MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21b/2017/11.20.11.52-TDI

FRAMEWORK DE INTEGRAÇÃO DE DADOS AMBIENTAIS, BASEADO EM METADADOS

José Luiz Moreira

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais, orientada pelo Dr. Maurício Gonçalves Vieira Ferreira, aprovada em 22 de novembro de 2017.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34P/3Q3J2U8>

INPE
São José dos Campos
2017

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/6921
E-mail: pubtc@inpe.br

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21b/2017/11.20.11.52-TDI

FRAMEWORK DE INTEGRAÇÃO DE DADOS AMBIENTAIS, BASEADO EM METADADOS

José Luiz Moreira

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais, orientada pelo Dr. Maurício Gonçalves Vieira Ferreira, aprovada em 22 de novembro de 2017.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34P/3Q3J2U8>

INPE
São José dos Campos
2017

Dados Internacionais de Catalogação na Publicação (CIP)

Moreira, José Luiz.

M813f Framework de integração de dados ambientais, baseado em metadados / José Luiz Moreira. – São José dos Campos : INPE, 2017.

xxii + 58 p. ; (sid.inpe.br/mtc-m21b/2017/11.20.11.52-TDI)

Dissertação (Mestrado em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2017.

Orientador : Dr. Maurício Gonçalves Vieira Ferreira.

1. Framework. 2. Integração. 3. Dados ambientais. 4. Metadados. I.Título.

CDU 001.103.2:502

Aluno (a): *José Luiz Moreira*

Título: "FRAMEWORK DE INTEGRAÇÃO DE DADOS AMBIENTAIS, BASEADO EM METADADOS".

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de *Mestre* em

Engenharia e Tecnologia Espaciais/Eng. Gerenc. de Sistemas Espaciais

Dr. Nilson Sant'Anna

Presidente / INPE / SJCampos - SP

Dr. Maurício Gonçalves Vieira Ferreira

Orientador(a) / INPE / SJCampos - SP

Dra. Lília de Sá Silva

Membro da Banca / INPE / São José dos Campos - SP

Dra. Magda Aparecida Silvério Miyashiro

*Convidado(a) / * / São Paulo - SP*

Este trabalho foi aprovado por:

( ) maioria simples

(X) unanimidade

São José dos Campos, 22 de novembro de 2017

“Nossa maior fraqueza está em desistir. O caminho mais certo de vencer é tentar mais uma vez”.

Thomas Edison

A minha esposa Miriam e a meus filhos Mariana, Bruno e Lívia que são a razão da minha existência.

AGRADECIMENTOS

A Deus por ter me abençoado com saúde e força espiritual para concluir esta etapa em um momento delicado da minha vida.

A minha esposa Miriam por todo amor e companheirismo nos momentos mais difíceis, sem seus "empurrões" eu não teria conseguido.

A meus filhos Mariana, Bruno e Lívia que com suas conquistas, se tornaram exemplos de perseverança para mim.

Aos meus amigos do INPE, pela compreensão, incentivo e apoio incondicional.

Por fim, um agradecimento especial ao meu orientador Maurício pela compreensão e paciência durante este período.

RESUMO

Sistemas de coleta e processamento de dados ambientais em operação no Centro de Missão e Coleta de Dados - CMCD do Instituto Nacional de Pesquisas Espaciais - INPE, geram diariamente um grande volume de dados. Estes sistemas trabalham de forma diferenciada, com formatos de dados e armazenamento próprios ou específicos, tornando mais complexas as atividades de acesso e disponibilização de dados por parte da equipe de desenvolvimento de software responsável pela automatização do processo operacional. Neste cenário, é de grande valia uma solução de software que propicie aumentar à eficiência, a qualidade e reduza os desperdícios de recursos envolvidos neste processo. Nesta dissertação, é proposta a arquitetura de um framework de software que seja capaz de: (i) fornecer aos usuários (sistemas clientes) acesso simultâneo a múltiplas fontes de dados; (ii) encapsular os procedimentos de acesso, extração e integração dos dados; (iii) apresentar uma visão uniforme e consistente dos dados; (iv) prover interoperabilidade sintática, estrutural e semântica entre sistemas; e (v) prover uma saída integrada de dados com opções de exportação para diversos formatos. Benefícios, tais como: a eliminação do trabalho repetitivo no acesso, seleção e disponibilização de dados são obtidos através da adoção da arquitetura definida, que prioriza a maximização do reuso, minimização de acoplamento e melhor consistência e compatibilidade entre os sistemas de coleta e processamento de dados ambientais em operação no CMCD. O estudo de caso apresentado descreve a aplicação do framework proposto na arquitetura de um sistema com interface para integração e visualização dinâmica de dados ambientais. Como conclusão desta dissertação, é apresentada uma análise das limitações encontradas, as contribuições da arquitetura proposta e também as considerações para trabalhos futuros.

Palavras-chave: Framework. Integração. Dados Ambientais. Metadados.

ENVIRONMENTAL DATA INTEGRATION FRAMEWORK, BASED ON METADATA

ABSTRACT

Environmental data collection and processing systems in operation at the Mission and Data Collection Center (CMCD) of the National Institute of Space Research (INPE) generate a large amount of data daily. These systems work in a differentiated way, using own or specific formats and data storage, making the activities of access and availability of data more complex performed by the software development team responsible for the automation of the operational process. In this scenario, a software solution that increases efficiency, quality and reduces the waste of resources involved in this process is of great value. This dissertation proposes the architecture of a software framework that is capable of: (i) providing users (client systems) simultaneous access to multiple data sources; (ii) encapsulating procedures of data access, extraction and integration ; (iii) presenting a uniform and consistent view of data; (iv) providing syntactic, structural and semantic interoperability between systems; and (v) providing an integrated data output with export options for various formats. Benefits such as: elimination of repetitive work on access, selection and availability of data are obtained through the adoption of the defined architecture, which prioritizes reuse maximization, coupling minimization and better consistency and compatibility between the environmental data collection and processing systems in operation in the CMCD. The present case study describes the application of the proposed framework in the architecture of a system with interface for integration and dynamic visualization of environmental data. As a conclusion of this dissertation, an analysis of the limitations found, the contributions of the proposed architecture and also the considerations for future work are presented.

Keywords: Framework. Integration. Environmental Data. Metadata.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE SIGLAS E ABREVIATURAS

CMCD	Centro de Missão e Coleta de Dados
CPTEC	Centro de Previsão de Tempo e Estudos Climáticos
DSA	Divisão de Satélites e Sistemas Ambientais
INPE	Instituto Nacional de Pesquisas Espaciais
PCD	Plataforma de Coleta de Dados
SBCDA	Sistema Brasileiro de Coleta de Dados Ambientais
SCD	Satélite de Coleta de Dados
SGBD	Sistema Gerenciador de Banco de Dados
SQL	Structured Query Language
XML	eXtensible Markup Language
JVM	Java Virtual Machine
PDF	Portable Document Format
CSV	Comma Separated Values
XLS	eXceL Spreadsheet
MVC	Model View Controller
CBERS	China-Brazil Earth Resources Satellite
RIA	Rich Internet Application
OGC	Open Geospatial Consortium
ORM	Object-Relational Mapping
XSD	Xml Schema Definition
SIG	Sistema de Informações Geográficas
TA	Trabalho Acadêmico
FM	Framework de Mercado
P2P	Peer to Peer
DW	Data Warehouse
JNI	Java Native Interface
JDBC	Java Database Connectivity
JPA	Java Persistence API
API	Application Programming Interface

SUMÁRIO

Pág.

1 INTRODUÇÃO

O Sistema Brasileiro de Coleta de Dados Ambientais - SBCDA, tem por finalidade receber, processar e disseminar dados provenientes de estações de coleta de dados, tais como Plataformas de Coleta de Dados (PCDs), estações meteorológicas (fixas e móveis), bóias (oceânicas, em rios, açudes etc.), dentre outros diversos tipos de estações.

As principais áreas de aplicação destes dados são:

- Meteorologia / Oceanografia, para previsão de tempo e de clima e análises oceanográficas;
- Hidrologia, para monitoramento e previsão de enchentes, secas e níveis de água em rios e reservatórios;
- Sismologia, para transmissão em tempo hábil de dados sobre tremores de terra;
- Biologia, para seguimento de animais;
- Silvicultura, para alarmes contrafogo;
- Agrometeorologia para monitoramento das melhores condições para semear, plantar e colher.

O SBCDA é constituído pela constelação de satélites SCD-1, SCD-2 e família CBERS, pelas diversas redes de plataformas de coleta de dados espalhadas pelo território nacional, pelas Estações Terrenas de Recepção de Cuiabá e de Alcântara, e pelo Centro de Missão e Coleta de Dados (CMCD). A homologação de novas plataformas de coleta de dados é realizada no Centro Regional do Nordeste (CRN), a instalação e a manutenção das plataformas são realizadas pelo Laboratório de Instrumentação Meteorológica (CPTEC/LIM) em Cachoeira Paulista e pelo CRN para a Região Nordeste.

O SBCDA tem por finalidade receber, processar e disseminar dados de diversas fontes, o que gera um enorme volume de dados, distribuídos em diversos sistemas, cada um com sua própria finalidade e tipo de armazenamento específico.

A Figura 1.1 ilustra o Sistema Brasileiro de Coleta de Dados Ambientais e todos os seus componentes.

Figura 1.1 - SBCDA, mostrando os círculos de visibilidade das estações de Cuiabá e de Alcântara.

Fonte: Yamaguti. et al (2009)

1.1. **A Rede de PCDs**

As PCDs ou Estações Ambientais Automáticas surgiram da necessidade de inúmeras empresas e instituições obter regularmente informações colhidas em lugares remotos ou espalhadas por uma região muito extensa.

Uma PCD é um dispositivo coletor, formatador e transmissor de dados, usado em aplicações de monitoramento ambiental tais como alerta sobre

enchentes, controle de irrigação, controle de recursos hídricos, medidas de qualidade do ar etc.

A função coleta de dados é realizada por um conjunto de sensores, selecionados de acordo com a operação de monitoramento a ser levada a efeito, e uma interface de aquisição de dados. Usualmente, os sensores são usados para monitorar o nível e a qualidade de água 'de rios e reservatórios, a temperatura e umidade do ar, pressão atmosférica, a velocidade e direção do vento, a radiação solar, a temperatura e umidade do solo, a concentração de ozônio, a precipitação etc.

Os dados adquiridos, incluindo a identificação e status da PCD, são armazenados e formatados em uma mensagem padrão equivalente àquela empregada no Sistema ARGOS e transmitidos para o satélite.

Os parâmetros das PCDs são estabelecidos de acordo com a sua posição geográfica e o desempenho do sistema, mensurado em termos das mensagens corretas recebidas ao longo do dia na Estação Terrena de Recepção.

As modernas Estações PCDs são em sua maior parte destinadas à aplicações com satélites, munidas com células solares e baterias para o seu suprimento de energia, possibilitando estender de forma quase indefinidamente sua vida útil. Existem vários satélites com equipamentos apropriados para receber transmissões de PCDs e retransmiti-las para a Terra, onde podem ser disponibilizadas aos usuários de diversas maneiras.

1.2. **Motivação**

Os contínuos avanços tecnológicos os quais resultam em sensores, transmissores e componentes que vão a bordo dos satélites, mais inteligentes e sofisticados, estão aumentando substancialmente os dados gerados pelos SCDs.

Um grande volume de dados é gerado rotineiramente nos sistemas de processamento de dados ambientais em operação no CMCD, o que demanda recursos computacionais (hardware e software) robustos para operacionalização e disseminação de informações para os usuários do SBCDA.

