroborando

esta descrição, em estudos realizados (BABOSA, 2006) descreve o SIAB (Sistema de Informação da Atenção Básica) onde os dados gerados pelo sistema são encaminhados a Secretária Municipal de Saúde, que por sua vez envia funcionários em datas predeterminadas para a coleta dos dados. Uma vez que os dados foram entregues ao órgão competente, estes não voltam para as suas unidades de origem impossibilitando análises ou discussões. Em (ALVANHAN ET AL., 2001) é mencionado o SINAN (Sistema Nacional de Agravos Notificáveis), ao qual coleta e envia os dados para uma autarquia estadual (regional ou central) que posteriormente segue para o âmbito federal, sem que haja uma análise local das informações.

No intuito de produzir uma base de dados centralizada e unificada destes diversos sistemas de informação, a partir de 1991, o SUS em conjunto com a FUNASA (Fundação Nacional de Saúde) criou o DATASUS (Departamento de Informática do SUS). Com esta ação o governo federal procurou uma forma de integrar os dados dos sistemas regionais para que as informações fossem visualizadas como um todo, para que a União pudesse alocar melhor seus recursos conforme as prioridades indicadas e montar uma estratégia de planejamento. Seguindo o fluxo que inicia nas unidades básicas de saúde, todas as instâncias administrativas de cada esfera da administração pública, analisam e os consolidam localmente para envio posterior ao gestor regional responsável até sua chegada ao Ministério da Saúde para uma consolidação nacional, conforme a indicação da Figura 9.

Figura 9: Fluxo de dados dentro da estrutura dos sistemas de saúde no Brasil.

A cultura de centralização de dados é antiga, e para o caso dos sistemas da saúde pública brasileira temos alguns fatores detectados (ALMEIDA, 1998) como:

- forte centralização da política nos anos de ditadura militar, onde alguns dos sistemas de informação foram concebidos,
- A tecnologia disponível na criação dos sistemas, onde a maioria foi baseada em *main-frames*.
- a política de que a centralização na produção de informação é uma boa estratégia para a padronização e a melhoria na qualidade das informações.

No intuito de descentralizar a informação alguns esforços foram aplicados como forma de possibilitar uma melhor gestão e definição de prioridades por parte dos demais órgãos que compõem o sistema único de saúde. Como parte deste esforço está à transposição dos sistemas federais para os sistemas estaduais ou locais, como ocorreu com alguns dos sistemas de mortalidade, nascidos vivos e cólera (BRANCO, 1996).

Os passos dados na melhoria dos sistemas de informação têm contribuído de maneira relevante para a administração e o planejamento. Toda esta contribuição poderia ser estendida aos demais participantes do setor. Para exemplificar, poderíamos criar uma situação onde um cidadão com domicílio e rotina médica em São Paulo e que por contingência fosse atendido no Rio de Janeiro e a partir de seu número SUS houvesse acesso ao seu cadastro e as informações pertinentes ao seu

atendimento. Essas informações acessadas auxiliariam não somente o atendimento ao paciente, mas os profissionais médicos poderiam possuir informações que poderiam auxiliar a definir um diagnóstico. Porém a utilização dessas informações visando auxiliar outras áreas como no apoio as atividades dos profissionais em saúde ou mesmo para um atendimento mais eficiente ao paciente, o compartilhamento, ainda é restrito e pouco difundido.

No intuito de extrair este tipo de informações, uma nota técnica publicado por (CAMPOS, M. ET AL., 2001), apresenta um procedimento para criar banco de dados gerado pelos arquivos do Sistema de Informações Hospitalares do Sistema Único de Saúde (SIH-SUS) onde estes dados estão disponíveis em CD fornecido pelo Ministério da Saúde. O resultado deste processo é um banco de dados que contém informações de vários arquivos de dados dispersos, com estruturas e objetivos diversos.

Com foco na tecnologia web, empresas de equipamentos para informática têm desenvolvido sistemas que atendem a um determinado domínio administrativo compartilhando os dados com os pares envolvidos neste ambiente. O Hygia (PINTO, 2000) é um sistema desenvolvido pela empresa Techne Engenharia e Sistema (<http://www.techne.com.br>), onde, através da internet procura interligar os diversos setores do sistema municipal de saúde em uma base de dados única, centralizada e compartilhada. Possui também mecanismos para a comunicação com os demais sistemas nacionais alimentando a base de dados do próximo nível governamental. Outro trabalho neste sentido (LEÃO ET AL., 2004) é o projeto de informatização da Secretária Municipal de Saúde de São Paulo (SMS-SP) onde existe a proposta e a viabilidade na construção de um único sistema integrado. Apesar destes sistemas possuírem o compartilhamento de dados, estes estão restritos ao seu domínio. Outro ponto relevante, é o fato do armazenamento de dados ficar centralizado em um único servidor expondo um ponto único, suscetível a falhas e gargalo provocado pelo grande número de acesso simultâneo.

7.1.2 Estrutura Hierárquica nos Hemocentros

Os Hemocentros possuem uma estrutura semelhante à descrita na seção anterior. Os medicamentos destinados aos hemofílicos são repassados pelo governo federal para os hemocentros, que por sua vez os distribuem para outras instituições, como hospitais por exemplo. O Hemocentro de Campinas denomina estas instituições, as quais recebem o repasse de medicamentos do gestor de medicamentos, de Hemonúcleos. Os Hemocentros que recebem o medicamento do Governo Federal, são responsáveis por realizar a consolidação mensal sobre o uso dos medicamentos, e repassar as informações ao Governo Federal.

As informações de um atendimento feito em um Hemonúcleo em geral é registrada em um prontuário local, com o passar do tempo os dados dos pacientes ficam espalhados pelos locais onde ele foi atendido, com o agravante de que qualquer informação referente ao atendimento a um hemofílico é de extrema importância para os médicos durante um atendimento.

Pode-se notar através da descrição acima o caráter hierárquico da organização dos Hemocentros, que estão subordinados às instituições governamentais e responsáveis pela distribuição dos medicamentos aos Hemonúcleos.

Em geral Hemonúcleos podem ter estrutura precária no que diz respeito a recursos computacionais, o que torna perigoso o armazenamento local de informações de pacientes e consumo de medicamentos.

7.1.3 As Especificidades dos Hemocentros e dos Pacientes Hemofílicos

As hemofilias são doenças hemorrágicas hereditárias resultantes da deficiência quantitativa e/ou qualitativa do fator VIII (hemofilia A) ou IX (hemofilia B) da coagulação. Pacientes acometidos por hemofilia podem apresentar sangramentos de gravidade variável, dependendo do nível de atividade do fator presente no plasma. Os sangramentos podem ser espontâneos ou pós-traumáticos, presentes ao nascimento ou somente diagnosticados ocasionalmente. A hemofilia ainda não

tem cura, o que leva os pacientes a serem dependentes da reposição do fator deficiente, por meio da administração de produtos derivados do plasma (hemoderivados) ou fatores recombinantes, por toda a vida.

É muito importante os Hemocentros manterem diversas informações sobre os seus pacientes hemofílicos e mantê-los acessíveis por outras instituições de saúde. Além disso, o controle dos medicamentos utilizados deve ser feito com precisão devido ao alto custo deste. Ao final de todo mês relatórios devem ser remetidos para os órgãos superiores informando detalhes do consumo do medicamento.

O objetivo deste estudo é a aplicação da arquitetura no caso descrito, facilitando o acesso aos dados, aumentar a confiabilidade e disponibilidade dos dados, bem como facilitar o acesso aos dados de controle dos medicamentos pelos órgãos superiores.

7.1.4 Adaptando a Arquitetura ao Problema dos Hemocentros

Para aplicar a arquitetura especificada, deve-se configurá-la de forma hierárquica no que diz respeito ao acesso aos dados e a criação das réplicas.

Ao fazer uma consulta ou receber uma dose do medicamento em um hemonúcleo, o que denominamos na figura 9 como nível operacional, o operador, ao inserir os dados no sistema, este automaticamente dispara o processo de replicação do dado em um nível mais seguro na hierarquia, em geral a estrutura computacional e a segurança dos recursos nos hemocentros são mais confiáveis, oferecendo uma melhor infra-estrutura para o armazenamento e segurança das informações. Por este motivo, o dado é então replicado no hemocentro, por exemplo. A partir de então, os dados se tornam acessíveis aos outros hemonúcleos, para acesso através do hemocentro que fica no nível denominado nível operacional/gestão.

O nível de gestão mostrado na figura 9 corresponde às entidades governamentais, que deverão ter acesso aos dados referentes ao controle dos medicamentos. Estes poderão realizar consultas em todos os nós que estejam em um nível abaixo, com o intuito de obter as informações de consumo do medicamento, bem como, estatísticas sobre a doença, tratamentos e pacientes.

A figura 10, exemplifica como ficaria a arquitetura aplicada na integração dos dados dos Hemocentros.

Figura 10: Visão de uma possível solução para a problemática dos Hemocentros utilizando a arquitetura proposta.

