2
Trabalhos Relacionados

A questão do acesso a dados científicos, especialmente para a integração de aplicações científicas, tem sido abordada sob diferentes perspectivas na busca por soluções. Dentre as soluções pesquisadas, algumas das quais influenciaram fortemente a arquitetura apresentada neste trabalho, é possível identificar quatro perspectivas principais de solução: independência no formato de armazenamento, empacotamento e desempacotamento eficientes, definição de modelos de dados genéricos e definição de arquitetura para servidores.

A *independência no formato de armazenamento* consiste em desacoplar a representação estruturada do dado da forma como este dado é armazenado, por exemplo em disco. Um servidor de dados utiliza bibliotecas como um intermediário que fica encarregado por efetivamente persistir e recuperar os dados. Tipicamente essas bibliotecas estruturam o dado em um formato mais apropriado para o tratamento pela aplicação, distinto do formato utilizado para o armazenamento propriamente dito. Dentre as soluções que oferecem independência no formato de armazenamento, podem ser citadas o HDF [7], o NetCDF [9] e o Protocol Buffers [10]; este último será descrito em detalhes adiante.

Além do meio de armazenamento, os dados estruturados são serializados também para serem transferidos através de uma rede. É importante, neste caso, que a serialização se efetue de maneira rápida e que a quantidade de *bytes* gerada seja pequena, para que a transferência seja eficiente. Por conta disso, algumas soluções se preocupam em oferecer mecanismos que atendam a estas demandas por *eficiência no empacotamento e no desempacotamento destes dados*. O Protocol Buffers, por exemplo, permite que se escolha se a representação gerada pela serialização será mais eficiente para armazenamento ou para transferência. Já o *valuetype*, que representa os objetos por valor em CORBA, além de oferecer empacotamento e desempacotamento eficientes, ainda permite que os servidores de dados implementem customizações nestes mecanismos. Os objetos por valor de CORBA também serão descritos em mais detalhes adiante.

A terceira perspectiva identificada é a *definição de modelos de dados*

genéricos. Esses modelos definem a representação a ser utilizada pelos dados científicos que serão oferecidos pelos servidores de dados. As aplicações lidam apenas com os dados na representação indicada por esses modelos e, com isso, ficam totalmente independentes da forma e da representação do armazenamento do dado. O HDF, o NetCDF, OpenSpirit [11] e o SDS [12] definem modelos de dados para uso das aplicações científicas.

Por fim, algumas soluções definem *arquiteturas para servidores de dados*, que apresentam os elementos que devem fazer parte dos servidores, o papel de cada elemento e como deve ser realizada a comunicação entre eles. O SDS apresenta uma arquitetura genérica para dados científicos, enquanto que o OpenSpirit apresenta a implementação de uma arquitetura para o domínio da Geologia e da Geofísica.

Nas seções seguintes serão analisados com mais detalhes os trabalhos mais relevantes, identificando as perspectivas abordadas por cada um deles.

2.1 Scientific Data Server (SDS)

O SDS [12] define uma arquitetura para a construção de servidores de dados científicos que tenham por objetivo oferecer esses dados para aplicações clientes através de uma rede. Essa arquitetura define os componentes que devem estar presentes em um servidor de dados e um modelo de dados científicos que contém os tipos de dados que podem ser oferecidos pelos servidores. A figura 2.1 apresenta os componentes definidos na arquitetura SDS.

O modelo de dados do SDS oferece os dados científicos sob a forma de objetos denominados *Scientific Data Objects*. Um dado científico é sempre representado como um *scientific dataset* e através dele se obtém as partes ou atributos do dado, que podem ser de quatro tipos: *multi-dimensional array, table, collection e text block*.

Os objetos de dados científicos possuem duas operações que são implementadas por todos os tipos definidos no modelo de dados: *Get* e *Describe*. A operação *Get* é responsável por transferir o objeto científico para a aplicação cliente. A operação *Describe*, por sua vez, é responsável por enviar para a aplicação cliente uma descrição do objeto contendo o tipo do dado, os seus metadados e, opcionalmente, um *thumbnail*, ou visão resumida, do objeto.

