Universidade Federal do Maranhão
Centro de Ciências Exatas e Tecnologia
Programa de Pós-graduação em Engenharia de Eletricidade

Bysmarck Barros de Sousa

Um Serviço de Metadados Integrado ao Middleware de Grade MAG

São Luís - MA
2006

Bysmarck Barros de Sousa

Um Serviço de Metadados Integrado ao Middleware de Grade MAG

Dissertação apresentada ao Programa de Pós-graduação em Engenharia de Eletricidade da UFMA, como requisito parcial para a obtenção do grau de MESTRE em Engenharia de Eletricidade.

Orientador: Francisco José da Silva e Silva

Doutor em Ciência da Computação - UFMA

Co-orientador: Mário Antônio Meireles Teixeira

Doutor em Ciência da Computação - UFMA

São Luís - MA

2006

Sousa, Bysmarck Barros de
Um Serviço de Metadados Integrado
ao Middleware de Grade MAG / Bysmarck Barros de Sousa - São Luis [s.n], 2006
125 f., il

1.Ciência da Computação 2. Sistemas Distribuídos 3. Grade de Computadores. I.Título.

CDU 004 (043)

Bysmarck Barros de Sousa

Um Serviço de Metadados Integrado ao Middleware de Grade MAG

Dissertação apresentada ao Programa de Pós-graduação em Engenharia de Eletricidade da UFMA, como requisito parcial para a obtenção do grau de MESTRE em Engenharia de Eletricidade.

Aprovado em 10 de julho de 2006

BANCA EXAMINADORA

Francisco José da Silva e Silva

Doutor em Ciência da Computação - UFMA

Mário Antônio Meireles Teixeira

Doutor em Ciência da Computação - UFMA

Djamel Fawzi Hadj Sadok

PhD em Ciência da Computação - UFPE

Maria Del Rosario Girardi Gutierrez

Doutora em Informática - UFMA

Aos meus pais, irmãs e sobrinhas

Resumo

A computação em grade permite a integração e compartilhamento de recursos como software, dados e periféricos, em ambientes instituicionais e multi-instituicionais. Recentemente, devido ao grande volume de dados gerado nos mais variados domínios de aplicação, aplicações que realizam computação intensiva sobre os dados tem sido largamente empregadas. Uma especialização de grade computadores, chamada Grade de Dados (*DataGrid*), tem proposto o desenvolvimento de uma infra-estrutura complementar às grades computacionais para permitir a integração e compartilhamento de grandes volumes de dados distribuídos geograficamente. Essa infra-estrutura complementar deve superar desafios como: heterogeneidade de sistemas de banco de dados, segurança, acesso aos dados com baixa latência, confiabilidade, transparência, publicação e descoberta dos dados. Em uma arquitetura de grade de dados, destacam-se dois serviços básicos e fundamentais: *serviço de dados* - que objetiva o armazenamento e recuperação de dados de forma transparente com relação à distribuição e heterogeneidade do ambiente; e *serviço de metadados* - que permite a publicação e descoberta dos dados que são compartilhados na grade através de metadados. Este trabalho descreve a arquitetura, implementação e aspectos de desempenho de um serviço de metadados que denominamos MagCat. O MagCat foca no desafio da publicação e descoberta dos dados compartilhados em uma grade de computadores. MagCat permite a publicação de dados usando como unidade básica arquivos. Ele permite o agrupamento desses arquivos de forma a facilitar a busca e permitir maior organização dos metadados. MagCat é independente do middleware de grade, e também do sistema de armazenamento sobre o qual são armazenados os metadados. MagCat foi desenvolvido usando a tecnologia de agentes de software, explorando a grande possibilidade de abstração desta tecnologia.

Palavras-chaves: grades de computadores, grades de dados, metadados, agentes.

Abstract

Grid computing allows the integration and sharing of resources, such as software, data, and peripherals, in multi-organization environments. Recently, a great amount of data is been generated in various application domains and data-intensive computing is being widely used. A grid computing specialization, called *DataGrid*, proposes a complementary infrastructure that allows the integration and sharing of large distributed data sets. Designers of such infrastructure must overcome several challenges, such as: highly heterogeneous database systems, security issues, provision of low-latency data access, reliability, transparency, data publication, and discovery. Two basic services can be distinguished in a *DataGrid* architecture: a *data service*, responsible for data storage and retrievement, and a *metadata service*, that allows data publication and discovery through the use of metadata. This work describes the architecture, implementation, and performance evaluation of a metadata service called MagCat. MagCat is responsible for data publication and discovery in grid environments. MagCat allows data publication using the concept of file as the basic unit of its data model. It allows file grouping, which facilitates data searching and organization. MagCat is self-contained and independent of the grid middleware. It is also independent of the storage system used to keep the metadata. MagCat was developed using the software agent paradigm, exploring the higher abstraction level of this technology.

Keywords: grid computing, data grid, metadata, software agents.

Agradecimentos

A meus pais, Maria Raimunda e José Luiz, pelo exemplo de honestidade, respeito, luta e incentivo em todos os momentos de minha vida.

Às minhas irmãs, Nadja e Goreth, pelo amor fraterno que tem me dedicado.

Ao professor Francisco José da Silva e Silva pela orientação, amizade e principalmente, pela paciência e confiança, sem as quais este trabalho não se realizaria.

Ao professor Mário Antônio Meireles Teixera, pela amizade e co-orientação, que contribuiram bastante para este trabalho.

Aos professores Maria Del Rosario Girardi Gutierrez e Djamel Fawzi Hadj Sadok, por terem aceito participar desta banca.

Aos membros do LSD, que não é uma droga, Rafael Fernandes, Stanley Araújo, Eduardo Viana e Estevão, pelo companheirismo, e Gilberto Cunha, pelo companheirismo e esforço desempenhado para o desenvolvimento deste trabalho.

A Carlos Eduardo dos Santos Araújo, pela sua amizade e estima.

A Emanoel Claudino e Alisson Lindoso pela amizade e discursões acerca deste trabalho que foram muito valorozas.

À CAPES, pelo financiamento desta pesquisa.

À Casa do Estudante Universitáro do Maranhão (CEUMA), sem a qual esta jornada talvez nem tivesse sido iniciada. Serei eternamente grato.

A Alcides Neto, pela amizade, momentos de descontração e contribuição para este trabalho no exercício de suas atividades como secretário do Programa de Pós-Graduação em Engenharia de Eletricidade.

A todos que direta ou indiretamente contribuiram para a conclusão deste.

"Sob a direção de um forte general, não haverá jamais soldados fracos".

Sócrates

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

Atualmente, dado o surgimento de aplicações cada vez mais complexas e destinadas a variados fins, existe uma crescente necessidade de poder computacional nos mais diversos domínios de atuação humana, seja no setor industrial ou no ambiente acadêmico. A satisfação dessas necessidades tradicionalmente vem aliada a altos custos e complexidade ligados à aquisição de hardware, software e infra-estrutura de rede sobre a qual as aplicações devem executar. Para evitar altos custos e complexidade, a computação em grade tem se tornado uma alternativa atraente, dado que pode-se utilizar o poder computacional existente no parque computacional das instituições, evitando assim, por exemplo, a aquisição de hardware destinado a supercomputação. Uma grade de computadores compreende uma infra-estrutura de hardware e software para integração e compartilhamento de recursos distribuídos como, software, dados e periféricos, entre ambientes instituicionais e multi-instituicionais.

Muitas aplicações científicas e industriais utilizam dados que se encontram geograficamente distribuídos, são computacionalmente intensivas e geram também um grande volume de dados. Para tratar com grandes volumes de dados distribuídos geograficamente, a computação em grade especializou-se, através de uma infra-estrutura complementar chamada Grade de Dados (*DataGrid*). Essa infra-estrutura complementar tem como desafios:

- Superar a heterogeneidade de sistemas de banco de dados: existe hoje uma variedade de sistemas de banco de dados no mercado, que diferem sensivelmente tanto na estruturação lógica dos dados (relacional, orientado a objeto, XML) quanto em termos de linguagem de acesso (SQL, OQL, XPath) [Bre90, OV91];

- Disponibilizar mecanismos de segurança: cada instituição possui políticas de segurança independentes para proteger os seus dados de usuários não autorizados, de forma que a possibilidade de compartilhamento, em um ambiente distribuído, impõe a criação de mecanismos de autenticação e autorização mais complexos, pois ao possibilitar o acesso aos dados em rede, são herdados problemas de manutenção de segurança das rede de computadores;

- Permitir o acesso aos dados com baixa latência: aplicações como, por exemplo, análise experimental e simulação em física de alta-energia, modelagem climática, engenharia de terremotos e astronomia, são utilizadas por comunidades de centenas de pesquisadores, os quais compartilham conjuntos maciços de dados. Tal compartilhamento impõe a necessidade de transferência de grandes conjuntos de dados para processamento, o que leva a uma alta latência para transferência;

- Apresentar alta confiabilidade: dada a característica distribuída do ambiente, falhas são comuns. Métodos de recuperação de falhas são necessários para tratar falhas da rede, bem como falhas na transferência dos dados;

- Prover diversos níveis de transparência: é desejável fornecer aos usuários uma interface uniforme para acesso aos dados distribuídos, que disponibilize o acesso, localização e concorrência de forma transparente aos usuários;

- Permitir a publicação e descoberta dos dados: dada a possibilidade do compartilhamento dos dados, um aspecto importante é como torná-los públicos. Assim, tem-se que fornecer mecanismos para que os pesquisadores e o público em geral venham a se beneficiar dos dados compartilhados, de modo que se faz necessário a existência de informações que possibilitem a publicação e descoberta desses dados.

A. Chervenak et. al [CFK+01] descrevem uma arquitetura geral para grades de dados e classificam dois serviços como fundamentais:

- *serviço de dados* - compreende mecanismos para acessar, gerenciar e executar transfrência dos dados armazenados no ambiente da grade, objetivando deixar transparente aos usuários da grade a característica distribuída do ambiente.

- *serviço de metadados* - metadado é definido popularmente como sendo "dados sobre dados". Em outras palavras, o metadado descreve ou qualifica um dado ou recurso qualquer, incorporando significado a ele. O serviço de metadados fornece mecanismos para nomear, publicar e acessar os diferentes tipos de metadados sobre os dados. Um dos desafios para o o ambiente de grade de dados é como identificar e localizar os dados que são de interesse para uma área particular . A solução é descrever as características dos dados através de metadados, e usar os metadados para identificar os dados relevantes.

1.1 Objetivo

O objetivo principal deste trabalho é a definição de uma arquitetura e implementação de um serviço de metadados a ser integrado ao middleware de grade denominado MAG (*Mobile Agents Technology for Grid Computing Environments*).

Os objetivos específicos deste trabalho são:

- Estudo do estado da arte em grades de dados;
- Explorar a tecnologia de agentes de software na implementação da arquitetura do serviço de metadados proposto, de forma a avaliar sua adequação e vantagens para este fim;
- Realização de avaliação de desempenho de aspectos ligados às funcionalidades a serem implementadas;
- Desenvolvimento de um estudo de caso compreendendo um aplicação computacionalmente intensiva sobre grande volume de dados para validação da implementação a ser realizada;
- Análise do impacto do mecanismo de execução de aplicações do *middleware* MAG para execução da aplicação a ser desenvolvida.

1.2 Estrutura da Dissertação

Esta dissertação descreve a arquitetura, implementação e avaliação de desempenho do serviço de metadados chamado MagCat. Ela está organizada como a seguir: no Capítulo 2 apresentamos o estado da arte em computação em grade; no Capítulo 3 apresentamos o *middleware* sobre o qual a implementação realizada foi integrada e testada; no Capítulo 4 descrevemos a arquitetura do serviço de metadados proposto neste trabalho e a implementação realizada; no Capítulo 5, apresentamos um estudo de caso e avaliação de desempenho da implementação realizada para validação do serviço de metadados; no Capítulo 6 apresentamos trabalhos relacionados, onde descrevemos características dos mesmos e fazemos um paralelo com o trabalho aqui proposto e; no Capítulo 7 apresentamos nossas conclusões e trabalhos futuros.

2 Grades de Dados

Atualmente, variados domínios de aplicação do conhecimento humano como, por exemplo, modelagem climática, física de alta energia, astronomia e aplicações médicas, tem gerado um grande volume de dados. Freqüêntemente, a análise dessas grandes quantidades de dados demandam por grande capacidade de processamento e resultam em mais informações a serem armazenadas. Tipicamente existe grande colaboração entre centros de pesquisas de áreas afins, de forma que o gerenciamento distribuído dos dados deve permitir não somente armazenamento e recuperação dos dados, mais também permitir o acesso colaborativo, rápido, catalogação com informações descritivas para facilitar a recuperação e ainda torná-los disponíveis para computação na grade.

Este capítulo apresenta o conceito de *middlewares* para grades computacionais e grades de dados, os desafios a serem tratados pela infra-estrutura destes middlewares e também alguns projetos de *middlewares* para grades de computadores e grades de dados. É explanado de forma sucinta a taxonomia de grades computacionais e é apresentada uma taxonomia de organização de grades de dados. Ademais, é apresentada uma arquitetura geral para grades de dados e introduzimos a motivação para o uso de metadados em grades de dados.

2.1 Introdução a Grades de Computadores

Atualmente, ramos de atividades científicas, comerciais e industriais como biologia, processamento de imagens para diagnóstico médico, previsão do tempo, física de alta energia, previsão de terremotos, simulações mercadológicas, prospecção de petróleo e computação gráfica, têm demandado por grande capacidade de processamento, compartilhamento de dados e colaboração entre usuários e organizações, motivando avanços tecnológicos em arquiteturas de hardware e software, bem como em redes de computadores e sistemas distribuídos.

Porém, a obtenção de maior capacidade computacional depende de altos investimentos em hardware de alto desempenho e software, chegando a custar milhares de doláres, que possivelmente compreendem recursos financeiros indisponíveis.

A Internet trouxe uma forma revolucionária de conexão entre sistemas, mas certos recursos, como softwares e periféricos, ainda não podem ser compartilhados e acabam sub-utilizados. Em uma rede que se encontra sob um único domínio administrativo, ou seja, um conjunto de máquinas que estão sujeitas a um determinado conjunto de políticas e restrições estabelecidas por alguma autoridade local, é comum que exista o compartilhamento de recursos como discos rígidos e impressoras, e colaboração entre os usuários, mas quando a rede ultrapassa um domínio administrativo, tal compartilhamento e colaboração tornam-se limitados, devido principalmente a questões técnicas e políticas de segurança.

Nesse contexto, a Computação em Grade (*Grid Computing*)[AI99, BBL00], em especial grades oportunistas [GKG+02], emerge como solução em infra-estrutura de hardware e software para integração de recursos distribuídos, aproveitamento de recursos ociosos, compartilhamento e colaboração. O *middleware*[1] de grade é o componente de software responsável pela integração dos recursos distribuídos, de modo a criar um ambiente unificado para o compartilhamento de dados, recursos computacionais e execução de aplicações, compondo ambientes dinâmicos e multi-institucionais, onde organizações compartilham entre si capacidade e recursos computacionais (software, dados, periféricos) colaborativamente. Além disso, propõe a utilização da quantidade considerável de capacidade computacional ociosa, representada pelo parque computacional de muitos ambientes organizacionais, onde computadores pessoais passam longos períodos sem utilização por parte de seus usuários, em favor da resolução de problemas. O *middleware* de grade permite que a capacidade individual de cada computador possa ser utilizada em conjunto, ao contrário de fazer-se altos investimentos em hardwares de alto desempenho. A origem do termo *Grid Computing* deriva de uma analogia com a rede elétrica (*Power Grid*), e reflete o objetivo de tornar

[1]middleware é um software que reside entre o sistema operacional e a aplicação a fim de facilitar o desenvolvimento de aplicações, escondendo do programador diferenças entre plataformas de hardware, sistemas operacionais, bibliotecas de comunicação, protocolos de comunicação, formatação de dados, linguagens de programação e modelos de programação.

o uso de recursos computacionais distribuídos tão simples quanto em qualquer lugar do globo ligar um aparelho na rede elétrica.

O compartilhamento multi-instituicional e dinâmico proposto pela computação em grade deve ser, necessariamente, altamente controlado, com provedores de recursos e consumidores definindo claramente e cuidadosamente o que é compartilhado, a quem é permitido compartilhar, e as condições sob as quais ocorre o compartilhamento. Ian Foster et. al [FKT01], conceituam os grupos formados por indivíduos e/ou organizações que compôem esses ambientes dinâmicos e multi-institucionais como Organizações Virtuais (OV).

Para possibilitar o compartilhamento de recusos computacionais e colaboração, a infra-estrutura de grades computacionais deve tratar questões como:

- Heterogeneidade de plataformas de hardware: um sistema de computação em grade pode, eventualmente, lidar com recursos heterogêneos, necessitando fornecer mecanismos que tratam essa diversidade de plataformas;
- Interoperabilidade: uma infraestrutura de grade de computadores deve seguir padrões na construção de suas interfaces, de forma a fomentar a interoperabilidade entre os *middlewares* de grade e o objetivo de integração em escala global;
- Custo reduzido: o uso de grade deve se constituir em meio alternativo aos altos custos de hardware de alto desempenho e software;
- Gerenciamento de recursos: mecanismos que possibilitem o controle de recursos e monitoramento são essenciais para a alocação correta de acordo com requisitos pré-estabelecidos, balanceamento e escalonamento de tarefas para grade [GKG+02, JF04];
- Qualidade de serviço: com uso colaborativo da grade, deve ser garantido que proprietários de estações de trabalho não tenham suas aplicações afetadas devido às aplicações da grade que executam em seus recursos. O eventual uso comercial da grade deve ser negociado por meio de *Service Level Agreement* (SLA) entre usuários e provedores de serviço para garantir níveis adequados de custo e prestação de serviço [MC04];

- Serviços de informação: informações sobre a estrutura e o estado dos recursos (configuração, disponibilidade, carga de trabalho, e políticas de uso) são essencias para o gerenciamento dos mesmos, dado que estas informações permitem aos gerenciadores de recursos alocá-los de maneira mais eficiente [JF04];

- Políticas de segurança: devido aos recursos computacionais encontrarem-se em diversos domínios administrativos, eventualmente, existem várias políticas de segurança. Assim, a infra-estrutura de grade deve tratar questões de integração de políticas de segurança locais, mapeando identidades e autorizações, provendo acesso autenticado e confiável, bem como garantindo privacidade e a autonomia administrativa local;

- Tolerância a falhas: tratar falhas em grades é bastante complexo, visto que as grades agregam uma enorme quantidade de componentes de hardware e software, eventualmente heterogêneos, que fazem a probabilidade de falhas aumentar proporcionalmente ao tamanho do sistema, bem como a identificação da origem da falha seja dificultada.

Há atualmente vários projetos de sistemas de computação em grade em desenvolvimento, entre os quais destacam-se:

- Globus [FKT01] é um projeto de computação em grade que atualmente é desenvolvido a partir do padrão *Open Grid Service Architecture* (OGSA) [FKNT02], que objetiva utilizar a tecnologia de serviços web para fomentar maior interoperabilidade entre as plataformas de grade;

- Legion [GLFK00, LG96] é um sistema de computação em grade baseado em objetos. Em Legion, todos os elementos da grade são representados por objetos, sejam eles computadores, dispositivos de armazenamento, dados, aplicações, entre outros. Tais objetos fazem uso do substrato básico provida por Legion, denominada camada de objetos núcleo;

- Condor [TTL03] é um sistema de computação em grade que possibilita a agregação de computadores, especialmente estações de trabalho ociosas, para utilizar a capacidade de processamento coletiva afim de executar aplicações de alto desempenho[ML97], mas sem prejudicar o proprietário do recurso;

- MyGrid [CPC+00] é um projeto de computação em grade desenvolvido na Universidade Federal de Campina Grande, que objetiva a execução de aplicações paramétricas, também conhecidas como *Bag of Task*s (BoT). Aplicações paramétricas são aplicações paralelas nas quais as tarefas são independentes uma da outra. Este projeto é diferente de outros projetos existentes, pois omite o suporte a aplicações arbitrárias em favor de suportar somente aplicações BoT;

- InteGrade [GKGF03, GKG+02] é um projeto de computação em grade, desenvolvido pela Universidade de São Paulo (USP), Universidade Federal do Mato Grosso do Sul (UFMS), Pontifícia Universidade Católica do Rio de Janeiro (PUC/RIO), Universidade Federal de Goiás (UFG) e Universidade Federal do Maranhão (UFMA). Seu objetivo é a construção de um middleware que permita a utilização de recursos computacionais ociosos, dedicados ou não, de maneira a aproveitar sua capacidade ociosa em tarefas de computação de alto desempenho. O InteGrade utiliza a tecnologia de objetos distribuídos CORBA como base de sua implementação, suporte a execução de aplicações pararelas, independentes ou não, além de aplicações convencionais.

Krauter et al. [KBM02] propõem a seguinte taxonomia para sistemas de grades de computadores:

- Grade Computacional (*Computing Grid*): é um sistema de computação em grade que objetiva prover maior capacidade computacional através da agregação de recursos computacionais distribuídos. Como exemplo de classes de aplicações temos: supercomputação distribuída, cujas aplicações usam grades computacionais para agregar recursos computacionais substanciais para resolver problemas que não poderiam ser resolvidos por um único sistema como, por exemplo, aplicações para planejamento e treinamento militar através de simulações interativas distribuídas e simulação precisa de processos físicos complexos; modelagem climática; alto rendimento, onde a grade é usada para escalonar grande número de tarefas independentes ou fracamente acopladas, com o objetivo de utilizar ciclos de processadores ociosos como, por exemplo, a resolução de problemas criptográficos;

- Grade de Serviços (*Service Grid*): provê serviços viabilizados pela integração de diversos recursos computacionais. Exemplos de classes de aplicações são:

  - Computação sob demanda: aplicações usam as capacidades da grade por períodos curtos de acordo com solicitações por recursos como, por exemplo, aplicações de instrumentação médica e requisições de utilização de software especializado por usuários remotos;
  - Computação colaborativa: aplicações que utilizam a grade para dar apoio a trabalhos cooperativos envolvendo diversos participantes, como, por exemplo, em projetos colaborativos e educação;
  - Multimídia: a grade provê a infra-estrutura para aplicações multimídia em tempo real.

- Grade de Dados (*Data Grid*): é um sistema de computação em grade que objetiva prover mecanismos especializados para publicação, descoberta, acesso e classificação de grandes volumes de dados distribuídos. A computação intensiva de dados cujo foco está na síntese de novas informações de dados que são mantidos em repositórios, bibliotecas digitais e bancos de dados geograficamente distribuídos são exemplos de grade de dados.

2.2 Introdução a Grades de Dados

Atualmente, existe um grande volume de dados gerado nos mais variados domínios de aplicação como, por exemplo, modelagem climática, física de alta energia, astronomia e aplicações médicas, originando grandes coleções de dados, que se encontram geograficamente distribuídas em diversos centros de pesquisa [BBL00]. Tipicamente, nesses centros, realizam-se experimentos que requerem computação de alto desempenho e que levam um longo tempo de processamento sobre conjuntos maciços de dados compartilhados entre as comunidades científicas. A combinação de grandes coleções de dados, distribuição geográfica de dados e usuários, e ainda a computação de alto desempenho e intensiva sobre os dados demanda por mecanismos sofisticados para superar desafios de acesso e compartilhamento de dados.

Os principais desafios para o acesso e compartilhamento de dados em ambiente distribuído são:

- Heterogeneidade de sistemas de banco de dados: existe uma variedade de sistemas de banco de dados no mercado, que diferem sensivelmente tanto na estruturação lógica

dos dados (relacional, orientado a objeto, XML) quanto em termos de linguagem de acesso (SQL, OQL, XPath) [Bre90, OV91];

- Segurança: cada instituição possui políticas de segurança independentes para proteger os seus dados de usuários não autorizados, de forma que a possibilidade de compartilhamento, em um ambiente distribuído, impõe a criação de mecanismos de autenticação e autorização mais complexos, pois ao possibilitar o acesso aos dados em rede são herdados problemas de manutenção de segurança das rede de computadores;

- Acesso aos dados com baixa latência: aplicações como, por exemplo, análise experimental e simulação em física de alta-energia, modelagem climática, engenharia de terremotos e astronomia, são utilizadas por comunidades de centenas de pesquisadores, os quais compartilham conjuntos maciços de dados. Tal compartilhamento impõe a necessidade de transferência de grandes conjuntos de dados para processamento, o que leva a uma alta latência para transferência;

- Confiabilidade: dada a característica distribuída do ambiente, falhas são comuns, métodos de recuperação de falhas são necessários para tratar falhas da rede, bem como falhas na transferência dos dados;

- Transparência: é desejável fornecer aos usuários uma interface uniforme para acesso aos dados distribuídos, de forma a superar aspectos de transparência como, por exemplo, acesso, localização e concorrência;

- Publicação e descoberta dos dados: dada a possibilidade do compartilhamento dos dados, um aspecto importante é como torná-los públicos. Assim, tem-se que fornecer mecanismos para que os pesquisadores e público em geral venham a se beneficiar dos dados compartilhados, de modo que se faz necessário a existência de informações que possibilitem a publicação e descoberta desses dados.

Para superar os desafios citadas anteriormente, a computação em grade especializou-se, objetivando fornecer também a infra-estrutura para manipular grandes volumes de dados[FK99, FKT01]. Essa especialização é chamada de Grade de Dados (*DataGrid*). Um *middleware* para grade de dados integra em sua arquitetura:

- Uma API uniforme: para acessar sistemas de banco de dados heterogêneos, ocultando do usuário a estruturação lógica dos dados, bem como a linguagem de acesso;

- Mecanismos de segurança: que garantam acesso autenticado e autorizado aos dados;

- Mecanismos de replicação e gerenciamento de réplicas: podem-se criar cópias locais ou réplicas para superar possíveis grandes latências de transferência, prover maior disponibilidade e confiabilidade. Dada a possibilidade de replicação, esse serviço deve permitir a localização de réplicas e determinar a melhor estratégia para acessar um dado, bem como garantir a consistência das várias réplicas[OV91, DHJM+01, ABB+02, CFK+01, GKL+02];

- Mecanismos de transferência de dados: a replicação geralmente envolve uma grande quantidade de dados (terabytes, petabytes), de forma que, aliado ao serviço de replicação, se faz necessário protocolos que garantam alto desempenho [ABB+01] face às transferências de grandes volumes de dados;

- Mecanismos de publicação e descoberta de dados: a descoberta dos dados é impraticável sem informações adequadas para descrevê-los, bem como mecanismos que possibilitem a publicação e acesso a essas informações. Geralmente essas informações são chamadas de metadados. Metadado é o dado que descreve o conteúdo, formato ou atributos de um dado ou recurso qualquer. Popularmente, metadado é conhecido como "dado sobre o dado".

2.3 Serviços para Grades de Dados

A. Chervenak et. al [CFK+01] propõem uma arquitetura geral para grades de dados, ilustrada na Figura 2.1, na qual os serviços são divididos em:

- Serviços básicos: que compreende serviços específicos de grades de dados e serviços gerais de grades de computadores, tais como:

  - Sistemas de armazenamento, que podem ser implementados por qualquer tecnologia de armazenamento, que ofereça um conjunto de funções para criar, apagar, ler, escrever e manipular dados. Exemplos de sistemas de armazenamento são:

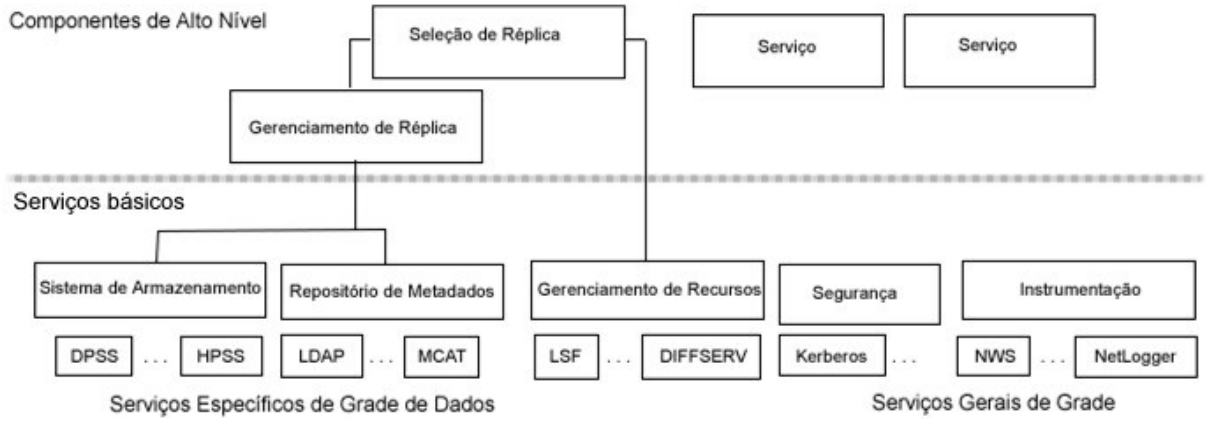


Figura 2.1: Arquitetura geral para grades de dados

* *Distributed Parallel Storage System* (DPSS) [TLC+99] e *High Performance Storage System* (HPSS), que proveêm acesso em alta velocidade a dados e transferência paralela e/ou utilização de múltiplos servidores;
* *Lightweight Directory Access Protocol*[2] (LDAP): um protocolo para acessar serviços de diretórios;
* *Storage Resource Broker* (SRB), que objetiva integrar sistemas de armazenamento heterogêneos através de uma API uniforme e um catálogo de metadados chamado MCAT para descrever e localizar os dados [BMRW98].

– API de acesso, que tem como requisito o suporte a requisições remotas para leitura e escrita, transferência entre sistemas de armazenamento, integração de políticas de segurança para acesso remoto, além de contemplar aspectos de tolerância a falhas, no que diz respeito a reportar erros ocorridos para as aplicações, para possível tratamento;

– Serviços gerais de grades: gerenciamento de recursos, segurança e instrumentação;

– Repositório de metadados: compreende o gerencimento de informações sobre a própria grade de dados, incluindo informações sobre as instâncias de arquivos, dos vários sistemas de armazenamento contidos na grade e informações de réplica, oferecendo um mecanismo uniforme para nomear, publicar e acessar os diferentes tipos de metadados, desde metadados que descrevem o

[2]http://www.ietf.org/rfc/rfc2251.txt.

conteúdo e estrutura dos dados àqueles que descrevem detalhes dos sistemas de armazenamento como capacidade e políticas de uso.