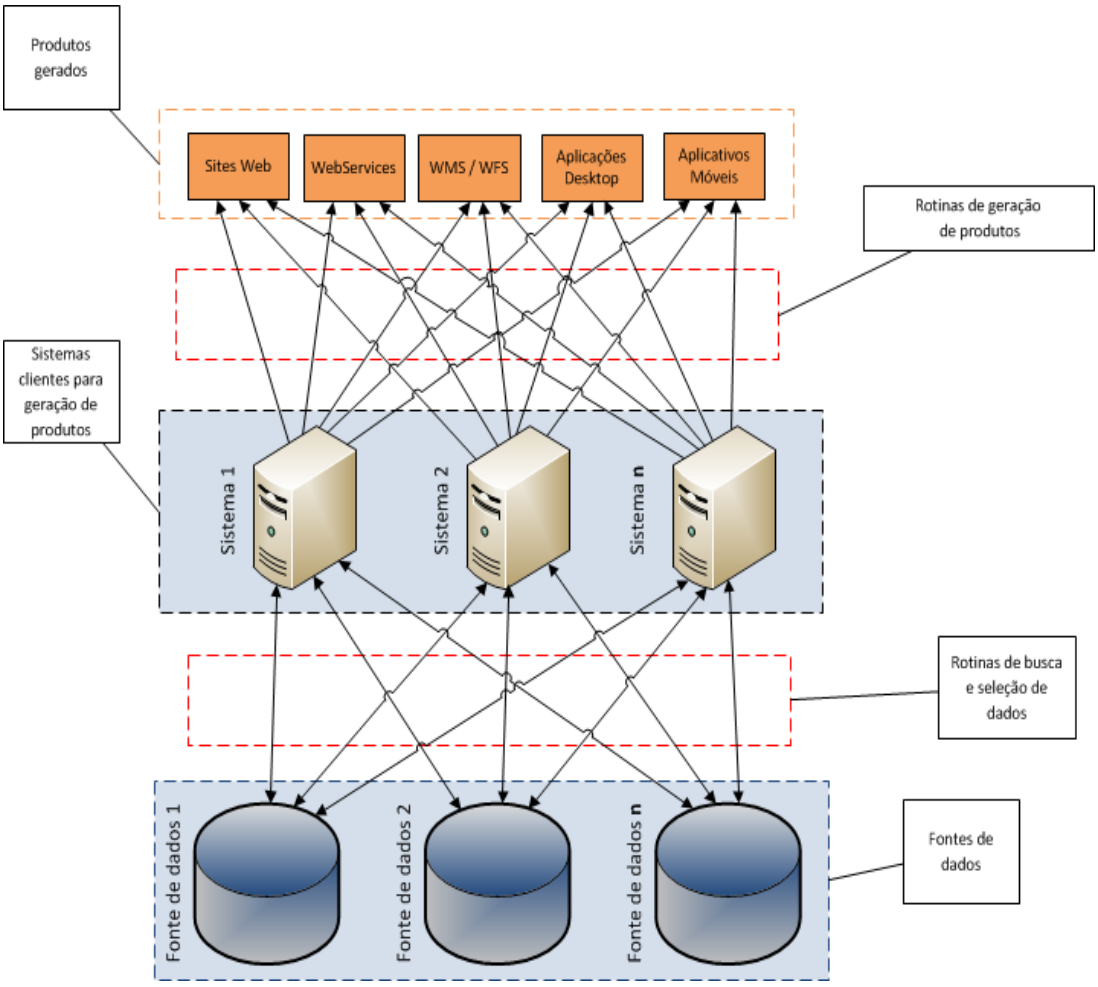
Os sistemas de processamento de dados ambientais, cada um com sua finalidade de pesquisa, geram fontes de dados com modelos e SGBDs distintos. Os dados disponíveis nas bases já foram decodificados, pelos respectivos sistemas, a partir das mensagens em formato original (dados bruto) transmitidos pelas PCDs e são divulgados em formato de dados de engenharia.

Neste ambiente heterogêneo, o armazenamento é efetuado de forma descentralizada, o que dificulta o acesso e a disseminação dos dados, gerando, muitas vezes, uma grande redundância nos aplicativos que são disponibilizados para o usuário final.

Outra questão é a falta de um padrão no formato de saída, pois a falta de integração das bases de dados gera informações que necessitam de processamento posterior a fim gerar o relacionamento devido, para que possam ser aplicadas com qualidade em diversas áreas de pesquisa.

O cenário apresentado na Figura 1.2 demonstra a redundância na geração de código e na complexidade de administração do ambiente de software no CMCD, pois cada sistema implantado deve implementar suas próprias rotinas de acesso às fontes de dados disponíveis no momento e também às rotinas de disponibilização dos dados consultados conforme o formato definido. Estas atividades devem ser executadas a cada nova fonte de dados inserida no ambiente e também a cada novo formato de disponibilização definidos para os usuários destes dados.

Figura 1.2 - Ilustração do ambiente de software no CMCD



Fonte: Produção do Autor

1.3. **Objetivo da Pesquisa**

Esta pesquisa tem por objetivo investigar, conceber e implementar um framework para a integração de bases de dados ambientais através de mapeamentos utilizando metadados para ser utilizado pela equipe de desenvolvedores, visando aumentar a eficiência, qualidade, simplicidade no uso, transparência nas conexões e reduzir os desperdícios de recursos envolvidos no processo de desenvolvimento de sistemas de informações em operação dentro de um Centro de Missão.

1.4. **Metodologia**

A metodologia adotada para o desenvolvimento deste trabalho baseou-se em cinco atividades de pesquisa:

- Levantamento bibliográfico;

Esta atividade baseou-se na investigação e análise de modelos de integração de dados, métodos de pesquisa científica, definições e arquiteturas de frameworks, artigos científicos, documentação oficial dos principais SGBDs utilizados nos sistemas de processamento do CMCD e demais trabalhos relacionados a este.

- Levantamento dos frameworks existentes;

Para esta atividade foi realizada uma pesquisa em dissertações, teses, artigos científicos e frameworks de mercado para o levantamento das principais características empregadas nestas soluções e identificar as que mais se adequavam para a solução do problema que motivou este trabalho de dissertação. Esta atividade teve como foco os processos de configuração através de metadados, de geração de códigos SQL, o tratamento das particularidades (sintaxe, os tipos de dados etc.) dos SGBDs em questão e também o esforço do processo de integração de um framework em um sistema cliente.

- Elaboração e implementação do projeto do framework;

Com o resultado das atividades anteriores iniciou-se projeto de software da arquitetura do framework com o levantamento dos requisitos funcionais e não funcionais, a concepção do design do projeto com a definição dos componentes do framework, a implementação do código do framework e os eventuais testes unitários e de integração específicos ao projeto.

- Verificação e validação;

A confirmação da adequação da arquitetura proposta foi verificada e validada através da implementação de protótipos de modo a comprovar a conformidade com os requisitos estabelecidos. Protótipos para a carga e validação de metadados, consulta a diferentes bases de dados,

integração de dados e geração de formatos específicos para disponibilização de dados foram desenvolvidos de modo a validar a arquitetura definida.

- Definição de um estudo de caso (aplicação em um cenário real) e avaliação dos principais resultados;

Por fim foi definido um estudo de caso simulando a aplicação do framework integrado a um cenário real. O sistema cliente em estudo tem por objetivo a integração e visualização dinâmica de dados ambientais. Foram analisadas as atividades básicas necessárias para a codificação do sistema cliente, comparando os cenários sem a integração com o framework e utilizando as funcionalidades do framework. Ao final deste estudo, foi efetuada uma avaliação da arquitetura do framework definida, considerando o objetivo deste trabalho de dissertação.

Vale ressaltar, que a redação desta dissertação foi efetuada paralelamente as atividades descritas, sendo realizada de forma incremental desde a concepção do projeto de pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo, apresenta os principais conceitos utilizados para a definição da proposta de implementação de um framework para integração de dados com a utilização de metadados.

2.1. Conceitos utilizados

2.1.1. Framework

Uma das definições mais clássicas para framework é, "Um conjunto de classes que definem um projeto abstrato de soluções para uma família de problemas relacionados, oferecem poderosa forma de reutilização de software, proporcionando níveis mais elevados de abstração nos projetos de aplicações, buscando facilitar o reuso de seu código" (JOHNSON, 1988; JANI, 2009). Com frameworks pode-se alcançar um maior ganho de produtividade no desenvolvimento de software, diminuição no tempo produção de software e maior qualidade das aplicações desenvolvidas (KIRK, 2007).

GAMMA et al. (1994) modificam a definição de Johnson em termos do que é um projeto e qual é a definição de se utilizar framework, a saber: Um conjunto de classes cooperando que compõe um projeto reusável para uma classe específica de software. Um framework fornece a orientação arquitetural dividindo o projeto em classes abstratas e definindo suas responsabilidades e colaborações. Um desenvolvedor personaliza o framework para uma aplicação particular através de subclasses e compondo instâncias de classes do framework.

Framework pode finalmente ser definido quanto à sua estrutura e sua utilidade. Quanto à estrutura, um framework é a reutilização de parte do projeto ou de sua totalidade, sendo representado por classes abstratas e pela forma como instâncias dessas se relacionam. Já quanto à utilidade, um framework é um esqueleto de uma aplicação que deve ser parametrizado pelo desenvolvedor da aplicação; ou seja, com um framework extrai-se e disponibiliza-se a essência de uma determinada

família de aplicações, de modo que o desenvolvedor possa partir de um patamar superior de implementação da arquitetura (PREE,1995)

Um ponto importante que deve ser esclarecido é a diferença básica que existe entre o uso de bibliotecas de classes (abstratas ou concretas) e framework.

Usar classes independentes, significa instanciar ou redefini-las (no caso de abstratas) em uma aplicação que está sendo desenvolvida. Dessa forma, a aplicação é que irá invocar essas classes.

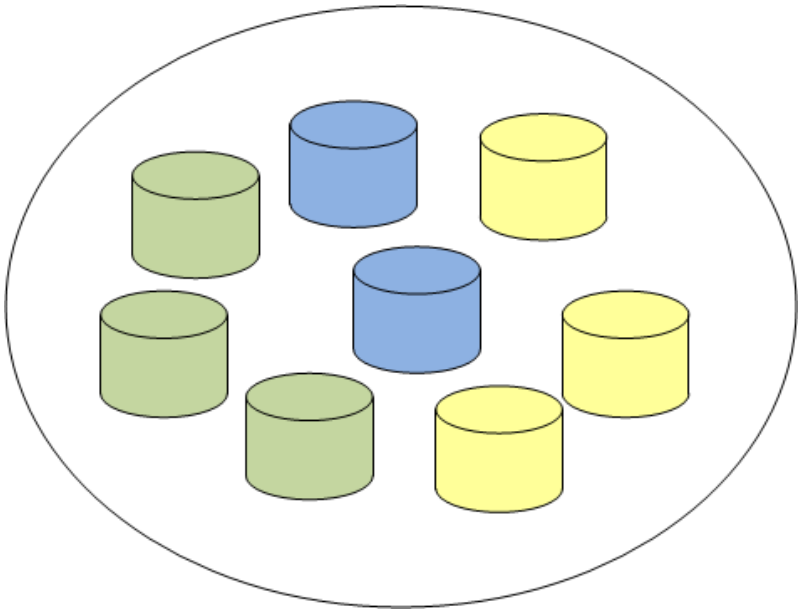
No caso do framework, existe uma inversão dos papéis. As classes que porventura serão estendidas (personalizadas pelo usuário do framework) serão chamadas por essa estrutura, onde a aplicação é o próprio framework que está sendo moldado para a produção de uma aplicação específica.

Quando um framework é usado, toda a estrutura de controle e as partes comuns às aplicações do domínio já estão definidas, e será escrito código particular à aplicação em desenvolvimento. Para isso o desenvolvedor usa nomes e convenções previamente especificadas pelo desenvolvedor do framework. O resultado é a produção de aplicações pelo framework.

2.1.2. Integração de dados

A integração de banco de dados envolve o processamento pelo qual as informações de bancos de dados participantes podem ser integradas conceitualmente para formar uma única definição coesa de um sistema formado por vários bancos de dados. Em outra palavra, é o processo de projetar o esquema conceitual global.

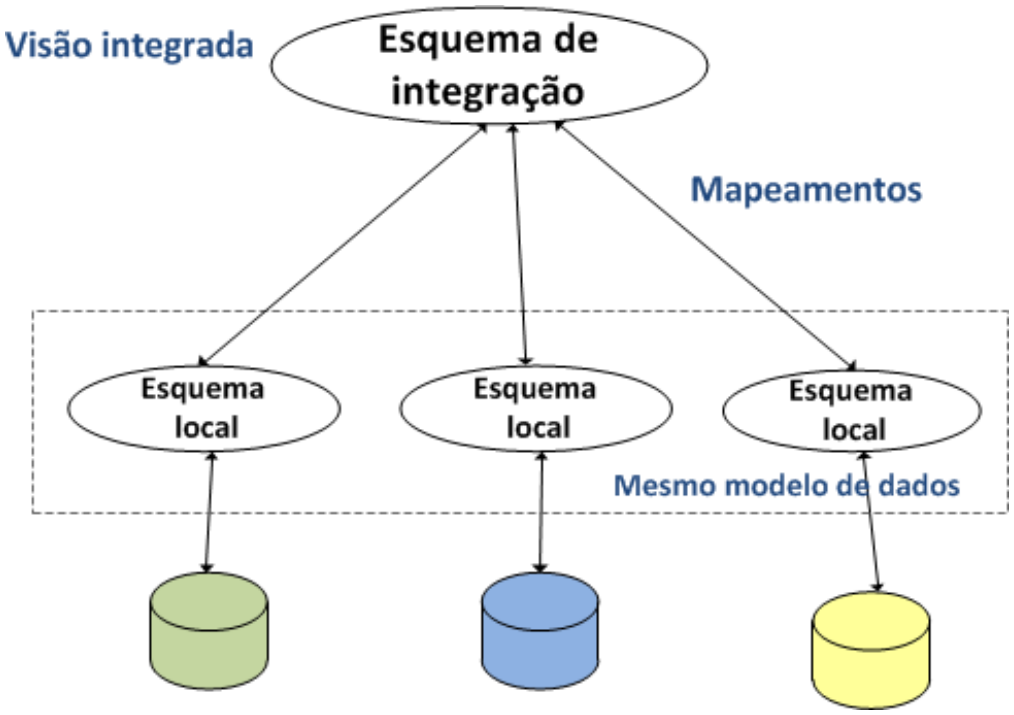
Figura 2.1 - Visão geral dos dados



Fonte: Produção do Autor

A Integração de Dados de múltiplas fontes é o mais antigo problema enfrentado pela comunidade de pesquisa em banco de dados. Ao longo do tempo, diversas soluções têm sido apresentadas e a principal questão é, como oferecer uma visão global de dados distribuídos em fontes de dados autônomas e heterogêneas?