Como se pode observar, a natureza hierárquica do sistema de saúde é obedecido. Os Hemonúcleos e Hospitais, que estão na base da figura, são geradores de informações, porém nem sempre os recursos computacionais neste nível serão confiáveis, podendo o mesmo não suportar um número alto de consultas ou até mesmo perder dados devido a falhas imprevistas. Por essas razões os participantes deste nível solicitam a replicação dos seus dados nos nós um nível acima. Os Hemocentros possuem uma infra-estrutura computacional melhor, podendo este suportar cargas maiores de consulta, bem como ser um local seguro para os dados dos pacientes.

No nível mais alto da hierarquia, está o Governo Federal, que possui interesse na obtenção de estatísticas sobre a doença bem como sobre o consumo dos medicamentos. Os dados estão disponíveis para aplicações diversas como mineração de dados, por exemplo.

Ainda pode-se notar que o fato dos participantes de um mesmo nível não possuir acesso um ao outro, não significa que dados gerados em um deles não possa ser

acessado pelo outro, pois ambos tem acesso ao banco de dados que contém as réplicas.

7.2 Considerações Finais

A aplicação da arquitetura no estudo de caso discutido, se beneficia das características hierárquicas e do fato de o volume de dados no sistema descrito é considerado baixo. A flexibilidade da arquitetura, por meio do seu sistema de configuração permite que o mesmo reproduza fielmente o sistema dos Hemocentros.

Foram realizados testes de integração do protótipo desenvolvido com o sistema computacional em desenvolvimento para dar suporte aos Hemocentros. Os testes de integração preliminares foram satisfatórios. A utilização do *Hibernate Framework* pelos desenvolvedores do sistema, não foi empecilho para a integração com o protótipo devido ao desenvolvimento do *Driver* JDBC. Apesar de o componente *Client* ter sido usado com sucesso, a aplicação do protótipo completo não foi possível ser concluída até a data da publicação desta dissertação.

8 Testes e Análises

Nesta seção descrever-se-á os testes realizados e os resultados obtidos. Os testes realizados visam comprovar a funcionalidade do protótipo desenvolvido para validação da arquitetura. Exemplos de códigos utilizados durante os testes podem ser encontrados no apêndice C.

8.1 Ambiente de Testes

Foi desenvolvido um ambiente de testes utilizando Máquinas Virtuais, com o intuito de validar a funcionalidade do sistema. A utilização de Máquinas Virtuais para a realização de testes funcionais mostrou-se bastante satisfatório. A

Figura 11: Infra-estrutura de testes para o sistema implementado.

figura 11 é uma representação do ambiente de testes desenvolvido. Todas máquinas virtuais e a máquina hospedeira foram definidas como nós da *grade*. Note que, utilizou-se endereços de rede privados, porém o importante para os testes é que os nós consigam se comunicar uns com os outros. Outras topologias seriam possíveis porém as definidas aqui são suficientes para validar a funcionalidade do sistema proposto.

A tabela 7 fornece detalhes de cada uma das máquinas na infra-estrutura de testes. Para validar o funcionamento do protótipo em ambientes heterogêneos, decidiu-se incluir computadores executando o PostgreSQL e MySQL. Também acrescentou-se

à infra-estrutura uma máquina contendo o sistema operacional *Windows XP Professional*. Note que os programas de teste são executados no nó 1 utilizando, se necessário, os demais nós.

Tabela 7: Máquinas utilizadas para teste dos componentes.

Nó	Nome	Descrição	Componentes em Execução
0	darkstar	Máquina Hospedeira - Processador Intel Core 2 Duo, 2 GB de Memória RAM, 160GB de Disco Rígido Sistema Operacional Linux Kernel 2.6.27	Configurator DataHandler Database (PostgreSQL) DataDictionary Programas de Testes
1	grid-node01	Maquina Virtual com 128MB de RAM Sistema Operacional Linux Kernel 2.6.27	DataHandler Database (PostgreSQL)
2	grid-node02	Máquina Virtual com 128MB de RAM Sistema Operacional Linux Kernel 2.6.27	DataHandler Database (MySQL)
3	grid-node03	Máquina Virtual com 512MB de RAM Executando DataHandler Sistema Operacional Windows XP Professional	DataHandler Database (MySQL)

A partir deste ambiente validou-se funcionalmente o protótipo da arquitetura, tanto na sua forma totalmente distribuída, como também quando configurado de forma hierárquica. Os testes foram realizados variando-se as configurações das máquinas em questão e verificando-se as funcionalidades do sistema, como mostram as subseções a seguir.

8.2 Testes Funcionais dos Componentes

As *activities* desenvolvidas tiveram as suas funcionalidades testadas. Cada conjunto de *activities* representam um componente da arquitetura especificada e o seu comportamento deve prover as funcionalidades descritas pela arquitetura. Os testes realizados são detalhados abaixo.

8.2.1 Teste do Componente Configurator

O componente *Configurator* é composto pelas *activities* *LoadConfiguration* e *StoreConfiguration*. Os testes consistiram em verificar o recebimento de configurações armazenadas no nó contendo o componente. As configurações são armazenadas em arquivos no nó. Quando solicitada uma configuração o nó da *grade* envia um identificador da configuração desejada juntamente com a solicitação.

Foram escritos testes para enviarem configurações para serem armazenadas utilizando-se a *activity StoreConfiguration*, sendo que algum tempo depois esta configuração era requisitada novamente utilizando-se a *activity LoadConfiguration*. Para estes testes foram envolvidos dois nós.

As *activities* funcionaram dentro das suas especificações, testes foram feitos solicitando configurações com identificadores inexistentes, e como esperado as mensagens de erro chegaram ao cliente informando-o sobre a falha.

8.2.2 Teste do Componente DataHandler

O componente *DataHandler* é composto por três componentes, sendo dois deles desenvolvido no escopo deste trabalho, são eles o *ReplicaManager* e o *DictionaryHandler*. Os testes desenvolvidos serão apresentados a seguir.

Testes do *DictionaryHandler*

O subcomponente *DictionaryHandler* foi testado utilizando-se dois métodos, o primeiro (*Dictionary Test* - DT1) consistiu em chamar diretamente a *activity*, e o segundo (DT2) verificar os resultados da chamada a uma *activity* do *ReplicaManager*. A figura 12 exemplifica os testes mencionados.

Este subcomponente realiza operações no componente *DataDictionary*, portanto, verificar se as informações estão sendo armazenadas de forma correta foi o primeiro

passo. Foram feitas chamadas diretas às *activities* e logo após os resultados foram inspecionados no *DataDictionary*.

As *activities* do *ReplicaManager* fazem uso das *activities* do *DictionaryHandler* quando desejam inserir informações de localidade, no *DataDictionary*, de um registro sendo replicado. Foram feitos testes de inserção, remoção e atualização de registros e observou-se os resultados no *DataDictionary*.

Figura 12: Estrutura dos testes realizados com o DictionaryHandler.

Testes do *ReplicaManager*

Fazem parte do *DataHandler* as *activities*: *InsertReplicate*, *DeleteReplicate* e *UpdateReplicate*. Para testá-los foram determinados dois conjuntos de testes para as *activities*. O primeiro consistindo em verificar a corretude na execução das *activities* implementadas, e o segundo para validar as configurações hierárquica e distribuída da arquitetura. Os quatro primeiros testes são mostrados na tabela 8 e fazem parte do primeiro conjunto de testes.

Tabela 8: Lista dos testes realizados para validar o componente *DataHandler*, suas descrições e resultados esperados.

Teste	Descrição	Resultado Esperado
AT1	Ambiente configurado para não criar réplicas. Aplicativo de teste realiza operações de inserção,	As <i>activities</i> responsáveis deverão executar as operações no banco de dados.

	atualização e remoção.	
AT2	Ambiente configurado para replicar a operação em um dos nós.	As operações devem ser executadas no banco de dados local e propagadas para o nó configurado para receber a réplica.
AT3	Envolve 3 nós onde todos estão configurados para replicar os dados.	As operações devem ser executada localmente e propagada, o mesmo processo deverá ocorrer no próximo nó, propagando a mudança para um terceiro nó.
AT4	Verificar se as operações dos testes AT1, AT2 e AT3, manipularam corretamente o dicionário de dados	Esperasse que a execução dos testes anteriores estejam atualizando corretamente as informações de localidade das réplicas.

Cada *activity* desenvolvida tem um programa de testes próprios. Nestes programas exploram-se as funcionalidades e os diferentes tipos de uso que a *activity* implementada pode oferecer. Tais testes foram denominados de *Activity Test* (teste de *activities*) e abreviados para AT, seguido de um número identificador do teste. Os testes são detalhados na tabela 8 e na figura 13.

A figura 13a exemplifica os testes realizados para validar funcionalmente a execução da operação solicitada no Banco de Dados exposto com o OGSA-DAI. Em caso de inserção uma as informações de localidade da réplica devem ser inseridas no dicionário de dados.

Figura 13: Detalhes do funcionamento dos testes executados.