- Serviços de alto nível, que compreendem serviços como:
  - Serviço de gerenciamento de réplica, que utiliza as funcionalidades do sistema de armazenamento e repositório de metadados. O papel do serviço de gerenciamento de réplica é criar réplicas dentro da grade de dados, levando em consideração características de desempenho e disponibilidade;
  - Serviço de localização de réplica, que objetiva otimizar o acesso aos dados por parte das aplicações, considerando critérios de desempenho, custo ou segurança. O dado selecionado pode ser local ou acessado remotamente. Alternativamente, o processo de seleção pode iniciar a criação de uma nova réplica dada a possibilidade de melhor desempenho em relação às cópias existentes. A ação de iniciar o processo de criação de nova cópia é assistido por serviços de informação sobre o desempenho da rede, largura de banda ou características do tamanho do dado.

Dos serviços apresentados, A. Chervenak et. al [CFK+01] classificam dois serviços como fundamentais:

- *serviço de dados* - compreende mecanismos para acessar, gerenciar e executar transfrência dos dados armazenados no ambiente da grade, objetivando deixar transparente aos usuários da grade a característica distribuída do ambiente.
- *serviço de metadados* - fornece mecanismos para nomear, publicar e acessar os diferentes tipos de metadados sobre os dados. Um dos desafios para o ambiente de grade de dados é como identificar e localizar os dados que são de interesse para uma área particular . A solução é descrever as características dos dados através de metadados, e usar os metadados para identificar os dados relevantes.

2.4 Taxonomia de Grades de Dados

Srikumar Venugopal et. al [VBR05] descrevem uma taxonomia para grades de dados classificando-as de acordo com a organização (p. ex. hierárquica, inter ou

intradomínio, colaborativa, etc.), tecnologias usadas para transporte de dados (p. ex. criptografia, cache dos dados, compressão, etc.), mecanismos e políticas de replicação e políticas de escalonamneto/alocação de recursos. Devido à existência de grandes volumes de dados e a presença de múltiplas réplicas o escalonamento de tarefas de computação intensiva sobre dados é diferente do escalonamento de outros tipos de tarefas, pois os escalonadores têm que levar em consideração a disponibilidade de largura de banda e a latência da transferência entre o nó onde a tarefa será executada e o sistema de armazenamento de onde o dado será obtido.

Nesta seção explanaremos apenas a classificação quanto à organização, haja visto o objetivo desta dissertação não compreender a construção do *middleware* de grade de dados em sua completude, mas de um serviço de metadados.

2.4.1 Taxonomia de organização

Figura 2.2: Taxonomia de organização de grades de dados

A figura 2.2 ilustra a classificação das grades de dados de acordo com aspectos organizacionais relacionados a: modelo, escopo, OV, fonte de dados e gerenciamento do *middleware*.

Modelo

É a forma como as fontes de dados estão organizadas no sistema. Dentre eles destacam-se:

- Hierárquica: neste modelo existe uma fonte centralizadora dos dados, que realiza a distribuição para as demais fontes;
- Federada: é o modelo mais comum, dado que cada instituição que deseja compartilhar dados já possui seus sistemas de armazenamento de dados. Neste modelo, cada instituição preserva sua autonomia sobre a base de dados local;
- Sensor: este é aplicado em projetos como *Network for Earthquake Engineering Simulat*ion (NEESgrid)[RWM+04], onde os dados obtidos de sensores são enviados para uma base de dados central, que é consultada por pesquisadores. Neste modelo, o fluxo dos dados é dos sensores para a base de dados, conferindo uma característica *bottom-up* à obtenção de dados;
- Híbrida: este modelo é a junção dos modelos hierárquico, federado e sensor;

Escopo

O escopo de grades de dados pode variar dependendo se é específica a um único domínio, ou se a infra-estrutura é genérica o suficiente para abranger várias áreas científicas.

Organização Virtual (OV)

Grades de dados são formadas por Organizações virtuais [FKT01]. De acordo com sua organização social, uma OV pode ser classificada como:

- Colaborativa: as OVs participantes compartilham e colaboram para alcançar um objetivo comum;
- Regulada: as OVs podem ser controladas por uma única organização que estabelece regras para acessar e compartilhar recursos;

- Baseada em economia: os provedores de recurso colaboram com os consumidores devido a motivos comerciais. Neste caso, acordos de serviço estabelecem requisitos de QoS na grade [MC04];

- Baseada em reputação: uma OV é criada a partir de convites de outras entidades, baseado sobre a qualidade dos serviços que elas fornecem.

Fonte de dados

As fontes de dados em uma grade de dados podem ser transientes ou estáveis. Um cenário para uma fonte de dados transiente é um satélite que transmite dados em periódos determinados do dia. Nesses casos, as aplicações necessitam estar cientes da vida curta do fluxo de dados, pois as informações são produzidas e armazenadas durante um certo intervalo de tempo;

Gerenciamento

O gerenciamento em grades de dados pode ser automático ou manual. Atualmente as grades de dados são gerenciadas através de intervenção humana, mesmo que minimamente, na execução de tarefas como monitoramento de recursos, autorização de usuários e replicação de dados.

2.5 Middlewares para Grades de Dados

2.5.1 Storage Resource Broker (SRB)

SRB [BMRW98, ASKF03] é um middleware desenvolvido pelo *San Diego Supercomputing Center* (SDSC), que provê uma interface uniforme para acessar e manipular de forma transparente dados distribuídos e replicados sobre sistemas de armazenamento heterogêneos.

A Figura 2.3 ilustra de forma simplificada a arquitetura do SRB, que é composta por três componentes:

- Um catálogo de metadados chamado MCAT, que é um repositório de metadados implementado sobre um sistema de banco de dados relacional. MCAT mantém informações sobre os dados, autenticação, autorização e auditoria
- Servidor SRB, que é responsável por gerenciar os sistemas de armazenamento;
- API de acesso aos servidores e ao catálogo de metadados MCAT.

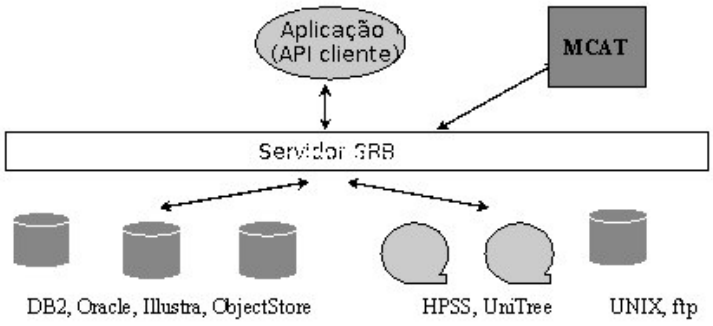


Figura 2.3: Arquitetura SRB

SRB é uma solução para grades de dados bem madura, e entre suas caracterísctcias destacamos:

- Provê quatro métodos de autenticação: senha criptografada e não criptografada, certificado SEA[3] e *Grid Security Infrastructure* (GSI) [FKTT98, BWE+00]. SEA é um mecanismo de autenticação e criptografia baseado em criptografia usando RSA[4] desenvolvido pelo SDSC;
- Possui um mecanismo para controle de acesso a dados através de um conceito chamado *ticket*, que representa privilégios sobre os dados;
- API's de acesso em C, C++, Java, e Python;
- Um catálogo de metadados chamado MCAT para armazenar metadados de localização, de autenticação de usuários, de log do sistema, e descrição dos dados;
- Suporta entrega de dados usando transferência serial ou paralela. Usando transferência paralela, múltiplas conexões são abertas entre o cliente e o sevidor, e os dados são enviado em pararelo sobre cada conexão;

[3]http://www.sdsc.edu/ schroede/sea.html.

[4]RSA é um algoritmo de criptografia baseado em chave pública, que deve o seu nome a três professores do Instituto MIT, Ron **R**ivest, Adi **S**hamir e Len **A**dleman, que inventaram este algoritmo.

- O acesso a recursos distribuídos é provido através de SRB federados, onde cada servidor gerencia um determinado tipo de sistema de armazenamento e permite que um servidor SRB atue como cliente de outro servidor SRB. A federação permite que uma aplicação cliente tenha acesso a um servidor SRB remoto sem conectar-se diretamente a ele;

- Possibilita replicação entre os servidores federados.

SRB é utilizado em vários projetos de grades de dados, em vários domínios de aplicação, como astronomia, bioinformática, biologia, ecologia, educação, geologia, física de partículas, estudos geodésicos e sísmicos, medicina molecular, neurociência, entre outros. Dentre os projetos há o *UK eScience* (eDiaMoND)[5], *BaBar*[6], *BIRN*[7], *IVOA*[8] e *California Digital Library* [RMLZ02].

2.5.2 OGSA-DAI

Open Grid Services Architecture Data Access and Integration (OGSA-DAI) é um middleware que provê uma interface uniforme para acessar sistemas de banco de dados em grades de dados, ocultando aspectos de heterogeneidade como formato dos dados, *drivers* de acesso e mecanismos de transporte dos dados. Entre suas características destacam-se:

- Permite o armazenamento e acesso a dados utilizando bancos de dados relacionais, XML e sistemas de arquivos;

- Disponibiliza transporte dos dados através de SOAP sobre HTTP, FTP e GridFTP;

- A segurança é provida por certificados X.509;

- Mecanismos de transformação dos dados (XSLT, ZIP, GZIP);

- O conceito *Activity*, que possibilita extensão das suas funcionalidades [HBM].

A Figura 2.4 ilustra a arquitetura do OGSA-DAI, que é composta por três componentes:

[5]http://www.ediamond.ox.ac.uk/.
[6]http://www.ppdg.net/.
[7]http://www.nbirn.net.
[8]http://www.ivoa.net/.

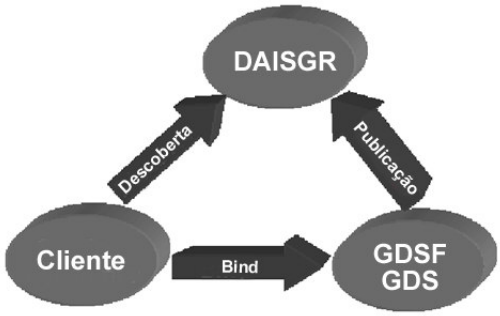


Figura 2.4: Arquitetura OGSA-DAI

- *Data Access and Integration Service Group Registry* (DAISGR): permite o registro e descoberta de fontes de dados. Um cliente pode usar o DAISGR para identificar o provedor que detém os dados desejados;
- *Grid Data Service Factory* (GDSF): atua como um ponto de acesso persistente à fonte de dados. Cada fonte e tipo de sistema de armazenamento de dados possui um GDSF independente;
- *Grid Data Service* (GDS): atua como um ponto de acesso transiente aos dados. É através do GDS que a aplicação cliente interage com a fonte de dados. GDS's são criados por um GDSF, e acessam o sistema de armazenamento usando *drives* específicos, e componentes adicionais como formatadores de dados e mecanismos de transporte. Aplicações podem usar GDS's diretamente para acessar sistemas de armazenamentos individuais, ou podem usar um mecanismo de processamento de consulta ditribuída, OGSA-DQP [NAN+03], para coordenar o acesso a múltiplos sistemas de armazenamento.

2.5.3 Globus

O Globus *toolkit* provê suporte para o gerenciamento de dados em grades de computadores através de componentes como:

- *Replica Location Service* (RLS) [CPB+04, CDF+02], que é um serviço para localização de réplicas que mantém informações sobre a localização física das cópias de dados. Os principais componentes são o *Local Replica Catalog* (LRC), que mapeia a representação lógica para a localização física, e o *Replica Location Index* (RLI), que indexa o próprio catálogo;

- GridFTP [Pro00, ABB+02, ABK+05], que é uma extensão do *File Transfer Protocol* (FTP) adicionando novas características como:
  - Mecanismo de segurança através de *Grid Security Infrastructure* (GSI) [FKTT98, BWE+00] que utiliza Kerberos para autenticação;
  - Transferência paralela;
  - Transferência parcial dos dados;
  - Transferência *third party*, que permite um usuário ou aplicação remota iniciar, monitorar e controlar a operação de transferência entre dois ambientes remotos distintos (a origem e o destino para a transferência);
  - Transferência *striped* de dados, o que significa que os dados são transferidos usando-se múltiplas *streams Transmission Control Protocol* (TCP) entre o servidor origem e o destino;
  - Configuração manual ou automática negociação de tamanhos de *buffer/window*.
- *Reliable File Transfer* (RFT) [RKM02], que propõe um mecanismo para tornar a transferência de dados com GridFTP mais eficiente, por detectar uma variedade de falhas e reiniciar a transferência do ponto onde ocorreu a falha;
- *Data Replication Service* (DRS) [CSK+05], que combina RLS e GridFTP para permitir replicação de dados;
- *Open Grid Service Architecture - Data Access and Integration* (OGSA-DAI).

2.6 Motivação para Uso de Metadados

Em termos simples, metadado é definido popularmente como sendo "dados sobre dados". Em outras palavras, o metadado descreve ou qualifica um dado ou recurso qualquer, incorporando significado a ele. Sem metadado, a informação se restringe a um conjunto de dados sem significado. Visto que metadado também é dado, é possível ter metadado sobre metadado do dado, metadado sobre o metadado do metadado, e assim sucessivamente. Para identificar e diferenciar metadado de dado podemos considerar a seguinte definição: se X é dado e $X \longrightarrow Y$ representa um relacionamento descritivo tal

que X descreve Y, então X é metadado sobre Y. Se não existe relacionamento com Y, então X é simplesmente dado e não metadado.

Haynes [Hay04] destaca a importância de metadados, argumentando o seguinte:

- Metadados tornam as consultas mais eficientes: metadados podem tornar as consultas mais eficientes por estabelecer um contexto através de descritores individuais, atributos. Por exemplo, a palavra "Silvestre" pode indicar o criador ou autor de um documento, indicando o nome individual, enquanto "silvestre" no título de um documento pode corresponder ao assunto ao qual pertence o documento. Metadados apropriados podem permitir aos mecanismos de busca localizar as informações de maneira discriminativa, servindo como filtros de busca e evitando o retorno de informações inúteis para o objetivo da busca. Contudo, a maior eficiência obtida pode ter como conseqüência uma perda de desempenho;

- Metadados fornecem uma forma para gerenciar documentos eletrônicos: sistemas de gerenciamento de conteúdo usam metadados para registar quando foram realizadas atualizações, quem foi o responsável pela criação e quais as permissões de acesso existentes;

- Metadados auxiliam na determinação da autenticidade do dado: permitem a realização de auditoria sobre a autenticidade e propriedades do recurso, por possibilitar o registro de informações acerca do histórico das manipulações sofridas pelo recurso durante seu ciclo de vida;

- Metadados possibilitam a interoperabilidade: interoperabilidade depende da troca de metadados entre os sistemas, de forma a estabelecer a natureza do dado sendo transferido e como ele pode ser identificado e tratado.

Como pode-se notar, existe uma variedade de propósitos para metadados, os quais podem estar relacionados a:

- Descrição dos recursos:
  - Com o crescente uso da *World Wide Web* (WWW), metadados têm obtido grande popularidade devido a necessidade de encontrar informações úteis

diante da grande quantidade de conteúdo disponível. Em bibliotecas digitais, tem-se usado o padrão de metadados *Dublin Core*[9] para descrição dos documentos. Em pesquisas sobre Web Semântica (*Semantic Web*)[10], padrões como *Resource Description Framework* (RDF)[11] e *Web Ontology Language* (OWL)[12] têm sido criados com a finalidade de descrever o conteúdo das páginas web e também prover relacionamento entre elas de forma a facilitar a descoberta e integração de informações por parte dos usuários e aplicações;

- Em ambientes de grades de computadores, o escalonamento de tarefas depende de descrições adequadas dos recursos computacionais disponíveis (capacidade de processamento, quantidade de memória, espaço em disco, etc), bem como informações acerca da aplicação como, parâmetros de I/O, restrições, preferências e plataforma de hardware e software requerida. Serviços como o *Monitoring and Discovery System* (MDS) [vLFF+97, ZS04] fornecem informações sobre a disponibilidade dos rescursos e serviços disponíveis;

- Segurança: o estabelecimento de informações adequadas que identifiquem usuários e também determinem quem é permitido executar que ação, assim como estabeleçam credenciais e permitam a realização de auditoria são essenciais para a manutenção da segurança. Metadados de segurança envolvem informações relacionadas à autenticação, autorização e auditoria;

- Propriedade intelectual: metadados que identifiquem propriedade intelectual têm por objetivo garantir ao autor ou criador o estabelecimento de condições sob as quais o dado pode ser usado por terceiros;

- Gerenciamento de informações: dada a criação de recursos, informações que permitam identificar o processo de criação e histórico de modificações realizadas ao longo da existência do dado, bem como procedimentos, parâmetros, pré-requisitos e restrições usados são necessárias para o correto gerenciamento dos recursos. Essas informações podem ser usadas para recriar o recurso, em caso de corrompimento, e para validar o resultado de transformações realizadas no recurso;

[9] http://dublincore.org/.
[10] http://www.w3.org/2001/sw/.
[11] http://www.w3.org/RDF/.
[12] http://www.w3.org/2004/OWL/.

- Replicação: tipicamente, a réplica é criada porque a nova localização oferece melhor desempenho ou disponibilidade em relação a uma localização particular e ainda para prover tolerância a falhas. Porém, a replicação traz consigo a necessidade de manutenção de consistência das várias cópias [CDF+02]. Logo, a existência de informações que identifiquem as cópias, o relacionamento entre elas, suas localizações e permitam adequada seleção são imprescindíveis para o uso efetivo dessa técnica;

- Catálogo de banco de dados relacional: em um banco de dados relacional, metadados são usados para descrever as tabelas (nomes, tamanho, número de colunas, linhas, tipo de dado armazenado em cada coluna, etc), bem como o relacionamento entre elas;

- Interoperabildade: atualmente, é grande a produção de dados em ambientes científicos, acadêmicos, industriais e comerciais. Por vezes, há o interesse na integração e compartilhamento de dados entre as diversas comunidades. Neste momento, os administradores de dados deparam-se com modelos de dados incompatíveis (relacional, hierárquico, orientado a objetos, arquivos, etc), estruturas de dados incompatíveis, assim como problemas de semântica do dado, pois cada comunidade produz seus dados de forma independente, obdecendo interpretações e requisitos particulares. Metadado assume um importante papel no contexto da interoperabilidade de dados e fontes de dados ao permitir uma melhor integração, troca, acesso e interpretação dos dados através do uso de convenções comuns a respeito da semântica, sintaxe e estruturação do dado. Padrões de metadados têm emergido para fomentar interoperabilidade, tais como:

  - *Common Warehouse Metamodel* (CWM)[13], cujo objetivo é permitir a integração de sistemas de data warehouse, e-business e sistemas de negócios inteligentes em ambientes heterogêneos e distribuídos;
  - *Meta-Object Facility* (MOF) [MOF], que é uma linguagem abstrata e um arcabouço para construir, especificar e gerenciar metamodelos independentes de tecnologia, bem como uma linguagem de especificação de estrutura semântica de meta-metamodelos;
  - *XML Metadata Interchange* (XMI) [XMI], que é um padrão criado com o objetivo de prover interoperabilidade entre repositórios de metadados e

[13] http://www.omg.org/technology/cwm/.

ferramentas de desenvolvimento, através da troca de metadados armazenados em sistemas de arquivos tradicionais ou no formato de fluxo (stream) de dados baseados no padrão XML. O padrão XMI foi projetado para permitir a troca de qualquer modelo de metadados especificado segundo o metamodelo MOF.

2.6.1 Metadados em Grades de Computadores

Dada a existência do dado e a necessidade de descrevê-lo para facilitar a descoberta e uso do mesmo, formatos padronizados de metadados e mecanismos que possibilitem o armazenamento e acesso a eles são requeridos. Comunidades científicas, principalmente ligadas a Internet, têm desenvolvido padrões de metadados baseados em XML, tais como RDF e OWL, bem como API's de acesso. Em pesquisas relacionadas a grades de computadores, têm-se usado mecanismos como LDAP, que usa um formato particular de definição de metadados bem como mecanismos de acesso. Sistemas de banco de dados distribuídos usam seus catálogos internos para armazenar metadados relacionados à replicação, possuindo por vezes uma estrutuda de armazenamento proprietária. Em geral, instituições usam sistemas de banco de dados como mecanismo de armazenamento e acesso, principalmente armazenando os metadados em relações. Pode-se notar aqui a falta de padronização quanto ao formato, amazenamento e acesso de metadados em ambientes multi-institucionais e multi-disciplinares. Existem propostas de formatos padrão baseados em XML para expressar metadados e fomentar a interoperabilidade entre os sistemas de acesso e armazenamento, como XMI [XMI], porém é uma questão ainda em aberto, que tem gerado muita pesquisa.

Em grades de dados, dada a heterogeneidade das plataformas de hardware e software das OV's, a flexibilização dos mecanismos de acesso aos metadados é necessária, favorecendo um fraco acoplamento entre os mesmos. Como apresentando por A. Chervenak et. al [CFK+01, CDK+02] e G. Singh et. al [SBC+03], mecanismos de acesso a metadados descritivos, de replicação, localização de réplica e segurança devem ser constituídos sobre serviços independentes, assim como serviços que fomentam o acesso aos sistemas de banco de dados e gerenciamento de recursos.

Toda a infraestrutura de uma grade de computadores depende de um conjunto de metadados para configuração, identificação de recursos, monitoramento da infraestrutura, escalonamento de tarefas, replicação, segurança e ainda informações que

possibilitem a composição de OV's dada a heterogeneidade de plataformas de hardware e software. Recentes pesquisas sobre o uso de metadados em grades de computadores conduziram a um novo conceito chamado Grade Semântica (*Semantic Grid*)[14], cujo objetivo é criar uma infra-estrutura que possibilite a formação de Organizações Virtuais (OV) mais rapidamente, facilmente e automaticamente dada a heterogeneidade dos recursos computacionais existentes. A chave é o uso de metadados para a adequada descrição de todos os recursos, incluindo serviços, de uma forma reconhecível pelos componentes da grade. As tecnologias de metadados aliadas à Web Semântica são formas de alcançar tal objetivo, que é a chamada interoperabilidade semântica.

2.7 Considerações Finais

Em variados ramos da atividade humana, a computação em grade tem se tornado uma alternativa bastante atrativa para suprir a demanda de grande capacidade computacional, colaboração e compartilhamento de recursos distribuídos, pois permite, em especial através de grade oportunistas, que se use a capacidade computacional disponível no parque computacional das organizações ao contrário de fazer-se altos investimentos em hardware de alto desempenho e software.

A porposta de compartilhamento de recursos computacionais e colaboração impõem à infra-estrutura de uma grade computacional o tratamento de questões como: heterogeneidade de plataformas de hardware; interoperabilidade; custo reduzido; gerenciamento de recursos; qualidade de serviço; serviços de informação; políticas de segurança; tolerância a falhas.

Atualmente existem classes de problemas que requerem computação de alto desempenho e que levam um longo tempo de processamento sobre grandes conjuntos de dados compartilhados entre organizações distribuídas geograficamente. Neste contexto, a computação em grade especializou-se de forma a fornecer em sua infra-estrutura mecanismos para permitir o acesso sistemas de banco de dados heterogeneos, segurança no acesso aos dados, baixa latência na transferência, confiabilidade, transparência, publicação e descoberta dos dados. Essa especialização é chamada de Grade de Dados.

[14] http://www.semanticgrid.org/.

Diante da possibilidade de acesso compartilhado e distribuídos aos dados, mecanismos que permitam a publicação e descoberta dos mesmos são necessários e compreendem um serviço essencial em grades de dados, o qual chamamos de Serviço de Metadados.

O serviço de metadados viabilizam seu objetivo através do uso de metadados. Metadados desempenho papel fundamental ao descrever e qualificar o dado, dando significado a ele. Em um cenário geral, metadados assumem diversos propósitos como: descrição de recursos, segurança, asseguram propriedade intelectual, permitem o gerenciamento de informações, são usados para permitir interoperabilidade na troca e acesso a dados, bem como usados em catálogos internos de sistemas de banco de dados. Em grades de computadores, metadados são usados para, por exemplo, configuração, identificação de recursos, monitoramento da infra-estrutura, escalonamento de tarefas e replicação.

Uma grade computacional compreende uma infra-estrutura de hardware e software que permitam o fornecimento de serviços para tratar as questões citadas anteriormente. A infra-estrutura de software compreende o que chamadas de *middleware* de grade. Atualmente destacam-se vários *middlewares* de grade como, por exemplo, o MAG (*A Mobile Agent based Computational Grid Platform*)[LSS05, Lop06].

3 Plataforma de Grade MAG

Este capítulo descreve a arquitetura do middleware MAG (*A Mobile Agent based Computational Grid Platform*)[LSS05, Lop06] sobre o qual o serviço de metadados proposto nesta dissertação foi integrado e testado. O MAG é um middleware para grades de computadores que está sendo desenvolvido pelo Departamento de Ciência da Computação da Universidade Federal Maranhão (DEINF/UFMA), cujo principal objetivo é o desenvolvimento de uma infra-estrutura de software livre baseada sobre a tecnologia de agentes de software que permita o reaproveitamento de recursos ociosos existentes em instituições para a resolução de problemas computacionalmente intensivos.

3.1 Visão Geral do MAG

O MAG está organizada em forma de uma pilha de camadas de software, apresentada na Figura 3.1.

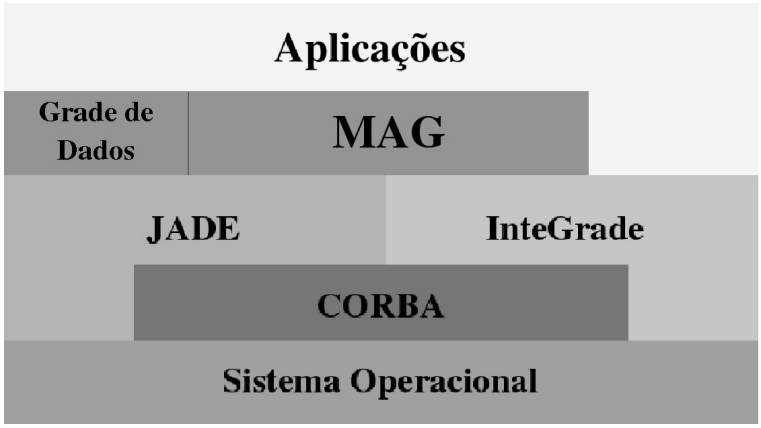


Figura 3.1: Arquitetura em camadas do *middleware* MAG

O MAG utiliza o *middleware* Integrade [GKGF03, GKG+04, CGA+04, Gol04] como fundação para sua implementação, objetivando evitar a duplicação de esforços no desenvolvimento de uma série de componentes. O Integrade é um *middleware* de grade desenvolvido pelo Instituto de Matemática e Estatística da Universidade de São Paulo (IME/USP), o Departamento de Computação e Estatística da Universidade Federal de Mato Grosso do Sul (DCT/UFMS), o Departamento de Informática da Pontifícia

Universidade Católica do Rio de Janeiro (DI/PUC-Rio), Departamento de Informática da Universidade Federal do Maranhão (DEINF/UFMA) e Instituto de Informática da Universidade Federal de Goiás (INF/UFG). Assim como o MAG, o objeto do projeto Integrade é o desenvolvimento de um *middleware* de grade que permite o aproveitamento do poder computacional ocioso em computadores existentes em intituições afim de resolver problemas computacionalmente intensivos.

O Integrade é estruturado em aglomerados, ou seja, unidades autônomas dentro da grade que contém todos os componentes necessários para funcionar independentemente. A visão geral da arquitetura de um aglomerado Integrade pode ser visto na Figura 3.2.

Figura 3.2: Arquitetura de um aglomerado Integrade

O nó "Gerenciador do Aglomerado" representa o nó responsável por gerenciar o aglomerado e por comunicar-se com os outros gerenciadores de aglomerados. Um "Nó de Usuário" é um nó pertencente a um usuário que submete aplicações para execução na grade. Um "Nó Compartilhado" é tipicamente um PC ou estação de trabalho em um laboratório compartilhado que exporta parte de seus recursos, tornando-os disponíveis aos usuários da grade. Um "Nó Dedicado" é aquele reservado para a execução de computações para a grade. Estas categorias podem sobrepor-se: um nó pode ser ao mesmo tempo um "Nó de Usuário" e um "Nó Compartilhado".

O MAG adiciona ao Integrade um novo mecanismo de execução de aplicações, através do uso da tecnologia de agentes móveis, para permitir a execução de aplicações Java, não suportadas nativamente pelo Integrade. O MAG disponibliza também um mecanismos de tolerância a falhas de aplicações, migração transparente (para prover suporte a nós não dedicados), além do acesso de clientes móveis e nômades à grade (através de dispositivos de computação portátil e da web).

A camada Grade de Dados compreende os serviços de suporte a dados, dentre os quais o serviço de metadados, que é o objeto deste trabalho.

A camada JADE (*Java Agent Development Framework*) é um arcabouço utilizado para a construção de sistemas multiagentes. Ela provê ao MAG várias funcionalidades como facilidades de comunicação, controle do ciclo de vida e monitoração da execução dos agentes móveis. O JADE é uma plataforma portável (pois foi desenvolvida utilizando tecnologia Java) e aderente à especificação FIPA[1].

A comunicação entre e com os componentes do Integrade é realizada através da tecnologia de objetos distribuídos CORBA, representada na camada CORBA. O Integrade utiliza o serviço de negócios (*trading*) [Obj00] desta tecnologia para catalogar informações dos recursos disponíveis para grade. Por último, a camada de sistema operacional pode ser variável, dado que o MAG é independente de plataforma, pois utiliza tecnologia Java.

3.2 Arquitetura Base do MAG

A arquitetura do MAG é ilustrada com o uso da metodologia de desenvolvimento de sistemas multiagente chamada *Agil PASSI* [CCSS04]. O motivo de se usar está metodologia é devido ao fato que as técnicas tradicionais de desenvolvimento de software não fornecerem recursos suficientes para a representação de diversos aspectos relativos a sistemas baseados em agentes.