O componente *Q/R Protocol (Query/Response Protocol)* é a porta de entrada de um servidor de dados científicos. Através dele é possível obter os dados e seus metadados oferecidos pelo servidor, que são enviados sob a forma de um *scientific dataset*. Uma aplicação pode analisar o *scientific dataset* para

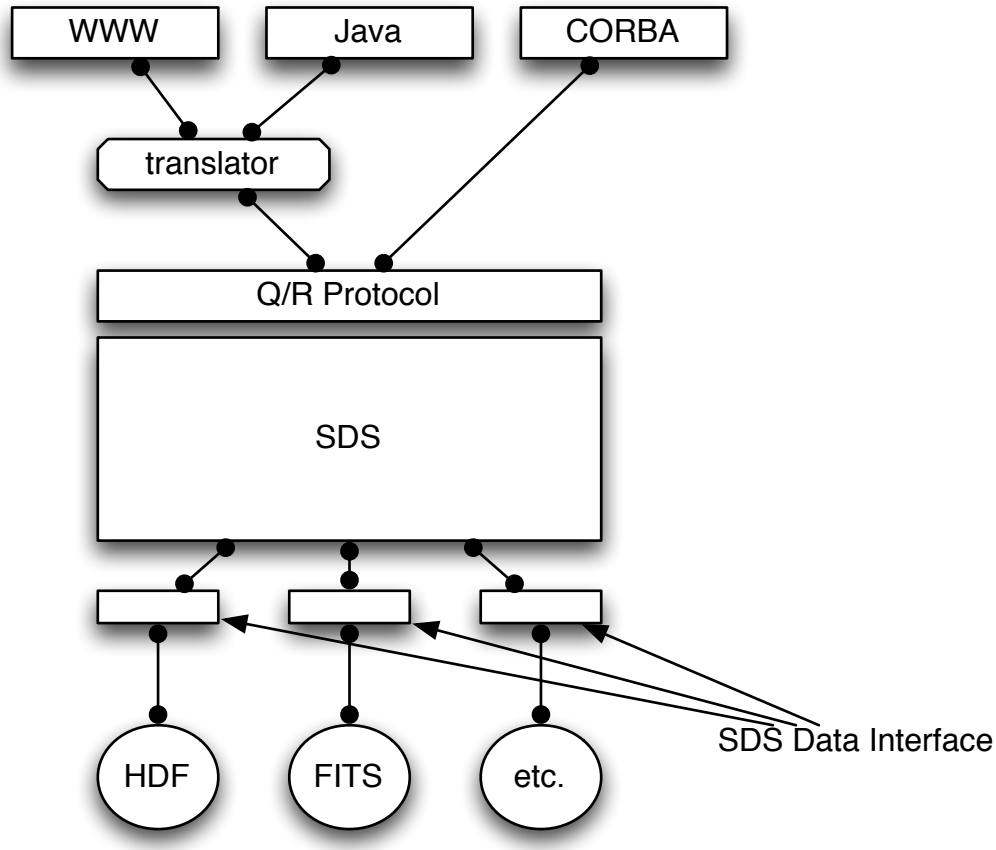

Figura 2.1: Arquitetura SDS

identificar um conjunto de atributos do seu interesse e então obter apenas a parte do dado que seja do seu interesse.

O componente *SDS Data Interface* é o responsável pela independência do SDS em relação aos meios e formatos de armazenamento dos dados. Esse componente define uma interface padrão com operações que devem ser implementadas para cada conjunto de dados físico, sendo o responsável pelo mapeamento entre o formato de armazenamento usado pelo servidor de dados e o modelo de dados genérico definido pelo SDS.

O SDS apresenta algumas limitações que podem inviabilizar o seu uso em alguns servidores de dados. Não é possível, por exemplo, estender o modelo de dados com a inclusão de novos tipos; os dados científicos devem ser sempre uma combinação dos tipos primitivos disponíveis. Além disso, não existem mecanismos para a criação, atualização e remoção dos dados científicos.