A figura 13b exemplifica os testes realizados para validar a replicação, sendo a máquina *darkstar* configurada para realizar a replicação de suas operações. Ao solicitar a replicação, o solicitante fica responsável por atualizar o dicionário de dados com informações dos dados armazenados nele mesmo e na máquina onde foi feita a réplica.

A figura 13c descreve o teste AT3, que é bastante semelhante ao teste AT2, porém o *grid-node01* que recebe a réplica também está configurado para replicar a informação. Fica claro no teste AT3 a responsabilidade de atualização do dicionário de dados por parte do solicitante da replicação.

Após a execução dos testes, inspeções no banco de dados foram realizadas para verificar se as operações executadas nos testes mencionados geraram os resultados esperados. As *activities* que manipulam o dicionário de dados passam pelos mesmos testes e as inspeções são feitas no dicionário para verificar se as operações são feitas corretamente, o que constitui o teste AT4.

Os resultados foram satisfatórios, as operações se propagam para os nós configurados para receberem as réplicas e os dicionários são manipulados corretamente. Vale salientar, que os testes estão sendo feitos em um ambiente controlado e estável. Testes em ambientes reais estão sendo planejados para as próximas etapas deste projeto.

8.3 Testes utilizando Hierarquias

Foram realizados testes para verificar o comportamento do protótipo quando configurado de maneira hierárquica. Para isso foram escritas diferentes configuração para as diferentes máquinas como mostra a tabela 9.

O funcionamento do sistema com a configuração em questão é mostrado na figura 14. A máquina *darkstar* mantém as configurações de todos os nós da *grade* e as disponibilizam a partir do componente *Configurator*. Está configurada também na máquina *darkstar* o componente *DataDictionary*.

Tabela 9: Configurações das máquinas para a definição de uma hierarquia de replicação.

Máquina	Chave	Valor
darkstar	<i>configurator.url</i>	local
	<i>configuration.id</i>	local
	<i>replication.site</i>	none
	<i>dictionary.url</i>	http://darkstar/wsrf/services/dai/dictionary
grid-node01	<i>configurator.url</i>	http://darkstar:8080/wsrf/services/dai/configurator
	<i>configuration.id</i>	grid-node01
	<i>replication.site</i>	http://grid-node02:8080/wsrf/services/dai/lbb
	<i>dictionary.url</i>	http://darkstar/wsrf/services/dai/dictionary
grid-node02	<i>configurator.url</i>	http://darkstar:8080/wsrf/services/dai/configurator
	<i>configuration.id</i>	grid-node02
	<i>replication.site</i>	http://darkstar:8080/wsrf/services/dai/lbb
	<i>dictionary.url</i>	http://darkstar/wsrf/services/dai/dictionary
grid-node03	<i>configurator.url</i>	http://darkstar:8080/wsrf/services/dai/configurator
	<i>configuration.id</i>	grid-node03
	<i>replication.site</i>	http://darkstar:8080/wsrf/services/dai/lbb
	<i>dictionary.url</i>	http://darkstar/wsrf/services/dai/dictionary

A configuração mostrada na tabela 9 faz com que as réplicas sejam feitas em determinadas máquinas, desta forma força-se a propagação hierárquica das réplicas. O *grid-node01* ao efetuar uma operação replica a mesma no *grid-node02* e atualiza o dicionário de dados com as informações do novo registro(em caso de inserção). O *grid-node02* então solicita a criação de mais uma réplica na máquina *darkstar* e também atualiza o dicionário de dados. Uma necessidade em esquemas de configuração como o apresentado é a detecção de ciclos, não permitindo que configurações cíclicas, ou seja, configurações em que um nó acabe solicitando indiretamente uma réplica em si mesmo. Nesta versão ciclos não são detectados e trabalhos neste sentido estão sendo realizados.

Figura 14: Teste realizado para verificar o comportamento do protótipo quando configurado hierarquicamente.

A figura 14 mostra como ficou a configuração hierárquica. As setas contínuas indicam o fluxo das réplicas, quando um registro é inserido, por exemplo, no *grid-node01* as réplicas começam a ser realizadas partindo deste nó para os demais, como consta nas configurações de cada um deles. As setas serrilhadas, indicam a atualização do dicionário de dados, o *grid-node01* ao replicar seus dados em *grid-node02* atualiza o dicionário colocando as informações de localidade da réplica que

acabou de solicitar. Pode-se notar que *grid-node02*, também está configurado para replicar as operações nele realizadas para o nó *darkstar*, percebe-se também que o mesmo também atualiza as informações no dicionário, que neste caso está instalado na máquina *darkstar*.

Foram submetidas ao sistema uma série de operações de inserção, remoção e atualização para verificar se as operações foram executadas corretamente e se o sistema se comporta de fato, de acordo com a configuração estabelecida.

8.4 Teste do Driver JDBC para OGSA-DAI

Foram definidos três tipos de testes para o *Driver* JDBC desenvolvido. O primeiro consistindo no teste das *activities* desenvolvidas para dar suporte ao *driver*. O segundo teste foi feito levando em consideração aplicações simples e rotineiras escritas em Java e que fazem acesso a bancos de dados utilizando a tecnologia JDBC, como realizar consultas, inserções e atualização em um banco de dados exposto usando o OGSA-DAI.

8.4.1 Testes das *activities* de suporte

Para testar as *activities* de suporte, que tem como função retornar metadados detalhados de um banco de dados exposto, escolheu-se um SGBD e criou-se um banco de dados de testes, então expôs-se o banco de dados na *grade*. Um programa foi desenvolvido solicitando todos os metadados possíveis, utilizando o *driver* do distribuidor do SGBD, bem como o *driver* JDBC para OGSA-DAI. Após obter os dados, o programa compara os resultados para verificar se as informações são as mesmas como mostra a figura 15.

Figura 15: Testes realizados para validar a funcionalidade das *activities* de suporte ao *driver* JDBC desenvolvido.

A comparação dos resultados mostrou que as *activities* desenvolvidas retornaram o mesmo resultado que o *driver* JDBC fornecido pelo fabricante.

8.4.2 Demais Testes

Foram escritos outros dois programas para validar o funcionamento do *driver*, como mostra a figura 16. O primeiro mostrado na figura 16a, é um programa java comum para acessar um banco de dados. Utilizou-se o *driver* JDBC desenvolvido e foram realizadas operações simples no Banco de Dados.

O teste da figura 16b foi realizado para testar o *Driver* sob uso intenso do ponto de vista de suas funcionalidades. Escolheu-se então o *Hibernate Framework*, que utiliza o JDBC como base para realizar mapeamento objeto-relacional entre os objetos Java e as tabelas de um banco de dados relacional. Após a realização do mapeamento, o programa utiliza o *Hibernate* para iniciar uma série de operações de inserções e atualizações. O gerenciamento de tais operações são feitas pelo *Hibernate*, que com base na criação de objetos de uma classe mapeada ou na modificação de um dos atributos desse mesmo objeto, cria instruções SQL para

serem executadas no SGBD através do *Driver* JDBC. Este teste revelou diversos erros, que foram corrigidos até a execução do teste obter sucesso.

Figura 16: Testes realizados para validar a funcionalidade do driver JDBC para OGSA-DAI.

8.4.3 Considerações Finais

O *Driver* encontra-se em versão de teste, sua primeira versão estável ainda não foi liberada. O *Driver* deve passar por um processo de refinamento para correção de erros e novas funcionalidades devem ser acrescentadas em versões futuras. O *driver* pode ser obtido a partir do seu sítio (<http://ogsadai-jdbc.sourceforge.net>) incluindo o seu código-fonte.

9 CONCLUSÃO

A necessidade de integração de dados é uma necessidade, tanto na academia quanto na indústria. Tecnologias para promover esta integração existem, porém algumas funcionalidades não são oferecidas, além de que a aplicação de padrões e como consequência a promoção da interoperabilidade devem servir de guia para o desenvolvimento de tais ferramentas, o que ocorre em muitas das ferramentas disponíveis.

Soluções de mercado não fazem uso, nem estabelecem padrões completamente abertos e não tem como fundamento a tecnologia de *grades*, sendo uma alternativa o uso do OGSA-DAI, que promove a discussão e aprovação destes padrões dentro da OGF, se comprometendo em segui-los, bem como ter toda a sua infra-estrutura implementada sob o conceito de *grades*.

Porém tal *middleware* não oferece todas as funcionalidades encontradas nas soluções comerciais, então aliar tais necessidades com o uso de padrões e da tecnologia de *grades* é uma contribuição relevante.

A escolha de uma implementação do JDBC utilizando a biblioteca de desenvolvimento do OGSA-DAI gerou resultados além do esperado. Concluimos que tal ferramenta não se aplica apenas à arquitetura em questão, mas sim para aquelas aplicações que desejam fazer uso de bancos de dados expostos pelo OGSA-DAI e que são desenvolvidos sob a plataforma Java. A mesma também oferece uma forma fácil de fazer com que aplicações legadas escritas em Java façam uso de bancos de dados expostos na *grade*, além de permitir que *frameworks* baseados no JDBC também o façam.