A Figura 3.3 ilustra a sociedade de agentes através do Diagrama de Identificação de Agentes, que é obtido através do agrupamento de casos de uso em pacotes. Os casos de uso representam funcionalidades atribuídas ao agente. As comunicações entre os casos de uso representam a comunicação existente entre os agentes da arquitetura. Os componentes e agentes que compõem o suporte a mobilidade não são ilustrados neste diagrama por não estarem diretamente relacionados a este trabalho.

Os desenvolvedores do MAG precisaram adaptar o Diagrama de Indetificação de Agentes para a realidade do MAG. Isto se deve ao fato deste projeto não ser formado apenas por agentes, mas também por componentes de software, pois reutiliza a infra-

[1] *Foundation for Intelligent Physical Agents* – organização sem fins lucrativos que objetiva a produção de padrões de interoperação de agentes de software heterogêneos. Disponível em http://www.fipa.org/.

estrutura do projeto Integrade. A representação dos componentes mostrados neste diagrama é a de um pacote com o esteriótipo *Component*.

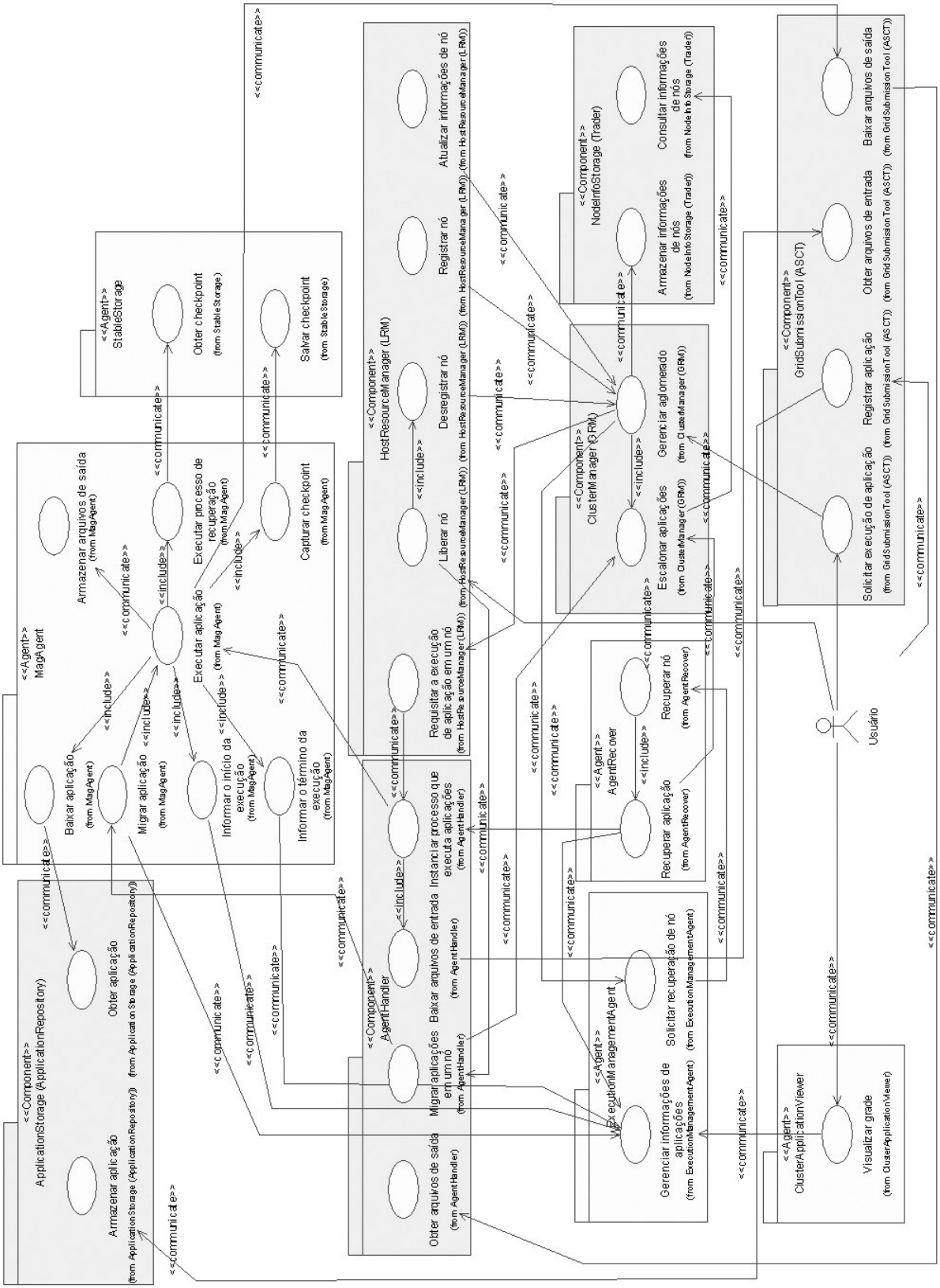


Figura 3.3: Diagrama de identificação de agentes do MAG

Cada agente e componente desta arquitetura recebe um nome relativo ao papel que desempenha. Alguns componentes desta arquitetura foram reutilizados do projeto Integrade, e têm seus nomes originais apresentados entre parênteses. Estes componentes foram modificados para funcionar em conjunto com a arquitetura do MAG. Os agentes / componentes identificados neste diagrama são:

- *ApplicationStorage (ApplicationRepository):* componente que permite o armazenamento e a recuperação de aplicações para execução na grade. Toda aplicação a ser submetida para execução deverá ser previamente armazenada neste repositório. O MAG utiliza o *ApplicationRepository* do Integrade para efetuar esta tarefa;

- *GridSubmissionTool (ASCT):* componente utilizado pelo usuário para a submissão e controle de aplicações. Através dele os usuários podem registrar e fazer o *upload* das aplicações para o *ApplicationStorage*, especificar pré-requisitos de execução (como a plataforma de hardware e software nas quais a aplicação deve ser executada e requisitos de memória necessários à execução da aplicação), requisitar a execução das aplicações registradas e coletar seus resultados. Foi reutilizada a ferramenta ASCT do Integrade;

- *NodeInfoStorage (Trader):* este componente é um armazém de informações de nós, permitindo o armazenamento e a recuperação destas informações. Estas informações são utilizadas para realizar o escalonamento de execução de aplicações nos nós da grade. Este componente foi reutilizado do projeto Integrade e compreende o serviço de negócios (*trading*)[Obj00] do *middleware* CORBA utilizadado;

- *ClusterManager (GRM):* componente responsável por efetuar tarefas de gerenciamento do aglomerado, tais como o registro, desregistro e manutenção de informações de nós conectados à grade, detecção de falhas de nós e escalonamento de tarefas de acordo com as informações mantidas pelo *NodeInfoStorage*. Foi reutilizado o componente GRM do Integrade;

- *HostResourceManager (LRM):* este componente deve executar em nós que compartilham recursos com a grade. Ele é responsável por registrar e desregistrar o nó junto à grade, informando que ele faz parte ou não de um aglomerado. Além disso, monitora os recursos da máquina em que executa, enviando periodicamente

estas informações ao *ClusterManager*, fazendo com que a grade sempre conheça a quantidade de recursos disponíveis. Ele é também responsável por requisitar a execução de aplicações no nó em que executa, e por iniciar o processo de liberação deste mesmo nó, caso tenha sido requerido por um usuário. Foi reutlizado o componente LRM do Integrade;

- *AgentHandler:* componente que mantêm a plataforma de agentes que executa em nós que compartilham recursos na grade. É responsável pela instanciação de agentes para a execução de aplicações, notificação da necessidade de migração para localidades remotas, obtenção de arquivos de entrada da aplicação e retorno de seus arquivos de saída para o usuário que originou a requisição;

- *MagAgent:* principal agente da arquitetura do MAG. Ele é responsável por instanciar e executar as requisições feitas à grade. Ele também monitora a execução destas aplicações, aguardando por seu término, para poder notificar o usuário deste fato, além de migrá-las caso isto seja requerido. Também captura e envia periodicamente o *checkpoint* das aplicações para o *StableStorage*, que o armazena, além de manter os dados de execução de sua aplicação atualizados junto ao *ExecutionManagementAgent*. Para garantir a recuperação de aplicações em caso de falha, o MAG propõe o uso da abordagem de *checkpointing*. Nesta abordagem, durante a execução das aplicações, seus estados de execução (*checkpoints*) são periodicamente salvos e enviados para ao *StableStorage* para armazenagem;

- *ExecutionManagementAgent:* este agente armazena dados relativos às aplicações em execução na grade, tais como: o identificador único da execução, nó em que a aplicação executa atualmente, arquivos de entrada e saída, usuário que requisitou a computação, etc. Estes dados são utilizados pelo processo de recuperação de execuções que tenham falhado, além de permitir a consulta de informações sobre as execuções em curso na grade, através do *ClusterApplicationViewer*;

- *ClusterApplicationViewer:* o *ClusterApplicationViewer* é uma ferramenta gráfica utilizada pelos administradores da grade para visualizar as aplicações que executam na mesma. Através dele é possível saber em que nós as aplicações da grade executam, além de consultar informações detalhadas sobre cada uma destas execuções;

- *AgentRecover:* é o agente responsável pelo tratamento de falhas de aplicações que ocorram em um nó da grade. Consulta o *ExecutionManagementAgent* para obter a informação de quais aplicações executavam no nó que falhou, solicita o escalonamento de novos nós para receber as computações que falharam e recria os agentes que executarão estas aplicações, informando a eles que deverão recuperar suas computações a partir de seus últimos *checkpoints*;

- *StableStorage:* é o agente responsável por armazenar os *checkpoints* das aplicações.

3.3 Modelo de Execução de Aplicações do MAG

Atualmente no MAG é possível executar duas classes distintas de aplicações: regulares e paramétricas. As aplicações regulares são aplicações sequenciais compostas de um binário que executa em uma única máquina. Já as aplicações paramétricas são aquelas que executam múltiplas cópias do mesmo binário em máquinas distintas com entradas diferentes. Esta classe de aplicações (também chamada de BoT ou *Bag-of-Tasks*) divide as tarefas em sub-tarefas menores que executam de forma independente, sem haver comunicação entre elas. Várias aplicações enquadram-se nesta categoria, como as de segmentação de imagens, simulações climáticas e sísmicas. Aplicações escritas na linguagem Java são executadas pelo MAG, enquanto que aplicações nativas do sistema operacional são executadas diretamente pelo *middleware* Integrade. O Integrade adota o modelo BSP (*Bulk Synchronous Parallelism*) [Val90] para a execução de computações paralelas, usando a biblioteca BSPLib [HMS+98] da Universidade de Oxford como referência. A Figura 3.4 ilustra o modelo de execução de aplicações no MAG.

Um usuário requisita a execução de uma aplicação através da interface do *GridSubmissionTool* (a aplicação deve ter sido previamente registrada no repositório de aplicações). O *GridSubmissionTool* então encaminha esta requisição ao *ClusterManager* (1), que de acordo com os pré-requisitos estabelecidos pelo usuário solicita ao *NodeInfoStorage* um nó para escalonar a execução da aplicação (2). O *ClusterManager* então repassa a requisição ao *HostResourceManager* do nó escalonado (3). Caso a aplicação a ser executada seja nativa do sistema operacional o *HostResourceManager* executa a aplicação sem nenhuma intervenção por parte do MAG. Entretanto, se esta for uma aplicação escrita usando a linguagem Java, a requisição é encaminhada ao

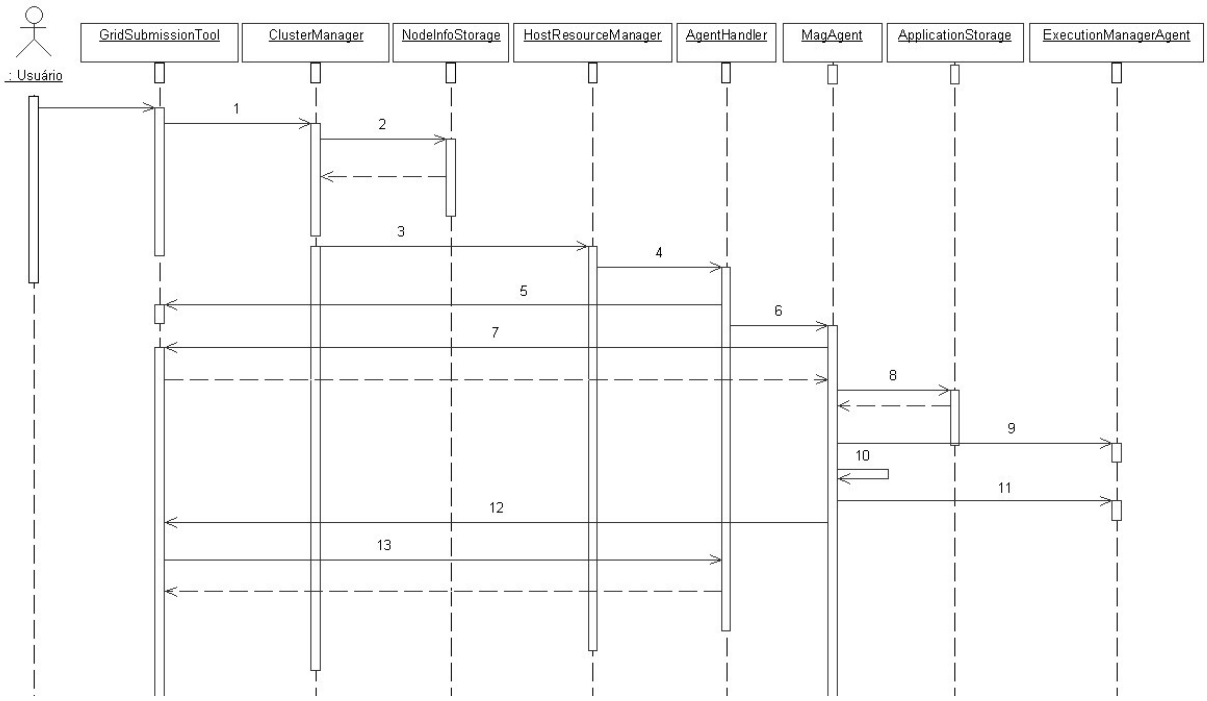


Figura 3.4: Modelo de execução de aplicações do MAG

AgentHandler local (4). Este *AgentHandler* então informa ao *GridSubmissionTool* que aceitou o seu pedido de execução (5), repassando a ele sua IOR CORBA. Em seguida o *AgentHandler* instancia um *MagAgent* responsável por executar a aplicação do usuário (6), que prepara o ambiente para a execução da aplicação baixando seus arquivos de entrada de dados (7) e seu *bytecode* (8), salvando-os no sistema de arquivos local.

Em seguida, o *MagAgent* registra os dados relativos à execução de sua aplicação junto ao *ExecutionManagementAgent* (9). Ao receber esta solicitação, o *ExecutionManagementAgent* a trata, armazenando os dados relativos à execução da aplicação. O *MagAgent* então instancia a aplicação e inicia sua execução em uma *thread* independente (10). Após o início da execução da computação, o *MagAgent* passa a monitorá-la, aguardando por seu término. Ao perceber o fim da execução da aplicação, o *MagAgent* passa a executar suas rotinas de término, solicitando a remoção do registro dos dados da execução junto ao *ExecutionManagementAgent* (11) e notificando o *GridSubmissionTool* que originou a requisição sobre o término da execução da aplicação solicitada (12). Em seguida, o *GridSubmissionTool* requisita os resultados da computação ao *AgentHandler* no qual a execução da aplicação terminou (13), exibindo por fim, estes dados a seu usuário.

3.4 Considerações Finais

O MAG é um *middleware* de grade que adota o paradigma de agentes de software em desenvolvimento e tem como objetivo o reaproveitamento de recursos ociosos para utilização na resolução de problemas computacionalmente intensivos.

Um aspecto interessante a ser ressaltado nesta proposta é que os seus desenvolvedores procuram evitar a duplicação de esforços no desenvolvimento de uma série de componentes e/ou serviços na constituição da infra-estrutura do *middleware*, o que o torna uma arquitetura híbrida com relação ao paradigma de desenvolvimento, apesar de sua proposta primordial do uso de agentes. MAG integrade em sua infra-estrutura, além dos agentes que constituem sua arquitetura base, componente providos pelo *middleware* InteGrade, que usa tecnologia de objetos e comunicação usando CORBA, e serviços providos pela plataforma de agentes JADE, que fornece o ambiente para execução, comunicação, mobilidade e comunicação entre os agentes. A comunicação entre os agentes é realizada através do padrão de comunicação FIPA e pode ser viabilidade por tecnologias de comunicação como, por exemplo, Java RMI (*Remote Method Invocation*) e CORBA.

A execução de aplicações no MAG é realizado através de agentes como *AgentHandler* e *MAGAgent*. Este é o agente que efetivamente executa a aplicação. É possível executar aplicações regulares e paramétricas.

O *middleware* MAG foi usado para a realização de testes e avaliação do serviço de metadados proposto neste trabalho, chamado MagCat.

4 MagCat: Um Serviço de Metadados para o Middleware MAG

A partir da década de 90, a Computação em Grade tem emergido como uma nova ferramenta para satisfazer a necessidade de grande capacidade computacional em variados ramos de atividades como, previsão do tempo, simulações sísmicas, física de alta energia e estudos colaborativos em biomedicina e diagnósticos de doenças (Esclerose Múltipla, Esquisofrenia, Doença de Parkinson, Câncer). Recentemente as pesquisas em grades de computadores tem dado atenção especial ao grande volume de dados usado e produzido por aplicações nos vários domínios de aplicação, desenvolvendo a chamada grade de dados, que é uma infra-estrutura complementar àquela proposta por *middlewares* de grade para o gerenciamento desse grande volume de dados. Essa infra-estrutura complementar engloba um conjunto de serviços para gerenciamento de dados em grades de computadores, que dentre os quais destacam-se como principais o serviço de dados, para transferência rápida e confiável dos dados, bem como replicação, e o serviço de metadados, para a publicação e descoberta dos dados na grade.

Este capítulo descreve a arquitetura e implementação do MagCat[dSdSeST+06], um serviço de metadados baseado na tecnologia de agentes de software que permite a publicação e descoberta de dados em grades de computadores.

4.1 Requisitos para o Serviço de Metadados

Quando uma grande quantidade de dados é compartilhada, existe a necessidade de mecanismos que tornem possível a publicação desses dados, de forma que usuários possam descobrir esses dados. Descoberta de dados é impraticável sem informações adequadas para descrevê-los e mecanismos que possibilitem o acesso a essa informações. Usualmente essas informações são chamadas de metadados, ou "dado sobre dado". O serviço de metadados provê os mecanismos para gerenciar os metadados.

Metadados podem ser usados para descrever dados gerados nos mais variados domínios. Cada domínio organiza e descreve seus dados da maneira mais adequada para seus estudos. Assim, identificamos os seguintes requisitos de um serviço de metadados para grades computacionais:

- Prover um esquema[1] de metadados genérico e independente de domínio;
- Prover um esquema de metadados extensível, de forma a possibilitar a criação de novos atributos de acordo com as necessidades dos usuários, apropriado para catalogação de dados de domínios diferentes;
- Ser escalável e distribuído com capacidade de armazenar grande quantidade de informações;
- Possuir alta disponibilidade;
- Prover baixa latência sobre consultas aos metadados;
- Prover mecanismos de controle de autenticação e autorização das operações sobre informações armazenadas;
- Possibilitar o agrupamento de dados relacionados, de forma a facilitar a recuperação dos mesmos;
- Prover uma clara separação entre componentes de armazenamento de metadados lógicos ou descritivos de metadados de armazenamento físico ou de réplica;
- Disponibilizar interface bem definida para publicação, armazenamento, descoberta e recuperação dos metadados;
- Prover um comum entendimento sobre o significado dos metadados, de forma a possibilitar a identificação não ambígua do tipo de dado e domínio ao qual pertence.

4.2 Modelo de Dados

Objetivando prover os requisitos de esquema genérico, extensível, possibilidade de agrupamento de dados relacionados e autorização definimos o modelo de dados

[1] Nós chamamos de esquema um conjunto de atributos que descrevem o dado.

ilustrado na Figura 4.1 como um diagrama de classe UML (*Unified Modeling Language*). Como podemos visualizar, a unidade conceitual básica para representação da informação é um arquivo (*File*). Cada arquivo pode ser agrupado em uma única coleção (*Collection*), que também podem ser agrupada em outras coleções. Cada usuário deve possuir permissões adequadas para manipular um determinado conjunto de informações, assim, o acesso a arquivos e coleções é sujeito a autenticação e autorização sobre operações permitidas sobre eles. As permissões estabelecidas para coleções são propagadas automaticamente para os aquivos pertencentes àquela coleção. Todos os usuários do serviço de metadados possuem permissão de leitura sobre todos os metadados armazenados. Para permitir que arquivos pertencentes a alguma coleção sejam organizados de acordo com a necessidade de especialistas de domínio e em vários agrupamentos diferentes, definimos o conceito chamado visão (*View*).

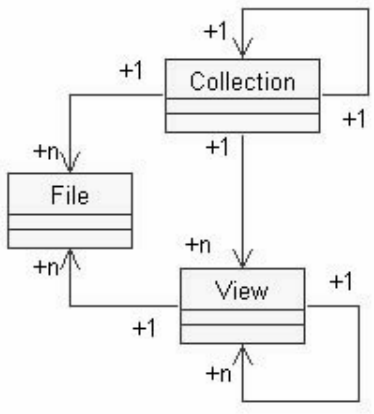


Figura 4.1: Modelo de dados

Cada conceito no modelo de dados apresentado tem um conjunto de atributos básicos, apresentados na Tabela 4.1. Especialistas de domínio podem definir atributos adicinais específicos de seu domínio através da estensão dos conceitos apresentados. Esta característica garante o requisito de estensibilidade de esquema.

Instâncias do modelo de dados podem ser armazenadas em diferentes sistemas de armazenamento como, por exemplo, banco de dados relacional, XML, arquivos e como objetos. Para possibilitar o acesso a esses sistemas de armazenamento nós criamos uma linguagem de consulta própria para o MagCat. Esta linguagem é independente de sistema de armazenamento. No MagCat, a linguagem é mapeada para linguaguem de consulta nativa do sistema de armazenamento como, por exemplo SQL, XPath ou OQL. A

Tabela 4.1: Atributos básicos de cada conceito

Conceitos	**Atributos**
File	name; description, format (binary, text, HTML, XML, jpeg, dicom, etc); is_valid, que indica se o dado ainda é válido; version; creator; creation timestamp; last update timestamp; user (usuário responsável pela última atualização); collection (coleção a qual pertence a visão, opcional)
Collection	name; description; creator; creation timestamp; last update timestamp; user (usuário responsável pela última atualização); collection (coleção a qual pertence a visão, opcional).
View	name; description; creator; creation timestamp; last update timestamp; user responsible for the last update; collection (coleção a qual pertence a visão, opcional).

gramática da linguagem do MagCat é ilustrada parcialmente na Figura 4.2 e apresentada completamente no Apêndice A.

```
<expression> := <create statement> | <update statement> |
          <delete statement> | <select statement> |
          <define statement>

<select statement> := SELECT <object_type> SCHEMA_NAME <schema_name> WHERE <condition>

<create statement> := CREATE <object_type> SCHEMA_NAME <schema_name> <left_paren>
                      <attribute_list> <right_paren> VALUES <left_paren> <attribute_value_list>
                      <right_paren>

<define statement> := DEFINE SCHEMA_NAME <schema_name> DESCRIPTION <left_paren> <string>
                      <right_paren> EXTENDS <schema_name> ATTRIBUTES <left_paren>
                      <attribute_list> <right_paren> ATTRIBUTE_TYPES <left_paren> <attribute_type_list>
                      <right_paren> ATTIBUTE_DESCRIPTION <left_paren> <attribute_description_list>
                      <right_paren>

<object_type> := LOGICAL_FILE | LOGICAL_COLLECTION | LOGICAL_VIEW

<condition>:= <attribute> <operator> <value> <connector> <condition>
```

Figura 4.2: Parte da gramática da linguagem de consulta do MagCat

- <... statement> representa as operações de criação (CREATE), atualização (UPDATE), remoção (DELETE) e consulta (SELECT) ao repositório de metadados, ou definição (DEFINE) de esquema de metadados. A operação DEFINE é usada para estender um esquema de metadado para um determinado domínio;
- <object_type> pode ser um arquivo (LOGICAL_FILE), uma coleção (LOGICAL_COLLECTION) ou uma visão (LOGICAL_VIEW);
- O <schema_name> identifica o esquema relacionado à operação de solicitada, enquanto <condition> especifica os atributos, seus valores e operadores usados na

cláusula WHERE, que atua como um filtro dos metadados solicitados. Se a operação DEFINE é solicitada, o primeiro <schema_name> identifica o esquema que será criado através da estensão do segundo <schema_name>. Assim, o novo esquema herdará todos os atributos do esquema base e terá também novos atributos descritos por <attribute_list>, <attribute_type_list>, e <attribute_list_description>.

4.3 Arquitetura

Por utilizar a tecnologia de agentes em sua concepção, a modelagem do serviço de metadados não poderia utilizar as técnicas tradicionais de desenvolvimento de software, dado que as mesmas não fornecem recursos suficientes para a representação de diversos aspectos relativos a sistemas baseados em agentes como, por exemplo, a modelagem de ontologias, necessária para definir o conhecimento dos agentes a respeito do domínio da aplicação.

Após analisar algumas metodologias para o desenvolvimento de sistemas multiagentes, optou-se por utilizar a *Agil PASSI* [CCSS04]. A *Agil PASSI* é uma adaptação da metodologia PASSI (*Process for Agents Societies Specification and Implementation*) [CSSC03, CSS03] original. Ela foi concebida para permitir o desenvolvimento de sistemas multiagentes através de um processo ágil[2].

A metodologia *Agil PASSI* é composta fundamentalmente por 4 modelos [CCSS04]:

- *Requisitos:* modelo dos requisitos envolvidos no desenvolvimento do sistema;
- *Sociedade de agentes:* uma visão dos agentes envolvidos na solução, suas interações e seu conhecimento sobre o sistema;
- *Codificação:* modelos que descrevem como o sistema deverá ser codificado;
- *Testes:* planejados antes da fase de codificação e executados logo após ela.

[2]Agile Manifesto website. Disponível em: http://www.agilemanifesto.org.

4.3.1 Modelo de Requisitos do MagCat

Este modelo é composto por duas atividades: *planejamento* e *descrição de requisitos de domínio* (*Domain Requirements Description* ou DRD). A etapa de planejamento é utilizada para efetuar o planejamento do desenvolvimento do projeto. É nesta atividade que são efetuadas a análise de riscos, a divisão das tarefas em sub-tarefas e o planejamento das interações entre os desenvolvedores durante a implementação do sistema. Já na atividade de descrição de requisitos de domínio são utilizados diagramas de casos de uso da UML para representar a descrição funcional do sistema. Neste diagrama, o relacionamento entre os casos de uso pode seguir basicamente três estereótipos: (a) *include*, que indica que o comportamento de um caso de uso está inserido dentro de outro, evitando a criação de casos de uso que contenham funcionalidades semelhantes; (b) *extend*, que indica que o comportamento de um caso de uso é estendido por outro que lhe adiciona novas funcionalidades; e (c) *communicate* que é utilizado na relação entre os atores e o sistema, e na comunicação entre funcionalidades distintas da aplicação. A Figura 4.3 ilustra o DRD do MagCat.

Um agente é o responsável por iniciar o processamento de todas as funcionalidades representadados pelos casos de uso. Ao solicitar o inicio de qualquer processamento, é necessário que o agente seja autenticado (*Autenticar agente*) e que seja também verificado as permissões do mesmo diante da operação solicitada (*Verificar autorização*).

Para obter as definições de esquema de metadados existentes como, por exemplo, as definições de arquivo, coleção e visão, o agente consulta as definições de esquema (*Consultar definições de esquemas de metadados*).

As definições encontradas serão utilizados como base para as demais definições quando o agente solicitar a criação de novos esquemas de metadados, ou seja, a estensão do esquema de metadados. Para que algum metadado seja publicado deve haver um esquema de metadados previamente criado. De posse das definições de esquema existentes, o agente gera uma expressão de acordo com a linguagem do MagCat para criar um esquema. Essa expressão será usada para solicitar a criação de um novo esquema de metadados (*Criar esquema de metadados*). Criado o esquema, ele é armazenado no repositório de esquemas (*Armazenar esquema de metadados*).