Figura 2.2 - Visão geral do problema de Integração de Dados



Fonte: Produção do Autor

A heterogeneidade de fontes de dados pode ser encontrada em:

- nível físico (diferentes plataformas de hardware e software).
- nível lógico (diferentes modelos de dados).
- nível conceitual (diferentes esquemas e conceitos).

Os tipos de abordagem definidos para solucionar este problema são:

- Abordagem Virtual, onde:
  - Os dados são recuperados diretamente das fontes.
  - As consultas são enviadas diretamente às fontes de dados.
  - Os resultados individuais obtidos são integrados e enviados ao usuário.

– Vantagem: Os dados estão sempre atualizados.

– Desvantagem: Os custos de processamento das consultas e de acesso às fontes são elevados.

•. Materializada

– Os dados das fontes distintas são extraídos e materializados localmente em repositórios chamados Data Warehouses.

– Vantagem: As consultas são realizadas sobre a base materializada (melhor desempenho).

– Desvantagem: Necessidade de manter a base materializada sempre atualizada

Exemplos de Arquiteturas para Integração de Dados, são:

a) Arquitetura de Mediadores –

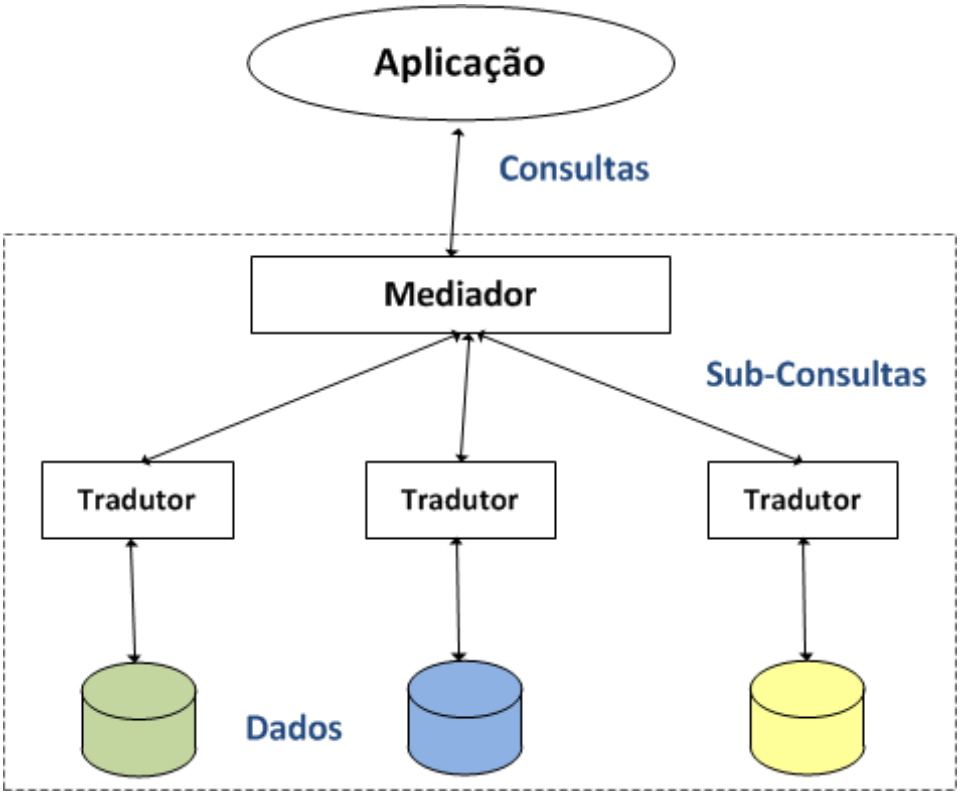
A arquitetura de mediadores é composta pelas fontes de informação, pelos tradutores e pelos mediadores. As fontes de informação podem ser autônomas e heterogêneas. Os tradutores convertem os dados das fontes de informação para um modelo de dados comum e convertem consultas de aplicações em consultas específicas da fonte de informação correspondente. Os mediadores são módulos de software que exploram o conhecimento representado em um conjunto ou subconjunto de dados para gerar informações para aplicações residentes em uma camada superior.

Na arquitetura de mediadores o acesso aos dados distribuídos nas diversas bases de dados é efetuado através de consultas que são submetidas ao sistema através dos mediadores, e estes as transformam em subconsultas a serem enviadas às fontes de informação. As subconsultas geradas pelos mediadores devem ser

traduzidas para as linguagens de consulta de cada SGBD componente. Ao final, os resultados das consultas são traduzidos e a resposta é retornada para o usuário.

O processo de atualização dos dados através de mediadores é semelhante ao de consulta; as atualizações submetidas ao sistema, via mediador, devem ser traduzidas em atualizações a serem executadas nas bases de dados correspondentes.

Figura 2.3 - Arquitetura de Mediadores



Fonte: Produção do Autor

b) Arquitetura de Data Warehouse

Segundo Date (2004) “Data Warehouse (que no português significa, literalmente armazém de dados) é um deposito de dados orientado por assunto, integrado, não volátil, variável com o tempo, para apoiar as decisões gerenciais”.

c) Orientado por assunto

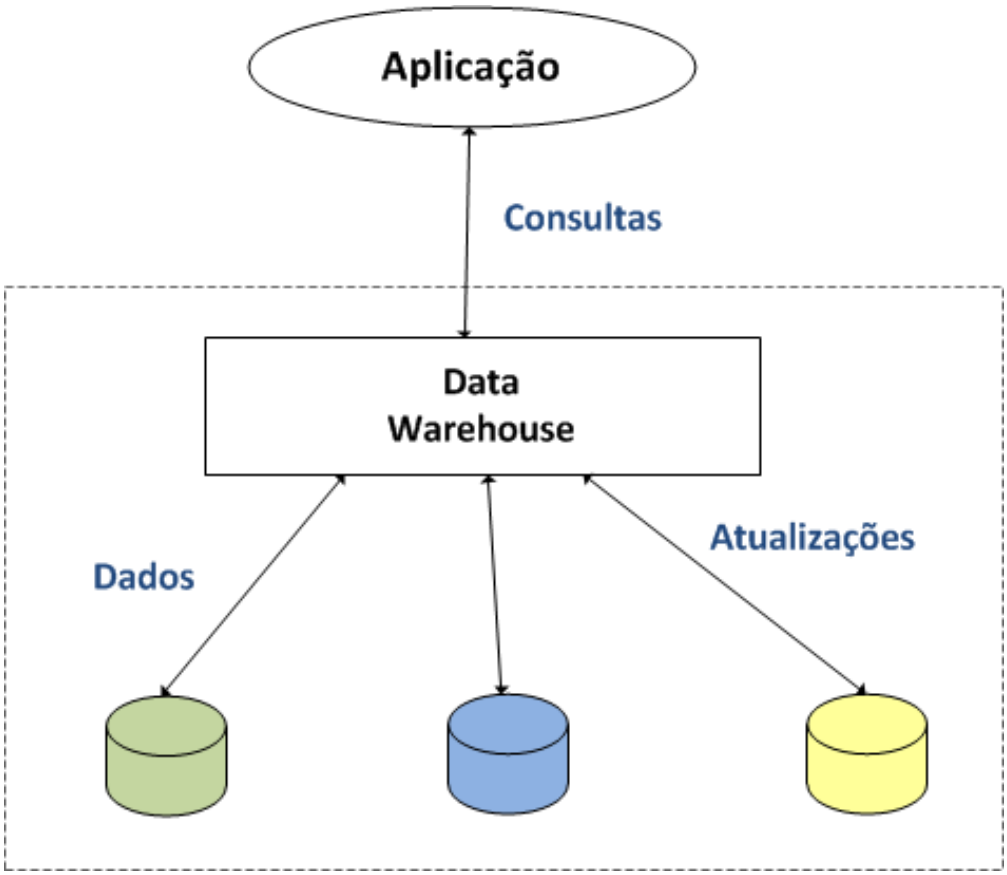
Refere-se aos sistemas transacionais organizados em uma determinada aplicação de uma empresa. A orientação por assunto é uma característica importante, pois toda a modelagem do DW é orientada a partir dos principais assuntos da empresa. Por exemplo uma empresa de arrecadação de impostos, onde os principais assuntos são os cadastros de contribuintes, impostos a recolher.

d) Integrado

Essa é a característica mais importante do Data Warehouse, pois trata da integração, que é feita do ambiente operacional para as aplicações do DW. A integração é realizada visando padronizar os dados dos diversos sistemas em uma única representação, para serem transferidos para a base de dados única do DW.

O armazenamento se dá num depósito único, que seja de rápido acesso para as análises. Tal armazenamento conterá dados históricos advindos de bancos de dados transacionais que servem como backend de sistemas corporativos. Quanto mais dados do histórico das operações da empresa, melhor será para que a análise destas informações reflita o momento da empresa.

Figura 2.4 - Arquitetura de Data Warehouse



Fonte: Produção do Autor

e) Arquitetura Peer to Peer (P2P)

Na arquitetura P2P, não há distinção entre clientes e servidores. Especificamente, nesta arquitetura, todas as máquinas têm todas as funcionalidades disponíveis no Sistema gerenciador de Banco de Dados (SGBD).

As entidades que participam desta arquitetura são chamadas de peers (pares) ou nós. Cada nó pode se comunicar com outras máquinas para executar transações e consultas, ocorrendo, portanto, uma distribuição da computação local.

Outra característica da arquitetura P2P é que a topologia da rede pode ser modificada dinamicamente e mesmo assim é possível continuar a realizar operações de BD sobre o sistema.

Vantagens desta arquitetura:

- Disponibilidade do Sistema e dos dados, porque a arquitetura P2P é formada por várias máquinas, então quando o sistema é acessado, se uma máquina não estiver disponível, outa assume e atende a requisição, porém se não existir replicação dos dados eficiente, a resposta de uma consulta pode não ser a representação real de todo o sistema.

- Escalabilidade, é possível adicionar novos peers, porém aumentará o tempo de resposta para realizar operações sobre esta arquitetura. A consulta é realizada inicialmente em um nó que repassa a requisição para seus nós vizinhos.

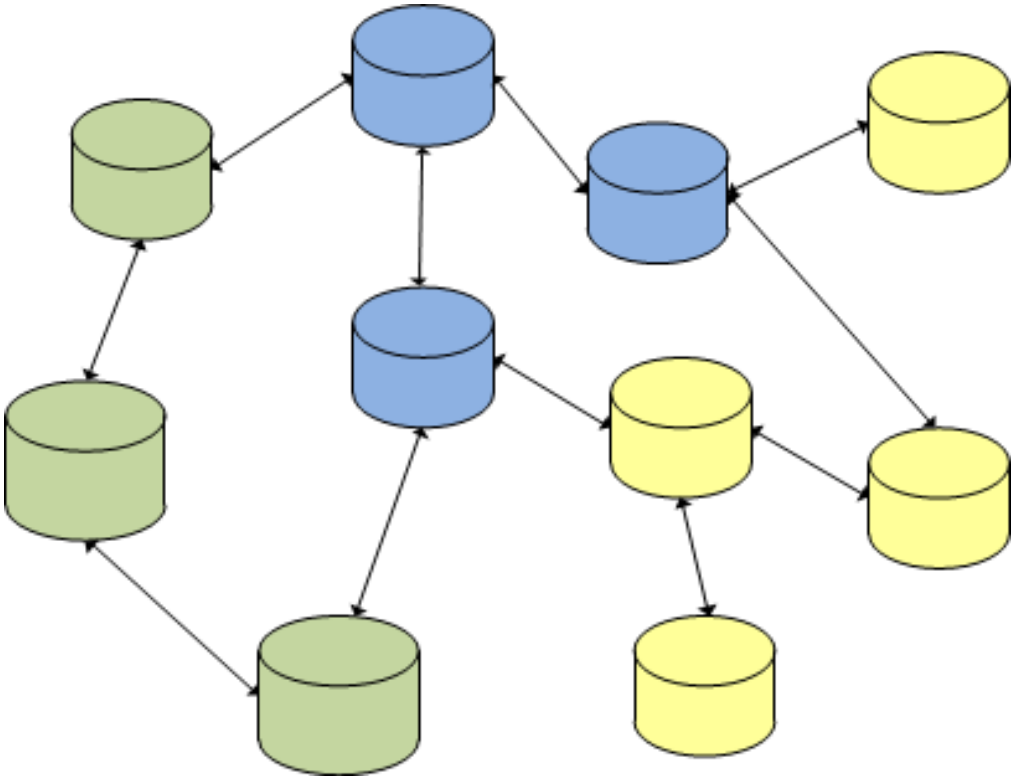
Desvantagens:

- Consistência dos Dados

- Otimização das Consultas

- Maior Complexidade nas Operações, se a taxa de atualização dos dados é alta, então um grande tempo será gasto no acesso a todos os pontos da rede.