Concluimos a partir deste trabalho, que o uso de *grades* é de fato uma boa opção para a problemática de integração de dados, e que o OGSA-DAI é uma excelente ferramenta para o desenvolvimento de aplicações para *grades* que fazem uso de bancos de dados com a necessidade de integração. Apesar de não oferecer recursos mais avançados de integração, o OGSA-DAI se mostrou flexível, devido

aos seus vários pontos de extensibilidade, o que nos permitiu o acréscimo das funcionalidades necessárias para a implementação do protótipo da arquitetura.

A configurabilidade de um sistema é importante, garantindo-lhe maior flexibilidade e a possibilidade de dispor os componentes de variadas formas. Este conceito foi comprovado ao se desenvolver um componente configurador no sistema, o que permitiu a configuração de instâncias do sistema obedecerem uma hierarquia, o que se tornou bastante útil ao se aplicar a arquitetura em um sistema de saúde, que tem uma hierarquia bem definida no que diz respeito à geração e repasse dos dados gerados pelo sistema de informação.

9.1 Publicações

Ainda como resultado do trabalho apresentado, houve a escrita de três artigos, sendo dois deles em inglês e submetidos para conferências internacionais, sendo um já publicado (BRITO; SATO, 2008) e um submetido aguardando resultado, e um artigo em português submetido para um simpósio nacional. A aceitação de um dos artigos comprova a contribuição do trabalho desenvolvido exposto neste documento.

9.2 Contribuições

Contribuições foram feitas durante a execução deste trabalho, sendo válido ressaltar a definição de uma arquitetura para integração de bancos de dados com suporte a replicação para *grades*.

Uma contribuição também relevante se refere ao desenvolvimento do *driver* JDBC para o *middleware* OGSA-DAI. Como já discutido, esta ferramenta possibilita a integração de dados em aplicações existentes e que não são habilitadas para a *grade*, bem como permite o uso de ferramentas baseadas no JDBC e o desenvolvimento rápido de aplicativos que façam uso de bancos de dados expostos na *Grade*.

Outra contribuição diz respeito às extensões feitas ao OGSA-DAI. Estas extensões na forma de *activities*, adicionam funcionalidades não existentes na versão atual do OGSA-DAI. O contínuo desenvolvimento e aperfeiçoamento destas *activities* possibilitará a sua inserção nos pacotes de extensões liberado pelos desenvolvedores do OGSA-DAI.

9.3 Trabalhos Futuros

Como trabalhos futuros, tem-se o refinamento do sistema recém desenvolvido, com o intuito de oferecer um produto e liberar uma versão estável e que possa ser colocado em produção.

Deverão ser realizados testes detalhados de desempenho, como tempo de consulta, tempo de propagação das réplicas e atualização do dicionário de dados, deverão ser realizados também, testes para verificar a taxa de integridade das réplicas durante o funcionamento do sistema. Tais testes devem ser feitos adicionando-se ruídos em uma rede controlada.

Uma vez finalizado o desenvolvimento do sistema, deverá iniciar-se um processo de discussão junto ao time de desenvolvimento do OGSA-DAI para avaliar a inclusão das *activities* desenvolvidas na distribuição principal ou em pacotes de extensão do OGSA-DAI. Além disso deverá ser definida uma agenda de atualizações para a inclusão de novos recursos e a correção periódica de erros, culminando em novas versões do sistema.

O protótipo desenvolvido, e seu posterior aperfeiçoamento, servirá de base para a investigação do uso da teoria de Transações em um ambiente de *grades* computacionais, pois o protótipo baseado na arquitetura apresentam um comportamento bastante peculiar, onde requisições disparam outras requisições que são executadas em outro recurso, e este último por fim também pode disparar outras requisições. Garantir que estas operações sejam realizadas através do uso da teoria de transações é objeto de estudos futuros.

REFERÊNCIAS

ALMEIDA, M. **Descentralização de sistemas de informação e o uso das informações a nível municipal**, Informe Epidemiológico do SUS. p.27-34. SUS, 1998.

ALVANHAN, R.; CAMPOS, J.; SOARES, D.; ANDRADE, S. Bases da Saúde Coletiva. **Vigilância epidemiológica**, p. 212-223, 2001.

ANTONIOLETTI, M.; ATKINSON, M.; BAXTER, R.; ET AL. The design and implementation of Grid database services in OGSA-DAI. **Concurrency and Computation: Practice and Experience**, v. 17, p. 357-376, 2005.

BABOSA, D. **Sistemas de informação em saúde: a percepção e a avaliação dos profissionais diretamente envolvidos na atenção básica de Ribeirão Preto / SP**. MSc Thesis, Faculdade de Medicina de Ribeirão Preto da Universidade de São Paulo, 2006.

BATINI, C.; LENZERINI, M.; NAVATHE, S. A comparative analysis of methodologies for database schema integration. **ACM Comput. Surv.**, v. 18, n. 4, p. 323-364. doi: 10.1145/27633.27634, 1986.

BOOTH, D.; HASS, H.; MCCABE, F.; ET AL. Web Services Architecture. **W3C Technical Reports and Publications**. Recuperado de <http://www.w3.org/TR/ws-arch/>, 2004.

BRADLEY, J.; BROWN, C.; CARPENTER, B.; ET AL. The OMII Software Distribution. **Proceedings of the UK e-Science All Hands Meeting**, p. 748-753, 2006.

BRANCO, M. **Sistemas de informação em saúde no nível local**, Caderno de Saúde Pública. v. 12, p.267-270, 1996.

BRITO, M.; SATO, L. Extending OGSA-DAI Possibilities with a JDBC Driver. In: Computational Science and Engineering, 2008. CSE '08. 11th IEEE International Conference on. p.155-162. doi: 10.1109/CSE.2008.55, 2008.

BUYYA, R.; VENUGOPAL, S. The Gridbus toolkit for service oriented grid and utility computing: an overview and status report. **Grid Economics and Business Models**,

2004. GECON 2004. 1st IEEE International Workshop on, p. 19-66, 2004.

CAMPOS, M.; MARTINS, M.; NORONHA, J.; TRAVASSOS, C. Proposta de integração de dados do sistema de informações hospitalares do Sistema Único de Saúde (SIH-SUS) para pesquisa. **Inf Epidemiol SUS.**, v. 9, n. 1, 2001.

CAREY, M. J.; LIVNY, M. Distributed Concurrency Control Performance: A Study of Algorithms, Distribution, and Replication. **Proceedings of the 14th International Conference on Very Large Data Bases**, p. 13-25, 1988.

CHEN, Y. IBM OGSA-DAI Replication. . Recuperado Janeiro 23, 2009, de http://www.aiai.ed.ac.uk/~ychen/ibm_ogsadai/ibm-ogsadai-index.html, 2008.

CHEN, Y.; BERRY, D.; DANTRESSANGLE, P. Transaction-Based Grid Database Replication. Nottingham, 2007.

CHINNICI, R.; GUDGIN, M.; MOREAU, J.; SCHLIMMER, J.; WEERAWARANA, S. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. **W3C Working Draft**, v. 3, 2004.

CZAJKOWSKI, K.; FERGUSON, D.; FOSTER, I.; ET AL. The WS-Resource Framework Version 1.0. **Global Grid Forum**, available at <http://www.globus.org/wsrf>, 2004.

DOGDU, E.; WANG, Y.; DESETTY, S. A Generic Database Web Service. **Proc. of the 2006 Int. Conf. On Semantic Web and Web Services, SWWS'06**, 2006.

ELLIS, J.; HO, L.; FISHER, M. JDBC 3.0 Specification. **Sun Microsystems, Santa Clara, CA**, 2006.

ELMAGARMID, A.; RUSINKIEWICZ, M.; SHETH, A. **Management of Heterogeneous & Autonomous Database Systems**. 1° ed., p.400. Morgan Kaufmann, 1999.

ERWIN, D.; SNELLING, D. UNICORE: A Grid Computing Environment. In: **Euro-Par 2001 Parallel Processing**. p.825-834. Recuperado Junho 9, 2008, de http://dx.doi.org/10.1007/3-540-44681-8_116, 2001.

FOSTER, I. Globus Toolkit Version 4: Software for Service-Oriented Systems. **Network And Parallel Computing: IFIP International Conference, NPC 2005, Beijing, China, November 30-December 3, 2005: Proceedings**, 2005.

FOSTER, I.; KESSELMAN, C. Globus: a Metacomputing Infrastructure Toolkit. **International Journal of High Performance Computing Applications**, v. 11, n. 2, p. 115, 1997.

FOSTER, I.; KESSELMAN, C. **The Grid 2: Blueprint for a New Computing Infrastructure**. 2 ed. Morgan Kaufmann, 2003.

FOSTER, I.; KESSELMAN, C.; NICK, J.; TUECKE, S. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. **Open Grid Service Infrastructure WG, Global Grid Forum**, v. 22, p. 2002, 2002.