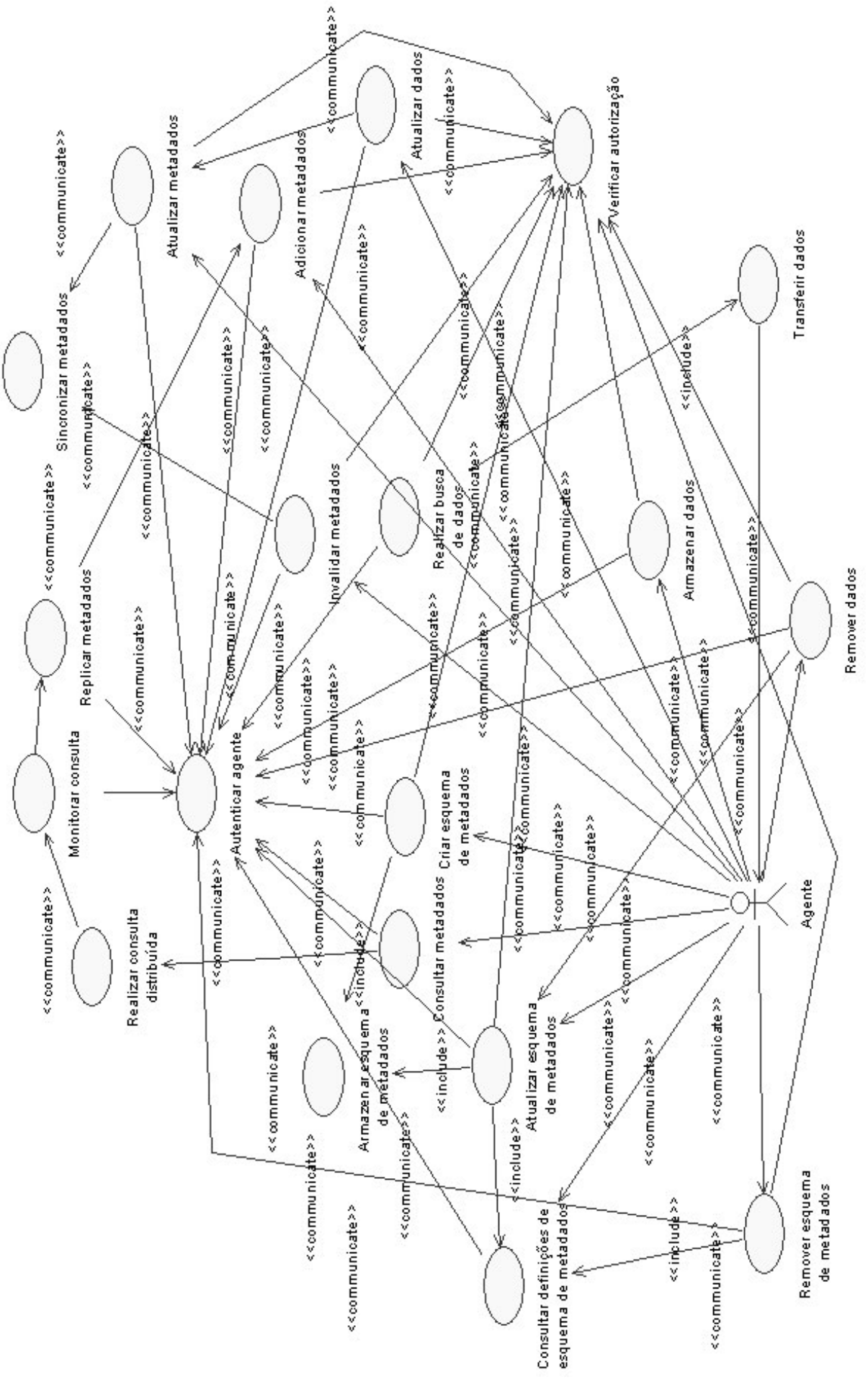


Figura 4.3: Diagrama de requisitos de domínio

Os esquemas de metadados podem ser atualizados. Assim, de posse das definições de esquema existentes, o agente gera uma expressão de acordo com a linguagem do *MagCat* para atualizar o esquema selecionado dentre os existentes. Essa expressão será usada para solicitar a atualização do esquema de metadados (*Atualizar esquema de metadados*). A existência do esquema a ser atualizado é verificada (*Consultar definições de esquemas de metadados*). O esquema então é atualizado junto ao repositório de esquemas (*Armazenar esquema de metadados*).

Dado a possibilidade que seja necessário muitas atualizações em um esquema, o agente pode solicitar a remoção do mesmo. O agente, de posse do nome do esquema a ser apagado solicita que ele seja apagado das definições de esquemas existentes (*Remover esquema de metadados*). A existência do esquema é confirmado através de uma consulta às definições de esquemas (*Consultar definições de esquemas de metadados*). O esquema é apagado caso exista.

A publicação de metadados tem seu início com a solicitação das definições existentes de esquemas de metadados (*Consultar definições de esquemas de metadados*) por um agente. De posse das definições de esquema existentes, o agente seleciona o esquema que corresponde ao tipo de dado que deseja publicar e gera uma expressão de acordo com a linguagem do MagCat para publicar um dado. Essa expressão será usada para solicitar a publicação do dado (*Adicionar metadados*). A publicação consiste em adicionar ao repositório de metadados um novo conjunto de metadados que correspondem a um tipo de dado definido por um esquema. Os metadados então são adicionados ao repositório e disponibilizados para consultas.

Os metadados publicados podem ser atualizados. Podemos partir do pressuposto que o agente possui a identificação do conjunto de metadados a ser atualizado. A identificação é única diante de todos os conjuntos de metadados armazenados na grade. O agente gera uma expressão de acordo com a linguagem do MagCat para atualizar metadados. Essa expressão será usada para solicitar a atualização de metadados armazenados no repositório (*Atualizar metadados*). Caso os metadados atualizados possuam réplicas, logo após a atualização as cópias são sincronizados (*Sincronizar metadados*).

Se é detectado que o dado representado por algum conjunto de metadados foi gerado incorretamente, sendo assim inadequado para análises, é permitido a invalidação

do dado. Para indicar a invalidade de um dado, é efetuada uma atualização em um atributo que indica se o conjunto de metadados referencia um dado válido. Para proceder com a invalidação, podemos partir do pressuposto que o agente possui a identificação do conjunto de metadados a ser removido. A identificação é única diante de todos os conjuntos de metadados armazenados na grade. De posse da identificação, ou mesmo de metadados que identifiquem o dado cujos metadados serão invalidados, o agente gera uma expressão de acordo com a linguagem do MagCat para invalidar metadados. Essa expressão será usada para solicitar que os metadados sejam invalidados no repositório (*Invalidar metadados*). Caso os metadados invalidados possuam réplicas, logo após a atualização as cópias são sincronizados (*Sincronizar metadados*).

Para efetuar a descoberta dos dados, o agente procede com consultas baseadas em metadados sobre o repositório de metadados. O agente deve gerar uma expressão de acordo com a linguagem do MagCat para consultar o repositório de metadados (*Consultar metadados*). O agente pode solicitar que a consulta seja realizado em toda a grade, de forma que possa encontar todos os dados existentes na grade que estejam de acordo com as restrições expressas na consulta (*Realizar consulta distribuída*). A consulta então é enviando para repositórios remotos previamente identificados. Cada consulta tem sua latência e repetição monitoradas (*Monitorar consulta*). As informações a cerca da latência e repetição da consulta serão utilizadas para a tomada de decisão de replicação de metadados para o repositório local.

A replicação é realizada objetivando fornecer disponibilidade e baixa latência nas consultas sobre os metadados disponíveis na grade. Cada consulta a repositórios remotos tem seu tempo de retorno de resultados e repetição monitorados (*Monitorar consulta*). Caso uma determinada consulta tenha seu tempo de resposta excedendo um tempo limite estabelicido pelo administrador do sistema, o processo de replicação daqueles resultados é iniciado e é criada uma cópia no repositório local (*Replicar metadados*) dos resultados retornados pela consulta monitorada (*Adicionar metadados*).

4.3.2 Modelo de Sociedade de Agentes

Este modelo é composto pelos diagramas de *identificação dos agentes* (*Agent Identification* ou AId) e de *descrição de ontologia do domínio* (*Domain Ontology Description* ou DOD).

Identificação dos Agentes (AId)

Na *Agil PASSI*, os casos de uso que compuseram os cenários apresentados no *Modelo de Requisitos* são agora no AId agrupados em pacotes de forma a definir os agentes de acordo com as suas funcionalidades. Os relacionamentos entre os casos de uso pertencentes a um mesmo agente são identificados através do esteriótipo "include" e os relacionamentos entre casos de uso de diferentes agentes possui o esteriótipo "communicate". A Figura 4.4 mostra o diagrama de identificação de agentes do MagCat.

Cada agente e componente desta arquitetura recebeu uma denominação relativa ao papel que desempenha. Os agentes identificados neste diagrama foram:

- *CatalogManagerAgent*: agente responsável por realizar as operações de adição, remoção, atualização e consulta sobre o repositório de metadados;
- *SchemaManagerAgent*: agente responsável pela estensibilidade do esquema de metadados para diferentes domínios de aplicação;
- *SearchAgent*: agente responsável pela execução de consultas em repositórios de metadados distribuídos;
- *ReplicationManagerAgent*: agente responsável pela replicação e sincronização dos metadados;
- *RequestMonitorAgent*: agente responsável pelo monitoramento das consultas requisitados pelos outros agentes. O *ReplicationManager* usa os dados coletados para a tomada de decisão de replicação local dos metadados;
- *SecurityAgent*: agente responsável pela autenticação e autorização das operações executadas pelos demais agentes na sociedade;
- *DataManagerAgent*: agente responsável pelo armazenamento e acesso aos dados. Este agente exerce aqui as funcionalidades básicas do *serviço de dados* na grade.

A partir do diagrama de identificação de agentes, o sistema passa a ser definido em função dos agentes que o compõem e das relações entre eles. Este modelo será refinado nos diagramas a seguir, de maneira que seja possível projetar a estrutura interna dos agentes do sistema e definir como será realizada a comunicação entre eles.

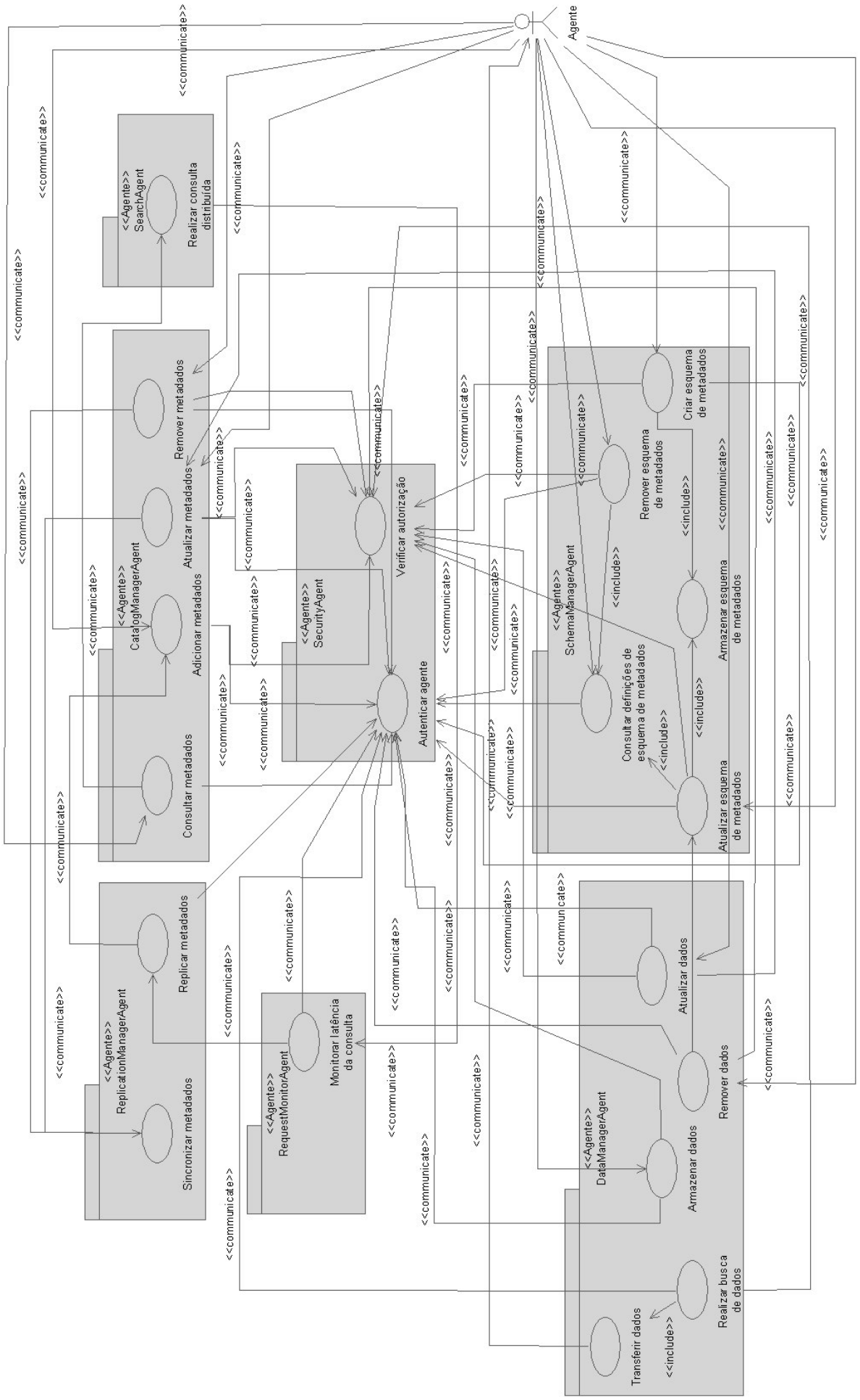


Figura 4.4: Diagrama de identificação de agentes

Descrição de Ontologia de Domínio (DOD)

Uma ontologia é uma especificação explícita dos objetos, conceitos e outras entidades que se assume existirem em uma área de interesse, além das relações entre estes conceitos e restrições, expressos através de axiomas[Gru93]. O uso de ontologia tem sido ampliado e popularizado como via de atingir a interoperabilidade de sistemas de informação e conseqüente compartilhamento de conhecimento.

A existência de ontologias em sistemas baseado em agentes possibilita a representação inequívoca do conhecimento dos agentes de forma a garantir que as informações trocadas pelos agentes não sejam interpretadas equivocadamente. O objetivo do diagrama de descrição de ontologia de domínio é modelar a ontologia presente no sistema, ou seja, representar o conhecimento dos agentes que compõem a sociedade. O diagrama DOD do MagCat é ilustrado na Figura 4.5.

A ontologia no diagrama DOD, um diagrama de classe UML, é descrita em termos de três tipos de elementos: conceitos, predicados e ações.

Os conceitos representam as entidades e categorias presentes no domínio do sistema. Eles aparecem no diagrama DOD com o estereótipo *Concept*. No diagrama DOD do MagCat podemos destacar os conceitos `LogicalFile`, `LogicalCollection` e `LogicalView` como principais, dado que como podemos perceber representam o modelo de dados apresentado na Seção 4.2 e são utilizados na comunicação entre os agentes que requisitam alguma operação sobre o repositório de metadados e o agente `CatalogManagerAgent`.

Os predicados aparecem na forma de inferências sobre os conceitos do domínio. Apresentam o estereótipo *Predicate* no diagrama DOD. Estão relacionados às propriedades dos conceitos e geralmente apresentam-se como questionamentos sobre estas propriedades. No DOD do MagCat temos o predicado `isExpression`, que é utilizado para verificar se uma expressão é válida de acordo com a gramática especificada na Seção 4.2.

As ações representam tarefas executadas pelos agentes. No diagrama DOD aparecem com o estereótipo *Action*. Na ontologia do MagCat, por exemplo, as ações `AddObjectAction`, `DeletObjectAction`, `UpdateObjectAction` e `QueryObjectAction` que correspondem, respectivamente, às operações de adição, remoção, atualização e consulta sobre o repositório de metadados executadas pelo agente `CatalogManagerAgent`.

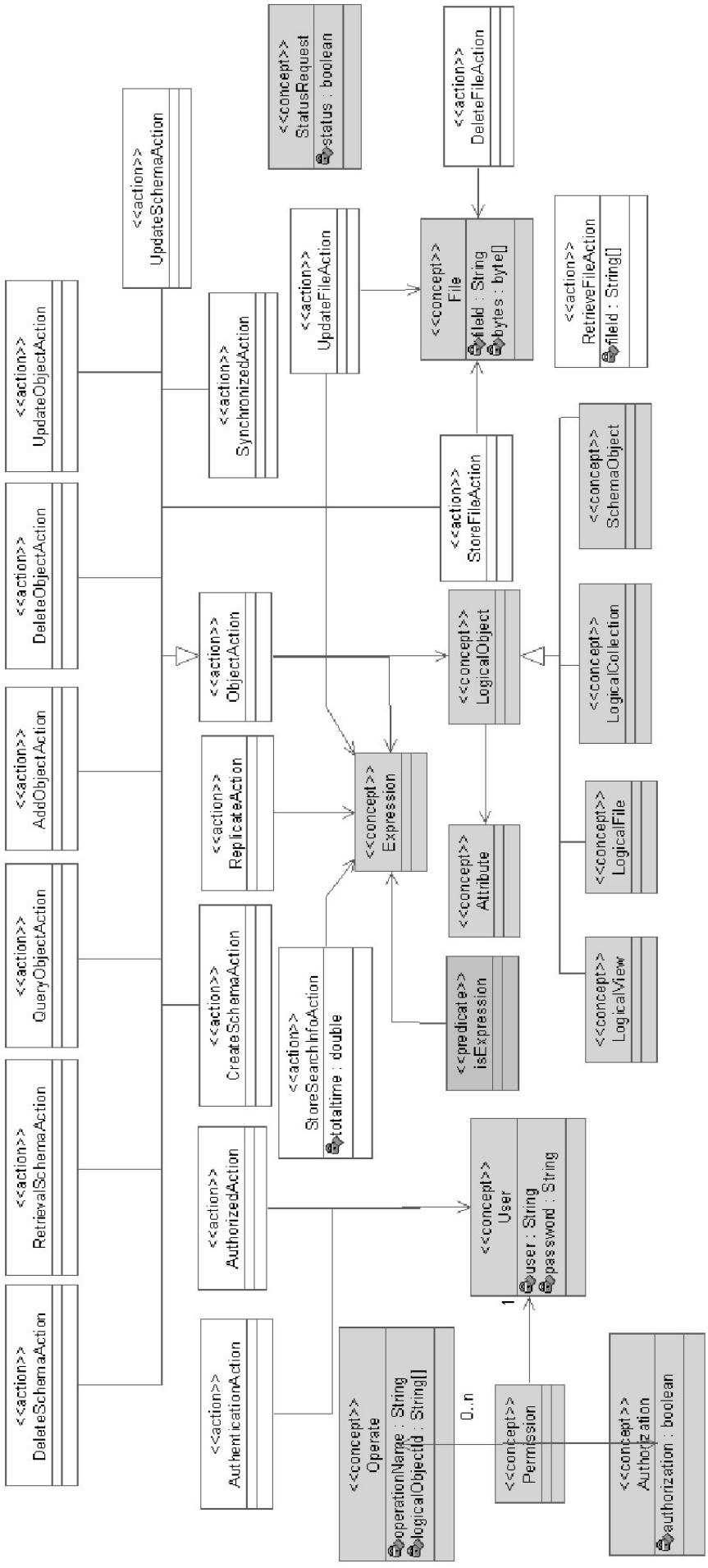


Figura 4.5: Diagrama de descrição de ontologia de domínio

Outro exemplo são as ações `ReplicateAction` e `SynchronizedAction` que são executados pelo agente `ReplicationManagerAgent`

Os elementos identificados no diagrama DOD são usados para definir o conhecimento dos agentes, como citado, e serão essenciais na distinção da semântica da comunicação nas mensagens trocadas.

4.4 Implementação

Na metodologia *Agil Passi*, a fase de implementação corresponde ao *Modelo de Código*, onde os desenvolvedores são encorajados a descobrir padrões na implementação dos agentes para reuso e efetivamente codificar as demais soluções que não se pode reutilizar. Neste modelo a solução arquitetural do sistema, estrutura dos agentes e de seus comportamentos, em termos de classes e métodos é criada. O *Modelos de Código* é formado pelos seguintes diagramas:

- Diagrama de Ontologia de Comunicação (*Communication Ontology Diagram* ou COD): um diagrama de classes representando os agentes, a comunicação entre eles e os parâmetros contidos nas mensagens trocadas;
- Diagrama de Definição de Estrutura Multiagente (*Multi-Agent Structure Definition* ou MASD): um diagrama de classe onde é possível ter uma visão completa da sociedade de agentes e seus relacionamentos, que representam a comunicação entre eles;
- Diagrama de Definição de Estrutura de Agente (*Single-Agent Structure Definition* ou SASD): um diagrama de classe que representa a estrutura interna de cada agente;
- Diagrama de Descrição de Comportamento Multiagente (*Multi-Agent Behaviour Description* ou MABD): um diagrama de atividade que representa o fluxo de execução e comunicação entre todos os agentes da sociedade.

Atualmente nós temos implementados os agentes `CatalogManagerAgent`, `SchemaManagerAgent` e `DataManagerAgent`, os quais fornecem as funcionalidades básicas necessárias para o *serviço de metadados* aqui proposto. A linguagem Java foi usada na implementação destes agentes.

4.4.1 Diagrama de Ontologia de Comunicação (COD)

O objetivo deste diagrama é associar o conhecimento dos agentes, aqui representado pela ontologia ilustrada na Seção 4.3.2, e as comunicações que ocorrem entre eles. No diagrama COD, dois tipos de elementos são possíveis: agente e comunicação, representados pelos estereótipos *Agent* e *Communication*, respectivamente. Como citado anteriormente, as comunicações entre os agentes possui parâmetros, os quais são descritos a seguir:

a) *Ontologia*: os elementos modelados no diagrama DOD são utilizados aqui para definir a semântica da comunicação, possibilitando que os agentes entendam o significado das mensagens trocadas, seja a solicitação de execução de alguma ação pelo receptor, transmissão de dados reconhecidos nos conceitos ou mesmo a solicitação de verificação de validade de algo através de predicados;

b) *Linguagem de conteúdo*: enquanto a ontologia define o significado da comunicação, a linguagem define as notações léxicas e sintáticas que podem ser utilizadas na comunicação. Como exemplos de linguagem temos a RDF (*Resource Description Framework*), baseada em XML, KIF[GF92] *Knowledge Interchange Format* e a linguagem SL (*Semantic Language*), criada pela FIPA com o intuito de torná-la a linguagem padrão para a comunicação entre agentes;

c) *Protocolo de interação*: define como os agentes devem interagir durante sua comunicação. A FIPA define uma série de protocolos de interação que podem ser utilizados na comunicação entre agentes. Exemplos destes protocolos são o FIPA-Request [FIPc], FIPA-Query [FIPb] e FIPA-Propose [FIPa].

No diagrama COD ilustrado na Figura 4.6 as comunicações entre os agentes do MagCat usam o protocolo FIPA-Request ou FIPA-Query.

O protocolo FIPA-Request permite que um agente solicite a outro a realização de uma operação. Este protocolo está ilustrado no diagrama de seqüência AUML da Figura 4.7(a): um agente iniciador (*Initiator*) efetua o procedimento de requisição (*request*) de um serviço a um agente participante (*Participant*). Este, por sua vez, pode aceitar ou não a requisição recebida (*agree* ou *refuse*). Em caso positivo, o agente participante efetuará o processamento da requisição e responderá ao agente iniciador com um dos três tipos de mensagens: (1) uma mensagem *failure* caso ocorra uma falha;

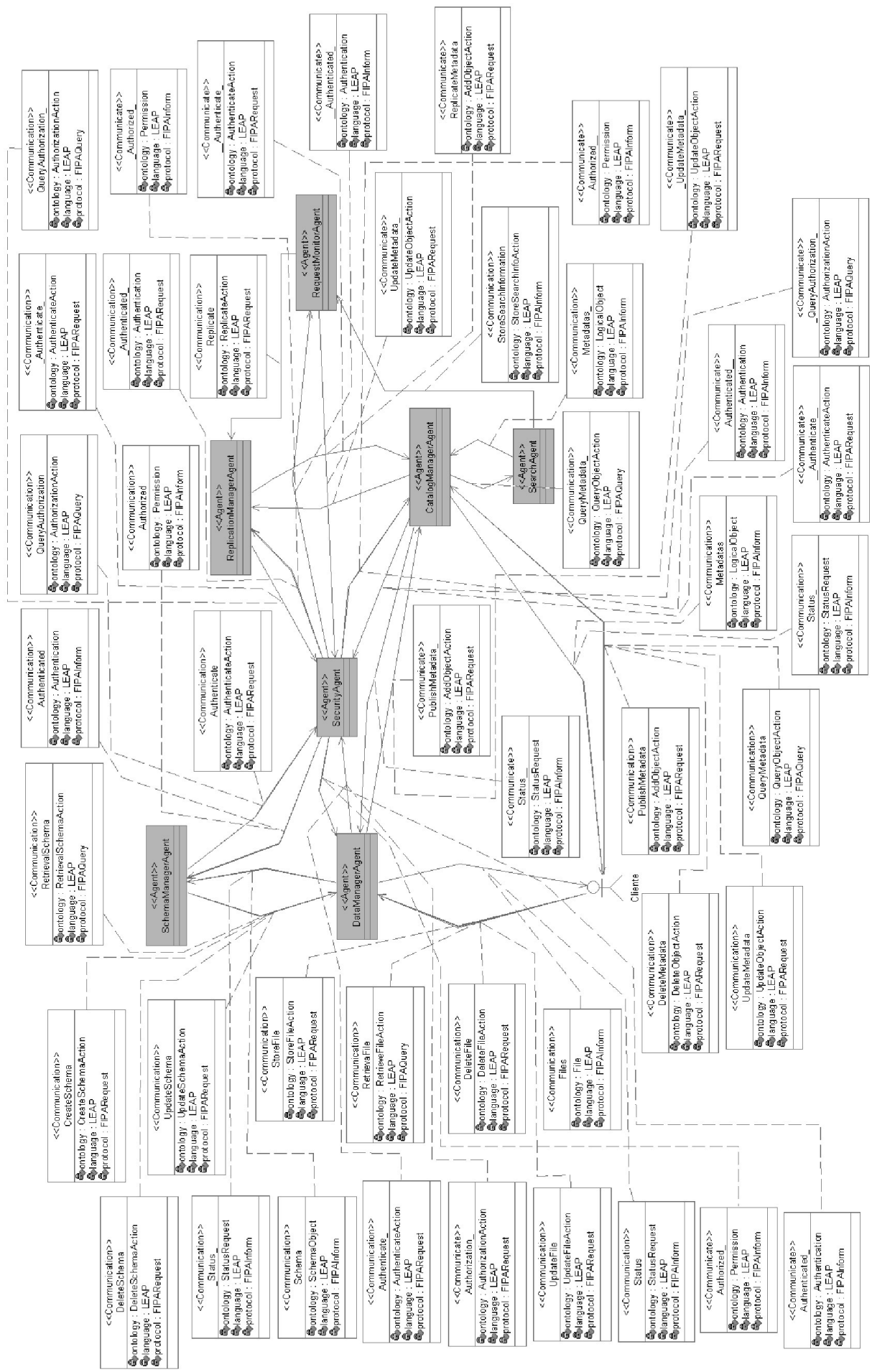
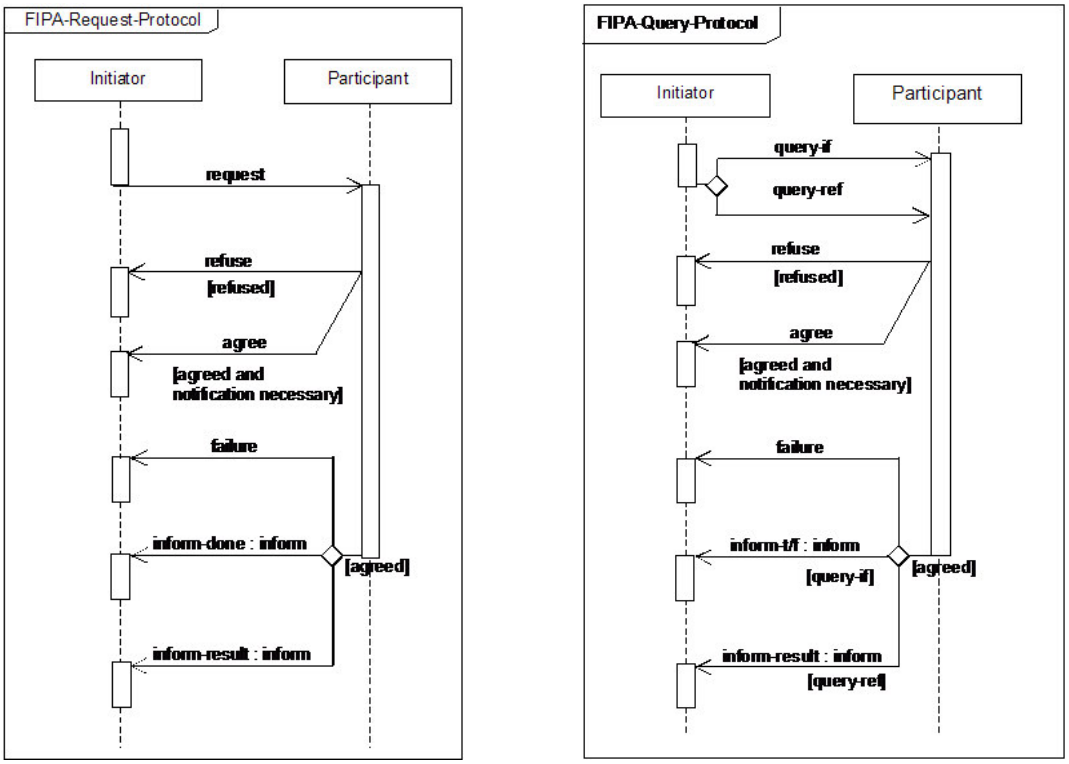


Figura 4.6: Diagrama de ontologia de comunicação

(2) uma mensagem informando que a operação foi efetuada com sucesso pelo agente participante (*inform-done*); (3) uma mensagem contendo o resultado do processamento (*inform-result*). No MagCat o protocolo FIPA-Request é utilizada nas comunicações como, por exemplo, `StoreFile`, `Authenticate`, `PublishMetadata` e `CreateSchema`.



(a) Protocolo FIPA-Request (b) Protocolo FIPA-Query

Figura 4.7: Protocolos FIPA

O protocolo FIPA-Query é utilizado para solicitar consultas a um agente. A Figura 4.7(b) mostra o diagrama de seqüência AUML deste protocolo. Utilizando o protocolo FIPA-Query é possível efetuar dois diferentes tipos de consultas a agentes. Para tanto, duas diferentes performativas são disponibilizadas: *query-if* e *query-ref*. A performativa *query-if* é utilizada para consultar um agente a respeito de uma dada proposição, cujo resultado é booleano. Este tipo de consulta não é utilizada na infraestrutura do *MagCat* e, portanto, não será explanado. Já as consultas baseadas na performativa *query-ref* são utilizadas quando um agente deseja realizar um consulta direcionada a outro. Como pode ser visto na Figura 4.7(b), a interação inicia-se com uma solicitação de consulta (*query-ref*) de um agente iniciador (*Initiator*) a respeito de algum objeto junto a um agente participante (*Participant*). Este por sua vez pode aceitar ou não a solicitação de consulta (*agree* ou *refuse*). Caso o agente participante aceite a solicitação,

ele então efetua o processamento da consulta e retorna seu resultado ao agente iniciador (*inform-result*). Caso uma falha ocorra no processamento da consulta, uma mensagem de falha deve ser retornada ao agente iniciador (*failure*). No MagCat as comunicações `QueryMetadata`, `RetrievalSchema` e `RetirevalFile` utilizam este protocolo.