Figura 2.5 - Arquitetura de Peer to Peer (P2P)



Fonte: Produção do Autor

A escolha da arquitetura depende de alguns fatores:

– A quantidade de fontes de dados a serem integradas;

– A frequência de atualização das fontes;

– A infraestrutura de comunicação.

2.1.3. Metadados

Metadados são habitualmente definidos simplesmente como dados descrevendo outros dados. No entanto, cada vez mais, especialmente no meio digital, o conceito tem sido empregado em variados contextos que envolvem diversos propósitos e tecnologias.

Dempsey e Heery (1997, p. 5) conceituaram metadados como “dados associados com objetos que desoneram os usuários potenciais de ter

conhecimento completo antecipado da existência e características desses objetos”.

Segundo Alves (2010, p.47) os metadados são atributos que representam uma entidade (objeto do mundo real) em um sistema de informação. Em outras palavras, são elementos descritivos ou atributos referenciais codificados que representam características próprias ou atribuídas às entidades; são ainda dados que descrevem outros dados em um sistema de informação, com o intuito de identificar de forma única uma entidade (recurso informacional) para posterior (acesso e) recuperação.

Seja o exemplo a seguir um trecho de código XML.

```
<pessoa sexo="feminino">
    <nome>Ana</nome>
    <sobrenome>Silva</sobrenome>
</pessoa>
```

Nesse código XML, pessoa, nome e sobrenome são marcas (elementos), onde, há uma ocorrência dos elementos nome e sobrenome aninhados dentro do elemento pessoa e sexo é um atributo, seguido de seu conteúdo.

Um ponto a destacar é que as marcas XML, não são fixas, ou seja, são convencionalmente criadas pelas comunidades de usuários. Isso aproxima a linguagem da semântica dos dados e tem ocasionado uma especialização e surgimento de padrões diversos adaptados a finalidades de cada domínio.

2.1.4. Balance Line

Balance Line é um algoritmo, uma técnica computacional para o gerenciamento de grandes volumes de dados sequenciais, que são grandes conjuntos de dados, com origem em uma ou diversas fontes, com uma chave em comum e que são ordenados por esta chave.

A melhoria do desempenho no processamento desses dados e a otimização do uso de recursos computacionais, são importantes pontos que devem ser considerados para a adoção desta técnica.

A aplicação desta técnica está relacionada às atividades de:

- Sincronização de dados;
- Carga de dados (parcial ou completa) e
- Integração de dados.

Os principais conceitos relacionados a esta técnica são:

1. Master File (Arquivo Principal)

É o principal conjunto de dados, que representa a visão final dos dados, é a referência no processamento com os demais conjuntos, é a origem dos dados.

2. Transaction File (Arquivo de Transação)

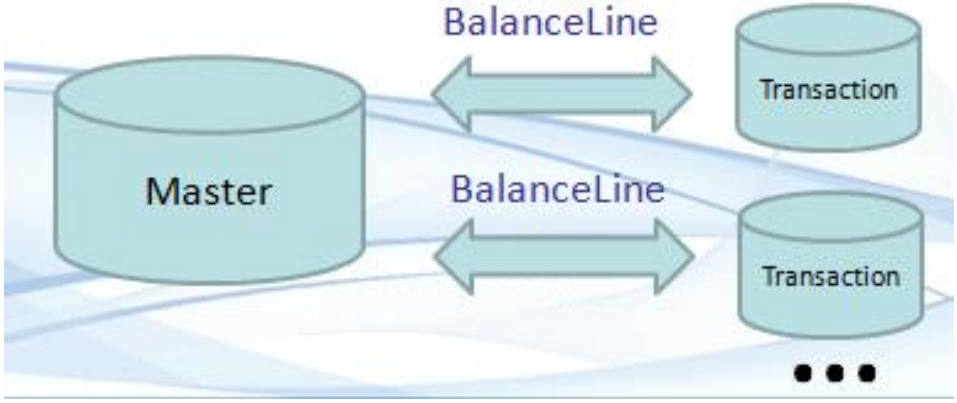
É o conjunto secundário de dados, que pode ser um ou mais arquivos, que representam as transações efetuadas e que contém os dados que serão sincronizados com a origem (Master File).

3. Key (Chave de registro)

É o identificador único de um registro no conjunto de dados, podendo ser um único atributo, uma junção de atributos etc.

A figura abaixo, ilustra a arquitetura de funcionamento desta técnica.

Figura 2.6 - Arquitetura Balance Line



Fonte: HOLMS (2011)

2.2. **Trabalhos Acadêmicos (TA) relacionados**

Nesta seção, são apresentados os principais trabalhos relacionados ao tema proposto, estudados para a avaliação do estado da arte e para fundamentar e orientar as etapas desse trabalho. Nos trabalhos apresentados, foram analisados aspectos relativos ao uso de frameworks na arquitetura e desenvolvimento de aplicações desacopladas e flexíveis, demonstrando, por exemplo, conceitos de reusabilidade, reflexão, inversão de controle e configuração por metadados.

Uma pesquisa bibliográfica sobre os frameworks, fornece uma perspectiva de como as novas abordagens para construí-los, complementa as técnicas anteriores, a fim de trazer mais benefícios para o aplicativo que usá-lo.

2.2.1. XGIS FLEX: Um Framework Livre para o desenvolvimento de Sistemas de Informações Geográficas para a Web (TA1)

Segundo Pascual (2013), em sua dissertação de mestrado, intitulada XGIS FLEX: Um Framework Livre para o desenvolvimento de Sistemas de Informações Geográficas para a Web onde são apresentados dois estudos de caso relacionados a arquiteturas de implantação de sistemas de informações geográficas (SIG) em ambiente web que culminaram no

desenvolvimento de um framework composto por um conjunto de componentes de apoio à construção de novos softwares de SIG para web. O primeiro estudo foi realizado com foco na interoperabilidade de serviços de mapas na web, contemplando principalmente as funções de consulta e disponibilização de dados em formatos diversos. Para viabilizar a troca e integração de informações espacializadas, foi necessário o desenvolvimento de alguns componentes que foram escritos sobre tecnologias proprietárias tais como Esri Flex API. Entendidas as limitações que um motor central de código fechado poderia trazer ao objeto principal do estudo, isto é, o desenvolvimento de uma arquitetura de intercâmbio de informações geográficas, surgiu a ideia de construir um conjunto de software que contornassem tais dificuldades. O segundo estudo foi realizado com foco na elaboração de um framework livre para desenvolvimento de SIG em ambiente web de padrão Rich Internet Application (RIA) e que seguisse conceitos da Web 2.0, possibilitando a indexação e visualização de dados provenientes de diferentes tipos de mídia (fotografias, vídeos, documentos) que fossem aderentes aos padrões da Open Geospatial Consortium (OGC), mas que, ainda assim, fossem possíveis de serem compartilhadas com outros formatos de dados. Este último estudo buscou ainda reduzir a complexidade e custos de implantação de projetos dessa natureza e facilitar a customização por parte de usuários finais. Ao conjunto de códigos desenvolvidos e sua arquitetura, foi dado o nome de XGIS Flex Framework.

2.2.2. Proposta de um Framework de Persistência de Objetos em bases de dados Objeto-Relacional (TA2)

Segundo Rombaldo J. (2012), em sua dissertação de Mestrado intitulada Proposta de um Framework de Persistência de Objetos em bases de dados Objeto-Relacional, que apresenta o desenvolvimento de um Framework de persistência que utiliza banco de dados objeto-relacional como mecanismo de persistência. Tendo por objetivo usar os conceitos de orientação a objetos descritos na norma SQL:2008, conceitos como: objeto (atributos e métodos), herança, agregação, composição, referências (REF) e estruturas multivaloradas (arrays e multiset). Para

tanto se desenvolveu e formalizou (XSD) um arquivo XML que representa um esquema de objetos através da norma SQL:2008. Da mesma forma definiu-se um conjunto de anotações Java, com o intuito de facilitar a utilização e configuração do Framework, o qual é chamado de O-ODBM (Object Object-Relational Database Management).

2.2.3. Utilizando Anotações em Linguagens Orientadas a Objetos para Suporte à Programação Orientada a Componentes (TA3)

Segundo Saldanha (2010), em sua dissertação de Mestrado, intitulada, Utilizando Anotações em Linguagens Orientadas a Objetos para Suporte à Programação Orientada a Componentes, define que em sistemas distribuídos baseados em componentes de software, o uso de linguagens de programação orientadas a objeto é bastante comum para definir, através de frameworks, interfaces de programação para construção e uso de componentes. No entanto, o que se percebe na maioria dos modelos de programação que seguem essa abordagem, é a utilização de construções das próprias linguagens orientadas a objeto, como classes e interfaces, para definir uma interface de programação que segue um paradigma orientado a componentes. Como consequência, o código fonte mistura aspectos da funcionalidade do componente com os mecanismos de implementação específicos do modelo de programação, o que impede a reutilização deste componente em outros frameworks, além de incluir uma complexidade extra no código. Recentemente, observamos uma tendência à adição de metadados às implementações dos componentes, utilizando marcações específicas no código fonte. Estes metadados proveem as informações necessárias para que alguma ferramenta, seja baseada em geração de código ou em mecanismos de reflexão computacional da própria linguagem de programação, realize a integração da implementação do componente com a infraestrutura de suporte do modelo de componentes. Essa técnica é denominada, por alguns autores, de Programação Orientada a Atributos. Linguagens como Java e C# já oferecem suporte nativo a esta técnica através das Anotações. O objetivo desta dissertação é investigar a adoção da técnica de programação orientada a atributos juntamente com uma linguagem orientada a objetos

para construção de aplicações baseadas em componentes. Como parte do estudo, foi desenvolvido um novo mecanismo de programação baseado em atributos para a versão Java do middleware SCS.

2.2.4. A Conceptual Model for Metadata-based Frameworks (TA4)

Segundo Guerra (2010), em sua tese de Doutorado, intitulada A Conceptual Model for Metadata-based Frameworks, um framework, pode ser considerado um software incompleto com alguns pontos que podem ser especializados para adicionar comportamento específico às aplicações, permitindo não só a reutilização de código fonte, mas também a reutilização de design de projetos. Técnicas para desenvolvimento de frameworks evoluiu, a partir do uso de herança e composição e passando por outras mais sofisticadas, como reflexão e introspecção. Frameworks mais recentes empregam a estratégia de definir um esquema de metadados específicos para aplicações para usar em suas classes e elementos de programação, permitindo a personalização do comportamento do framework. Apesar desta técnica está sendo amplamente utilizada, não há modelos, padrões de projeto ou diretrizes de desenvolvimento que visam ajudar na criação deste tipo de framework. Esta tese propõe um modelo conceitual de framework baseado em metadados que tem como objetivo, identificar soluções adequadas para a sua estrutura interna e cenários em que é adequado. A linguagem de padrões apresenta soluções de design visando uma maior flexibilidade na estrutura deste tipo de framework e uma coleção de padrões de arquitetura propondo cenários para seu uso. Alguns frameworks foram desenvolvidos por estudantes que usam as soluções no padrão proposto e os resultados da sua utilização foram avaliados através de questionários. Além disso, foi conduzido um experimento avaliando o uso de framework baseado em metadados nos cenários identificados, observando as consequências comparativamente às abordagens sem o uso de framework e com um framework tradicional. Como resultado, com este modelo conceptual, os arquitetos de software e desenvolvedores de frameworks devem ser capazes de identificar situações onde o uso de

metadados é apropriado e projetar soluções adequadas que proporciona flexibilidade na leitura de metadados e de processamento.

Um dos resultados apresentados, foi a criação de uma Linguagem Padrão para Frameworks baseados em metadados que documenta as melhores práticas de como estruturar internamente este tipo de componente de software, que terão aplicabilidade na concepção e na execução do trabalho proposto.

2.2.5. Frameworks de mercado (FM)

A adoção de frameworks de mapeamento Objeto-Relacional (ORM) se proliferou no desenvolvimento de software, com o objetivo de suprir a necessidade de implementação de uma camada de persistência que abstraia o processo de tradução do modelo de dados Orientado a Objeto (OO) para o modelo de dados Relacional, que é o mais utilizado para o armazenamento das informações, minimizando o acoplamento da aplicação em relação ao mecanismo de armazenamento de dados.