FOSTER, I.; KESSELMAN, C.; TUECKE, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. **International Journal of High Performance Computing Applications**, v. 15, p. 200-222, 2001.

FOSTER, I.; KESSELMAN, C.; TUECKE, S. The Open Grid Services Architecture. **The Grid2: Blueprint for a New Computing Infrastructure**, p. 215-257, 2004.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison-Wesley Professional, 1994.

GRAY, J.; HELLAND, P.; O'NEIL, P.; SHASHA, D. The dangers of replication and a solution. In: Proceedings of the 1996 ACM SIGMOD international conference on Management of data. p.173-182. ACM New York, NY, USA, 1996.

GRAY, J.; REUTER, A. **Transaction Processing: Concepts and Techniques**. 1^o ed., p.1070. Morgan Kaufmann, 1993.

GUDGIN, M.; HADLEY, M.; MENDELSON, N.; MOREAU, J.; NIELSEN, H. SOAP Version 1.2. **W3C Working Draft**, 2002.

HONG, C.; HUME, A.; JACKSON, M.; ET AL. OGSA-DAI 3.0 - The What's and Whys. In: UK e-Science All Hands Meeting, 2007.

JACKSON, M.; LLOYD, A.; SLOAN, T. Enabling Access to Federated Grid Databases: An OGSA-DAI ODBC Driver. **AHM2005: Innovating through e-Science, Nottingham**, 2005.

JORGE, M.; GOTLIEB, S.; SOBOLL, M.; ALMEIDA, M.; ROSÁRIO, M. **Aaliação do Sistema de Informação sobre Nascidos Vivos e o Uso de Seus Dados em Epidemiologia e Estatísticas da Saúde**, Rev. Saúde Pública. v. 27. São Paulo, 1993.

LAURE, E.; FISHER, S.; FROHNER, A.; ET AL. Programming the Grid with gLite. **Computational Methods in Science and Technology**, v. 12, p. 33-45, 2006.

LEÃO, B.; VALENTE, G.; EVANGELISTI, L.; ET AL. O Desafio de Integrar os Sistemas de Informação em Saúde. Ribeirão Preto, SP, 2004.

OGSA-DAI Documentation. <http://www.ogsadai.org.uk/documentation/>.

OZSU, M.; VALDURIEZ, P. **Principles of Distributed Database Systems**. 2 ed., p.666. Prentice Hall, 1999.

PINTO, I. **Os Sistemas Públicos de Informação em Saúde na Tomada de Decisão - Rede Básica de Saúde do Município de Ribeirão Preto-SP**. Mestrado, Escola de Enfermagem de Ribeirão Preto, Universidade de São Paulo., 2000.

SHETH, A.; LARSON, A. Federated database systems for managing distributed, heterogeneous, and autonomous databases. **ACM Comput. Surv.**, v. 22, n. 3, p. 183-236. doi: 10.1145/96602.96604, 1990.

TUECKE, S.; CZAJKOWSKI, K.; FOSTER, I.; ET AL. Open Grid Services Infrastructure (OGSI). **Global Grid Forum**, 2003.

WHITE, S.; HAPNER, M. **JDBC TM 2.1 API**. Sun Microsystems, 1999.

WIESMANN, M.; PEDONE, F.; SCHIPER, A.; KEMME, B.; ALONSO, G. Understanding replication in databases and distributed systems. In: Proceedings of the The 20th International Conference on Distributed Computing Systems (ICDCS 2000). p.464. IEEE Computer Society Washington, DC, USA, 2000.

XSTREAM <http://xstream.codehaus.org/>. . Recuperado Janeiro 23, 2009, de <http://xstream.codehaus.org/index.html>.

YEONG, W.; HOWES, T.; KILLE, S. RFC1777: Lightweight Directory Access Protocol. **Internet RFCs**, 1995.

APÊNDICE A - Activities dos Componentes Configurator, DataHandler e seus Subcomponentes

O componente *DataHandler* é responsável por executar operações que envolvam a replicação de uma determinada operação. Este componente é implementado como sendo uma instância em execução do *container* de serviços do *Globus Toolkit 4* executando os serviços do *middleware OGSA-DAI*.

O *OGSA-DAI* oferece serviços na forma de *activities*, que nada mais é do que uma abstração na linguagem java dos *Web Services* oferecidos pelo *OGSA-DAI* com suas respectivas entradas e saídas. Para implementar as funcionalidades do *DataHandler*, novas *activities* foram desenvolvidas e implantadas no *OGSA-DAI*. A seguir uma lista destas *activities* e suas respectivas descrições.

Foram especificados então subcomponentes do *DataHandler*. Sendo que a especificação destes componentes deu-se conforme a funcionalidade de uma determinada *activity*, ao agrupar-se as *activities* funcionalmente, chamou-se cada um destes agrupamentos de subcomponentes.

1 Activities do subcomponente ReplicaManager

ReplicaManager é uma subcomponente do componente *DataHandler*, ele é responsável por processar as requisições que envolvem replicação, executar as requisições e solicitar o registro das réplicas no diretório de dados. Tal componente possui as *activities* apresentadas a seguir.

1.1 InsertReplicateActivity

Sumário

Esta *activity* recebe uma *query* a ser executada, neste caso uma inserção, executa-a e solicita a replicação da operação em uma lista de servidores enviadas pelo cliente

ou lidas das configurações do servidor.

Entrada

A entrada consiste em uma *query* SQL do tipo INSERT, seguida opcionalmente de uma lista de servidores para onde as réplicas devem ser realizadas e o endereço de um diretório de dados. O servidor irá sobrescrever a segunda e terceira entrada se configurado para isso.

Saída

Como saída a *activity* retornará o status da operação, verdade (*true*) ou falso (*false*).

Comportamento

A *activity* executará a operação no recurso especificado e solicitará a replicação da operação nos servidores especificados pelo subcomponente *Configurator* do componente *DataHandler*, ou pelo cliente. A *activity* irá aguardar pelo resultado das replicações e irá escrever no dicionário de dados os recursos que agora tem o registro resultante da execução da *query*. Note que somente aqueles que informarem que a operação de replicação foi efetuada com sucesso serão cadastradas no dicionário de dados.

1.2 UpdateReplicateActivity

Sumário

Esta *activity* receberá uma *query* SQL do tipo UPDATE, que realizara uma modificação em uma linha ou mais de uma em um recurso determinado pelo cliente. O cliente poderá informar recursos onde a operação deverá ser replicada. Não é necessário modificações no dicionário de dados.

Entrada

A *activity* receberá como entrada uma *query* SQL do tipo UPDATE e uma lista de servidores onde se deverá realizar a replicação da *query*.

Saída

A *activity* retornará o resultado da execução, verdade (*true*) se tudo ocorreu como o

previsto ou falso (*false*) em caso de falha.

Comportamento

A *activity* em questão executara a *query* (UPDATE) no recurso especificado, após esta operação irá iniciar a tentativa de replicação nos recursos informados pelo cliente, ou então nos recursos especificados na configuração do componente *DataHandler* a partir do subcomponente *Configurator*. Não é necessária a modificação do dicionário, visto que uma atualização requer que ao registro já se encontre lá. A *activity* então retornará um resultado indicando sucesso ou falha no processo.

1.3 DeleteReplicateActivity

Sumário

Esta *activity* se faz necessária pois uma operação de remoção deve solicitar a remoção das informações, pertinentes ao registro, do diretório de dados. Uma remoção deve ser propagada para os servidores informados pelo cliente ou de uma lista obtida através do componente *Configurator*.

Entrada

A *activity* recebe como entrada uma *query* SQL do tipo DELETE, seguido de uma lista de servidores onde deve ser replicada a operação de remoção, seguido do endereço do diretório de dados. A lista de servidores para replicação e o endereço do diretório de dados podem ser alterados pelas configurações do servidor.

Saída

A saída desta *activity* é um valor verdadeiro (*true*), caso a operação tenha sido realizada com sucesso, ou falso (*false*) caso ocorra alguma falha durante a operação.

Comportamento

Ao receber a *query* juntamente com as listas de servidores a replicar e o endereço do diretório de dados, a *activity* executa a *query* e solicita a replicação da mesma nos servidores indicados, removendo as informações pertinentes do diretório de

dados. Caso o servidor especifique servidores onde serão feitas as réplicas e um diretório de dados, as informações passadas na requisição são sobrescritas. A *activity* então retorna para o cliente se a operação foi executada com sucesso ou não.

2 Activities do Subcomponente DictionaryHandler

DictionaryHandler é um subcomponente do componente DataHandler, este componente é responsável por se comunicar com o componente DataDirectory, sendo este responsável por inserir, buscar, modificar e remover informação do componente DataDirectory.

2.1 DictionaryLookupActivity

Sumário

Esta *Activity* busca em um diretório de dados por um determinado padrão recebido como entrada. A *Activity* então retorna as informação relevantes dos registros encontrados.

Entrada

A *Activity* recebe como entrada o padrão a ser buscado na forma de uma *string* bem como o endereço do diretório de dados a ser pesquisado.