Todas as comunicações entre agentes utilizam elementos da ontologia definida no diagrama DOD. Por exemplo, para solicitar a execução de operações sobre o repositório de metadados, um agente interage com o agente `CatalogManagerAgent` usando as seguintes comunicações: `PublishMetadata`, `QueryMetadata`, `UpdateMetadata` e `DeleteMetadata`, que representam, respectivamente as operações de adição, consulta, atualização e remoção de metadados no repositório. As informações transmitidas nestas comunicações seguem especificações definidas por elementos da ontologia do MagCat, neste caso, as ações `AddObjectAction`, `QueryObjectAction`, `UpdateObjectAction` e `DeleteObjectAction`. Dessa forma, ao receber uma mensagem, o `CatalogManagerAgent` é capaz de identificar seu significado através da semântica definida pela ontologia. A linguagem utilizada na comunicação entre os agentes do MagCat é a LEAP [CC04].

4.4.2 Diagrama de Definição de Estrutura Multiagente (MASD)

Através do diagrama de definição de estrutura multiagente é possível ter uma visão completa da sociedade de agentes e dos atores que compõem o sistema. Este diagrama é representado através de um diagrama de classes, com as classes simbolizando os agentes identificados através do diagrama AId apresentado na Seção 4.3.2, e com os relacionamentos entre classes representando a comunicação entre os mesmos. Os métodos dos agentes com inicial maiúscula representam as tarefas ou comportamentos dos mesmos, enquanto que os métodos com inicial minúscula representam métodos exportados como insterfaces CORBA. O objetivo de disponibilizar alguns métodos para serem invocados remotamente é possibilitar que *middlewares* não orientados a agentes possam interagir com o MagCat, de forma a fomentar a independência de *middleware*, tecnologia de desenvolvimento e também linguagem de programação.

A Figura 4.8 ilustra o diagrama MASD do MagCat. Neste diagrama a sociedade de agentes que compõe a arquitetura do MagCat são simbolizados através de classes e setas indicando a comunicação entre eles, cujo conteúdo é ilustrado no diagrama COD apresentada na Seção 4.4.1. Os agentes presentes

neste diagrama são: `CatalogManagerAgent`, `SchemaManagerAgent`, `DataManagerAgent`, `ReplicationManagerAgent`,`RequestMonitorAgent`, `SearchAgent` e `SecurityAgent`.

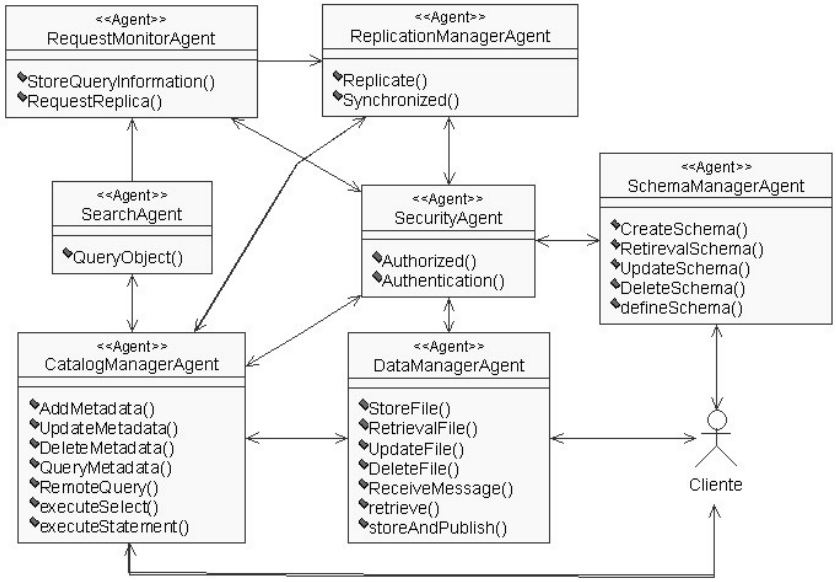


Figura 4.8: Diagrama de definição de estrutura multiagente

A Figura 4.9 ilustra a interface CORBA exportada pelo MagCat. O agente `CatalogManagerAgent` exporta um método para executar uma consulta (`executeSelect()`), que retornará um objeto `LogicalObjectDescriptionList`, e um método para executar comando para atualização, remoção e adição (`executeStatement()`). O agente `SchemaManagerAgent` exporta um método que permite a definição de novos esquemas de metadados (`defineSchema()`). O agente `DataManagerAgent` exporta um método que permite armazenar dados e publicá-los junto ao `CatalogManagerAgent` (`storeAndPublish()`) e um método que permite a recuperação do dado (`retrieve()`).

```
1
2  module dataservice{
3
4
5          struct PublishData{
6                  octet fileBytes;
7                  string publishStatement;
8
9          };
10
11         typedef sequence<PublishData> PublishDataList;
12
13     struct File{
14                 octet bytes;
15                 string fileKey;
16         };
17
18         typedef sequence<File> FileList;
19         typedef sequence<string> FileKeyList;
20
21
22         interface DataManager {
23                 long retrieve ( in FileKeyList fileKeyList, out FileList fileList );
24                 long storeAndPublish(in PublishDataList publishDataList );
25         };
26 };
27
28
29 module metadataservice{
30         typedef string Statement;
31
32         struct AnAttribute{
33            string name;
34            string value;
35         };
36
37
38         typedef sequence<AnAttribute> LogicalObjectDescription;
39
40         typedef sequence<LogicalObjectDescription> LogicalObjectDescriptionList;
41
42
43         interface CatalogueManager {
44                 long executeStatement(in Statement statement);
45                  LogicalObjectDescriptionList executeSelect(in Statement statement);
46         };
47
48         interface SchemaManager{
49                 long defineSchema(in Statement statement);
50         };
51
52 };
```

Figura 4.9: IDL do MagCat

4.4.3 Diagrama de Definição de Estrutura de Agente (SASD)

Para cada agente do diagrama MASD deve ser criado um diagrama de definição de estrutura de agente (SASD), que corresponde à representação da estrutura interna do mesmo. Neste diagrama são apresentadas as tarefas (ou comportamentos) dos agentes, bem como suas interfaces. A representação do SASD é realizada também através de diagramas de classes.

Nesta seção serão apresentados os agentes de software `CatalogManagerAgent` e `SchemaManagerAgent`, que formam a infra-estrutura básica do serviço de metadados MagCat, e o agente `DataManagerAgent` que compreende um protótipo básico do serviço de dados para o *middleware* MAG.

CatalogManagerAgent

O `CatalogManagerAgent` é o principal agente da arquitetura do MagCat. Ele é responsável por gerenciar o repositório de metadados, permitindo a realização das operações de adição, consulta, remoção e atualização.

O diagrama SASD do `CatalogManagerAgent` é apresentado na Figura 4.10. O `CatalogManagerAgent` é composto por nove classes, sendo uma delas a classe principal do agente e as outras classes de comportamentos que podem ser executados pelo mesmo.

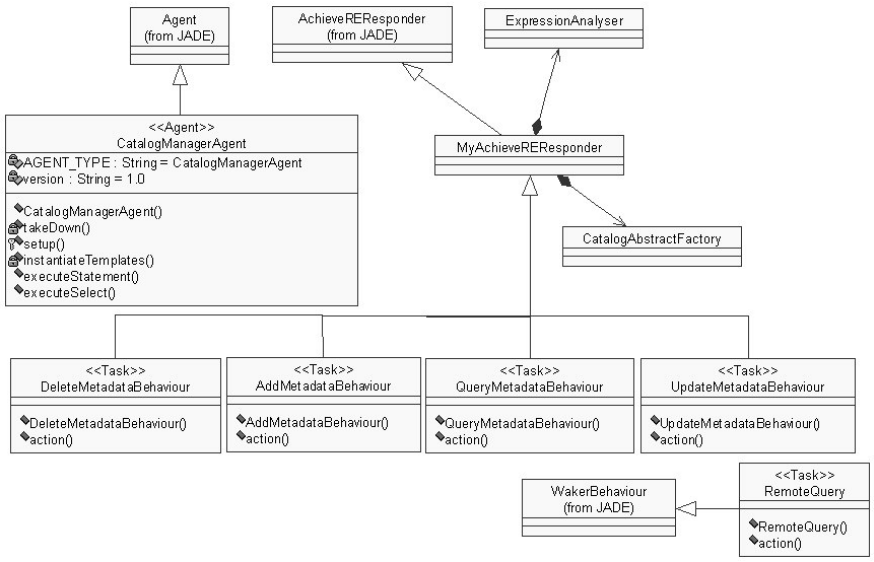


Figura 4.10: Diagrama de definição de estrutura do agente `CatalogManagerAgent`

As classes que compõem o diagrama SASD do `CatalogManagerAgent` são explicadas a seguir:

- `CatalogManagerAgent:` nesta classe encontram-se os métodos que controlam o ciclo de vida do agente, como os métodos de inicialização (`setup()`) e destruição (`takeDown()`) do agente. Esta é a classe principal do agente, responsável por adicionar os comportamentos para realização das tarefas de gerenciamento do repositório;
- `MyAchieveReResponder:` é uma classe que estende o tipo de comportamento `AchieveReResponder` da API do JADE, o qual implementa os protocolos FIPA;
- `AddMetadataBehaviour:` comportamento que realiza a operação de adição de metadados no repositório;

- `QueryMetadataBehaviour:` comportamento que realiza a operação de consulta sobre os metadados;
- `DeleteMetadataBehaviour:` comportamento que realiza a operação de remoção de metadados do repositório;
- `UpdateMetadataBehaviour:` comportamento que realiza a operação de atualização dos metadados sobre o repositório;
- `RemoteQuery:` comportamento que realiza consultas remotas. Ela extende a classe `WakerBehaviour`, que é um comportamento que é executado apenas uma vez, pois depois de um tempo estabelecido (*timeout*) é finalizado. Quando a consulta remota é realizada, o `CatalogManagerAgent` espera um *timeout*, que é estabelecido pelo administrador do sistema, caso a resposta remota não chegue, ele entrega somente os resultados locais;
- `CatalogAbstractFactory:` esta classe é a responsável por instanciar os `wrappers` que encapsulam o tipo de sistema de armazenamento usado como repositório de metadados. Isto possibilita, juntamente com a linguagem de consulta especificada na Seção 4.2 obter-se independência com relação ao sistema de armazenamento;
- `ExpressionAnalyzer:` está classe é responsável por realizar o *parse* da *string* de consulta enviado ao *CatalogManagerAgent* pelos outros agentes.

SchemaManagerAgent

O `SchemaManagerAgent` é o agente responsável pela estensibilidade do esquema de metadados, ou seja, permite que sejam definidos novos atributos que caracterizem um dado.

O diagrama SASD do `SchemaManagerAgent` é apresentado na Figura 4.11. O `SchemaManagerAgent` é composto por oito classes, sendo uma delas a classe principal do agente e as outras classes de comportamentos que podem ser executados pelo mesmo.

As classes que compõem o diagrama SASD do `SchemaManagerAgent` são explicadas a seguir:

- `SchemaManagerAgent:` nesta classe encontram-se os métodos que controlam o ciclo de vida do agente, como os métodos de inicialização (`setup()`) e destruição

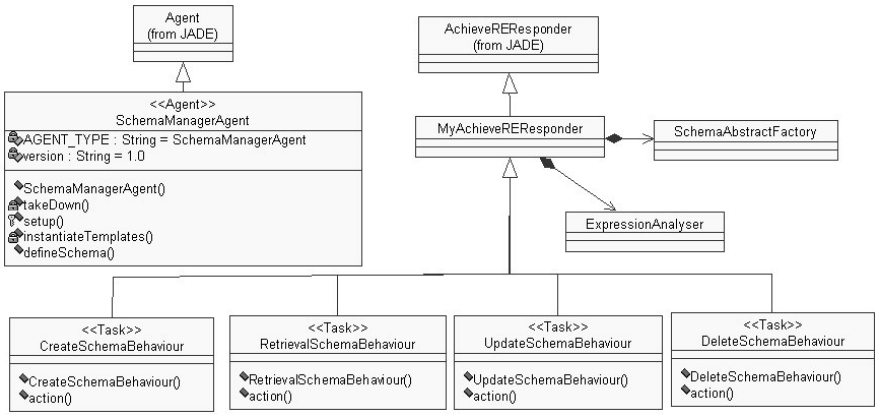


Figura 4.11: Diagrama de definição de estrutura do agente `SchemaManagerAgent`

(`takeDown()`) do agente. Esta é a classe principal do agente, responsável por adicionar os comportamentos para realização das tarefas de gerenciamento do repositório;

- `MyAchieveReResponder:` é uma classe que estende o tipo de comportamento `AchieveReResponder` da API do JADE, o qual implementa os protocolos FIPA;
- `CreateSchemaBehaviour:` comportamento que executa a tarefa de criação de novos esquemas de metadados;
- `RetrievalSchemaBehaviour:` comportamento executado para consultar os esquemas de metadados disponíveis;
- `DeleteSchemaBehaviour:` comportamento executado para remover um esquema de metadados;
- `UpdateSchemaBehaviour:` comportamento executado na atualização de um esquema de metadados;
- `SchemaAbstractFactory:` está classe é a responsável por instanciar os *wrappers* que encapsulam as funcionalides de criação de esquema de metadados para cada tipo de sistema de armazenamento usado como repositório de metadados;
- `ExpressionAnalyzer:` está classe é responsável por realizar o *parse* da *string* de consulta enviado ao `SchemaManagerAgent` pelos outros agentes.

DataManagerAgent

O `DataManagerAgent` é o agente responsável pelo armazenamento dos arquivos publicados junto ao `CatalogManagerAgent`.

O diagrama SASD do `DataManagerAgent` é apresentado na Figura 4.12. O `DataManagerAgent` é composto por seis classes, sendo uma delas a classe principal do agente e as outras classes de comportamentos que podem ser executados pelo mesmo.

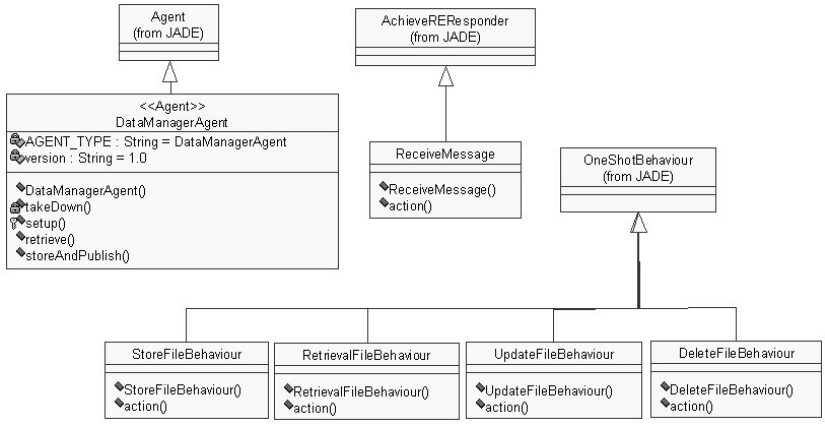


Figura 4.12: Diagrama de definição de estrutura do agente `DataManagerAgent`

As classes que compõem o diagrama SASD do `DataManagerAgent` são explicadas a seguir:

- `DataManagerAgent:` nesta classe encontram-se os métodos que controlam o ciclo de vida do agente, como os métodos de inicialização (`setup()`) e destruição (`takeDown()`) do agente. Esta é a classe principal do agente, responsável por adicionar os comportamentos para realização das tarefas de gerenciamento do repositório de arquivos publicados junto ao agente `CatalogManagerAgent`;

- `ReceiveMessage:` comportamento responsável por receber as mensagens que solicitam a execução de alguma tarefa. Pode ser entendido como um *listener* neste caso. Quando a mensagem é recebida, o agente identifica qual a ação a ser executada e então instancia uma dos comportamento que herdam da class `OneShotBehaviour` da API do JADE, o qual é um comportamento atômico e que só executa uma única vez após instanciado. Assim para cada requisição é criado um comportamente `OneShotBehaviour` para tratá-la individualmente;

- `StoreFileBehaviour:` comportamento executado para armazenar os arquivos no repositório local de arquivos, que corresponde, nesta implementação, a simplesmente o sistema de arquivos local;
- `RetrievalFileBehaviour:` comportamento executado para solicitar a busca e transferência por um arquivo armazenado;
- `UpdateFileBehaviour:` comportamento executado na atualização de um arquivo armazenado;
- `DeleteFileBehaviour:` comportamento executado para remoção de um arquivo armazenado.

4.4.4 Diagrama de Descrição de Comportamento Multiagente (MABD)

O diagrama de descrição de comportamento multiagente (MABD) expressa o fluxo de execução e comunicação dos agentes do sistema, através da interação entre seus agentes. É representado através de um diagrama de atividades UML. Estes diagramas são divididos em *swimlanes*, que correspondem aos comportamentos dos agentes, enquanto que as atividades correspondem aos métodos destes comportamentos. As setas entre atividades indicam alguma forma de comunicação entre os agentes ou invocação de métodos internos ao comportamente.

Dois cenários serão apresentados através do diagrama MABD: publicação de dados e descoberta com posterior transferência dos dados. Estes cenários foram desenvolvidos a partir do uso deste sistema integrado ao *middleware* MAG.

Publicação de Dados

Neste cenário um agente interage com os agentes `DataManagerAgent` e `CatalogManagerAgent` objetivando realizar o armazenamento de um determinado dado e publicação do mesmo. A Figura 4.13 ilustra o MABD realizado para publicação de dados.

O agente solicitante envia uma mensagem ao agente `DataManagerAgent` solicitando o armazenamento e publicação de um dado (`StoreFileAction`). O conteúdo

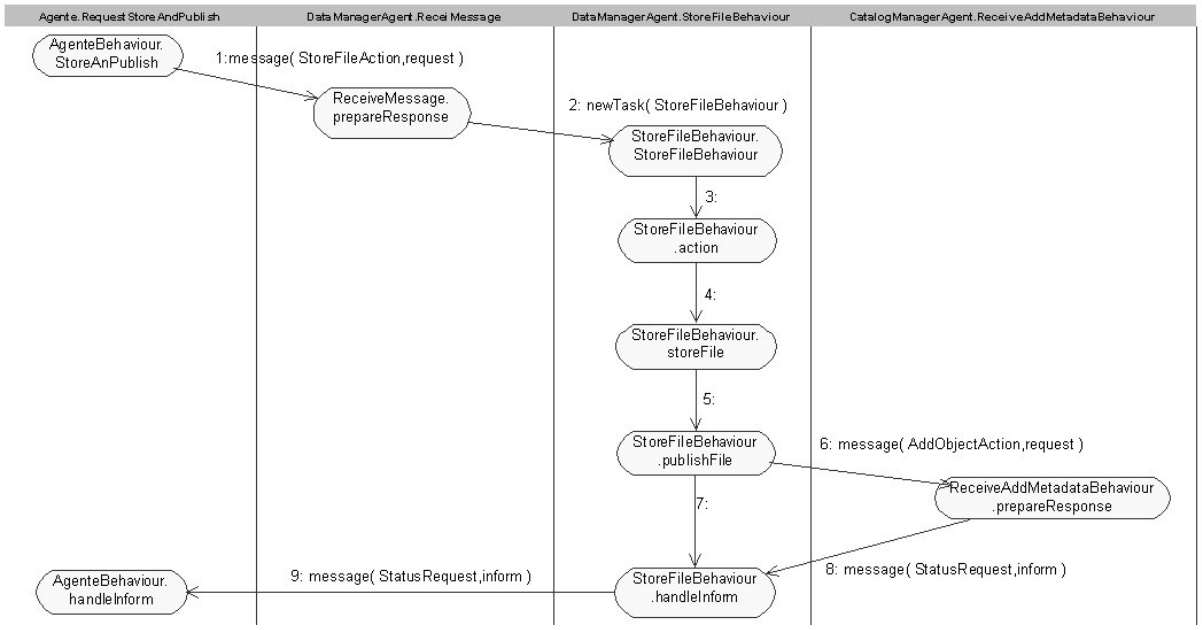


Figura 4.13: Diagrama MABD – publicação de dados

da solictação é o arquivo que ele deseja armazenar e uma *string* que é gerada a partir da grámatica da linguagem de consulta apresentada na Seção 4.2 para a publicação de dados junto ao `CatalogManagerAgent` (1). Ao receber a mensagem o `DataManagerAgent` identifica que o comportamento adequado para o atendimento da solicição é o de armazenamento de arquivos (`StoreFileBehaviour`), de forma que este é instanciado para atendimento àquela requisição (2). Na instanciação do comportamento, o método `action()` é invocado (3), que então invocanda o método `storeFile()` (4), que armazena o arquivo. Este após armazenar o arquivo, invoca o método `publishFile()` (5), que recebe a string enviado pelo agente e a envia para o `CatalogManagerAgent` (6) inserida em objeto *AddLogicalObjectAction*

Ao receber a mensagem, o `CatalogManagerAgent` processa a *string* e armazena os respectivos metadados no repositório. Concluído o processo de publicação, é enviada uma mensagem confirmando ou não a publicação dos dados (8), a qual é encaminhada para o agente que iniciou o cenário (9).

Descoberta de Dados

Neste cenário, novamente um agente interage com os agentes `DataManagerAgent` e `CatalogManagerAgent`, porém objetivando encontrar um dado com

características específicas. A Figura 4.14 ilustra o MABD realizado para a descoberta de dados.

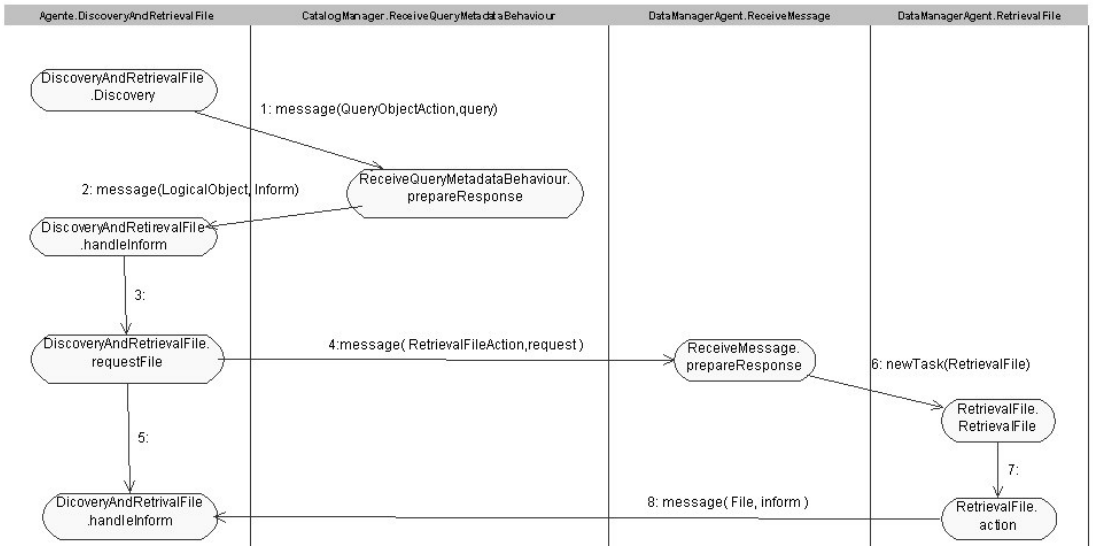


Figura 4.14: Diagrama MABD – descoberta de dados

Neste cenário, um agente realizada um consulta ao `CatalogManagerAgent` (1) enviando uma mensagem na qual contém uma *string* de consulta gerada a partir da gramática especificada na Seção 4.2. Nesta *string* deverá ser expresso as características do dado a ser encontrado. O `CatalogManagerAgent` recebe a mensagem, realiza a consulta ao repositório, e envia uma mensagem (2) com o resultado da consulta. Caso tenha encontrado algum dado publicado que esteja de acordo com as características especificadas na consulta enviada pelo agente solicitante, o `CatalogManagerAgent` envia todos os metadados que descrevem o dado encontrado e a identificação do dado, que pode ser utilizada para recuperar o conteúdo do dado junto ao `DataManagerAgent`.

Ao receber a mensagem informando o resultado da busca, se houve algum, o agente solicitante invoca o método `requestFile` (3) para o envio da mensagem de requisição do dado para o `DataManagerAgent` (4) e posterior recebimento do resultado (5). O método `requestFile` recebe como parâmetro a identificação do dado junto ao `DataManagerAgent` e o insere na mensagem de requisição do dado juntamento com a solitação de recuperação de dados (`RetrievalFileAction`).

Ao receber a mensagem o `DataManagerAgent` identifica que o comportamento adequado para o atendimento da solicição é o de recuperação de arquivos (`RetrievalFile`), de forma que este é instanciado para atendimento àquela requisição (6). Na instanciação do comportamento, o método `action()` é invocado (7) para recuperar o

arquivo do sistema de arquivos local e então, finalmente, enviá-lo para o agente solicitante (8).

4.5 Considerações Finais

O MagCat é um serviço de metadados para grades de dados e foi desenvolvido sobre o paradigma de agentes de software.

O MagCat foi projetado de forma a contemplar o conjunto de requisitos listadados na Seção 4.1, que permitem a descrição de dados gerados nos mais variados domínios de aplicação, escalabilidade e distribuição do serviço, disponibilidade, baixa latência sobre as consultas, segurança no acesso aos dados, independência entre os componentes de armazenamento de metadados lógicos e de localização física, um entendimento comum sobre o significado dos metadados, uma interface bem definida para publicação, armazenamento e recuperação dos metadados, bem como o agrupamento de dados relacionados.

O modelo de dados que compreende os conceitos Visão, Coleção e Arquivo permitiu satisfazer os requisitos de extensibilidade do esquema de metadados e agrupamento de dados relacionados.

Para possibilitar o acesso a diferentes sistemas de armazenamento onde podem estar armazenados os metadados, foi criado uma linguagem de consulta, que permitiu a manutenção da independência de sistema de armazenamento.

Devido à utilização do paradigma de agentes de software no desenvolvimento do MagCat, a metodologia de desenvolvimentos de sistemas multiagentes *Agil* PASSI foi empregada, pois técnicas e metodologias tradicionais não fornecem recursos suficientes para a representação de diversos aspectos relativos a sistemas baseados em agentes como, por exemplo, a modelagem da ontologia que compreende a representação do conhecimento dos agentes, entendimento comum sobre o significado dos metadados e identificação não ambígua do tipo de dado.

De acordo com a *Agil* PASSI, os agentes identificados foram: *CatalogManagerAgent*; *SchemaManagerAgent*; *SearchAgent*; *ReplicationManagerAgent*; *RequestMonitorAgent*; *SecurityAgent*; *DataManagerAgent*. Estes agentes são responsáveis por fornecer mecanismos que dêem suporte a satisfação dos requisitos como extensibilidade

do esquema de metadados, consultas em repositórios de metadados distribuídos, replicação e sincronização, monitoramento das consultas, autenticação, autorização, armazenamento e acesso aos dados.

A infra-estrutura básica implementada da arquitetura proposta neste trabalho compreende os agentes `CatalogManagerAgent`, `SchemaManagerAgent` e `DataManagerAgent`, os quais permitem a execução das principais funcionalidades de um serviço de metadados, que são a publicação e descoberta de dados. A interface para acesso a estas funcionalidades é implementada através do padrão de comunicação entre agentes FIPA, usando tecnologias de comunicação como Java RMI. Porém, com o objetivo de permitir que *middlewares* não orientados a agentes possam interagir com o MagCat, fazendo uso das funcionalidade básicas fornecidas, foram disponibilizadas interfaces CORBA.

5 Estudo de Caso e Avaliação de Desempenho

Neste capítulo apresentamos o uso do serviço de metadados proposto neste trabalho a partir de uma aplicação *Picture Archive and Communication System* (PACS), que foi integrada ao middleware MAG objetivando o uso da capacidade de armazenamento e recuperação de dados, bem como do mecanismo de execução de aplicações para executar duas aplicações: uma aplicação que extrai de imagens metadados baseados em textura; e outra aplicação que usa os metadados extraídos para comparar a similaridade entre as imagens. Este capítulo apresenta também avaliação de desempenho do estudo de caso com relação a sua execução no MAG. Além disso, apresentamos diversos experimentos que avaliam o desempenho do serviço de metadados no tocante às operações de armazenamento e consulta sobre os metadados.

5.1 Estudo de Caso

Atualmente, o diagnóstico médico através do uso de imagens tem aumentado, devido principalmente aos avanços no uso da computação na área da saúde, desenvolvimento de novas técnicas de aquisição, a pesquisa e aperfeiçoamento de algoritmos de manipulação de imagens. Assim, em ambientes hospitalares o diagnóstico baseado em imagens tem se tornado uma importante ferramenta na identificação de patologias.

5.1.1 Sistemas PACS

Um *Picture Archive and Communication System* (PACS) é composto por dispositivos de aquisição, processamento, armazenamento e visualização de dados interligados em rede. PACS aplicado à medicina armazena imagens de vários tipos como, por exemplo, raio-X, tomografia computarizada, ressonância magnética e ultrassonografia [Hua04].

O uso de sistemas PACS abriu novas possibilidades para a utilização de imagens médicas, pois os mais modernos permitem o acesso on-line a um grande volume de imagens.

Médicos realizam comparação de imagens para identificar doenças nos pacientes em estudo. Porém, as técnicas atuais para recuperação de imagens são bastante limitadas, permitindo somente busca simples baseadas, por exemplo, no nome do paciente, número de identificação e doença. Conseqüêntemente, para fazer uso mais eficiente do PACS ao analisar os dados de um paciente, os médicos necessitam combinar manualmente informações dos bancos de dados do hospital com um grande conjunto de informações correspontes às imagens disponibilizadas pelo PACS. Isto pode consumir muito tempo e em situações críticas pode ter como conseqüência a perda do paciente. Assim, é essencial permitir um rápido e eficiente acesso às imagens médicas, visto que as imagens são agora indispensáveis para muitos diagnósticos, planejamento de intervenção cirúrgica e tratamento de doenças.