Na plataforma Java, proposta para este trabalho, alguns frameworks têm sua adoção estabelecida no processo de desenvolvimento de software de diversas empresas da área. Exemplos de projetos de frameworks ORM na plataforma Java são:

- **Hibernate (FM1):** projeto open source liderado pela equipe de desenvolvimento da Red Hat;

- **EclipseLink (FM2):** projeto open source da Eclipse.org

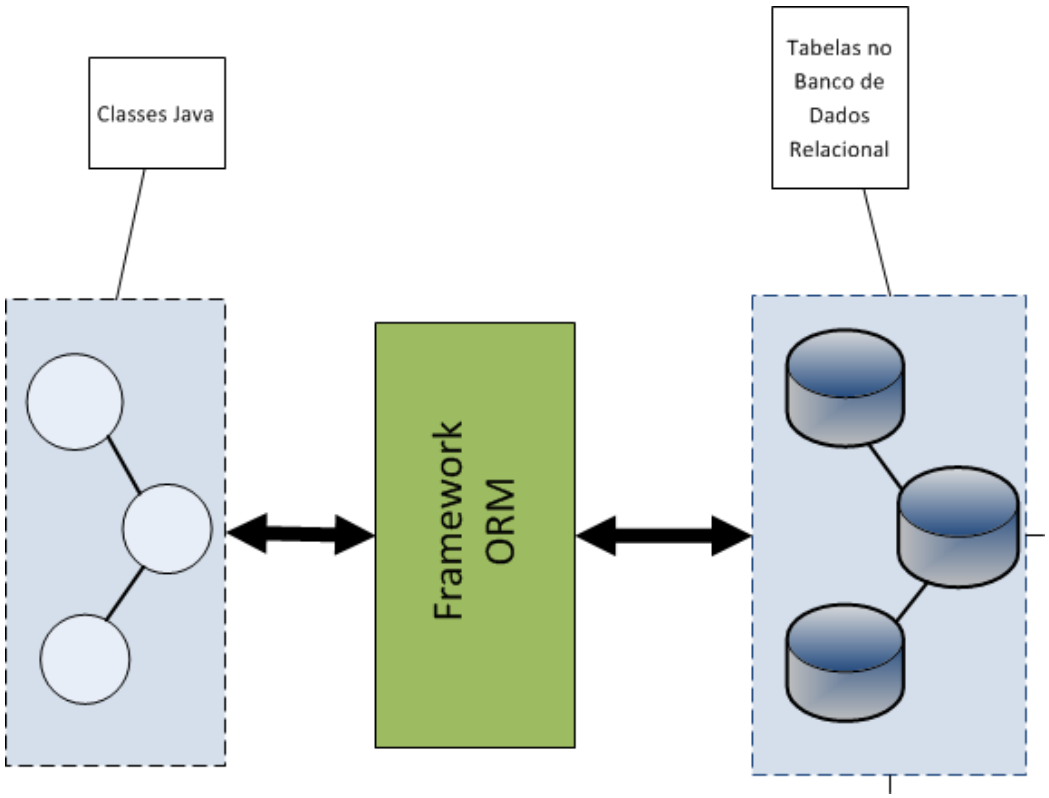
- **OpenJPA (FM3):** projeto open source da Apache Software Foundation

A arquitetura destes projetos é semelhante e se baseia em fornecer uma camada de persistência para acesso às bases de dados, através do mapeamento, das classes implementadas no código, por Anotações Java ou um arquivo em formato XML, encapsulando o processo de tradução de

código Java para instruções SQL (Structured Query Language), necessárias para acesso a base de dados no modelo Relacional.

Neste processo, os métodos que implementam as lógicas de negócios específicas de cada aplicação, ainda são de responsabilidade da equipe de desenvolvimento.

Figura 2.7 - Ilustração do mapeamento Objeto-Relacional



Fonte: Produção do Autor

2.2.6. Considerações

Após a análise dos trabalhos acadêmicos relacionados e os frameworks de mercado na plataforma Java, ficou clara a natureza empírica das informações a respeito da implementação e a adoção de frameworks no processo de desenvolvimento de sistemas da informação. No entanto, a importância do domínio do negócio do qual o sistema em desenvolvimento tem como objetivo automatizar as atividades do respectivo processo, é ressaltada nesses trabalhos.

Portanto, trabalhos com o objetivo de simplificar o processo de desenvolvimento de software para áreas de domínio de negócios específicos, como é o abordado por esta dissertação, através da adoção de frameworks, são oportunos e relevantes para a comunidade científica.

Uma comparação entre os trabalhos acadêmicos (TA) e os frameworks de mercado (FM) citados com a esta dissertação, é demonstrada na Tabela 2.1.

Tabela 2.1 - Comparativo entre os frameworks

Pontos relevantes	**Proposta deste trabalho**	**Trabalhos acadêmicos**	**Frameworks de Mercado**
Baixo Acoplamento	Sim, todos os métodos de acesso a dados são encapsulados no framework	Sim, métodos específicos devem ser implementados no software cliente **(Todos TA)**	Sim, métodos específicos devem ser implementados no software cliente **(Todos FM)**
Esforço de integração	Baseado em configuração	Baseado em implementação **(Todos TA)**	Baseado em implementação **(Todos FM)**
Reuso de código	Sim	Sim **(Todos TA)**	Sim **(Todos FM)**
Uso de metadados	Sim, arquivo XML	Sim, arquivo XML **(TA1 e TA2)** ou Anotação Java **(TA2, TA3 e TA4)**	Sim, arquivo XML e ou Anotação Java **(Todos FM)**
Classes para mapeamento	Encapsuladas no framework	Implementadas no software cliente **(TA2, TA3 e TA4)**	Implementadas no software cliente **(Todos FM)**

Fonte: Produção do Autor

3 ARQUITETURA DO FRAMEWORK

Neste capitulo são apresentadas as principais características do Framework, dando ênfase aos principais requisitos para implementação, aos benefícios da implantação no cenário do CMCD e as funcionalidades de cada módulo da arquitetura interna

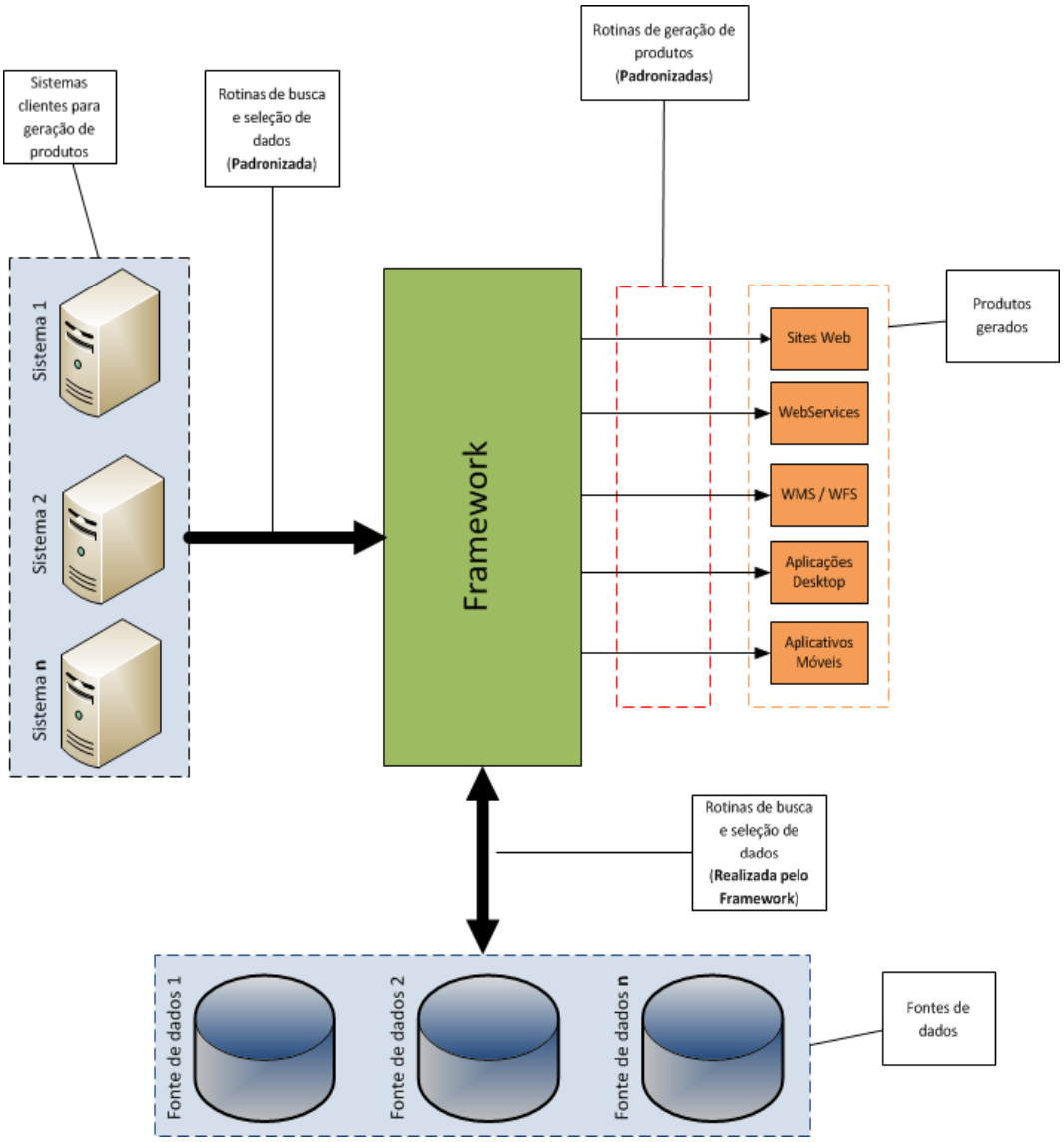
3.1. **Características do Framework**

Os principais requisitos definidos para a implementação deste trabalho são:

- permitir que usuários consultem simultaneamente múltiplas fontes de dados
  - Heterogêneas
  - Distribuídas
  - Autônomas
- manter transparentes os procedimentos de acesso, extração e integração dos dados;
- apresentar uma visão uniforme e consistente dos dados;
- prover interoperabilidade sintática: adoção de um modelo de dados comum
- prover interoperabilidade estrutural: definição de mapeamentos
- prover interoperabilidade semântica: uso de vocabulários

O cenário apresentado na Figura 3.1 demonstra a implantação do framework no ambiente de software do CMCD.

Figura 3.1 - Cenário com a implantação do Framework



Fonte: Produção do Autor

São alguns dos principais ganhos para a adoção desta ferramenta:

- redução do custo de desenvolvimento e manutenção dos sistemas clientes, eliminando trabalho repetitivo no acesso, seleção e disponibilização de dados;
- maximização do reuso;
- minimização de acoplamento;
- melhor consistência e compatibilidade entre os sistemas

As principais limitações estão relacionadas à tecnologia e ao ambiente computacional onde o software cliente é executado, podemos citar:

– o software cliente deve ser desenvolvido na plataforma JAVA ou em outra tecnologia que possibilite acesso à um framework JAVA;

– as fontes de dados mapeadas devem possuir o driver necessário para acesso via JAVA;

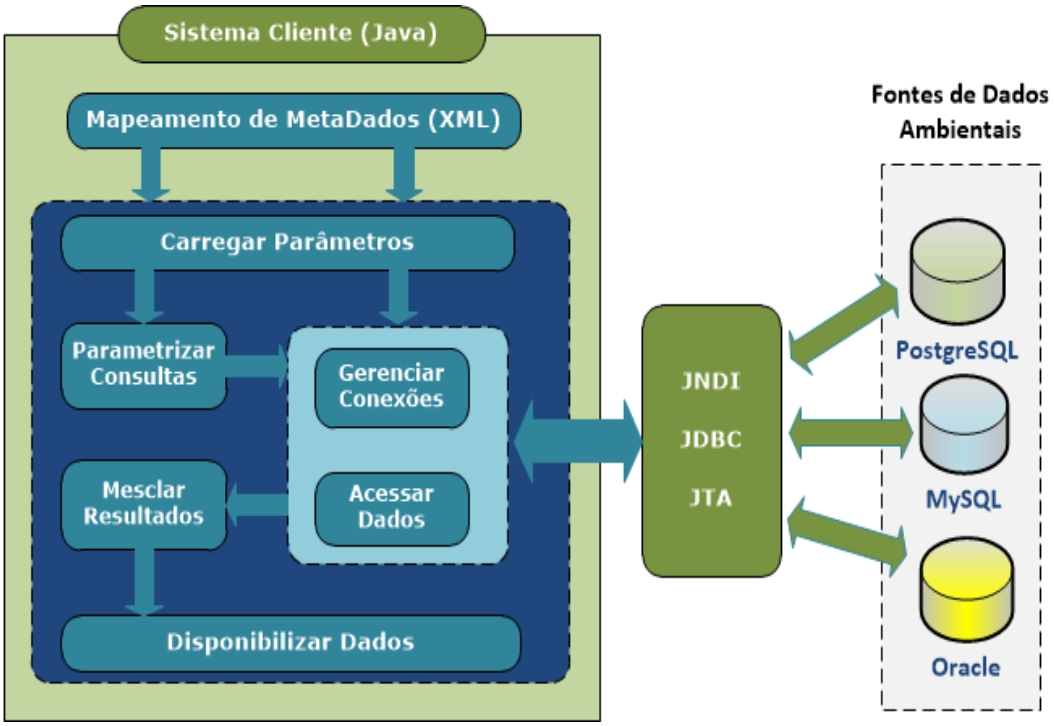
– o acesso às fontes de dados depende da infraestrutura de rede disponível no ambiente computacional onde o software cliente é executado;

– o número de conexões e a quantidade de dados retornados nas consultas, estão limitados à capacidade de memória RAM disponível no ambiente computacional onde o software cliente é executado;

3.2. **Arquitetura Interna**

A Figura 3.2 ilustra a arquitetura interna do framework, integrado à um sistema cliente, representada pela funcionalidade principal de cada módulo.