Saída

A *Activity* retorna uma lista contendo as informações relevantes dos registros encontrados que coincidem com o padrão especificado.

Comportamento

Ao receber a requisição, a *activity* realiza uma operação de busca pelo padrão informado em um diretório de dados. Ao executar tal operação a *activity* retorna para o cliente as informações relevantes dos registros encontrados no diretório de dados.

2.2 DictionaryInsertActivity

Sumário

Insere uma nova entrada informada em um diretório de dados especificado.

Entrada

A *activity* recebe como entrada um registro a ser inserido e o endereço do diretório de dados onde deve ser inserido tal registro.

Saída

Como saída, a *activity* retornará verdadeiro (*true*) em caso de sucesso, ou falso (*false*) em caso de falha.

Comportamento

A *activity* ao receber os dados de entrada, imediatamente envia um comando para o diretório de dados solicitando a inserção de um novo registro no mesmo, retornando ao cliente se a operação foi bem sucedida ou não.

2.3 DictionaryUpdateActivity

Sumário

Esta *activity* atualiza um registro inserido anteriormente no diretório de dados.

Entrada

Esta *activity* tem como entrada a informação a ser atualizada, bem como o diretório de dados onde será realizada a operação.

Saída

Como saída a *activity* irá retornar verdadeiro (*true*) em caso de sucesso e falso (*false*) em caso de erro durante a execução da operação.

Comportamento

A *activity* irá informar o diretório de dados sobre a atualização, repassando para o mesmo as informações necessárias para a realização da operação, e irá responder ao cliente com o resultado da operação, sucesso ou falha.

2.4 DictionaryRemoveActivity

Sumário

Esta *activity*, será responsável por remover uma entrada no diretório de dados.

Entrada

O registro ou um padrão para se remover os registros, juntamente com o endereço do dicionário de dados a ser manipulado.

Saída

Como saída a *activity* irá retornar o número de registros removidos, ou -1 caso a operação tenha sido malsucedida.

Comportamento

Esta *activity* poderá receber informações de um registro a ser removido, ou um padrão a ser buscado e todas os registros que estejam de acordo com esse padrão serão removidas do diretório de dados especificados, retornando ao cliente o número de registros removidos, ou -1 em caso de falha.

3 Activities do Componente Configurator

O componente *Configurator* é independente, e sua função é armazenar e fornecer configurações para o componente *DataHandler*, ou qualquer outro que necessite de armazenar ou recuperar configurações específicas. A definição de um ou mais *Configurators* permite a administradores definir estruturas estáticas para a uma infraestrutura baseada na arquitetura proposta.

3.1 StoreConfigurationsActivity

Sumário

Armazena as configurações recebidas sob uma determinada identificação.

Entrada

Configurações a serem armazenadas.

Saída

A *activity* retornará verdadeiro (*true*) caso a operação tenha sido realizada com sucesso, ou falso (*false*) caso a mesma tenha falhado.

Comportamento

A *activity* em questão irá armazenar as configurações enviadas sob um determinado identificador, para posterior recuperação de tais configurações. Configurações recebidas com um identificador já existente sobrescreverão as informações mantidas sob aquela mesma identificação.

3.2 LoadConfigurationsActivity**Sumário**

Esta *activity* irá retornar as configurações solicitadas para o cliente.

Entrada

O identificador das configurações a serem retornadas.

Saída

Configurações com identificador especificado.

Comportamento

Ao receber uma requisição, esta *activity* verificará a existência de configurações sob o identificador informado na requisição, caso exista irá retorná-los, caso não exista irá retornar uma configuração em branco.

APÊNDICE B – Exemplo de Implantação do Protótipo

1 Itens Obrigatórios em cada Nó

Para a implantação do protótipo apresentado, é obrigatório a instalação e configuração do OGSA-DAI. Este por sua vez poderá ser implantado nos *middleware Globus Toolkit 4.0.x* ou no Apache Tomcat 5.0.x com Apache Axis 1.4.

O *driver* tem suporte à infra-estrutura de segurança do *Globus Toolkit* e cabe ao administrador da infra-estrutura configurá-la. Esta etapa porém é opcional, pois o *driver* também funcionará no modo não seguro.

2 Instalando as *activities* dos componentes do sistema

Para exemplificar a montagem de uma infra-estrutura que faça uso do protótipo em questão utilizar-se-á como base a infra-estrutura montada para a realização dos testes, porém com algumas modificações, como por exemplo, o uso apenas de um SGBD e somente do sistema Linux. A infra-estrutura consiste em 4 nós, sendo que todos eles contém o componente *DataHandler*, e os último além de conter o *DataHandler*, também contém o componente *Configurator* e o componente *DataDictionary*. Apesar de uma das máquinas na estrutura real possuir windows, nesta seção abordaremos apenas configurações para a plataforma *Linux*.

As máquinas a serem configuradas se chamam, darkstar, grid-node01, grid-node02 e grid-node03.

2.1 Instalando e Configurando a Máquina darkstar

A máquina darkstar, terá os seguintes componentes: *DataHandler*, *Configurator*, *DataDictionary*. Foram desenvolvidas *activities* para os subcomponentes *ReplicaManager* e *DictionaryHandler* do *DataHandler*, além de outras duas *activities* desenvolvidas para o *Configurator*, *LoadConfiguration* e *StoreConfiguration*, as quais

será detalhado o processo de instalação.

Em seu pacote de instalação, o OGSA-DAI, oferece um *script* Ant para a implantação de novas *activities* em sua estrutura, este *script* é utilizado para fazer a implantação do protótipo. Nas subseções que seguem serão abordados os passos para a implantação e a configuração dos componentes.

2.1.1 Expondo um Banco de Dados

Para expor um banco de dados na *grade* utilizando o OGSA-DAI, deve-se primeiro definir qual será o sistema gerenciador de bancos de dados, obter o *driver JDBC* oferecido pelo fornecedor e criar o banco de dados. A tabela abaixo detalha as escolhas feitas para esta exemplificação:

SGBD	PostgreSQL v8.2
Driver JDBC (classe/arquivo)	org.postgresql.Driver/psql-8.3-driver.jar
Banco de Dados	exemplo

Para expor o banco de dados 'exemplo', deve-se utilizar o *script* oferecido no pacote de instalação do OGSA-DAI. Para isso criou-se o banco de dados de nome 'exemplo', e definiu-se uma tabela simples contendo quatro colunas, criou-se também um usuário chamado 'exemplo' com senha 'exemplo' para acesso ao banco de dados. A exposição então é feita da seguinte maneira:

```
$ant -Dgt.dir=$GLOBUS_LOCATION -Dgt.target=true -Ddai.resource.id=exemplo \
-Ddai.db.product=PostgreSQL \
-Ddai.db.vendor=PostgreSQL -Ddai.db.version=8.2 \
-Ddai.db.uri=jdbc:postgresql://localhost/exemplo \
-Ddai.db.driver=org.postgresql.Driver \
-Ddai.resource.jar.dir=./libs \
-Ddai.user=exemplo dai.password=exemplo \
deployRelationalResource
```

O significado dos atributos são:

- *gt.dir* – Diretório contendo a instalação do *Globus Toolkit 4*.
- *dai.target.gt* – Middleware onde está implantado o OGSA-DAI é o *Globus Toolkit 4*
- *dai.resource.id* – Nome que representará o banco de dados exposto, não precisa ser necessariamente o mesmo nome do banco de dados.

- `dai.db.product` – SGBD utilizado.
- `dai.db.vendor` – Distribuidor do SGBD.
- `dai.db.uri` – URI contendo a localização do banco de dados.
- `dai.db.driver` – *Driver* JDBC do SGBD.
- `dai.resource.jar.dir` – Diretório contendo o pacote do tipo *jar* contendo o *driver*.
- `dai.user` – Nome de usuário a ser utilizado para o OGSA-DAI conectar ao banco de dados.
- `dai.password` – Senha para o usuário informado no atributo **`dai.user`**.

A partir destes dados o OGSA-DAI irá expor o banco de dados na *grade*.

2.1.2 Implantando o Componente Configurator

Para a implantação do componente *Configurator* na máquina *darkstar*, os seguintes comandos devem ser executados para implantar as *activities* *LoadConfiguration* e *StoreConfiguration*:

```
ant -Dgt.dir=$GLOBUS_LOCATION -Ddai.target.gt=true \
-Ddai.activity.id=LoadConfiguration \
-Ddai.activity.class=br.usp.pcs.lahpc.iicd.server.activities. \
LoadConfigurationActivity -Ddai.activity.jar.dir=./jars deployActivity
```

```
ant -Dgt.dir=$GLOBUS_LOCATION -Ddai.target.gt=true \
-Ddai.activity.id=StoreConfiguration \
-Ddai.activity.class=br.usp.pcs.lahpc. \
iicd.server.activities.StoreConfigurationActivity \
-Ddai.activity.jar.dir=./jars deployActivity
```

O último parâmetro dos comandos indicam a ação a ser tomada, neste caso ***deployActivity***, a qual irá implantar a *activity*. Para implantá-la os parâmetros que devem ser passados para o *script* `ant` do OGSA-DAI são:

- `dai.activity.id` – Um identificador único para a *activity*.
- `dai.activity.class` – A classe que representa a *activity*, e a qual contém todas as instruções de execução da mesma.
- `dai.activity.jar.dir` – Diretório contendo todas as bibliotecas as quais a *activity* depende para a sua execução.