2.2 Service Data Objects (SDO)

A especificação *Service Data Objects (SDO)* [13] define uma arquitetura e um modelo de dados independente dos meios e formatos de armazenamento dos dados. Além disso, a arquitetura define também a API que deve ser oferecida para que as aplicações possam utilizar os dados representados no modelo. A figura 2.2, extraída de [13], apresenta os componentes definidos na especificação SDO: *Data Object*, *Data Graph* e *Data Access Service*.

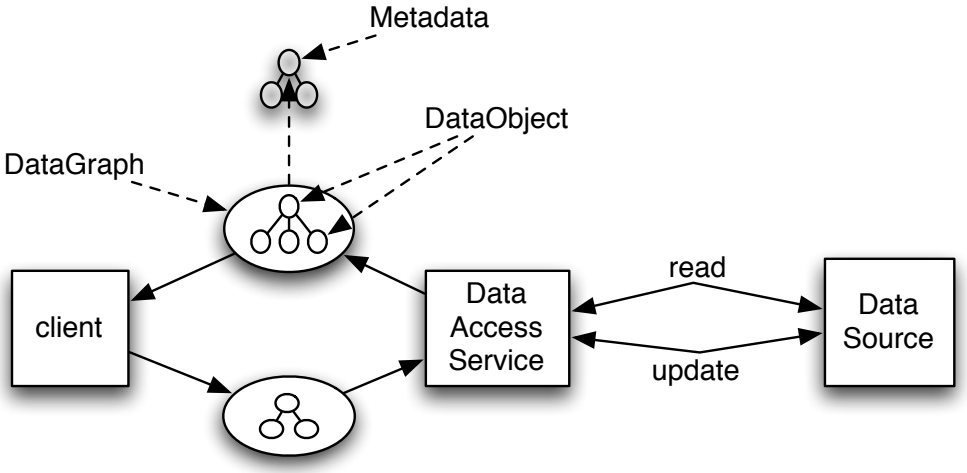

Figura 2.2: Arquitetura SDO

O *Data Object* é o componente que representa o dado propriamente dito. Esse componente é formado por um conjunto de atributos ou propriedades, cada uma contendo um nome e um valor, que pode ser de um tipo primitivo ou uma referência para outro Data Object. Opcionalmente, os *Data Objects* podem ter referências para seus metadados.

Um outro componente da arquitetura, o *Data Graph*, pode ser visto como um envelope ou uma coleção de *Data Objects*. Um *Data Graph* é a unidade transportada entre os servidores de dados e as aplicações. *Data Graphs* são conjuntos *data objects* de múltiplas raízes. Data Graphs podem rastrear mudanças realizadas nos grafos de DataObjects. Mudanças incluem inserção, remoção e alteração de propriedades dos Data Objects. Um *Data Graph* representa um conjunto de dados. São tipicamente a unidade de transferência entre componentes em um sistema.

O terceiro e último componente da arquitetura é o *Data Access Service (DAS)*. Este componente é responsável por carregar os dados a partir de um *data source* e mapear esses dados em *data graphs*. Além disso, esse componente

deve aplicar mudanças realizadas em um *data graph* de volta ao seu *data source* de origem.

A arquitetura SDO é baseada no padrão *disconnected data graphs (grafos desconectados de dados)*. Esse padrão define que uma aplicação cliente pode obter um *Data Graph*, proveniente de uma fonte de dados qualquer, modifica-lo e então enviá-lo de volta para a fonte de dados, para que as modificações sejam realizadas.

De acordo com a especificação, os *Data Objects* devem oferecer, no mínimo, uma API dinâmica de acesso a dados para leitura e modificação de objetos, incluindo suas propriedades. A API provê mecanismos para a manipulação dinâmica dessas propriedades. Opcionalmente, interfaces Java para *Data Objects* podem ser geradas a partir de modelos ou esquemas. A especificação SDO não define a geração de interfaces Java para os *Data Objects*. Apesar da importância do DAS na arquitetura SDO, não existe especificação para os *data access services*.