5.1.2 Content-Based Image Retrieval (CBIR)

Atualmente, têm sido desenvolvidas técnicas para busca de imagens baseadas no conteúdo visual das mesmas, usando atributos e características da própria imagem como, por exemplo, forma, tamanho, contorno e textura, além das técnicas baseadas em descrições textuais.

A técnica chamada *Content-Based Image Retrieval* (CBIR)[GR95] propõe a recuperação de imagens através de características extraídas diretamente dos pixels da imagem. As características extraídas são usadas para recuperação de imagens como, por exemplo, consulta por similaridade entre imagens a partir dessas característcias. As características extraídas através da técnica CBIR baseam-se em textura, contornos dos objetos, geometria, histogramas baseados em níveis de cinza e coloridos, assim como também características mais abstratas como, por exemplo, coeficientes das transformadas de Fourier e Wavelets.

Sistemas baseados em CBIR são diferentes de sistemas de recuperação de imagens convencionais. Primeiramente, o método usado para indexar as imagens usa características extraídas do pixel da imagem, enquanto que em sistemas convencionais, a indexação é realizada com entradas textuais que descrevem o

conteúdo da imagem. Segundo, o método usado para recuperar basea-se na análise comparativa das características extraídas das imagens, enquanto que, em métodos convencionais, a recuperação é baseada em consultas textuais sobre bases de dados [Li01], [FF98], [MRD00], [Gal75].

5.1.3 IWA

IWA é um protótipo de um sistema PACS desenvolvido pelo grupo de pesquisa em processamento de imagens do Departamento de Engenharia de Eletricidade da UFMA. O objetivo deste sistema é auxiliar o diagnóstico médico, permitindo aos médicos a identificação, recuperação e análise das imagens similares a de um caso em estudo. A busca de imagens similares é baseada na comparação de características extraídas das imagens.

Atualmente o IWA permite somente o armazenamento, a recuperação e a extração de características de imagens de tomografia computarozadas no padrão *Digital Image and COmmunication in Medicin* (DICOM)[Clu00]. O padrão DICOM descreve um formato e protocolo de armazenamento e recuperação de imagens de servidores de imagens médicas. O arquivo é composto por um cabeçalho que contém metadados de duas espécies:

- relacionados ao paciente: nome, sexo, idade, hospital;
- relacionados à imagem: dispositivo de aquisição, parâmetros usadas na aquisição, data, número de imagens armazenadas, tamanho.

IWA disponibiliza as seguintes funções: integração de imagens com registros de pacientes; armazenamento e recuperação de imagens no formato DICOM; gerenciamento de informações dos pacientes, médicos e exames; realização de tratamento de imagem, como zoom, brilho e contraste; extração dos metadados contidos no cabeçalho da imagem DICOM.

5.1.4 PACS e Grades

Tecnologias de grades de dados estão sendo aplicadas em várias áreas de pesquisa como, por exemplo, estudos colaborativos em biomedicina e tratamento de

doenças (Esclerose Múltipla, Esquizofrenia, Doença de Parkinson, ADHD, Disordem de Tourette's, câncer cerebral), modelagem climática e astronomia. Pesquisas recentes também têm integrado grades de dados a *Picture Archiving and Communication systems* (PACS).

Tradicionalmente, PACS são implementados usando uma arquitetura cliente/servidor. Porém, a capacidade de processamento e armazenamento é limitada ao hardware local. As técnicas de extração de características de imagens e comparação, aliadas a um grande volume de imagens representam aplicações computacionalmente intensivas, as quais requerem grande capacidade de processamento, não disponível nos ambientes hospitalares. Grades de dados podem oferecer a PACS uma infra-estrutura para armazenar, publicar, recuperar, e processar grandes volumes de dados distribuídos, que podem estar dispersos por múltiplas instituições médicas.

Em nossas pesquisas, integramos o IWA ao *middleware* MAG, de forma a permitir o armazenamento, recuperação e processamento de imagens sobre a grade. O IWA mantinha em seu código as funcionalidade de armazenamento e recuperação de imagens, que usava um banco de dados relacional, e os algoritmos de extração de características baseadas em texturas e comparação através das características extraídas de imagens no padrão DICOM. Diante das possibilidades de armazenamento, publicação e recuperação de imagens e ainda a capacidade de processamento disponilizadas pelo *middleware*, o IWA foi refatorado e as funcionalidades citadas foram adaptadas para execução na grade.

O serviço de metadados proposto neste trabalho foi utilizado para armazenar metadados relacionados à imagem e relacionados ao paciente, bem como metadados que correspondem às características extraídas da imagem. Uma vez que as imagens estejam armazenadas e publicadas, tornam-se disponíveis para o público que possui acesso a grade. O serviço de dados do MAG, implementado através do agente `DataManagerAgent`, é o responsável pelas funcionalidades simples de armazenamento e recuperação de imagens na grade.

Para aproveitar a capacidade de processamento disponibilizada pelo MAG, foi retirado e adaptado para execução na grade o código do IWA que realizava a extração das características e o que realizava a comparação.

5.1.5 Esquema de Metadados para Imagens Médicas

A Tabela 5.1 mostra o esquema de metadados para o IWA, o qual estende os esquemas básicos definidos pelo MagCat.

Tabela 5.1: Esquema para imagens médicas

Esquema	**Extende**	**Atributos**
Exam	Collection	exam_type (e.g: x-rays, computer tomography, magnetic resonance, and ultra-sound), diagnosis, region
Medical Image	File	patient age, sex, image_type, region, symptom, diagnosis, treatment, size_x, size_y, slice, slice_thickness, device, aquisition_parameters, segmentation_method and segmentation_metadata, a field that contains: direction-0, angle_moment-0, contrast-0, entropy-0, homogeneity-0, variance-0, direction-45, angle_moment-45, contrast-45, entropy-45, homogeneity-45, variance-45, direction-90, angle_moment-90, contrast-90, entropy-90, homogeneity-90, variance-90, direction-135, angle_moment-135, contrast-135, entropy-135, homogeneity-135, variance-135

Quando se realiza um exame em um paciente como, por exemplo, de tomografia computarizada ou ressonância magnética, uma coleção de imagens é gerada. O esquema *Exam* representa uma coleção de imagens no esquema de metadados definido para o IWA. O esquema *Medical Image* descreve uma única imagem, armazenando metadados como, por exemplo, a região do corpo humano (*region*), o número da fatia (*slice*), dispositivo de aquisição (*acquisition device*), parâmetros de aquisição (*acquisition parameters*). Também são armazenados metadados correspondentes às características baseadas em textura, extraídas das imagens.

5.1.6 Publicação e Armazenamento de Imagens

Para extrair os metadados contidos no cabeçalho de imagens DICOM e as características baseadas em textura de imagens de tomografia computarizada, O IWA tinha eu código a implementação de um algoritmo de matriz de co-ocorrência. Este código foi retirado e adaptado para ser registrado no *ApplicationStorage* apresentado na Seção 3.2 e poder executar sobre o *middleware* MAG.

A matriz de co-ocorrência ou *Spatial Gray Level Dependence Method* (SGLDM) é uma técnica de análise de textura que tem sido usada em segmentação de imagens 2D [MGHW+99, Jai89, HSD73]. Matriz de co-ocorrência mostra a dependência espacial sobre níveis de cinza ao longo de diferentes relações ângulares (horizontal, verical e diagonal)

na imagem. A matriz de co-ocorrência é especificada pela freqüência relativa $P(i, j, r, \Theta)$ com que dois pixels, separados pela distância r, ocorrem ao longo do ângulo Θ, um com nível de cinza i e outro com nível de cinza j. A matriz de co-ocorrência é então uma função da distância r, ângulo Θ e escalas de cinza.

O esquema de metadados definido permite o armazenamento das características ao longo dos ângulos 0^0, 45^0, 90^0 e 135^0, como pode ser observado na Tabela 5.1.

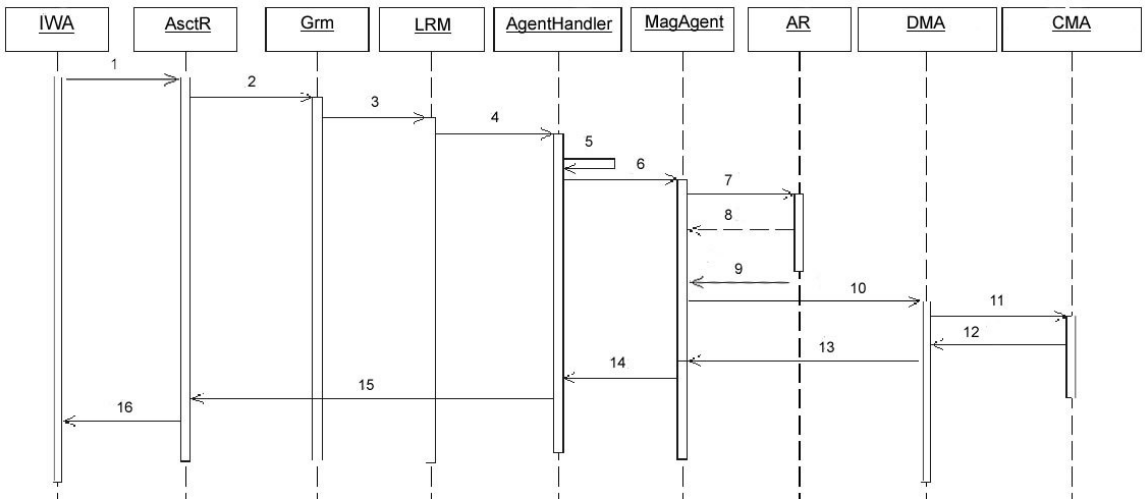


Figura 5.1: Diagrama de seqüência – publicação de imagens

A Figura 5.1 ilustra o processo de publicação das imagens, através de seus metadados. O usuário requisita a execução da publicação ao `IWA`, que redireciona a requisição ao `ASCTR` (a ferramenta de submissão e controle sem GUI, `ASCT`) (1),

O `ASCTR` redireciona a requisição para o `GRM` (2), que executa a seleção do nó para execução da aplicação. O `GRM` então redireciona a requisição para o `LRM` selecionado (3), que redireciona para `AgentHandler local` (4). O `AgentHandler` cria um novo `MagAgent` e armazena sua referência (5) para futuras comunicações. O `AgentHandler` passa para o `MagAgent` a identificação para a requisição do *bytecode* da aplicação (6) junto ao repositorio de aplicações (7,8), salvando no sistema de arquivos local (9). Ele então instancia a aplicação e inicia a sua execução, que realiza a extração dos metadados correspondentes ao método SGLDM sobre os ângulos 0^0, 45^0, 90^0 e 135^0 e os metadados do cabeçalho da imagem DICOM relacionados ao paciente e à própria imagem.

O `MagAgent` requisita ao `DataManagerAgent` (DMA) o armazenamento dos arquivos da imagem (10) e a publicação dos metadados. O `DataManagerAgent` (DMA)

armazena as imagens e requisita ao `CatalogManagerAgent` (CMA) a publicação dos metadados (11, 12). Então, o `DataManagerAgent` (DMA) notifica o `MagAgent` da finalização do processo (13). O `MagAgent` notifica o `AgentHandler` local quando a aplicação finaliza sua execução (14). A notificação é redirecionada para o `ASCTR` que originou a requisição (15), o qual notifica o `IWA` (16)

5.1.7 Descoberta e Recuperação de Imagens

Para acessar o histórico e imagens do paciente no `IWA`, o médico usa a identificação do paciente. São mostrados todas as informações do paciente como, por exemplo, exames anteriores. Durante a análise, o médico pode desejar ter acesso a informações de casos similares. Neste momento, a comparação de imagens de casos similares é muito importante. Na Figura 5.2 ilustramos a descoberta e recuperação de imagens baseada em metadados, através do mecanismo de execução de aplicações do MAG.

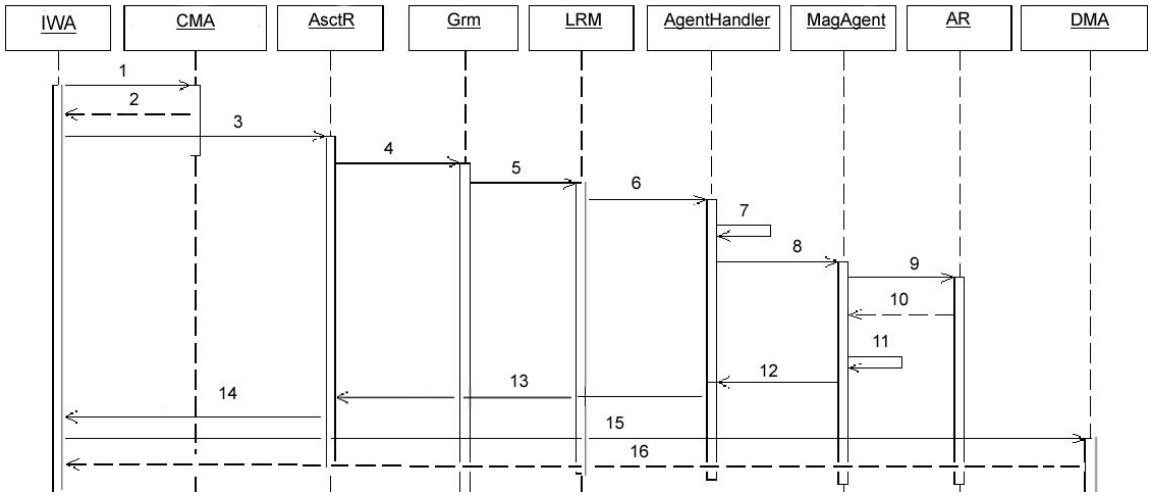


Figura 5.2: Diagrama de seqüência – descoberta de imagens

O médico requisita ao `IWA` a consulta por imagens similares ao caso em estudo. `IWA` redireciona a consulta para o `CatalogManagerAgent` (CMA) (1). Para realizar a consulta, o médico necessita informar metadados que identificam a imagem de seu interesse. É usada a linguagem de consulta descrita na Seção 4.2 para solicitar, por exemplo, " retorne as imagens de pacientes com idade entre 20 e 25 anos com diagnóstico de câncer nos pulmões". O `CatalogManagerAgent` (CMA) responde (2) com metadados de imagens como, idade do paciente, sexo, tipo de exame, sintomas, diagnóstico, região

do corpo humano a qual corresponde a imagem, tratamento, metadados extraídos com a técnica baseada em textura SGLDM e a identificação lógica da imagem junto ao `DataManagerAgent` (DMA).

`IWA` requisita a execução da descoberta da imagem através do `ASCTR` (3). Este, redireciona a requisição para o `GRM` (4). O `GRM` então redireciona a requisição para o `LRM` selecionado (5), que então se comunica com o `AgentHandler` local (6). O `AgentHandler` cria, instancia, armazena a referência do `MagAgent` (7, 8) e também passa a identificação da apliacação a ser executada, bem como os dados de entrada e parâmetros de execução. Os dados de entrada correspodem aos metadados extraídas com a técnica de análise de textura, os quais serão usadas para comparação entre as imagens. `MagAgent` então requisita a aplicação a ser executada (9, 10). A aplicação é então instanciada pelo `MagAgent` (11) e passa a executar o algoritmo que compara os metadados de uma imagem fonte, a do paciente em análise, e cada resultado da consulta ao `CatalogManagerAgent` (CMA). O resultado da execução é a porcentagem de similaridade entre as imagens. Finalizada a execução da aplicação, o `MagAgent` notifica o `AgentHandler` (12), que informa o `ASCTR` do término (13). O `ASCTR` então notifica o `IWA` (14), que finalmente requisita ao `DataManagerAgent` (DMA) a recuperação das imagens com maior porcentagem similaridade com a imagem do paciente em estudo (15, 16). Após recuperar as imagens, o `IWA` mostra as imagens, porcentagem de similaridade e metadados relacionados ao paciente.

5.1.8 Avaliação de Desempenho do IWA

Aplicações de extração e comparação de imagens como as usadas pelo IWA compreendem tarefas computacionalmente intensivas, dado um grande volume de imagens. É necessário que o *middleware* de grade execute essas aplicações de forma transparente, permitindo o acesso à capacidade computacional e aos dados sem negligenciar questões de desempenho, principalmente.

Durante a integração do IWA à grade, foram feitas avaliações de desempenho da aplicação de extração dos metadados. A aplicação foi executada de forma *standalone* e também sobre o MAG. Em nossos experimentos tentamos detectar o impacto do mecanismos de execução de aplicações do MAG sobre o comportamento da aplicação.

Os experimentos coletaram o tempo de execução para a aplicação quando da extração de metadados em lotes de 80, 100 e 120 imagens de tomografia computadorizada.

O primeiro experimento analisa a execução da aplicação de forma *standalone*, isto é, sem a interferência da grade. O segundo é a mesma aplicação, porém executada por um único agente, que usa o MAG para o processamento. Os outros experimento analisam, respectivamente, os casos para a aplicação sendo executada por dois, três e quatro agentes sobre o MAG, de forma paramétrica.

Na Tabela 5.2 apresentamos os resultados obtidos. Se considerarmos a execução de 120 imagens, pode-se observar que quando a aplicação é distribuída sobre duas máquinas, existe um ganho de 45%, o que mostra a eficiência do *middleware*. Quando três máquinas são usadas, nota-se um ganho de 59%, e ao usarmos 4 máquinas obtemos um ganho de 65%. Como pode-se ver, o MAG mostra-se estável e com desempenho consistente com adição de novas máquinas para execução da aplicação, influenciando positivamente o desempenho na execução da aplicação.

Tabela 5.2: Tempo de execução da aplicação de extração de metadados

Nº de imagens	Tempo (ms)				
	standalone	1 máquina	2 máquinas	3 máquinas	4 máquinas
80	638481.5	637097.2	411754.5	277400.0	247514.3
100	800331.4	819856.5	422096.2	342636.1	263334.2
120	923199.5	951755.8	515638.3	385773.4	316394.1

Pode-se observar, na Figura 5.3, que os resultados para execução da aplicação de forma *standalone* e a executada por um único agente sobre a grade são bem próximos, porém não chegam a ser sobrepostos. Nós concluímos que o MAG impõe uma penalidade mínima sobre a execução da aplicação usando apenas um agente.

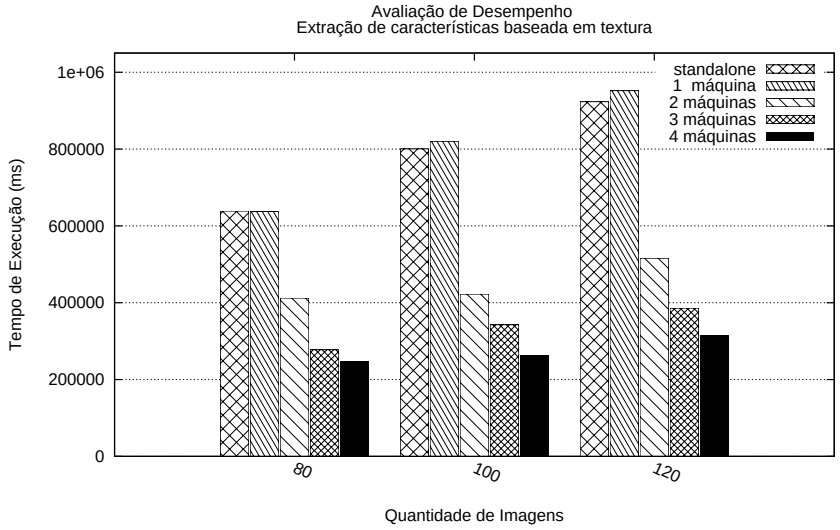


Figura 5.3: Avaliação de desempenho – Extração de características baseadas em textura

5.2 Avaliação de Desempenho do MagCat

Com o objetivo de obter informações que demonstrassem o comportamento do serviço de metadados, MagCat, diante da utilização de diferentes sistemas de acesso e armazenamento dos metadados, foram realizados vários testes que visaram avaliar o desempenho do serviço no tocante às operações de armazenamento e consulta dos metadados.

As operações avaliadas são funcionalidades de responsabilidade do agente `CatalogManagerAgent`, de forma que os testes realizados permitem inferir apenas a eficiência deste, dado o objetivo proposto.

Foram avaliados as implementações de acesso a um sistema de armazenamento relacional, utilizando o banco de dados MySQL e em LDAP, utilizando openLDAP[1].

Os testes foram realizados no Laboratório de Sistemas Distribuídos (LSD), pertencente ao Departamento de Informática da Universidade Federal do Maranhão. Para a realização dos experimentos foram utilizadas 2 máquinas, cujas especificações estão listadas na Tabela 5.3. Estas máquinas estavam interligadas através da tecnologia de rede Fast-Ethernet, que permite o tráfego de dados a 100 Mbps.

[1]http://www.openldap.org/.

Tabela 5.3: Especificação das Máquinas

Máquina	Processador	Memória RAM	Sistema
renoir	Intel Pentium 4 2.8 GHz	1 GB	Linux 2.6.10
monet	Intel Pentium 4 2.8 GHz	1 GB	Linux 2.6.10

Para avaliar as operações de armazenamento e consulta foram considerados fatores como variação da carga de requisições, quantidade de registros já existente na base de dados e quantidade de atributos a serem armazenados ou usados como filtro da consulta aos metadados. A métrica selecionada para análise da eficiência foi o tempo de resposta. O tempo de resposta analisado corresponde ao tempo transcorrido entre uma solicitação de armazenamento ou consulta aos metadados e o seu posterior resultado para o cliente.

5.2.1 Avaliação do Tempo de Resposta Mediante a Variação da Taxa de Chegada de Requisições

Operação de armazenamento de metadados

O objetivo deste experimento é analisar o comportamento do sistema, no tocante a operação de armazenamento de metadados, mediante a variação da taxa de chegada de requisições.

Para proceder com a medição do tempo de resposta estabelecemos um intervalo de geração de carga de 60 segundos. As cargas aplicadas durante intervalos distintos foram de 1000, 4000, 5000 e 7000 requisições. A geração de carga foi realizada através da implementação de um simples agente que realizou as requisições. Para nos aproximarmos de um ambiente real, onde vários clientes independentes realizariam as requisições, simulamos a independência das requisições por modelar o intervalo entre cada requisição usando uma distribuição exponencial [Jai91]. A distribuição exponencial possui a propriedade de não necessitar do momento do último evento ocorrido para inferir o instante do evento seguinte, o que foi ideal para o objetivo de simulação das requisições independentes. A expressão matemática utilizada para a geração deste intervalo foi calculada a partir da função de densidade de probabilidade da distribuição exponencial.

Foi criado um esquema de metadados que estendia o conceito `LogicalFile`, apresentado na Seção 4.2, para representar um arquivo qualquer a ser publicado. Em cada requisição de armazenamento foi inserida uma expressão de adição de metadados que seguiu a especificação da linguagem de consulta apresentada na Seção 4.2, com um número total de 16 atributos.

A Tabela 5.4 ilustra os resultados obtidos. A coleta dos tempos foi realizado pelo próprio agente que realizava as requisições ao final de cada rodada de submissões de um total de 30. Sobre os dados obtidos, foram calculadas estatísticas como média, desvio padrão, erro padrão e intervalo de confiança com grau de confiança de 0,95 ou 95%.

Tabela 5.4: Tempo de resposta (em ms) da operação de armazenamento de metadados mediante a variação da taxa de chegada de requisições

MySQL						
Nº de requisições	Max	Min	Des. padrão	Erro padrão	Média	Int. de confiança
1000	42,0	3,0	2,21	0,137	4,73	[4,59 ; 4,86]
4000	52,8	2,8	2,14	0,06	4,18	[4,12 ; 4,25]
5000	63,0	2,6	2,31	0,06	4,22	[4,15 ; 4,28]
7000	72,0	2,0	2,66	0,06	4,32	[4,26 ; 4,38]
LDAP						
Nº de requisições	Max	Min	Des. padrão	Erro padrão	Média	Int. de confiança
1000	46,4	6,6	2,75	0,17	10,12	[9,95 ; 10,29]
4000	75,8	6,6	6,08	0,18	12,5	[12,37 ; 12,74]
5000	122,8	6,6	11,94	0,33	15,93	[15,60 ; 16,26]
7000	197,6	6,6	28,49	0,66	28,75	[28,09 ; 29,42]

O gráfico da Figura 5.4 ilustra os tempos de resposta médios da operação de armazenamento de metadados mediante a variação da taxa de chegada de requisições no intervalo de 60 segundos.

Como podemos observar, os tempos de resposta médios mantém-se baixos e próximos com a variação da taxa de chegada de requisições para o MySQL, enquanto que para o LDAP há uma perda de desempenho. O desvio padrão mostra-nos pequena variabilidade em relação a média ao usar-se MySQL, enquanto que para o LDAP temos uma maior variabilidade. Analisando comparativamente, o uso do MySQL tem melhor

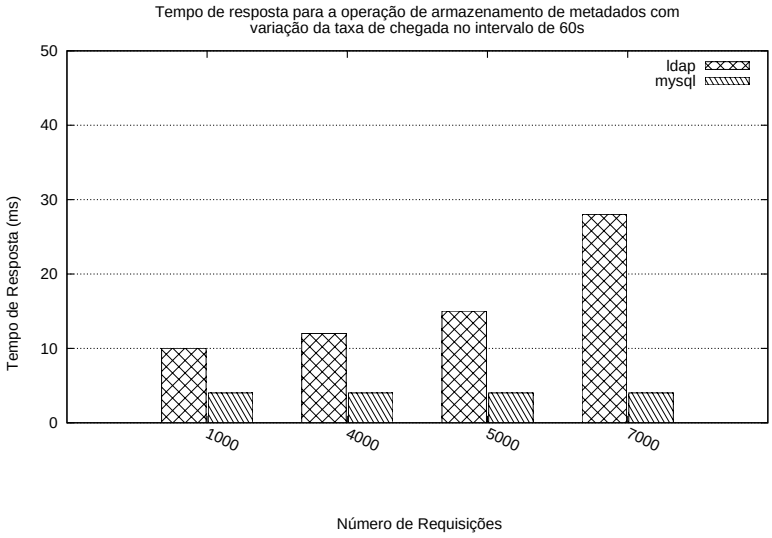


Figura 5.4: Tempo de resposta (em ms) da operação de armazenamento de metadados mediante a variação da taxa de chegada de requisições

desempenho. Se tomarmos como parâmetro a taxa de chegada de 7000 requisições em 60s, o tempo de resposta médio usando LDAP corresponde a quase sete vezes o tempo de resposta médio usando MySQL.

Operação de consulta sobre os metadados

A descrição deste experimento é como no experimento anterior, porém a operação a ser avaliada é a consulta sobre os metadados.

A Tabela 5.5 ilustra os resultados obtidos. Pode-se observar que ambos os sistemas de armazenamento apresentaram resultados similares. Assim, a variação da taxa de chegada de requisições não representa impacto considerável no tempo de resposta da consulta com o uso do MySQL ou LDAP.

O gráfico da Figura 5.5 ilustra os tempos de resposta médios da operação de consulta de metadados mediante a variação da taxa de chegada de requisições no intervalo de 60 segundos.

Tabela 5.5: Tempo de resposta (em ms) da operação de consulta sobre os metadados mediante a variação da taxa de chegada de requisições

MySQL						
Nº de requisições	Max	Min	Des. padrão	Erro padrão	Média	Int. de confiança
1000	56,0	4,2	4,55	0,28	8,21	[7,93 ; 8,50]
1500	60,2	4,0	2,86	0,14	6,33	[6,18 ; 6,47]
2000	63,0	4,0	2,77	0,12	6,17	[6,05 ; 6,29]
LDAP						
Nº de requisições	Max	Min	Des. padrão	Erro padrão	Média	Int. de confiança
1000	62,6	5,0	4,38	0,27	7,20	[6,92 ; 7,47]
1500	64,0	4,2	3,63	0,18	6,54	[6,36 ; 6,72]
2000	84,8	4,0	3,44	0,15	6,33	[6,18 ; 6,48]

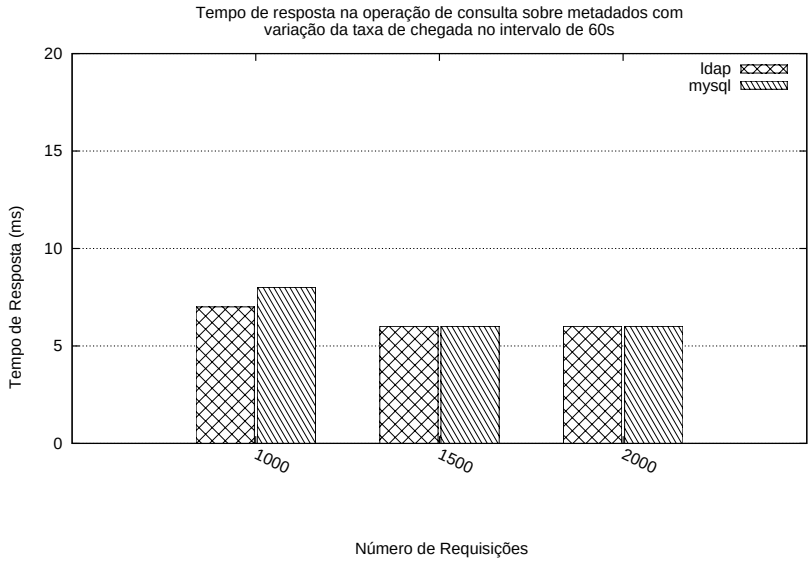


Figura 5.5: Tempo de resposta (em ms) da operação de consulta sobre os metadados mediante a variação da taxa de chegada de requisições

5.2.2 Avaliação do Tempo de Resposta para Operação de Armazenamento de Metadados Mediante a Variação da Quantidade de Registros na Base

O objetivo do experimento é analisar o comportamento do sistema, no tocante a operação de armazenamento de metadados mediante a variação da quantidade de registros armazenados na base.

Para proceder com a medição do tempo de resposta variamos a quantidade de registros armazenados no repositório, a qual correspondereu a uma base com 2500, 10000 e 20000 registros. Um agente simples foi o responsável por requisitar o armazenamento de metadados, através do uso da linguagem de consulta apresentada na Seção 4.2 e um esquema de metadados simples que estendeu o conceito `LogicalFile`. Cada requisição de armazenamento de metadados possuiu 16 atributos.