Figura 3.2 - Arquitetura Interna do Framework



Fonte: Produção do Autor

A implantação do Framework na arquitetura, não impacta nas funcionalidades definidas para o Sistema Cliente, ficando restrito aos métodos de acesso às fontes de dados ambientais.

Adotou-se o XML como formato padrão para o mapeamento dos metadados de forma a propiciar a interoperabilidade entre o framework e o sistema cliente de forma explícita e desacoplada do código fonte do sistema cliente, o que facilita o entendimento e a definição dos metadados por parte da equipe de desenvolvimento.

O Framework interage com as APIs disponíveis no sistema cliente, o que propicia independência no que diz respeito ao tipo de Gerenciador de Banco de Dados (SGBD) utilizado em cada fonte.

As seções a seguir, descrevem o funcionamento de cada módulo da arquitetura do Framework.

3.2.1. Mapear Fontes de Dados

O mapeamento dos metadados é efetuado pela equipe de desenvolvimento do Sistema Cliente através da definição de valores para os atributos referentes às fontes de dados, definindo o tipo, as credenciais de acesso, as tabelas e os respectivos campos que serão utilizados para o processo de acesso e integração de dados.

O arquivo de configuração dos metadados deve ser copiado para um diretório que esteja definido dentro do CLASSPATH, que é uma variável de ambiente do sistema operacional onde o sistema cliente é executado. Esta variável indica à Máquina Virtual Java (JVM) onde procurar bibliotecas, pacotes, arquivos e classes definidos durante o processo de desenvolvimento do sistema.

A Figura 3.3, ilustra o modelo do arquivo de configuração dos metadados, em formato XML, que deve ser definido na implementação, onde os dados dos respectivos atributos serão carregados durante a execução do sistema cliente do framework.

Figura 3.3 - Modelo do arquivo de mapeamento XML

```xml
<datasources>
    <datasource>
        <name>dt1</name>
        <url>url1</url>
        <driver>drv1</driver>
        <user>user1</user>
        <passwd>passwd1</passwd>
        <table name="dt1_tb1">
            <field>
                <fullname>Campo1 tabela1 dt 1</fullname>
                <dataname>field1</dataname>
                <frequency>30</frequency>
                <type>String</type>
                <source>Modelo</source>
            </field>
            <field>
                <fullname>Campo2 tabela1 dt 1</fullname>
                <dataname>field2</dataname>
                <frequency>15</frequency>
                <type>Integer</type>
                <source>Satelite</source>
            </field>
        </table>
    </datasource>
</datasources>
```

Fonte: Produção do Autor

O processo de validação do arquivo de metadados exige e existência de todos os elementos definidos, mesmo os que têm o preenchimento opcional. A descrição de cada elemento exigido para a definição dos metadados é:

a) **<datasource>** - elemento identificador da fonte de dados;

b) **<name> -** nome da fonte de dados que será utilizado como referência no código fonte durante a implementação. **Preenchimento obrigatório**;

c) **<url> -** caminho a ser utilizado na classe de conexão. Padrão utilizado em conexões JDBC para o respectivo tipo de gerenciador de banco de dados. **Preenchimento obrigatório**;

d) **<driver>** - nome do driver específico para a fonte de dados. Padrão utilizado em conexões JDBC para o respectivo tipo de gerenciador de banco de dados. **Preenchimento obrigatório**;

e) **<user>** - nome do usuário no banco de dados. **Preenchimento obrigatório**;

f) **<passwd>** - senha do usuário do banco de dados. **Preenchimento obrigatório**;

g) **<table>** - nome da tabela no banco de dados. **Preenchimento obrigatório**

h) **<field>** -elemento identificador para cada campo na respectiva tabela do banco de dados. **Somente os campos identificados serão utilizados no processo de integração**;

i) **<fullname>** - nome completo do campo utilizado opcionalmente no arquivo gerado durante o processo de disponibilização dos dados. Se vazio, será utilizado o nome original do campo. **Preenchimento opcional**;

j) **<dataname>** - nome original do campo na tabela do banco de dados. **Preenchimento obrigatório;**

k) **<frequency>** - a frequência de geração dos dados estabelecida para cada sistema de coleta. Informar o valor em minutos. **Preenchimento opcional**;

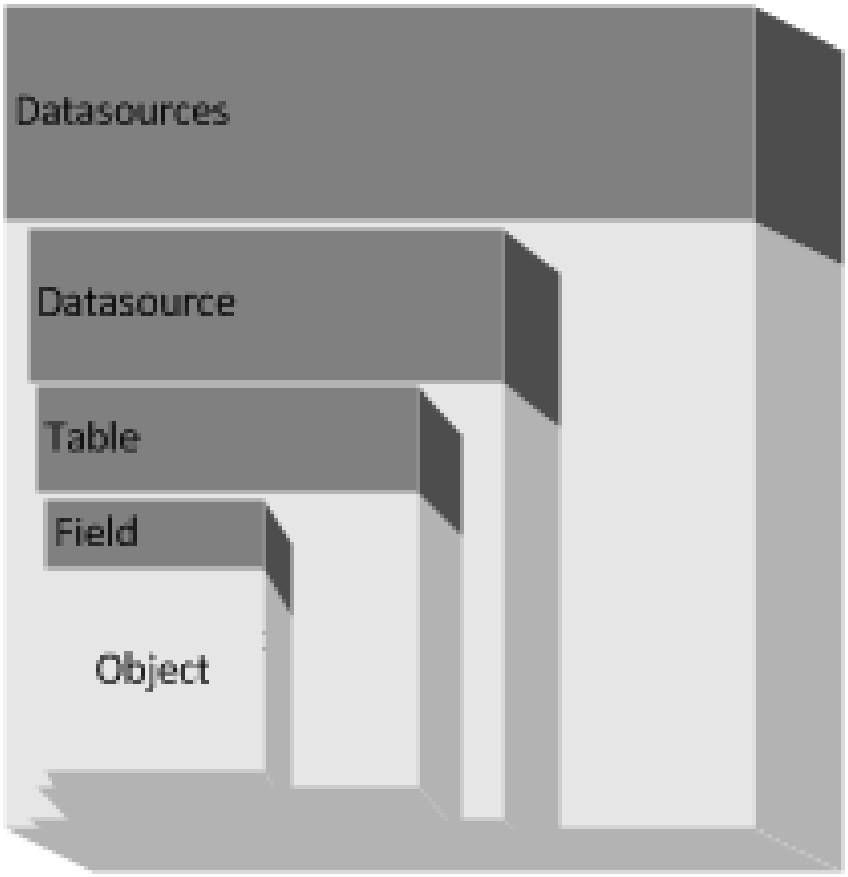
l) **<type> -** tipo do dado definido na tabela do banco de dados (utilizar o nome da classe Wrapper Java equivalente). **Preenchimento obrigatório;**

m) **<source>** - fonte original do dado, por exemplo, satélite, modelo etc. **Preenchimento opcional**;

3.2.2. Carregar parâmetros

Etapa de inicialização do sistema que carrega os metadados de mapeamento do arquivo de configuração.

Figura 3.4 - Hierarquia de classes de metadados



Fonte: Produção do Autor

Esta etapa retorna ao sistema cliente, a seguinte hierarquia de classes que representam os metadados de configuração:

- **Classe Datasources**: coleção de um ou mais objetos da classe Datasource;
- **Classe Datasource:** classe que define a fonte de dados e seus atributos;
- **Classe Table**: classe que define as tabelas do banco de dados para cada fonte;

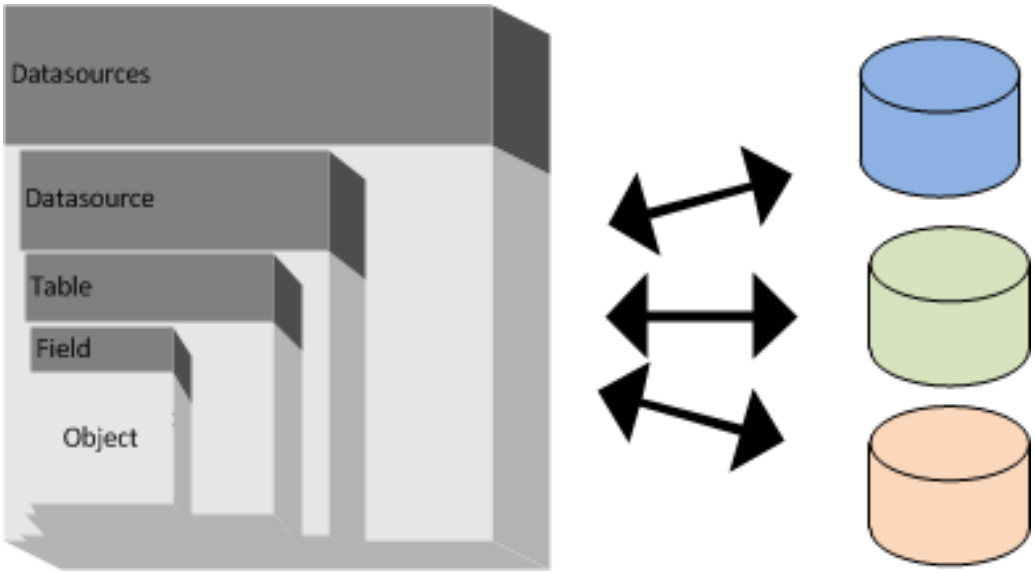
- **Classe Field**: classe que define os atributos relacionados às respectivas tabelas do banco de dados.

3.2.3. Gerenciar conexões

O pool de conexões é estabelecido conforme os tipos de gerenciadores de banco de dados definidos, com os respectivos parâmetros de acesso.

O framework utiliza os recursos do pool de conexões disponíveis para o sistema cliente, assim delegando o gerenciamento das conexões e interagindo com as APIs existentes.

Figura 3.5 - Abertura de conexões das fontes de dados



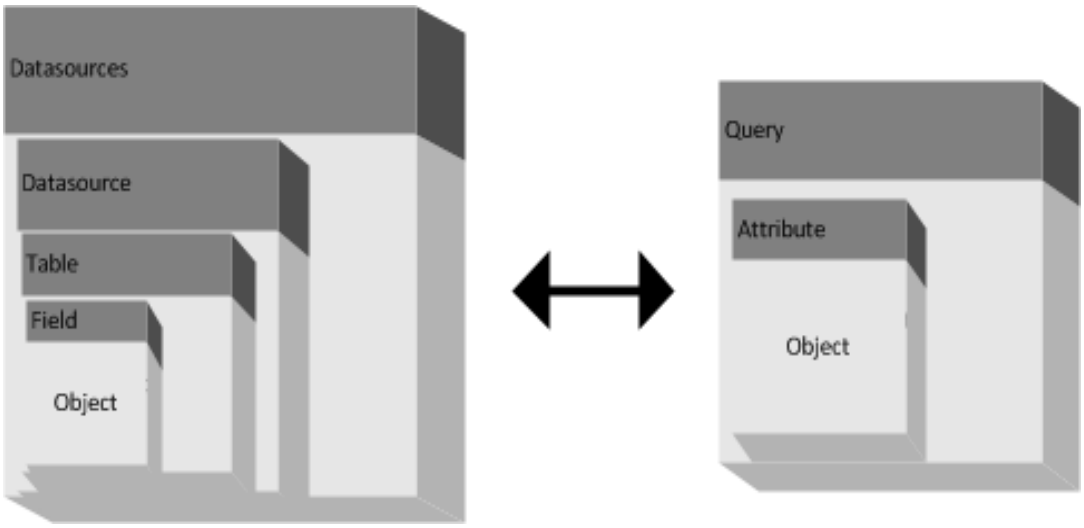
Fonte: Produção do Autor

A partir deste momento, todas as conexões às fontes de dados, definidas no mapeamento, estão disponíveis e prontas para uso pelo sistema cliente.

3.2.4. Parametrizar consultas

Todas as consultas são construídas dinamicamente através de chamadas de métodos parametrizados.