2.3 OpenSpirit

O *OpenSpirit* [11] é um *middleware* para integração de aplicações através do compartilhamento de dados mantidos pelas próprias aplicações. O *middleware* provê a infra-estrutura de suporte à integração de aplicações e dados através de uma série de serviços. Estes serviços incluem conexão de aplicações ao ambiente de integração, a disponibilização de informações sobre os repositórios de dados acessíveis, a difusão de mensagens entre aplicações, dentre outros.

Os dados oferecidos pelo *OpenSpirit* são definidos em um modelo genérico e independente de quaisquer repositórios de dados, chamado de *OpenSpirit Data Model (ODM)*. Nesse modelo, os dados são oferecidos através de uma visão única e fica a cargo de cada repositório de dados fazer o mapeamento corresponde entre o ODM e seu modelo interno. Vale ressaltar que o ODM contempla apenas o domínio da Geologia e Geofísica, atualmente.

Além do modelo de dados genérico, a infra-estrutura oferece um mecanismo de *query*, através do qual as aplicações podem criar, ler, atualizar e excluir dados com base no modelo de dados citado anteriormente. Este mecanismo, entretanto, é limitado a dados de pequeno volume; o acesso a *bulk data*, ou dado de grande volume, é realizado através de objetos remotos disponibilizados através de uma biblioteca (API).

O mecanismo de troca de mensagens permite ainda que as aplicações recebam mensagens notificando alterações nos dados de um repositório (criação,

alteração e exclusão).

Os *Data Connectors*, ou conectores de dados, são os elementos responsáveis por implementar os serviços de acesso aos diferentes repositórios, ou *datastores*. Os *Data Connectors* realizam, consequentemente, o mapeamento do repositório específico, para o qual foi criado, para o ODM.

Por fim, os *Application Adapters* são os elementos responsáveis por integrar as aplicações ao ambiente *OpenSpirit*. Através desses elementos, as aplicações têm acesso aos serviços de dados disponibilizados pelo *middleware*.

2.4 Protocol Buffers

O *Protocol Buffers* [10] é uma tecnologia de serialização de dados estruturados para uso principalmente em protocolos de comunicação e armazenamento de dados. Os dados são definidos utilizando-se uma linguagem independente de plataforma e de linguagem de programação e essas definições são materializadas em tipos de linguagens de programação específicas.

A tecnologia oferece bibliotecas de apoio, específicas para cada linguagem de programação suportada pelo compilador, que são utilizadas para gerar uma representação do dado estruturado e para fazer a serialização dessa representação através de *streams* de dados.

Os dados estruturados são especificados através de um tipo denominado *mensagem* e são definidos em arquivos de extensão *proto*. Uma mensagem é um pequeno registro lógico de informação, contendo uma série de atributos, muito similar a uma *struct* em linguagem C. Um atributo possui um tipo, um nome e um identificador unívoco dentro da mensagem. Além disso, um atributo possui um modificador que indica se esse atributo é obrigatório, opcional ou uma sequência. Os atributos podem ser numéricos, lógicos, textuais e até mesmo outros tipos de mensagens, possibilitando que o dado seja estruturado hierarquicamente.

Os tipos de mensagens podem ser atualizados sem que se prejudique as aplicações que usam o formato anterior; os *parsers* simplesmente ignoram campos com identificadores desconhecidos. Além disso, campos opcionais podem ser removidos sem nenhum ônus a tais aplicações.

A codificação do *Protocol Buffers* é baseada no conceito de *varints*. *Varint* é um método para serializar inteiros usando um ou mais *bytes*: quanto menor o número, menor a quantidade de *bytes*.

Cada *byte* em um *varint*, exceto o último *byte*, tem o *bit* mais significativo definido – isto indica que ainda restam *bytes* por vir. Os outros 7 *bits* de cada

byte são usados para armazenar o número na representação *complemento a 2*. Os *bytes* menos significativos vêm primeiro (*little endian*).

Testes comparativos realizados por seus desenvolvedores mostram que o *Protocol Buffers* é bastante eficiente em comparação a *XML*. Tais testes mostraram o *parser Protocol Buffers* sendo de 20 a 100 vezes mais rápido do que um *parser* XML [1].