Neste experimento, o agente fazia a requisição do armazenamento e esperava pela resposta de confirmação para em seguida realizar a nova requisição. Para que o tamanho da base mantivesse seu tamanho fixo, cada novo registro armazenada era apagado em seguida. Foram realizadas um total de 100 requisições para obtenção dos dados apresentados na Tabela 5.6. O Intervalo de confiança é de 0,95 ou 95%.

Tabela 5.6: Tempo de resposta (em ms) da operação de armazenamento de metadados mediante a variação do tamanho de base

MySQL						
Nº de registros	Max	Min	Des. padrão	Erro padrão	Média	Int. de confiança
2500	41	6,00	5,48	1,07	11,54	[10,46 ; 12,61]
10000	41	6,00	5,15	1,01	11,21	[10,19 ; 12,22]
20000	41	6,00	4,96	0,97	11,34	[10,36 ; 12,31]
LDAP						
Nº de registros	Max	Min	Des. padrão	Erro padrão	Média	Int. de confiança
2500	59	19	4,19	0,86	52,62	[51,75 ; 53,48]
10000	59	19	4,26	0,88	52,93	[52,05 ; 53,81]
20000	58	20	4,14	0,85	53,05	[52,20 ; 53,90]

Pode-se notar, pelas médias, que o MySQL possui um melhor desempenho se comparado ao LDAP. O desvio padrão mostra a pequena variabilidade dos tempos de respostas em relação as médias encontradas para ambos os sistema LDAP e MySQL.

O gráfico da Figura 5.6 ilusta o tempo de resposta em milessegundos da operação de armazenamento de metadados mediante a variação do tamanho da base.

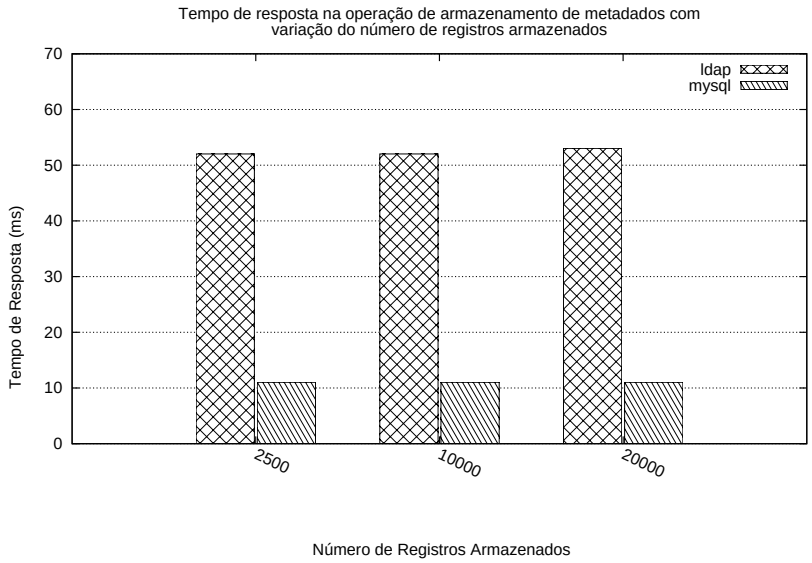


Figura 5.6: Tempo de resposta (em ms) da operação de armazenamento de metadados mediante a variação do tamanho de base

5.2.3 Avaliação do Tempo de Resposta Mediante a Variação da Quantidade de Atributos

Operação de armazenamento de metadados

O objetivo deste experimento é analisar o comportamento do sistema mediante a variação do número de atributos em uma requisição de armazenamento de metadados.

A quantidade de atributos em um registro compreendeu a 11, 16, 56 e 106 atributos em cada registro. Um agente simples foi o responsável por requisitar o armazenamento de metadados, através do uso da linguagem de consulta a apresentada na Seção 4.2 e um esquema de metadados simples que estendeu o conceito *LogicalFile*.

Neste experimento, o agente fazia a requisição do armazenamento e esperava pela resposta de confirmação para em seguida realizar a nova requisição. Para que o tamanho da base mantivesse seu tamanho fixo, cada novo registro armazenada era apagado em seguida. Foram realizadas um total de 100 requisições para obtenção dos dados apresentados na Tabela 5.7.

Como pode ser obervado, em ambos os sistemas, MySQL e LDAP, a quantidade de atributos em uma requisição de armazenamento influe no tempo de resposta, chegando ao dobro se compararmos o tempo de resposta para 11 e 106 atributos. Contudo, o

Tabela 5.7: Tempo de resposta (em ms) da operação de armazenamento de metadados mediante a variação da quantidade de atributos

MySQL				
Nº de atributos	Max	Min	Desvio padrão	Média
11	45	6	4,95	9,25
16	40	7	3,81	9,42
56	46	8	4,06	10,12
106	58	14	5,77	18,77
LDAP cliente				
Nº de atributos	Max	Min	Desvio padrão	Média
11	43	8	4,53	11,77
16	46	9	3,83	12,19
56	50	11	3,83	12,75
106	73	16	7,68	22,83

desempenho dos dois sistemas é similhar, não apresentando diferença significava para o usuário final. O gráfico da Figura 5.7 ilustra os tempos médios de resposta para este experimento.

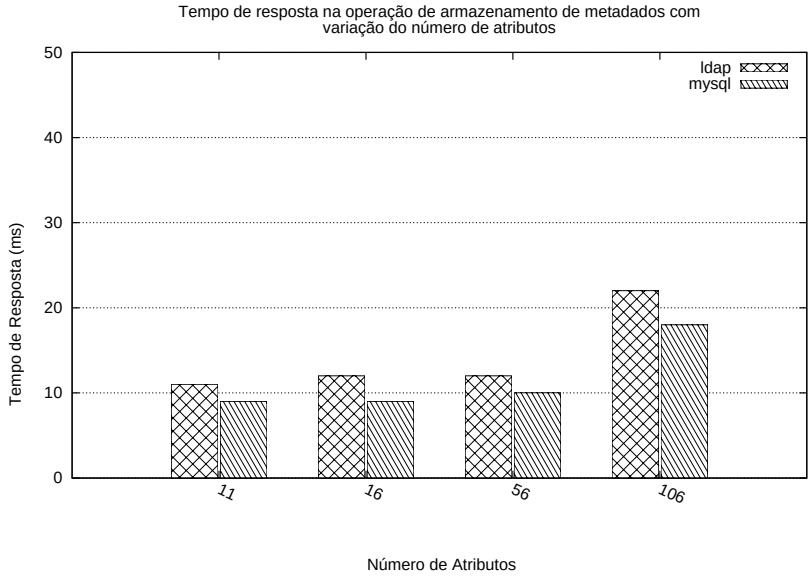


Figura 5.7: Tempo de resposta (em ms) da operação de armazenamento de metadados mediante a variação da quantidade de atributos

Operação de consulta sobre os metadados

O objetivo deste experimento é analisar o comportamento do sistema, no tocante à operação de consulta mediante a variação do número de atributos como critério de filtragem do resultado da consulta.

Para realizar este experimento variamos o número de atributos na requisição de consulta ao sistema. A quantidade de atributos em uma consulta compreendeu a 11, 16, 56 e 106 atributos em cada registro. Estabelecemos uma base com 1000 registros para realizar o experimento, e condicionamos as consultas a retornaram apenas um único registro.

Um agente simples foi o responsável por requisitar as consultas sobre os metadados, através do uso da linguagem de consulta apresentada na Seção 4.2 e um esquema de metadados simples que estendeu o conceito `LogicalFile`. Foram realizadas 100 requisições para cada quantidade de atributos, cujos resultados obtidos estão apresentados na Tabela 5.8.

Tabela 5.8: Tempo de resposta (em ms) da operação de consulta sobre os metadados mediante a variação da quantidade de atributos

MySQL						
Nº de atributos	Max	Min	Des. padrão	Erro padrão	Média	Int. de confiança
11	105	8	11,79	2,31	15,87	[13,55 ; 18,18]
16	108	9	11,75	2,30	16,16	[13,85 ; 18,46]
56	115	14	17,56	3,44	24,51	[21,06 ; 27,95]
106	134	24	17,34	3,40	34,8	[31,39 ; 38,20]
LDAP						
Nº de atributos	Max	Min	Des. padrão	Erro padrão	Média	Int. de confiança
11	74	9	9,09	1,78	16,02	[14,23 ; 17,80]
16	75	10	8,96	1,75	16,40	[14,64 ; 18,15]
56	86	17	10,06	1,97	23,66	[21,68 ; 25,63]
106	113	31	10,97	2,15	38,00	[35,84 ; 40,15]

O gráfico da Figura 5.8 ilustra os tempos médios de resposta para este experimento.

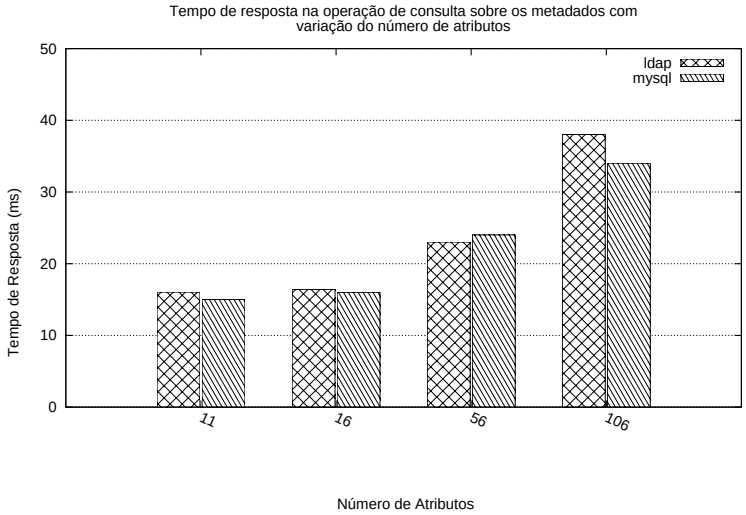


Figura 5.8: Tempo de resposta (em ms) da operação de consulta sobre os metadados mediante a variação da quantidade de atributos

Como pode ser obervado, em ambos os sistemas, MySQL e LDAP, a quantidade de atributos em uma requisição de consulta, como era de se esperar, influi no tempo de resposta, chegando a mais que o dobro se compararmos o tempo de resposta para 11 e 106 atributos. Contudo, o desempenho dos dois sistemas é similhar, não apresentando diferença significava para o usuário final.

5.3 Considerações Finais

O estudo de caso que compreende a aplicação PACS para o domínio médico nos permitiu constatar que os mecanismos do MagCat são suficientemente flexíveis e adaptáveis a diversos domínios. Esta flexibilidade é um dos principais requisitos apresentados para um serviço de metadados conforme descrito na Seção 4.1.

Comprovamos também que o uso do *middleware* de grade MAG permite disponibilizar recursos computacionais às aplicações, resultando em ganho de desempenho na execução das mesmas.

Observamos ainda que o processo de reengenharia das aplicações necessário para adaptá-las para execução na grade mostrou-se ser um procedimento de fácil realização.

A avaliação do MagCat usando tanto MySQL quanto LDAP como sistemas de armazenamento dos metadados, demonstrou que o serviço apresenta desempenho estável e tempos de resposta satisfatórios nas operações de armazenamento e consulta diante da variação da carga de requisições, tamanho da base de metadados e quantidade de atributos utizados para descrever os dados.

6 Trabalhos Relacionados

Este capítulo descreve relevantes projetos de serviços de metadados para grade de computadores. Após um resumo de cada trabalho faz-se uma análise comparativa com a proposta desta dissertação.

6.1 MCAT

MCAT [RWMS04] é um catálogo de metadados que compõe a arquitetura do middleware *Storage Resource Broker* (SRB) [BMRW98]. MCAT é o responsável por fornecer informações de autorização, autenticação, e informações sobre os dados ao middleware SRB.

A arquitetura do MCAT é ilustrada na Figura 6.1, cujos componentes são:

- MAPS (`Metadata Attribute Presentation Structure`): é uma estrutura de dados usada pelas aplicações para obter informações do MCAT. Esta estrutura é mapeada para o modelo de dados do sistema de armazenamento usado pelo MCAT como, por exemplo, o modelo relacional de dados. Existem três tipos de estruturas:
  - `MAPS_Query_Struct`: usada para realizar uma consulta;
  - `MAPS_Result_Struct`: usada pra obter os resultados de uma consulta;
  - `MAPS_Update_Struct`: usada para adicionar e atualizar os metadados;
  - `MAPS_Definition_Struct`: usada para definir o mapeamento entre o esquema de dados usado pelo sistema de armazenamento e as demais estruturas MAPS.
- `MAPS to Schema Converter`: é o módulo responsável por mapear a estrutura MAPS para o esquema de dados do sistema de armazenamento;
- `Dynamic Query Generator`: é o módulo usado para gerar consultas para o sistema de armazenamento utilizado pelo MCAT como, por exemplo, consultas SQL para Oracle ou DB2. Ele recebe o mapeamento realizado pelo `MAPS to Schema Converter` e gera as consultas específicas para o sistema de armazenamento em uso;

- `Answer Extractor and Cursor Control`: interage com o sistema de armazenamento e recebe as respostas das consultas requisitas pelos clientes. Ao requisitar uma consulta, o cliente especifica o número de tuplas retornadas de cada vez e se existem mais tuplas que as especificadas pelo cliente na consulta. O `Answer Extractor and Cursor Control` possibilita o acesso às demais tuplas restantes;

- `Schema to MAPS Converter`: recebe as tuplas a serem retornadas para o cliente e as insere na estrutura MAPS de retorno de resultados;

- `MAPS semantics` e `Schema semantics`: são módulos usados para associar os metadados no MCAT a metadados específicos de domínio e específicos para aplicações.

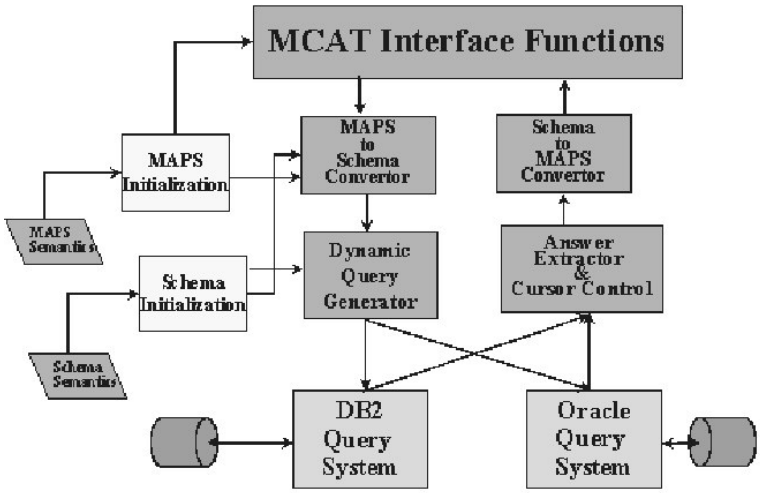


Figura 6.1: Arquitetura MCAT

MCAT tem como características: um esquema de metadados extensível; provê o conceito de coleções de dados; provê um espaço de nomes global, onde cada dado é identificado unicamente em toda a federação de servidores SRB; gerencia metadados de descrição de conteúdo dos dados; gerencia metadados de localização física como, por exemplo, qual *SRB Master* controla o dado e em qual nó este se encontra; gerencia metadados de autorização através de listas de controle de acesso e *tickets*, os quais representam privilégios sobre as coleções de dados e dados individuais; mantém metadados

de autenticação (usuário e senha) do SRB; possui uma API que disponibiliza consultas baseadas em atributos.

Em sua versão 3.0, foi implementada a aquitetura distribuída do MCAT através da federação de zonas. Zonas correspondem a sistemas SRB, cada um com seus próprios servidores SRB, um único MCAT, usuários, dados e administração local. Essas zonas são organizadas em uma federação. Com a federação de MCAT's, torna-se possível: melhorar o desempenho nas consultas, pois nas versões anteriores o catálogo se encontrava centralizado, e caso fosse necessária a consulta a múltiplos catálogos, incorreria em alta latência em um ambiente WAN; disponibilizar controle local aos *sites* que queiram compartilhar seus recursos, pois a gerencia do catálogo não se encontra centralizada; prover maior escalabilidade, devido a federalização de catálogos distribuídos. A implementação da federação de catálogos segue um modelo ponto-a-ponto, onde cada MCAT possui metadados que identificam todas as zonas da federação como, por exemplo, o nome da zona e endereço de rede.

A utilização do MCAT está fortemente acoplada à dos demais componentes do SRB. Isto porque para obter informações dos dados requeridos, autenticar e autorizar o cliente requisitante, o servidor consulta o MCAT. Toda a interação das aplicações clientes com o MCAT é realizada através do SRB Agente e dado que este agente é instanciado pelo servidor SRB pelo menos um servidor SRB deve ser instalado sobre a mesma máquina que hospeda o MCAT.

A implementação do catálogo, embora o projeto objetive o uso de qualquer sistema de armazenamento, é realizada sobre sistemas de banco de dados relacional. Pode-se utilizar sistemas como Oracle, IBM DB2, Sybase and Postgres. A federação não possibilita também a integração com outros serviços de metadados. Não há divisão entre os componentes de armazenamento de metadados lógicos ou descritivos de metadados de armazenamento físico ou de réplica. MagCat, por sua vez, é independente de middleware de grade, pode ter os metadados armazenados em qualquer sistema de armazenamento, bem como pode interagir com outros serviços de metadados através do encapsulamento destes por outras agentes e uso de sua linguagem de consulta. MagCat armazena apenas metadados lógicos, deixando sob a responsabilidade do serviço de replicação a gerência de metadados de localização física do dado. MagCat é implementado sobre o paradigma de agentes de software, onde a interação entre os agentes segue o padrão de mensagem FIPA ACL e Java RMI.

6.2 Metadata Catalogue Service (MCS)

MCS [SBC+03, CDK+02] é o serviço de metadados mais utilizado em projetos de grades de computadores atualmente. É desenvolvido como parte dos projetos *Grid Physics Network* (GriPhyN)[1] e NVO[2].

MCS é implementado como um serviço Web. Sua implementação usa o servidor Web Apache[3] e o sistema de banco de dados relacional MySQL. Os componentes da arquitetura do MCS são ilustrados na Figura 6.2. A aplicação cliente consulta o MCS através da API MCS. O Cliente envia consultas para o servidor MCS usando *Simple Object Access Protocol* (SOAP) [SOA]. O servidor MCS então interage com o sistema de banco de dados para executar a consulta, recebe os resultados e os envia de volta para o cliente, usando SOAP novamente.

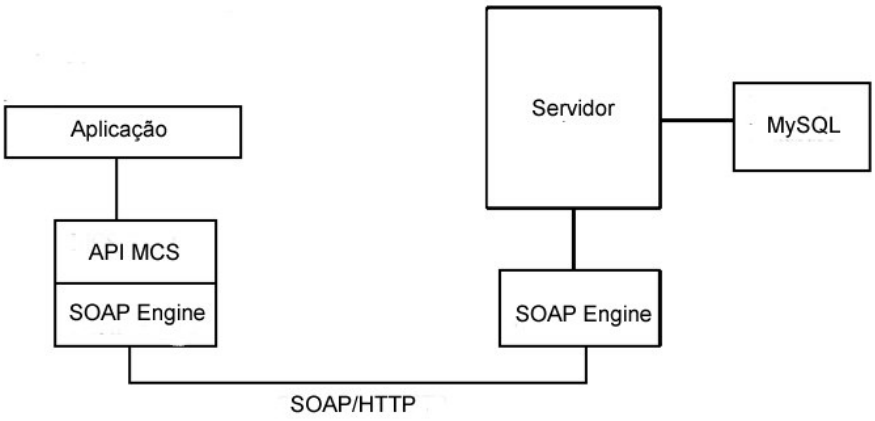


Figura 6.2: Arquitetura MCS

MCS tem como características: considera como mecanismos distintos o serviço de armazenamento e acesso de metadados descritivos do serviço de armazenamento e acesso a metadados de localização física, que geralmente é localizado em serviços de replicação [CDF+02]; assume o arquivo como modelo de dado básico, organizando-o logicamente em coleções e visões; disponibiliza uma API simples para armazenamento e consulta aos metadados; disponibiliza um esquema de metadados genérico que engloba metadados para arquivo, coleção, visão, autorização, autenticação, usuários, ciclo de vida de um arquivo, identificação de catálogo de metadados externo e metadados definidos pelo

[1] http://www.griphyn.org/
[2] http://www.us-vo.org/
[3] http://www.apache.org/

usuários, os quais permitem que o esquema de metadados seja extensível. Embora seja desenvolvido no seio dos projetos *Grid Physics Network* (GriPhyN) e NVO, MCS pode ser utilizado para dar suporte ao serviço de metadados em qualquer middleware de grade.

Algumas das idéias encontradas no MagCat tem relação direta com o MCS. Ambos têm um esquema de metadados extensível, são independentes de middleware de grade e proveêm uma clara separação entre metadados descritivos e metadados de localização física. Porém MagCat e MCS são projetos bem distintos na proposta de suas arquiteturas. MCS é desenvolvido sobre o conceito de serviços web. Seus componentes comunicam-se através do protocolo SOAP sobre HTTP. Uma aplicação cliente deve utilizar a API do MCS para se comunicar com o servidor MCS. Seu sistema de armazenamento é restrito a um sistema relacional. MagCat propõe uma abordagem baseada em agentes de software. Toda a comunicação entre os agentes é feita através dos padrões FIPA-ACL e Java RMI. Para que um agente, cliente, se comunique com os agentes que compõem a arquitetura do MagCat, é necessário que eles conheçam a ontologia de domínio especificada na Seção 4.2. MagCat é independente de sistema de armazenamento, visto que ele usa uma linguagem de consulta própria, reconhecida pelos agentes, a qual é mapeada para a linguagem de consulta particular do sistema de armazenamento.

6.3 Grid-Based Metadata Services

Ewa Deelman et. al [DSA+04] propõem a integração do MCS ao *Open Grid Service Architecture - Data Access Integration* (OGSA-DAI) [ADG+03], que por sua vez tem como base de implementação o middleware de grade Globus. A Figura 6.3 ilusta uma visão em alto nível das camadas da proposta.

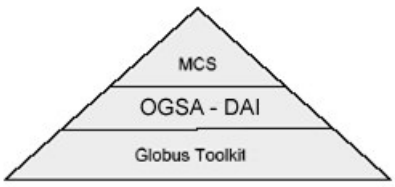


Figura 6.3: Visão em camadas da implementação do MCS sobre OGSA-DAI

A integração do MCS ao OGSA-DAI é possibilitada através de atividades (*activity*), as quais permitem adicionar funcionalidades ao OGSA-DAI. Para a camada

MCS no topo do OGSA-DAI, foi definida uma atividade específica que contém funções correspondentes à API do MCS, como, por exemplo, adição, remoção e consulta por arquivos lógicos, coleções e visões. Com a integração com OGSA-DAI, MCS mantém as caracteríticas citadas na Seção 6.2, exceto quanto à independência de middleware, pois passa a ter forte dependência do middleware OSGA-DAI. Nós pretendemos integrar o OGSA-DAI ao MAG para fornecimento do serviço de acesso a dados, encapsulando-o em um agente responsável pela interface entre o serviço Web e a sociedade de agentes. Assim, mantem-se a independência de middleware.

6.4 Artemis

Artemis é um sistema que objetiva integrar catálogos de metadados distribuídos. Artemis [TTGD04] explora técnicas como mediador de consultas, ontologias e ferramentas da web semântica para formular consultas sobre os catálogos. Artemis é integrado com o MCS.

A argumentação em favor desta proposta é que arquitetura flexível do MCS possibilita a adição de novos atributos de metadados, o que dificulta a integração dos catálogos por rapidamente mudar o esquema de metadados não possibilitando definir um modelo de conteúdo previamente e ainda não provê especificação semântica sobre os metadados. Para tratar esses problemas, Artemis cria dinamicamente modelos de informação disponível em vários catálogos e usa ontologias para descrever, mapear, e organizar informações de metadados. A geração dinâmica do modelo consiste em criar um modelo relacional de domínio através da consulta a vários catálogos. O modelo de domínio lista entidades, visões, coleções e itens. Para cada tipo de entidade, Artemis cria um predicado de domínio agrupando os respectivos atributos encontrados. Após listar cada tipo de entidade, Artemis cria o modelo de domínio a partir da junção de atributos de entidades comuns aos catálogos consultados. Tal modelo de domínio é utilizado pelo mediador de consulta. Como exemplo, veja a Tabela 6.1, que ilustra duas instâncias do MCS. Após a criação do modelo de domínio, tem-se o resultado expresso na Figura 6.4.

Como se pode observar, o modelo de domínio é uma combinação de atributos dos catálogos consultados. Este modelo será utilizado pelo mediador para realizar as consultas. O mapeamento para ontologias é feito entre os termos definidos no catálogo e os

Tabela 6.1: Exemplo de duas instâncias do MCS

MCS	Tipo de objeto	Atributos
MCS1	Item	Keyword, starttime, endtime
MCS2	Item	ci, about, sttime, etime

```
items(keyword, starttime, endtime,ci,about, sttime, etime)
:-
Mcs1items(keyword, starttime, endtime)^
(ci = "empty") ^
(about = "empty")^
(sttime = "empty")^
(etime = "empty")

items(keyword, starttime,endtime,ci, about, sttime, etime)
:-
Mcs2items(ci, about, sttime, etime)^
(keyword = "empty")^
(starttime = "empty")^
(endtime = "empty")
```

Figura 6.4: Modelo de domínio gerado pelo Artemis

conceitos representados em uma ontologia por perguntas interativas ao usuário. Artemis ainda utiliza dois outros compomentes: Prometeus, um sistema mediador; e Theseus, que executa consultas em paralelo a várias fontes de dados.

A proposta do Artemis de associação de semântica é bastante interessante, pois possibilita ao usuário filtrar as consultas de acordo com o domínio desejado. Apesar do Artemis prover consultas distribuídas, não contempla o acesso a serviços de metadados heterogêneos, possuindo forte dependência da arquitetura MCS. A arquitetura do MagCat é totalmente independente de sistema de armazenamento devido ao mapeamento da linguagem de consulta para a linguagem de consulta do sistema de armazenamento em uso. MagCat pode interagir com outros serviços de metadados através de *wrappers* instanciados pelo agente `CatalogManager`.

6.5 Distributed Medical Data Manager - DM^2

DM^2 [JJL+, HJV+, DMP+03] é um sistema distribuído desenvolvido como parte do projeto MEDIGRID, que é financiado pelo *IST European DataGrid*[4], *French*

[4] www.edg.org

ministry for research ACI-GRID[5] e *ECOS-Nord Committee* (*action* C03S02), com o propósito de gerenciar dados médicos. Ele prove interface entre servidores de imagens médicas, usualmente imagens no padrão DICOM, e middlewares de grade como se pode observar na Figura 6.5. Uma preocupação constante na gerência de imagens médicas em grades é a confidencialidade das informações dos pacientes, de forma que a interface de armazenamento do DM^2 assegura a segurança dos dados por anominizar e criptografar os dados enviados para armazenamento e acesso na grade. As imagens são replicadas usando o serviço de replicação do middleware de grade.

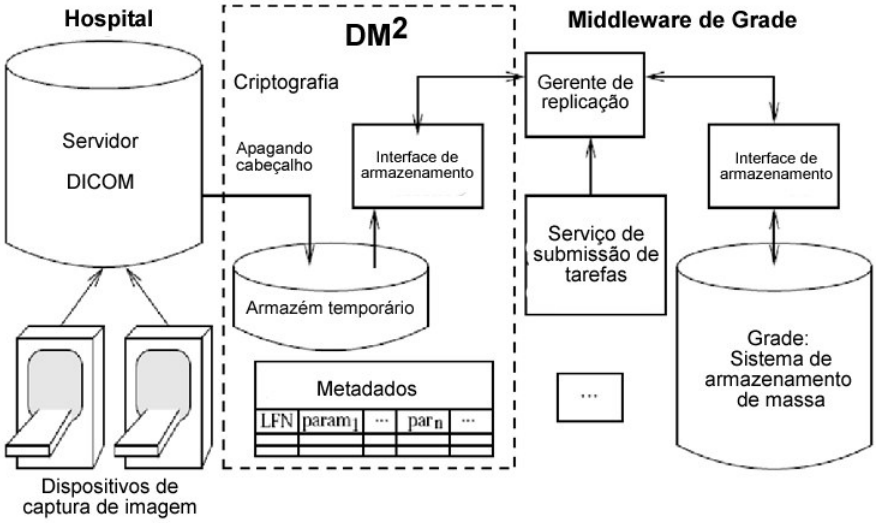


Figura 6.5: DM^2 - Interface entre servidores de imagens DICOM e a grade

DM^2 possui um serviço de metadados [PSDM04] integrado a sua arquitetura, o qual gerencia metadados relacionados a: pacientes (idade, sexo); à imagem (tamanho, resolução); à aquisição de imagens (dispositivo usado, parâmetros de aquisição, data); ao hospital (departamento responsável pela aquisição); ao registro médico (informações para interpretar a imagem); segurança (autorização, login de acesso a dados, chaves criptográficas); ao histórico do dado (origem da imagem, algoritmo usado, parâmetros), de forma a poder recuperar o dado original; e metadados relacionados à otimização de consultas (identificador da imagem armazenada, cache de consultas, cache de processamento). DM^2 disponibiliza dois tipos de consultas: basicas e híbridas. As consultas básicas envolvem metadados como, por exemplo, relacionados à imagem e paciente. As consultas híbridas associam computação sobre as imagens usando a grade. Um exemplo de consulta híbrida seria: o usuário fornece uma imagem fonte para o sistema,

[5] www-sop.inria.fr/aci/grid/public

e quer encontrar todos os pacientes com idades entre 40 e 60 anos que tenham imagens similares à imagem fornecida. Assim, o sistema seleciona os pacientes que tem idade no intervalo dado, executa o cálculo de similaridade entre as imagens, e então recebe os resultados. O resultado do cálculo é também armazenado. A implementação foi realizada sobre o middleware da European DataGrid e o bando de dados relacional MySQL. Todos os componentes do DM^2 são implementados em C++, e o acesso ao sistema é realizado também através de uma API em C++.