Os comandos acima são a base para a implantação de todas as outras *activities* dos

demais componentes e subcomponentes. Após implantadas devemos definir quais serão os recursos as *activities* poderão manipular, para isso o seguinte comando deve ser executado.

```
ant -Dgt.dir=$GLOBUS_LOCATION -Ddai.target.gt=true \
-Ddai.resource.id=exemplo -Ddai.activity.id=LoadConfiguration \
-Ddai.activity.name=LoadConfiguration exposeResourceActivity
ant -Dgt.dir=$GLOBUS_LOCATION -Ddai.target.gt=true \
-Ddai.resource.id=exemplo -Ddai.activity.id=StoreConfiguration \
-Ddai.activity.name=StoreConfiguration exposeResourceActivity
```

O último argumento indica que o comando a ser executado é o ***exposeResourceActivity***, tal comando expõe um *resource* para uso em uma *activity*. Pode-se notar a especificação do identificador do recurso ao qual as *activities* poderão fazer uso, bem como identificado da *activity*. Os atributos para este comando são:

- dai.resource.id – Identificador do *resource* que poderá ser utilizado durante a execução de uma *activity*.
- dai.activity.id – Identificador da *activity* que fará uso do *resource*.
- dai.activity.name – Nome da *activity*.

2.1.3 Implantando os Demais Componentes

A implantação das demais *activities* desenvolvidas para executar a função dos demais componentes, seguem a mesma lógica da implantação do componente *Configurator* mostrada na seção anterior.

Para a implantação do *DataHandler*, as *activities* dos seus subcomponentes devem ser implantadas.

<i>Subcomponente</i>	<i>Noma da Activity</i>
DictionaryHandler	DictionaryInsert
	DictionaryRemove
	DictionaryUpdate
	DictionaryLookup

ReplicaManager	InsertReplicate
	UpdateReplicate
	DeleteReplicate

Para implantá-las utiliza-se a mesma sintaxe mostrada para a implantação do componente *Configurator*, modificando-se apenas o identificador (atributo **dai.activity.id**) e a classe da *activity* (atributo **dai.activity.class**).

É necessário também informar qual recurso deve ser utilizado pelas *activities* implantadas, segue também a mesma sintaxe executada para o componente *Configurator*, modificando-se apenas os atributos **dai.activity.id** e **dai.activity.name**.

2.1.4 Configurando a Infra-Estrutura

Uma vez instalados os componentes na máquina *darkstar*, o próximo passo consiste em configurar a infra-estrutura, definindo-se as configurações para as demais máquinas. O protótipo irá ler os arquivos de configuração que estejam no diretório de trabalho (*home*) do usuário dono da instalação do OGSA-DAI e do *Globus Toolkit* 4. Os arquivos são nomeados da seguinte forma:

<id>.iicd.configuration

Onde *id* é um identificador único para aquela configuração, definimos então 4 arquivos de configuração, são eles:

local.iicd.configuration
grid-node01.iicd.configuration
grid-node02.iicd.configuration
grid-node03.iicd.configuration

A máquina *darkstar* mantém as configurações dos nós da *grade*. Para que possa exercer a função de configurador foi instalado o componente *Configurator*, como detalhado anteriormente.

2.2 Instalando e Configurando as Demais Máquinas

As demais máquinas da infra-estrutura sendo montada, devem conter apenas o

componente *DataHandler*, para isso deve-se instalar e configurar o OGSA-DAI e o *Globus Toolkit 4*, bem como implantar as *activities* referentes a este componente. São elas: *DictionaryInsert*, *DictionaryRemove*, *DictionaryUpdate*, *DictionaryLookup*, *InsertReplicate*, *UpdateReplicate*, *DeleteReplicate*. O procedimento é o mesmo explicado na seção 2.1.2 e 2.1.3 deste apêndice.

Todas as máquinas pertencentes a infra-estrutura mantêm um arquivo de configuração local. Este arquivo deve conter informações sobre quem é o Configurador, caso não seja especificado, a configuração local deverá prevalecer. A configuração para as máquinas constituem em arquivos com pares “chave=valor”. As configurações destas máquinas são mostradas na tabela 9 deste documento.

3 Instalando as *activities* de suporte para *Driver JDBC*

Nesta seção abordaremos os passos necessários para o uso do *Driver JDBC* para OGSA-DAI.

2.1 Obetendo e Compilando o Driver

2.1.1 Instalando a Partir dos Pacotes Binários

São disponibilizados dois binários do *driver*, significando que o seu uso é feito diretamente sem necessidade de compilação. Os dois binários são, um para ser utilizado com o Apache Axis 1.4 implantado no Tomcat 5.x e outro para ser utilizado com o Globus Toolkit 4. Estes binários podem ser obtidos a partir do site do driver (<http://ogsadai-jdbc.sourceforge.net>).

As dependências para execução do driver estão anexadas nos binários, isso significa que para usá-lo, não é preciso configurar manualmente as bibliotecas utilizadas pelo *driver*. Porém em algumas situações esse comportamento não foi observado, sendo que as dependências tiveram que ser explicitadas manualmente, como é o caso do uso do *driver* em aplicativos *web* implantados no Apache Tomcat.

Neste caso os passos definidos na documentação do OGSA-DAI devem ser realizados.

2.1.2 Obtendo um Binário a Partir do Código-Fonte

É possível obter a versão mais recente utilizando-se a ferramenta SVN, o *driver* é gerenciado por um repositório *Subversion*. O *Subversion* é um gerenciador de versões. Para obter e compilar a versão mais atual, os seguintes passos devem ser feitos:

```
$svn co https://ogsadai-jdbc.svn.sourceforge.net/svnroot/ \
ogsadai-jdbc/tags/0.1beta1 ogsadai-jdbc-0.1beta1
```

```
$cd ogsadai-jdbc-0.1beta1
```

```
$export JAVA_HOME=/PATH/TO/JDK_1.4
```

```
$ant
```

Os passos acima consideram que esteja sendo utilizado um sistema *Unix*, com um *shell* do tipo *bash* e o pacote desenvolvimento Java versão 1.4.x.

2.1.3 Instalando as *Activities* de Suporte ao Driver JDBC

Nesta versão, tem-se duas *activities* que dão suporte à operação do *Driver* e que devem ser implantadas no servidor, são elas: *PackDatabaseMetaData* e *DatabaseMetaDataMethod*. Deve-se obter o pacote contendo as *activities* (deve ser obtido a partir do site oficial do driver <http://ogsadai-jdbc.sourceforge.net>) e seguir os seguintes passos para implantá-las no OGSA-DAI OGSA-DAI.

Supondo que o pacote de instalação do OGSA-DAI foi descompactado em um diretório chamado */home/globus/ogsa-dai* e que o arquivo com as *activities* esteja no diretório */tmp* faça:

```
$cd /home/globus/ogsa-dai
```

```
$mkdir jdbc-metadata-activities
```

```
$mv /tmp/jdbc_metadata_activities.jar ./jdbc-metadata-activities
```

As *activities* possuem uma dependência sobre a biblioteca *Xstream* cujo pacote deve ser baixado e também copiado para o diretório onde foi copiado o pacote com as *activities*, para isso execute:

```
$mv /tmp/xstream.jar ./jdbc-metadata-activities
```

O próximo passo é informar o OGSA-DAI sobre a existência das duas novas *activities*. Para isso dentro do diretório de instalação do OGSA-DAI, deve-se executar os seguintes comandos.

```
$ant -Dgt.dir=$GLOBUS_LOCATION \
-Ddai.activity.jar.dir=./jdbc-metadata-activities \
-Ddai.activity.id= \
br.usp.pcs.lahpc.ogsadai.activities.PackDatabaseMetaData \
-Ddai.activity.class= \
br.usp.pcs.lahpc.ogsadai.activities.metadata.PackDatabaseMetaDataActivity \
-Ddai.target.gt=true deployActivity
```

O comando acima irá informar ao OGSA-DAI da existência da *activity* *PackDatabaseMetaData* e irá copiar as dependências para o diretório de bibliotecas do *Globus Toolkit*. A variável de ambiente ***\$GLOBUS_LOCATION*** se refere ao diretório onde o *Globus Toolkit* foi instalado.