Protocol Buffers é a *lingua franca* para dados no Google. Existem, aproximadamente, 50000 diferentes tipos de mensagens definidas através de quase 15000 arquivos de definição na árvore de código do Google. Estas mensagens são usadas tanto em sistemas de RPC (*remote procedure call*) quanto em sistemas de armazenamento de dados.

2.5 Representação de dados em CORBA

Aplicações que utilizam a tecnologia CORBA [8] trabalham, tipicamente, com objetos remotos. Ou seja, objetos instanciados em um servidor que são acessados pelos clientes através de referências. Essas referências são responsáveis por tratar os detalhes da localização do servidor e do protocolo de comunicação, para que se repasse para o objeto servidor as chamadas realizadas por um cliente. Os objetos remotos são todos derivados de um tipo básico chamado *Object*.

Existem algumas ocasiões, entretanto, onde é necessário transferir para o cliente uma cópia do dado para ser utilizado localmente. Esse recurso é particularmente útil quando o propósito principal de um objeto é encapsular dados. Nesse caso, são usados os chamados objetos por valor, representados em CORBA pelo tipo *valuetype*.

Um *valuetype* pode ser considerado um tipo híbrido entre o tipo *struct* e o tipo *interface*, que representa os objetos remotos. Os *valuetypes* podem ser de dois tipos: *concretos* ou *abstratos*. As principais diferenças entre um *valuetype* abstrato e um concreto é que o primeiro não pode ser instanciado e nem possuir estado (atributos), sendo constituído, apenas, de um conjunto de operações. Já um *valuetype* concreto pode rescrever o modelo padrão de empacotamento e desempacotamento e prover seu próprio meio para codificar e decodificar seu estado.

O ORB é responsável por criar os *streams* de dados tanto para a leitura quando para a escrita, agindo, implicitamente, como uma fábrica. Não existem operações para a criação de *streams* de dados.

[1] O parser XML utilizado nos testes não foi especificado.

O ORB é responsável também por construir efetivamente a codificação dos valores, inserindo cabeçalhos e tags, de acordo com o protocolo especificado para a comunicação; estas informações não ficam expostas para a aplicação.

As *interfaces abstratas* oferecem um meio para tratarmos de objetos CORBA tanto por referência (interface) quanto por valor (*valuetype*). Ou seja, podemos definir em tempo de execução se o objeto a ser passado será uma referência ou um objeto por valor.

2.6 Considerações Finais

Dentre todos os trabalhos pesquisados, o SDS e o OpenSpirit são os que atendem ao maior número de requisitos definidos para a solução desejada de integração de aplicações científicas através do compartilhamento de dados. A principal limitação destas duas soluções é a impossibilidade de se adicionar novos tipos de dados aos serviços de dados. O OpenSpirit oferece apenas tipos de dados no domínio da Geologia e da Geofísica e o SDS oferece tipos de dados ainda mais básicos.

O SDO, ao contrário do OpenSpirit e do SDS, permite que sejam adicionados novos tipos de dados aos serviços de dados. Porém, o SDO apresenta outras duas limitações que não são apresentadas pelas duas soluções anteriores: a imposição do uso da linguagem Java e a falta de especificação para os serviços de acesso a dados. A falta de uma especificação para o DAS traz dificuldades para as aplicações que pretendem acessar dados pois a comunicação poderá ser diferente em cada serviço acessado.

Finalmente, o Protocol Buffers e as abstrações para representação de dados em CORBA, embora não sejam soluções completas de serviços de acesso a dados, podem ser utilizadas na implementação da arquitetura destes serviços, seja na comunicação entre as aplicações e os serviços, seja no empacotamento e no desempacotamento dos dados. Entretanto, o Protocol Buffers não possui uma definição para as chamadas remotas entre as aplicações e os serviços, forçando os desenvolvedores das aplicações e dos serviços a criação e a implementação de um protocolo para permitir essa comunicação. Como veremos, a definição da arquitetura dos serviços de dados apresentada neste trabalho será realizada utilizando-se as funcionalidades oferecidas pelo *middleware* CORBA: abstração para tipos representando dados, comunicação distribuída e independência de linguagem de programação e plataforma de execução.