MagCat e o serviço de metadados do DM^2 diferem significamente na proposta de suas arquiteturas. No DM^2 o catálogo faz parte de uma arquitetura dedicada ao gerenciamento de dados médicos. O catálogo gerencia metadados de imagens médicas, sendo que o esquema de metadados não é extensível. Ele também gerencia metadados de sistemas como, metadados de autorização, autenticação e de otimização de consulta. Para a acessar o catálogo do DM^2, uma aplicação deve usar a API DM^2 e interagir com todo o sistema, pois a aquitetura é fortemente acoplada. Seu sistema de armazenamento é restrito a um sistema de banco de dados relacional. MagCat propõe uma abordagem baseada em agentes de software. Toda a comunicação entre os agentes é feita através dos padrões FIPA-ACL. MagCat é totalmente independente de middleware de grade. Tem um esquema de metadados extensível, ideal para uso independente de domínio. Provê uma clara separação entre metadados descritivos e metadados de sistema. Para que um agente, cliente, se comunique com os agentes que compõem a arquitetura do MagCat é necessário que eles conheçam a ontologia de domínio especificada na Seção 4.2. MagCat é independente de sistema de armazenamento, visto que ele usa uma linguagem de consulta própria, reconhecida pelos agentes, a qual é mapeada para a linguagem de consulta particular do sistema de armazenamento tornando MagCat mais flexível quanto ao sistema de armazenamento.

7 Conclusões e Trabalhos Futuros

O surgimento de aplicações que requerem grande poder computacional e realização de computação intensiva sobre dados compartilhados e geograficamente distribuídos tem evidenciado novos desafios para grades de computadores. Uma especialização das grades de computadores, chamada grade de dados, tem sido desenvolvida de forma a superar esses novos desafios, como: a heterogeneidade de sistemas de banco de dados, segurança, acesso aos dados com baixa latência, alta confiabilidade, acesso transparente, publicação e descoberta dos dados. A superação deste último desafio, constitui um apescto muito importante para a efetiva aplicação de grades de dados. Para que o público em geral possa descobrir a existência dos dados e possam manipulá-los é necessário que existam informação que permitam a correta descrição dos dados e mecanismos que tornem o acesso a essas informações possível. Tipicamente, essas informações são chamadas de metadados e em uma grade de dados compreendem informações gerenciadas pelo serviço de metadados.

Este trabalho apresentou o MagCat, um serviço de metadados baseado em agentes para middlewares de grade. Os agentes que compreendem a arquitetura do MagCat permitem a realização da publicação e descoberta de dados em ambientes de grades de dados. Os agentes do MagCat mantém independência de sistema de armazenamento de metadados. Realizou-se experimentos para avaliação do desempenho com o uso de dois diferentes sistemas de armazenamento de metadados, que mostraram resultados de bom desempenho do MagCat diante dos sistemas de armazenamento usados. A implementação realizada corresponde a um serviço de metadados que oferece as funcionalidades básicas, e foi validada através da publicação e descoberta de imagens médicas de tomografia computarizada. Alguns metadados das imagens eram extraídas através de uma aplicação computacionalmente intensiva sobre dados, que foi adaptada para execução sobre o *middleware* MAG. Além disso, um outro experimento foi realizado com o objetivo de medir a eficácia do *middleware* MAG para execução da aplicação de extração de metadados das imagens. A partir dos resultados foi possível perceber que o mecanismo de execução de aplicações do MAG influencia positivamente no desempenho

das aplicações e que o tempo de execução desta classe de aplicações é inversamente proporcional ao número de nós que compõem o aglomerado.

As principais contribuições deste trabalho foram:

- A definição de uma arquitetura para um serviço de metadados para o *middleware* MAG através da qual fosse possível a publicação e descoberta de dados no ambiente da grade. Para a definição desta arquitetura optamos por usar a metodologia de desenvolvimento de sistemas multi-agentes *Agil Passi*, seguindo o paradigma de desenvolvimento adotado pelo *middleware* MAG. Assim, temos como resultado a modelagem de uma sociedade de agentes que devem cooperar de forma a contemplar os requisitos que especificamos para um serviço de metadados na Seção 4.1, como:
  - Esquema de metadados genérico e independente de domínio, requisito que compete ao agente `SchemaManagerAgent`;
  - Esquema de metadados estensível, de forma a possibilitar a criação de novos atributos de acordo com as necessidades dos usuários, apropriado para catalogação de dados de domínios diferentes, requisito que compete ao agente `SchemaManagerAgent`;
  - Permitir o agrupamento de dados relacionados em coleções e visões;
  - Escalável e distribuído com capacidade de armazenar grande quantidade de informações, requisito que compete aos agentes `ReplicationManagerAgent` e `CatalogManagerAgent`;
  - Alta disponibilidade e baixa latência sobre consultas aos metadados, requisito que compete aos agentes `ReplicationManagerAgent`,`RequestMonitor` e `CatalogManagerAgent`;
  - Existência de mecanismos de controle de autenticação e autorização das operações sobre informações sobre o catálogo, requisito que compete ao agente `SecurityAgent`;
  - Clara separação entre componentes de armazenamento de metadados lógicos ou descritivos de metadados de armazenamento físico ou de réplica;
  - Interface bem definida para armazenar e consultar os metadados. Neste trabalho os agentes se comunicam pela troca de mensagens, as quais informam

ao agente para executar um ação bem definida como, por exemplo, a de consulta e adição de metadados;

- Definição de uma ontologia de domínio e comunicação, através da qual os agentes adquirem um comum entendimento sobre os metadados e as ações possíveis sobre os mesmos de forma a possibilitar a identificação não ambígua do tipo de dado e domínio ao qual pertence, a ontologia de dominínio e comunicação servem a este fim, porém para satisfazer por completo esse requisito, deveria ser possível a estensão da ontologia de acordo com o domínio. Atualmente a ontologia de domínio é muito restritiva;

- A implementação de parte da arquitetura proposta, que corresponde à implementação dos agentes `CatalogManagerAgent` e `SchemaManagerAgent`, os quais permitem a execução das principais funcionalidades de um serviço de metadados, que são a publicação e descoberta de dados no ambiente de grade. Esta implementação realizada corresponde a um serviço de metadados simples;

- Realização de avaliação de desempenho de aspectos ligados às funcionalidades implementadas: avaliou-se qual o impacto no desempenho do sistema quanto à substituição do tipo de sistema de armazenamento e recuperação de metadados usado. Os resultados mostraram que o sistema mostrou-se eficiente independente de sistema de armazenamento adotado;

- Adaptação de uma aplicação computacionalmente intensiva sobre grande volumes de dados para validação da implementação: foi adaptada uma aplicação de segmentação de imagens baseado em textura para auxilio a diagnóstico médico. Essa aplicação serviu como estudo de caso para teste do sistema. Foi testada a estensibilidade do esquema de metadado gerenciado pelo agente `SchemaManagerAgent`, o qual possibilitou a criação de um esquema de metadados para armazenamento e recuperação de imagens médica;

- Comprovação da eficácia e eficiência do uso do *middleware* MAG para execução da aplicação desenvolvida: a aplicação de segmentação de imagens nos proporciou provar a eficácia e eficiência do mecanismo de execução aplicações do middleware MAG para a resolução de aplicações computacionalmente intensivas;

Ressalta-se que durante o desenvolvimento deste trabalho gerou-se três publicações:

- *MagCat: An Agent-based Metadata Service for Data Grids*: artigo completo publicado no *4th International Workshop on Agent Based Grid Computing (CCGrid'06)*[dSdSeST+06] que descreve a arquitetura do MagCat;
- *MAG: A Mobile Agent based Computational Grid Platform* [LSS05]: artigo completo publicado no *4th International Conference on Grid and Cooperative Computing*, que descreve a arquitetura geral e a implementação do projeto MAG;
- *MAG, um middleware de grade baseado em agentes: estado atual e perspectivas futuras*[dSeSLdS+06]: artigo completo publicado no *IV Workshop on Computational Grids and Applications no Brazilian Symposium on Computer Networks* (SBRC2006), que descreve o estado atual da sociedade multiagente que compõem o *middleware* MAG.

7.1 Trabalhos Futuros

Com o avanço das pesquisa para o desenvolvimento deste trabalho, indentificamos diversas oportunidades de extenção, entre as quais destacamos:

- A implementação realizada corresponde somente às funcionalidades básicas para permitir a publicação e descoberta de dados na grade. Assim, planejamos realizar a implementação dos demais agentes que compõem a sociedade de agentes modelada neste trabalho;
- Estender o modelo de descrição de metadados, que hoje possui apenas os conceitos de arquivo, coleção e visão, de forma a aumentar sua expressividade. O serviço poderia ser utilizado não apenas para a descrição de dados produzidos pelas aplicações, mas também para outras tarefas na grade como, por exemplo, a descrição das aplicações a serem executadas e suas dependências, e o armazenamento de informações de contexto em grades que integram dispositivos de computação móvel;
- Investigar o uso de heurísticas de acesso a dados, de forma a permitir a redução do custo da transferência de dados na grade é uma área de pesquisa em aberto em

grade de dados. As heurísticas possibilitariam decidir se a melhor alternativa é a transferência dos dados para local onde a computação está sendo executada, migrar a computação até o dados ou ainda realizar uma combinação das duas alternativas anteriores.

- O serviço de dados implementado aqui corresponde a funcionalidades simples de armazenamento e recuperação de arquivos. Para prover o serviço de dados que contemple acesso a diferentes fontes de dados para o MAG, a intregação do acesso ao *Open Grid Service Architecture Data Access Integration* (OGSA-DAI), apresentado na Seção 2.5.2 sob o `DataManagerAgent` constitui-se em importante passo para completar a infra-estrutura de grade de dados provida pelo MAG;

- Integração com o GridFTP, apresentado na Seção 2.5.3, de forma a permitir a transferência rápida e confiável de grandes volumes de dados.

Referências Bibliográficas

[ABB+01] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data management and transfer in highperformance computational grid environments. 2001.

[ABB+02] Bill Allcock, Joe Bester, John Bresnahan, Ann L. Chervenak, Ian Foster, Carl Kesselman, Sam Meder, Veronika Nefedova, Darcy Quesnel, and Steven Tuecke. Data management and transfer in high-performance computational grid environments, 2002.

[ABK+05] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. The globus striped gridftp framework and server. In *Proceedings of Super Computing 2005 (SC05)*, November 2005.

[ADG+03] Malcolm P Atkinson, Vijay Dialani, Leanne Guy, Inderpal Narang, Norman W Paton, Dave Pearson, Tony Storey, and Paul Watson. Grid database access and integration: Requirements and functionalities. In *Global Grid Forum*, 2003.

[AI99] Chervenak A. and Foster I. The grid: Blueprint for a future computing infrastructure. 1999.

[ASKF03] Ratnadeep Abrol, Terry Sloan, Amy Krause, and Fritz Ferstl. Storage resource broker report. Technical report, EPCC - University of Edinburgh, Novembro 2003. EPCC-SUNGRID-WP1-5-SRB 1.0.

[BBL00] M. Baker, R. Buyya, and D Laforenza. The grid: International efforts in global computing. In *In Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, Julho 2000.

[BMRW98] Chaitanya Baru, Reagan Moore, Arcot Rajasekar, and Michael Wan. The sdsc storage resourse broker. In *CASCON'98 Conference*, 1998.

[Bre90] Y. Breitbart. Multidatabse interoperability. In *SIGMOD Record*, pages 53–60, September 1990.

[BWE+00] Randy Butler, Von Welch, Douglas Engert, Ian Foster, Steven Tuecke, John Volmer, and Carl Kesselman. A national-scale authentication infrastructure. *IEEE Computer*, 33(12):60–66, December 2000.

[CC04] Giovanni Caire and David Cabanillas. *Application-defined content languages and ontologies.* TILAB, novembro 2004. Disponível em: http://jade.tilab.com/doc/CLOntoSupport.pdf. Acesso: 12 de novembro de 2005.

[CCSS04] Antonio Chella, Massimo Cossentino, Luca Sabatucci, and Valeria Seidita. From PASSI to Agile PASSI: Tailoring a Design Process to Meet New Needs. In *IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'04)*, pages 471–474, 2004.

[CDF+02] Ann Chervenak, Ewa Deelman, Ian Foster, Leanne Guy, Wolfgang Hoschek, Adriana Iamnitchi, Carl Kesselman, Peter Kunszt, Matei Ripeanu, Bob Schwartzkopf, Heinz Stockinger, Kurt Stockinger, and Brian Tierney. Giggle: a framework for constructing scalable replica location services. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–17, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[CDK+02] Ann Chervenak, Ewa Deelman, Carl Kesselman, Laura Pearlman, and Gurmeet Singh. A metadata catalog service for data intensive applications. Technical report, GriPhyn, 2002.

[CFK+01] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, 23:187–200, 2001.

[CGA+04] Raphael Y. de Camargo, Goldchleger, Andrei, Kon, Fabio, Goldman, and Alfredo. Checkpointing based rollback recovery for parallel applications on the integrade grid middleware. *5th ACM/IFIP/USENIX International Middleware Conference*, 2004.

[Clu00] David A. Clunie. *DICOM Structered Reporting.* PixelMed Publishing, Pennsylvania, 2000.

[CPB+04] Ann L. Chervenak, Naveen Palavalli, Shishir Bharathi, Carl Kesselman, and Robert Schwartzkopf. Performance and scalability of a replica location service. In *13th IEEE International Symposium on High Performance Distributed Computing (HPDC-13'04)*, pages 182–191. 2004.

[CPC+00] Walfredo Cirne, D. Paranhos, L. Costa, E. Santos, Nazareno Nigini A., Andrade F. Brasileiro, and Jacques. U Sauvé. Using mygrid to run bag of tasks applications on computational grids. Technical report, Universidade Federal de Campina Grande. Departamento de Sistemas e Computação. Campina Grande - PB, 1500.

[CSK+05] A. Chervenak, R. Schuler, C. Kesselman, S. Koranda, and B. Moe. Wide area data replication for scientific collaborations. In *Proceedings of 6th IEEE/ACM International Workshop on Grid Computing (Grid2005).* November 2005.

[CSS03] Massimo Cossentino, Luca Sabatucci, and Valeria Seidita. Spem description of the passi process rel. 0.3.6. Technical Report RT-ICAR-20-03, Consiglio Nazionale delle Ricerche Istituto di Calcolo e Reti ad Alte Prestazioni, December 2003.

[CSSC03] M. Cossentino, L. Sabatucci, S. Sorace, and A. Chella. Patterns reuse in the PASSI methodology. In *Proceedings of the Fourth International Workshop Engineering Societies in the Agents World (ESAW'03)*, pages 29–31, Imperial College London, UK, October 2003.

[DHJM+01] Dirk Dullmann, Wolfgang Hoschek, Javier Jaen-Martinez, Ben Segal, Asad Samar, Heinz Stockinger, and Kurt Stockinger. Models for replica synchronisation and consistency in a data grid. pages 67–75. IEEE, 2001.

[DMP+03] H. Duque, J. Montagnat, J. M. Pierson, L. Brunie, and I. E. Magnin. Dm2: A distributed medical data manager for grids. In *CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid*, page 606, Washington, DC, USA, 2003. IEEE Computer Society.

[DSA+04] Ewa Deelman, Gurmeet Singh, Macolm P. Atkinson, Ann Chervenak, Neil P. Chue Hong, Carl Kesselman, Sonal Patil, Laura Pearlman, and Mei-Hui Su. Gird-based metadata services. In *Proceedings of the 16th International Conference on Scientific and Statical Database Management (SSDBM'04)*, pages 393–402. IEEE Computer Society, 2004.

[dSdSeST+06] Bysmarck Barros de Sousa, Francisco José da Silva e Silva, Mário Meireles Teixeira, , and Gilberto Cunha Filho. MagCat: An Agent-based Metadata Service for Data Grids. In *AgentGrid2006: 4th International Workshop on Agent Based Grid Computing. IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'06)*, Singapore, May 2006. IEEE Computer Society Press. to appear.

[dSeSLdS+06] Francisco José da Silva e Silva, Rafael Fernandes Lopez, Bysmarck Barros de Sousa, Antônio Eduardo Bernardes Viana, and Stanley Araújo de Sousa. Mag, um middleware de grade baseado em agentes: estado atual e perspectivas futuras. In *WCGA2006: Anais doIV Workshop on Computational Grids and Applications no Brazilian Symposium on Computer Networks (SBRC2006)*, Curitiba, maio 2006. Sociedade Brasileira de Computação. to appear.

[FF98] Peter A. Freeborough and Nick C. Fox. MR texture analysis to the diagnosis and tracking of alzheimer's disease. *IEEE Transactions on Medical Imaging*, 17(3):475–479, 1998.

[FIPa] FIPA Propose Interaction Protocol Specification. Disponível em: http://fipa.org/specs/fipa00036/index.html. Acesso: 01 de fevereiro de 2006.

[FIPb] FIPA Query Interaction Protocol Specification. Disponível em: http://fipa.org/specs/fipa00027/index.html. Acesso: 01 de fevereiro de 2006.

[FIPc] FIPA Request Interaction Protocol Specification. Disponível em: http://fipa.org/specs/fipa00026/index.html. Acesso: 01 de fevereiro de 2006.

[FK99] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco, 1999.

[FKNT02] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. Open Grid Service Infrastructure WG, Global Grid Forum, Jun 2002.

[FKT01] I. Foster, C. Kesselman, and S. Tueke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputing Applications*, 2001.

[FKTT98] Ian Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A security architecture for computational grids. In *CCS '98: Proceedings of the 5th ACM conference on Computer and communications security*, pages 83–92, New York, NY, USA, 1998. ACM Press.

[Gal75] Mary M. Galloway. Texture analysis using gray level run lenghts. *Computer Graphics and Image Processing*, 4:172–179, 1975.

[GF92] M. R. Genesereth and R. E. Fikes. Knowledge Interchange Format, Version 3.0 Reference Manual. Technical Report Logic-92-1, Stanford, CA, USA, 1992.

[GKG+02] A. Goldchleger, F. Kon, A. Goldman, Finger M., and Siang Wun. Song. Integrade: Rumo a um sistema de computação em grade para aproveitamento de recursos ociosos em máquinas compartilhadas. Technical report, Departamento de Ciência da Computação. Instituto de Matemática e Estatística - Universidade de São Paulo, Set 2002.

[GKG+04] A. Goldchleger, F. Kon, A. Goldman, M. Finger, and G. C. Bezerra. Integrade: Object-oriented grid middleware leveraging idle computing power of desktop machines. *Concurrency and Computation: Practice & Experience. Vol. 16, pp. 449-459*, 2004.

[GKGF03] A. Goldchleger, F. Kon, A. Goldman, and M. Finger. Integrade: Object-oriented grid middleware leveraging idle computing power of desktop machines. *ACM/IFIP/USENIX International Workshop on Middleware for Grid Computing. Rio de Janeiro, Brazil*, 2003.

[GKL+02] Leanne Guy, Peter Kunszt, Erwin Laure, Heinz Stockinger, and Kurt Stockinger. Replica management in data grids, july 2002. CERN, European Organization for Nuclear Research.

[GLFK00] Andrew S. Grimshaw, Michael J. Lewis, Adam J. Ferrari, and John F. Karpovich. Architectural support for extensibility and autonomy in wide-area distributed object systems. In *Proceedings of the 2000 Network and Distributed System Security Symposium (NDSS2000)*, February 2000.

[Gol04] Andrei Goldchleger. InteGrade: Um Sistema de Middleware para Computação em Grade Oportunista (InteGrade: a Middleware System for Opportunistic Grid Computing). Master's thesis, Department of Computer Science - University of São Paulo, December 2004. in Portuguese.

[GR95] V.N. Gudivada and V.V. Raghavan. Content based image retrieval systems. In *IEEE Computer*, volume 28, pages 18–22, Sep 1995.

[Gru93] T. R. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.

[Hay04] David Haynes. *Metadata for Information Management and Retrieval*. Neal-Schuman Publishers, october 2004.

[HBM] Neil Hardman, Andrew Borley, and James Magowan. Ogsa-dai: A look under the hood: Part 2: Activities and results. Disponível em: http://www-128.ibm.com/developerworks/grid/library/gr-ogsadai2/.

[HJV+] Montagnat H., Duque J.M., Pierson V., Breton L., Brunie I.E., and Magnin. Dm2: Medical image content-based queries using the grid. In *Healthgrid 2003*, January. Lyon, France.

[HMS+98] J. M. D. Hilll, B. Mccolll, D. C. Stefanescu, M. W. Goudreau, K. Lang, S. B. Rao, T. Suel, T. Tsantilas, and R. H. Bisselin. Bsplib: The bsp programming library. 1998.

[HSD73] R.M. Haralick, K. Shanmugam, and I. Dinstein. Textural features for image classification. *SMC*, 3(6):610–621, November 1973.

[Hua04] H. K. Huang. *PACS and Imaging Informatics: Basic Principles and Applications.* John Wiley & Sons, New Jersey, 2004.

[Jai89] A. K. Jain. *Fundamentals of Digital Image Processing.* Prentice Hall, Englewood Cliffs, NJ, USA, 1989.

[Jai91] Raj Jain. *The Art of Computer Systems Performance Analysis.* John Wiley and Sons, 1991.

[JF04] Joshy Joseph and Craig Fellenstein. *Grid Computing*, chapter Introduction to Grid Computing. Dec 30, 2003;, 1st edition, 2004.

[JJL+] H. Duque J., Montagnat J.M., Pierson L., Brunie I.E., and Brunie. Dm2: A distributed medical data manager for grids. In *Biogrid 2003*, may. Tokyo, Japan.

[KBM02] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computin. In *Software - Practice and Experience*, volume 32, pages 135–164. February 2002.

[LG96] Michael J. Lewis and Andrew Grimshaw. The core legion object model. In *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing (HPDC '96)*, pages 551 – 561. IEEE Computer Society Press, Agosto 1996.

[Li01] Xiangfeng Li. Texture analysis for optical coherence tomography image. Master's thesis, The University of Arizona, 2001.

[Lop06] Rafael Fernandes Lopes. MAG: uma grade computacional baseada em agentes móveis. Master's thesis, Universidade Federal do Maranhão, Sã Luís, MA, Brasil, January 2006. In Portuguese.

[LSS05] Rafael Fernandes Lopes, Francisco José da Silva Silva, and Bysmarck Barros de Sousa. MAG: A Mobile Agent based Computational Grid Platform. In *Proceedings of the 4th International Conference on Grid and Cooperative Computing*, Lecture Notes in Computer Science (LNCS Series), Beijing, November 2005. Springer-Verlag.

[MC04] Daniel A. Menascé and Emilliano Casalicchio. Qos in grid computing. *IEEE Internet Computing*, pages 85–87, July - August 2004.

[MGHW+99] Michael F. McNitt-Gray, Eric M. Hart, Nathaniel Wyckoff, James W. Sayre, Jonathan G. Goldin, and Denise R. Aberle. The effects of co-occurrence matrix based texture parameters on the classification of solitary pulmonary nodules imaged on computed tomography. *Computerized Medical Imaging and Graphics*, 23:339–348, 1999.

[ML97] Rajesh Raman e Todd Tannenbaum Miron Livny, Jim Basney. Mechanisms for high throughput computing. *PEEDUP Journal*, 11(1), Junho 1997.

[MOF] Meta object facility (mof) specification. Disponível em: http://www.omg.org/cgi-bin/doc?formal/00-04-03. Acesso: 19/09/2005.

[MRD00] Naga R. Mudigonda, Rangaraj M. Rangayyan, and J. E. Leo Desautels. Gradient and texture analysis for the classification of mammographic masses. *IEEE Transactions on Medical Imaging*, 19(10):1032–1043, 2000.

[NAN+03] Alpdemir N., Mukherjee A., Paton NW., Watson P., Fernandes AAA., Gounaris A., and Smith J. Service-based distributed query processing on the grid. In *Proceedings of the 1st International Conference on Service-Oriented Computing (ICSOC)*, pages 467–482. Springer, 2003.

[Obj00] Object Management Group. *Trading Object Service Specification*, Junho 2000. OMG document formal/00-06-27, version 1.0.

[OV91] M. Tamer Ösu and Patrick Valduriez. *Principles of Distributed Databse Systems*. Prentice Hall, 1991.

[Pro00] Globus Project. Gridftp universal data transfer for the grid. Technical report, Globus Alliance, Sep 2000.

[PSDM04] Jean-Marc Pierson, Ludwig Seitz, Hector Duque, and Johan Montagnat. Metadata for efficient, secure and extensible access to data in a medical grid. In *DEXA Workshops*, pages 562–566, 2004.

[RKM02] W. E. Allcock R. K. Madduri, C. S. Hood. Reliable file transfer in grid environments. In *27th Annual IEEE International Conference on Local Computer Networks (LCN'02)*, 2002.

[RMLZ02] A. Rajasekar, R. Moore, B. Ludascher, and I. Zaslavsky. The grid adventures: Sdsc's storage resource broker and web services in digital library applications. In *in Proceedings of the Fourth All-Russian Scientific Conference (RCDL'02) Digital Libraries: Advanced Methods and Technologies, Digital Collections*, 2002.

[RWM+04] A. Rajasekar, M. Wan, R. Moore, , and W. Schroeder. Data grid federation. In *Proceedings of the 11th International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2004)*. Las Vegas, USA: CSREA Press, June 2004.

[RWMS04] A. Rajasekar, M. Wan, R. Moore, and W. Schroeder. Data grid federation. In *Special Session on New Trends in Distributed Data Access*. PDPTA, Las Vegas NV, June 2004.

[SBC+03] Gurmeet Singh, Shishir Bharathi, Ann Chervenak, Ewa Deelman, Carl Kesselman, Mary Manohar, Sonal Patil, and Laura Pearlman. A metadata catalog service for data intensive applications. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 33, Washington, DC, USA, 2003. IEEE Computer Society.

[SOA] Simple object access protocol (soap) 1.1. Disponível em: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/. Acesso: 25 de novembro de 2005.

[TLC+99] B. Tierney, J. Lee, B. Crowley, M. Holding, J. Hylton, and F. Drake. A network-aware distributed storage cache for data intensive environments. In *Proc. of IEEE High Performance Distributed Computing Conference (HPDC-8)*, August 1999.

[TTGD04] Rattapoom Tuchinda, Snehal Thakkar, Yolanda Gil, and Ewa Deelman. Artemis: Integrating scientific data on the grid. In *AAAI*, pages 892–899, 2004.

[TTL03] Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the grid. In Fran Berman, Geoffrey Fox, and Anthony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., April 2003.

[Val90] Leslie G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 8(33):103–111, 1990.

[VBR05] Srikumar Venugopal, Rajkumar Buyya, and Kotagiri Ramamohanara. A taxonomy of data grids for distributed data sharing, management and processing. In *Procurar*, page 11111, mês 2005.

[vLFF+97] Gregor von Laszewski, Steve Fitzgerald, Ian Foster, Carl Kesselman, Warren Smith, and Steve Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. In *Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing*, pages 365–375, Portland, OR, 5-8 1997.

[XMI] Mof 2.0 / xmi mapping specification, v2.1. Disponível em:http://www.omg.org/technology/documents/formal/xmi.htm. Acesso: 20/09/2005.

[ZS04] X. Zhang and J. Schopf. Performance analysis of the globus toolkit monitoring and discovery service, mds2. In *Proceedings of the International Workshop on Middleware Performance (MP 2004), part of the 23rd International Performance Computing and Communications Workshop (IPCCC)*, April 2004.

A Grámatica da Linguagem de Consulta do MagCat

```
<expression> := <create statement> | <update statement> | <delete statement>
        | <select statement> | <define statement>

<create statement> := CREATE <object_type> SCHEMA_NAME <schema_name>
        <left_paren> <attribute_list> <right_paren> VALUES
        <left_paren> <attribute_value_list> <right_paren>

<update statement> := UPDATE <object_type> SCHEMA_NAME <schema_name>
        <left_paren> <attribute_list> <right_paren> VALUES
        <left_paren> <attribute_value_list> <right_paren>
        WHERE <condition>

<delete statement> := DELETE <object_type> SCHEMA_NAME <schema_name>
        WHERE <condition>

<select statement> := SELECT <object_type> SCHEMA_NAME <schema_name>
        WHERE <condition>

<define statement> := DEFINE SCHEMA_NAME <schema_name> DESCRIPTION
        <string> EXTENDS <schema_name> ATTRIBUTES <left_paren> <attribute_list>
        <right_paren> ATTRIBUTE_TYPES <left_paren> <attribute_type_list>
        <right_paren> ATTIBUTE_DESCRIPTION <left_paren>
        <attribute_description_list> <right_paren>

<update schema statement> := UPDATE SCHEMA_NAME <schema_name> ATTRIBUTES <left_paren> <at
        <right_paren> ATTRIBUTE_TYPES <left_paren> <attribute_type_list>
        <right_paren> ATTIBUTE_DESCRIPTION <left_paren>
        <attribute_description_list> <right_paren>

<delete schema statement> := DELETE SCHEMA_NAME <schema_name>

<object_type> := LOGICAL_FILE | LOGICAL_COLLECTION | LOGICAL_VIEW

<schema_name> := <character_set>

<character_set> := <character_set> | <letter> | <letter><character_set>
        | <digit><character_set>

<attribute> := <character_set>

<attribute_list> := <attribute> | <attribute> <comma> <attribute>

<attribute_value_list> := <value> | <value> <comma> <value>

<attribute_type>  := <string> | <integer> | <boolean> | <float>

<string> := <double quote> <character_set> <double quote>

<attribute_type_list>  := <attribute_type> | <attribute_type> <comma>
        <attribute_type>

<attribute_description_list>  := <descripton> | <descripton> <comma>
        <descripton>

<descripton> := <string>

<condition>:= <attribute> <operator> <value> <connector> <condition>

<value> := <digit> | <letter> | <letter><valor>  | <digit><value>
        | <value> <letter> | <value> <digit>

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<letter> :=  a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q
        | r | s | t | u | v | w | x | y | z

<left_paren> := (

<right_paren> := )

<comma> := ,

<double quote> := "
```

```
75 <operator> := <equal> | <less> | <greather> | <not_equal> | <less_than>
76         | <greather_than>
77
78 <connector> := AND | OR
79
80 <equal> :=  =
81
82 <less> := <
83
84 <greather> := >
85
86 <not_equal> := <>
87
88 <less_than> := <=
89
90 <greather_than> := >=
```