Figura 3.6 - Parametrização das consultas



Fonte: Produção do Autor

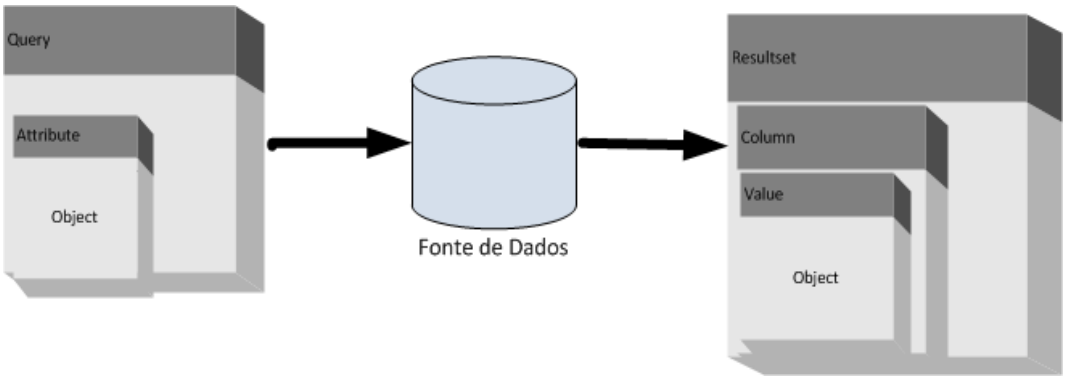
A parametrização das consultas é efetuada através da definição dos objetos das classes Query e Attribute, sendo:

- **Classe Query:** classe que define a fontes de dados, a tabela e o período desejado;
- **Classe Attribute:** classe que define os atributos das tabelas e, se necessário, especifica a faixa de valores para cada um.

3.2.5. Acessar a dados

É a Camada que executa o acesso a dados e retorna as listas de resultados.

Figura 3.7 - Execução das consultas nas fontes de dados



Fonte: Produção do Autor

Com todas as Querys definidas, o sistema cliente invoca o método responsável pelo acesso aos dados e a partir daí os comandos SQL são criados dinamicamente e as consultas são executadas em todas as fontes de dados especificadas.

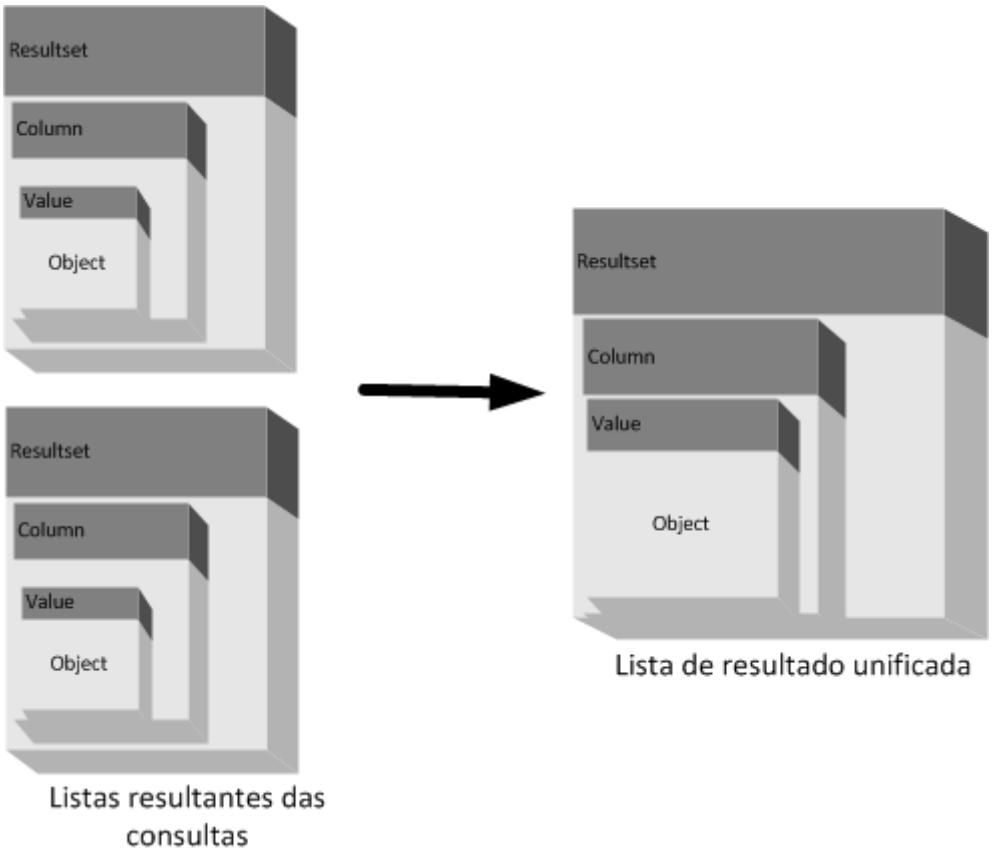
O resultado destas consultas é retornado nas classes Resultset, Column e Value, sendo:

- **Classe Resultset:** classe identificada por um nome e que recebe o conjunto de dados retornado na consulta;
- **Classe Column:** classe que recebe os nomes dos atributos dos dados retornados na consulta;
- **Classe Value:** classe que recebe a data e hora da coleta do dado e os valores de cada um.

3.2.6. Mesclar resultados

Este módulo do framework efetiva a integração dos dados consultados em cada respectiva fonte, mesclando uma lista definida como principal e preenchendo as lacunas nos intervalos de horários de coleta com os dados coletados nas demais fontes.

Figura 3.8 - Geração da lista unificada dos dados



Fonte: Produção do Autor

Nesta etapa, o sistema cliente, com os objetos da classe Resultset já populados com os dados resultantes de cada consulta, invoca o método responsável pela mescla dos dados, passando como parâmetro de entrada uma coleção da classe Resultset e obtém como retorno, um objeto da classe Resultset com todos os dados mesclados.

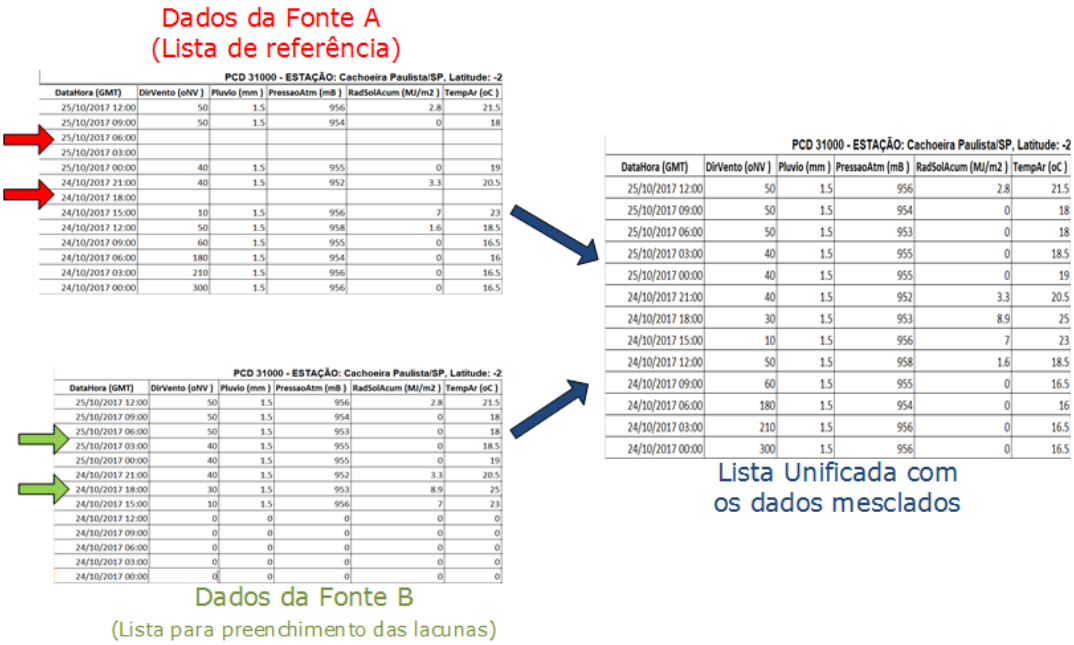
Por convenção adotada neste framework, os seguintes pontos devem ser considerados para o método que mescla os dados:

a) O primeiro elemento da coleção passada como parâmetro de entrada, é a lista principal de dados e será adotada como referência e as demais listas serão utilizadas para a geração da lista final, obedecendo a ordem de inclusão dos objetos;

b) O atributo de data e hora (timestamp) da coleta de dados é a chave de comparação entre as listas.

O resultado do processo de integração dos dados a partir de duas fontes de dados é ilustrado na Figura 3.9.

Figura 3.9 – Dados integrados em uma lista unificada

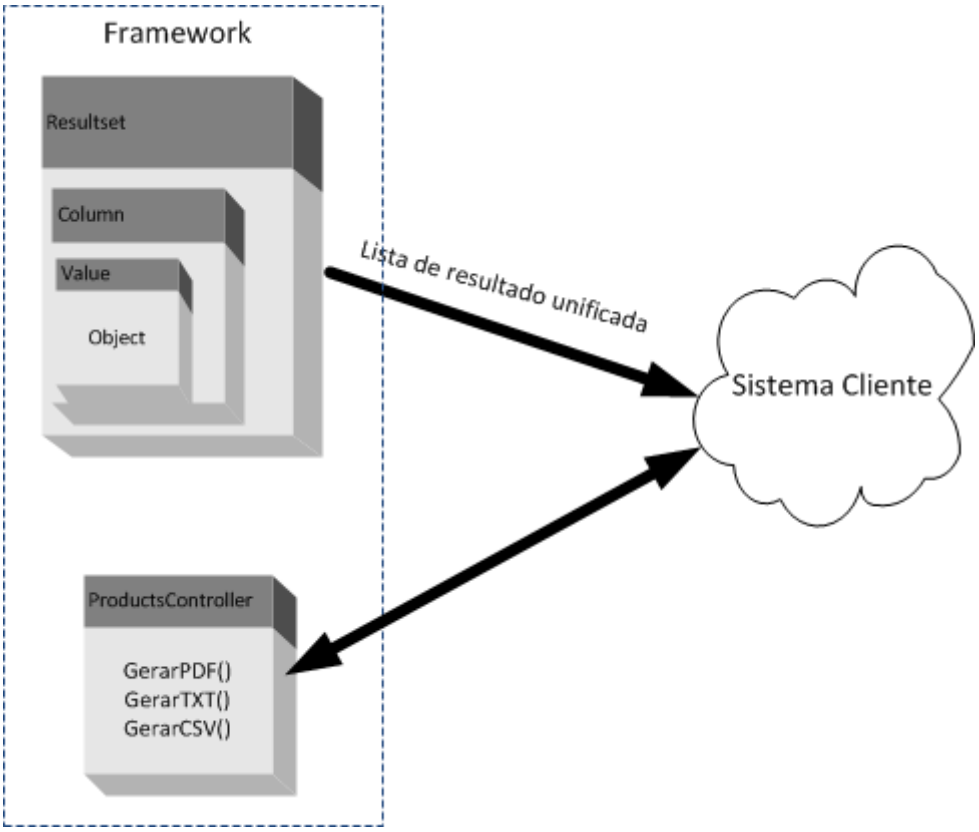


3.2.7. Disponibilizar dados

Os dados podem ser visualizados ou exportados para formatos definidos, incluindo xls, txt, csv, pdf, shape file, entre outros.

Os métodos de para geração do formato de saída de dados é parte da arquitetura principal do framework, ou seja, já são previamente implementados no código do framework em produção, em caso da necessidade da geração dos dados em novo formato, uma nova versão será gerada com o respectivo método.

Figura 3.10 - Geração de produtos



Fonte: Produção do Autor

O sistema cliente pode invocar os métodos de geração de produtos para os formatos pré-estabelecidos no framework, a qualquer momento, podendo ser aplicados na lista unificada de dados ou também na lista originada a partir de uma consulta simples, sempre passando como parâmetro um objeto da classe Resultset.