Para implantar a segunda *activity*, deve-se seguir os mesmos passos feitos para a *activity* anterior modificando somente alguns parâmetros, como mostra a lista de comandos a seguir:

```
$ant -Dgt.dir=$GLOBUS_LOCATION \
-Ddai.activity.jar.dir=./jdbc-metadata-activities \
-Ddai.activity.id= \
br.usp.pcs.lahpc.ogsadai.activities.DatabaseMetaDataMethodActivity \
-Ddai.activity.class= \
br.usp.pcs.lahpc.ogsadai.activities.metadata. \
DatabaseMetaDataMethodActivity \
-Ddai.target.gt=true deployActivity
```

O último passo a ser seguido é informar ao OGSA-DAI quais recursos (no caso Bancos de Dados Relacionais) que podem utilizar as *activities* recém implantadas. Para isso os seguintes comando devem ser executados:

```
$ant -Dgt.dir=$GLOBUS_LOCATION \
-Ddai.resource.id=my-data-resource-ID \
-Ddai.activity.id= \
br.usp.pcs.lahpc.ogsadai.activities.PackDatabaseMetaData \
-Ddai.activity.name= \
br.usp.pcs.lahpc.ogsadai.activities.metadata.PackDatabaseMetaData \
-Ddai.target.gt=true exposeResourceActivity \
```

```
$ant -Dgt.dir=$GLOBUS_LOCATION \
-Ddai.resource.id=my-data-resource-ID \
-Ddai.activity.id= \
br.usp.pcs.lahpc.ogsadai.activities.DatabaseMetaDataMethodActivity \
-Ddai.activity.name= \
br.usp.pcs.lahpc.ogsadai.activities.metadata. \
DatabaseMetaDataMethodActivity \
-Ddai.target.gt=true exposeResourceActivity
```

Após os passos listados, o *driver* poderá ser utilizado para uso em aplicações clientes.

APÊNDICE C - Exemplos de Testes Executados

1 Formas de Acessar uma Infra-Estrutura de Grade Montada utilizando-se o Protótipo da Arquitetura

Existem duas maneiras de acessar os recursos oferecidos pelo protótipo descrito. A primeira é utilizando-se as classes clientes das *activities*. A segunda é utilizando-se o *driver JDBC*.

1.2 Utilizando as Classes Cliente

Toda *activity* desenvolvida tem uma classe para utilização pelo cliente. O uso dessas classes permitem que programadores possam executar as *activities* em um servidor onde a mesma foi implantada.

O uso dessas classes seguem a mesma lógica de uso das classes do pacote de desenvolvimento cliente do OGSA-DAI. Os passos são os seguintes:

1. Cria-se um objeto para intermediar os acessos ao servidor
2. Solicita-se desta classe uma referência ao executor de requisições
3. Instancia-se as *activities* clientes.
4. Monta-se um *workflow*.
5. Envia-se o *workflow* para execução no servidor.

Segue um exemplo utilizado para testar as funcionalidades da activity *InsertReplicate*:

```
package br.usp.pcs.lahpc.activities;

import java.net.URL;

import uk.org.ogsadai.client.toolkit.DataRequestExecutionResource;
import uk.org.ogsadai.client.toolkit.PipelineWorkflow;
import uk.org.ogsadai.client.toolkit.RequestExecutionType;
import uk.org.ogsadai.client.toolkit.ServerProxy;
import
uk.org.ogsadai.client.toolkit.activities.delivery.DeliverToRequestStatus;
```

```

import uk.org.ogsadai.resource.ResourceID;
import br.usp.pcs.lahpc.iicd.client.activities.InsertReplicate;

public class InsertReplicateTest {

    public static void main(String[] args) throws Exception {
        ServerProxy server = new ServerProxy();
        server.setDefaultBaseServicesURL(new
URL("http://192.168.10.1:8080/wsrf/services/dai"));

        DataRequestExecutionResource drer =
server.getDataRequestExecutionResource(
        new ResourceID("DataRequestExecutionResource"));

        InsertReplicate insert = new InsertReplicate();
        insert.setResourceID("lbb");
        insert.setQuery("insert into littleblackbook values(100058,
'Mathias Brito'," +
            "'Francisco Benicio, 141', '9999999')");
        insert.setDictInfo("100058");
        DeliverToRequestStatus delivery = new DeliverToRequestStatus();

        delivery.connectInput(insert.getDataOutput());

        PipelineWorkflow pipeline = new PipelineWorkflow();
        pipeline.add(insert);
        pipeline.add(delivery);

        drer.execute(pipeline, RequestExecutionType.SYNCHRONOUS);

        System.out.println(insert.getRequestStatus());
    }
}

```

1.3 Utilizando o *Driver* JDBC

Uma versão modificada do *Driver JDBC* para OGSA-DAI está sendo desenvolvida para que o mesmo ao invés de chamar as *activities* regulares do OGSA-DAI, chamem as *activities* implementadas para o protótipo desenvolvido. O exemplo a seguir demonstra a utilização *driver*:

```

1: /* ... código acima omitido... */
2: Class.forName("br.usp.pcs.lahpc.ogsadai.jdbc.Driver");
3: Connection conn = DriverManager.getConnection(
        "jdbc:ogsadai://localhost:8080/dai/services/LBB_WN00");
4: Statement stmt = conn.createStatement();
5: stmt.execute("SELECT * FROM littleblackbook WHERE id < 10;");
6: ResultSet rs = stmt.getResultSet ();
7: while(rs.next()) {

```



```

8:      System.out.println( rs.getInt("id") + "\t"
+      rs.getString("name") + "\t"
+      rs.getString("address") + "\t"
+      rs.getString("phone"));
9:  }
10: /* ... código abaixo omitido ... */

```

2 Exemplos de Testes Realizados

Nesta seção abordaremos em mais detalhes os testes realizados para validar as funcionalidades do protótipo utilizado. Abaixo estão listados os testes realizados:

Componente/Subcomponente	Nome do Teste
DataHandler/ReplicaManager	DeleteReplicateTest
	InsertReplicateTest
	UpdateReplicate
DataHandler/DictionaryHandler	DicitionaryLookupTest
	DicitionaryInsertTest
	DicitionaryRemoveTest
	DicitionaryUpdateTest
	UpdateReplicateTest
Configurator	LoadConfigurationTest
	StoreConfigurationTest
JDBC Driver para OGSA-DAI	JDBCSupportTest
	JDBCTest
	CompareDriversTest

Além dos testes já citados, foram feitos testes de unidade no *Driver* JDBC, onde algumas funcionalidades eram testadas ao passo que eram desenvolvidas. A seguir está listado o teste *CompareDriversTest*.

```

import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.SQLException;

public class CompareDrivers {

    public static void main(String[] args) {

        Connection postgresCon = null;
        Connection ogsadaiCon = null;
        DatabaseMetaData postgresMD = null;
        DatabaseMetaData ogsadaiMD = null;

        try {
            Class.forName("org.postgresql.Driver");

            postgresCon = DriverManager.getConnection(
                "jdbc:postgresql://192.168.10.129/ogsadai",
"ogsadai",
                "ogsadai");

            Class.forName("br.usp.pcs.lahpc.ogsadai.jdbc.Driver");

            ogsadaiCon = DriverManager
                .getConnection("jdbc:ogsadai:gsi://192.168.10
.129:8443/wsrf/services/dai/lbb");
            postgresCon = DriverManager
                .getConnection("jdbc:postgresql://192.168.10.129/ogsadai"
, "ogsadai", "ogsadai");

            postgresMD = postgresCon.getMetaData();
            ogsadaiMD = ogsadaiCon.getMetaData();

        } catch (ClassNotFoundException e2) {
            e2.printStackTrace();
            System.exit(1);
        } catch (SQLException e2) {
            e2.printStackTrace();
            System.exit(1);
        }

        String posResult = new String();
        String ogsResult = new String();
        Method[] databaseMetaDataMethods =
DatabaseMetaData.class.getMethods();

        for (int i = 0; i < databaseMetaDataMethods.length; i++) {

            if (databaseMetaDataMethods[i].getParameterTypes().length
== 0
                &&
databaseMetaDataMethods[i].getReturnType().isPrimitive()) {

```

```

        try {
            int modifiers =
databaseMetaDataMethods[i].getModifiers();

            posResult =
databaseMetaDataMethods[i].invoke(postgresMD,
                                   null).toString();
            ogsResult =
databaseMetaDataMethods[i].invoke(ogsadaiMD,
                                   null).toString();

            System.out.print(databaseMetaDataMethods[i].getName()
                             + ": Postgres(" + posResult + ")"
                             + "<<<<>>> Ogsadai(" + ogsResult
+ ")");

            if
(posResult.toString().trim().equalsIgnoreCase(ogsResult.toString().trim()))
{
                System.out.println(" -- OK!");
            } else {
                System.out.println(" -- DIFF!!");
            }
        } catch (InvocationTargetException e) {
            e.getCause().printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
}
}

```

O teste apresentado anteriormente realiza chamadas a todos os definidas pela interface *DatabaseMetaData*, utilizando o *driver* do fabricante, no caso em questão do PostgreSQL e utilizando o *driver* desenvolvido para o OGSA-DAI, comparando os resultados e imprimindo na tela se eles diferem ou não.

O código fonte pode ser solicitado aos autores, para maiores detalhes sobre os testes realizados, bem como sobre a implementação do sistema.