Figura 3.11 – Exemplo de arquivo de saída no formato PDF

PCD 31000 - ESTAÇÃO: Cachoeira Paulista/SP, Latitude: -22.675, Longitude: -45.002, Altitude: 563

DataHora (GMT)	DirVento (oNV)	Pluvio (mm)	PressaoAtm (mB)	RadSolAcum (MJ/m2)	TempAr (oC)	TempMax (oC)	TempMin (oC)	UmidRel (%)	VelVento10m (m/s)	VelVentoMax (m/s)
25/10/2017 12:00	50	1.5	956	2.8	21.5	25	18	74	10.6	19.1
25/10/2017 09:00	50	1.5	954	0	18	25	16.5	89	4.1	9.7
25/10/2017 06:00	50	1.5	953	0	18	25	16	86	5.9	11.5
25/10/2017 03:00	40	1.5	955	0	18.5	25	16	84	6.9	12.5
25/10/2017 00:00	40	1.5	955	0	19	25	16	85	7.1	17.8
24/10/2017 21:00	40	1.5	952	3.3	20.5	25	16	75	8.7	20.6
24/10/2017 18:00	30	1.5	953	8.9	25	25	16	61	10.1	18.7
24/10/2017 15:00	10	1.5	956	7	23	23	16	64	11	19.8
24/10/2017 12:00	50	1.5	958	1.6	18.5	22.5	16	81	6.9	13
24/10/2017 09:00	60	1.5	955	0	16.5	22.5	16	90	4.6	8.8
24/10/2017 06:00	180	1.5	954	0	16	22.5	16	94	2.9	6.2
24/10/2017 03:00	210	1.5	956	0	16.5	22.5	16.5	94	2.8	6.4
24/10/2017 00:00	300	1.5	956	0	16.5	22.5	16.5	93	2	7.5
23/10/2017 21:00	290	1.5	954	0.2	18	22.5	18	86	2.9	9.3
23/10/2017 18:00	30	1.5	952	1.8	21	25	19	82	4.2	29.5
23/10/2017 15:00	60	1.5	954	2.2	22	26.5	19	76	5	9.2
23/10/2017 12:00	40	1.5	956	0.4	20	26.5	19	89	2.6	1.4
23/10/2017 09:00	20	1.5	952	0	19	26.5	19	91	1.1	8.5

Figura 3.12 – Exemplo de arquivo de saída no formato XLS

1	PCD 31000 - ESTAÇÃO: Cachoeira Paulista/SP, Latitude: -22.675, Longitude: -45.002, Altitude: 563										
2	DataHora (GMT)	DirVento (oNV)	Pluvio (mm)	PressaoAtm (mB)	RadSolAcum (MJ/m2)	TempAr (oC)	TempMax (oC)	TempMin (oC)	UmidRel (%)	VelVento10m (m/s)	VelVentoMax (m/s)
3	25/10/2017 12:00	50	1.5	956	2.8	21.5	25	18	74	10.6	19.1
4	25/10/2017 09:00	50	1.5	954	0	18	25	16.5	89	4.1	9.7
5	25/10/2017 06:00	50	1.5	953	0	18	25	16	86	5.9	11.5
6	25/10/2017 03:00	40	1.5	955	0	18.5	25	16	84	6.9	12.5
7	25/10/2017 00:00	40	1.5	955	0	19	25	16	85	7.1	17.8
8	24/10/2017 21:00	40	1.5	952	3.3	20.5	25	16	75	8.7	20.6
9	24/10/2017 18:00	30	1.5	953	8.9	25	25	16	61	10.1	18.7
10	24/10/2017 15:00	10	1.5	956	7	23	23	16	64	11	19.8
11	24/10/2017 12:00	50	1.5	958	1.6	18.5	22.5	16	81	6.9	13
12	24/10/2017 09:00	60	1.5	955	0	16.5	22.5	16	90	4.6	8.8
13	24/10/2017 06:00	180	1.5	954	0	16	22.5	16	94	2.9	6.2
14	24/10/2017 03:00	210	1.5	956	0	16.5	22.5	16.5	94	2.8	6.4
15	24/10/2017 00:00	300	1.5	956	0	16.5	22.5	16.5	93	2	7.5
16	23/10/2017 21:00	290	1.5	954	0.2	18	22.5	18	86	2.9	9.3
17	23/10/2017 18:00	30	1.5	952	1.8	21	25	19	82	4.2	29.5
18	23/10/2017 15:00	60	1.5	954	2.2	22	26.5	19	76	5	9.2
19	23/10/2017 12:00	40	1.5	956	0.4	20	26.5	19	89	2.6	1.4
20	23/10/2017 09:00	20	1.5	952	0	19	26.5	19	91	1.1	8.5

Figura 3.13 – Exemplo de arquivo de saída no formato CSV

```
PCD 31000 - ESTAÇÃO: Cachoeira Paulista/SP, Latitude: -22.675, Longitude: -45.002, Altitude: 563;;;;;;;;;;
DataHora(GMT);DirVento(oNV);Pluvio(mm);PressaoAtm(mB);RadSolAcum(MJ/m2);TempAr(oC);TempMax(oC);TempMin(oC);UmidRel(%);VelVento10m(m/s);VelVentoMax(m/s)
25/10/2017 12:00;50;1.5;956;2.8;21.5;25;18;74;10.6;19.1
25/10/2017 09:00;50;1.5;954;0;18;25;16.5;89;4.1;9.7
25/10/2017 06:00;50;1.5;953;0;18;25;16;86;5.9;11.5
25/10/2017 03:00;40;1.5;955;0;18.5;25;16;84;6.9;12.5
25/10/2017 00:00;40;1.5;955;0;19;25;16;85;7.1;17.8
24/10/2017 21:00;40;1.5;952;3.3;20.5;25;16;75;8.7;20.6
24/10/2017 18:00;30;1.5;953;8.9;25;25;16;61;10.1;18.7
24/10/2017 15:00;10;1.5;956;7;23;23;16;64;11;19.8
24/10/2017 12:00;50;1.5;958;1.6;18.5;22.5;16;81;6.9;13
24/10/2017 09:00;60;1.5;955;0;16.5;22.5;16;90;4.6;8.8
24/10/2017 06:00;180;1.5;954;0;16;22.5;16;94;2.9;6.2
24/10/2017 03:00;210;1.5;956;0;16.5;22.5;16.5;94;2.8;6.4
24/10/2017 00:00;300;1.5;956;0;16.5;22.5;16.5;93;2;7.5
23/10/2017 21:00;290;1.5;954;0.2;18;22.5;18;86;2.9;9.3
23/10/2017 18:00;30;1.5;952;1.8;21;25;19;82;4.2;29.5
23/10/2017 15:00;60;1.5;954;2.2;22;26.5;19;76;5;9.2
23/10/2017 12:00;40;1.5;956;0.4;20;26.5;19;89;2.6;1.4
23/10/2017 09:00;20;1.5;952;0;19;26.5;19;91;1.1;8.5
23/10/2017 06:00;50;1.5;952;0;20.5;26.5;20;83;6.8;10.5
23/10/2017 03:00;50;1.5;952;0;20.5;26.5;20;83;6.8;10.5
23/10/2017 00:00;90;1.5;952;0;21;26.5;20;89;1.9;7.3
```

3.3. **ESTUDO DE CASO**

Este capítulo apresenta um estudo de caso para exemplificar a aplicação da arquitetura proposta neste trabalho através da demonstração da lógica de funcionamento interno de um sistema cliente real, integrado ao framework para a disponibilização de dados ambientais.

3.3.1. Aplicação do framework em um cenário real

Sistemas com interface para integração e visualização dinâmica de dados ambientais, é o melhor cenário de exemplo para a aplicação do framework proposto neste trabalho. A inclusão do suporte às funcionalidades disponíveis ao código fonte do software cliente minimiza e simplifica a codificação de operações básicas e repetitivas efetuadas durante o processo de desenvolvimento do software.

A Tabela 2, demonstra a relação de atividades básicas necessárias que deverão ser implementadas durante a codificação do sistema cliente, comparando os cenários sem a integração com o framework e utilizando as funcionalidades do framework.

Observa-se que após a inclusão de suporte das funcionalidades disponíveis na biblioteca do framework no código fonte do sistema cliente simplifica este processo, pois atividades repetitivas são encapsuladas e delegadas para a execução do framework.

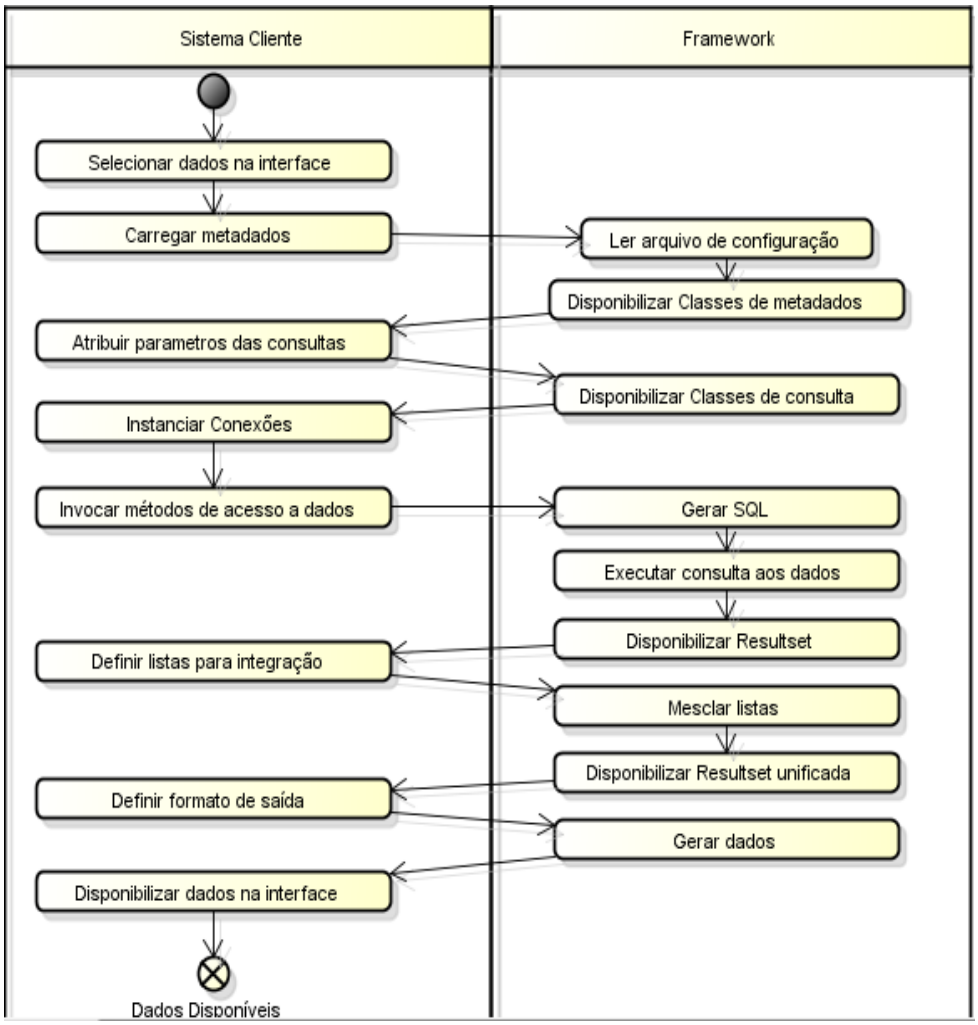
Tabela 3.1 - Comparação das operações sem e com o suporte do framework

Sem framework	Com framework
Definir conexões	Definir metadados
Abrir conexões	Parametrizar consultas
Definir querys (SQL ou classes)	Invocar método de consulta
Executar querys	Definir formato saída
Popular objetos	Dados disponíveis
Fechar conexões	
Mesclar resultado	
Ordenar lista resultante	
Definir formato de saída (desenvolver método)	
Dados disponíveis para uso	

Fonte: Produção do Autor

A Figura 3.14, ilustra o fluxo das atividades que devem ser implementadas pela equipe de desenvolvimento e as que têm a execução delegada ao framework durante a codificação do sistema cliente.

Figura 3.14 – Atividades entre sistema cliente e o framework



Fonte: Produção do Autor

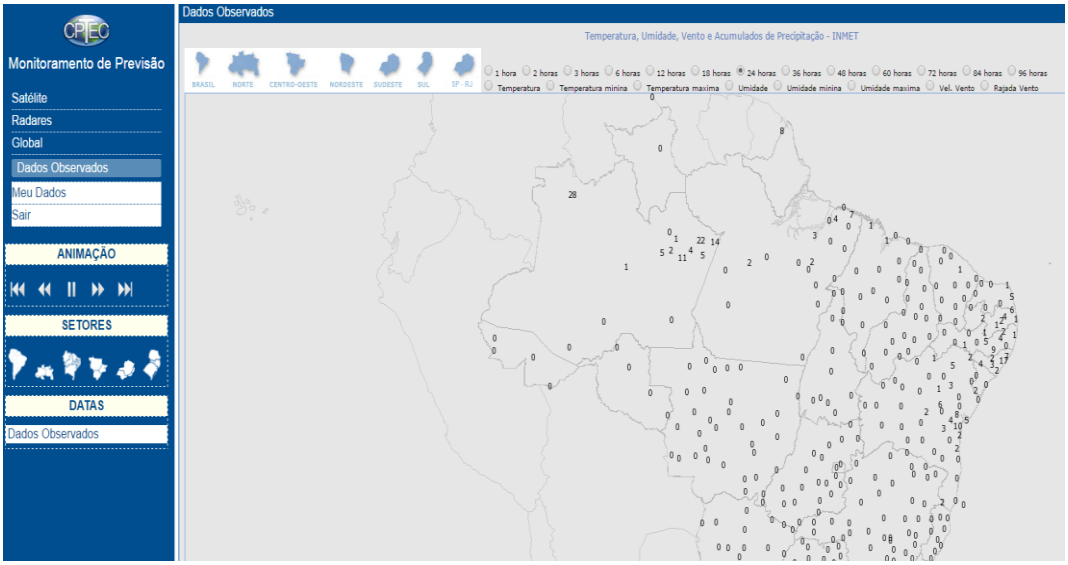
3.3.2. O sistema cliente

A Divisão de Sistemas e Satélites Espaciais – DSA, do CPTEC/INPE, desenvolveu um software, para ambiente Web, para integração e visualização de dados e produtos de Satélites Meteorológicos, sendo a equipe de Previsão de Tempo, o principal cliente desta aplicação.

A visualização de dados observados é uma opção deste pacote, podendo o usuário selecionar a região do Brasil com a respectiva rede instalada para a coleta de dados ambientais, a PCD que os dados serão

apresentados, o período de observação e as variáveis ambientais (Temperatura do Ar, Umidade, Precipitação etc.).

Figura 3.15 - Interface do sistema cliente



Fonte: Produção do Autor

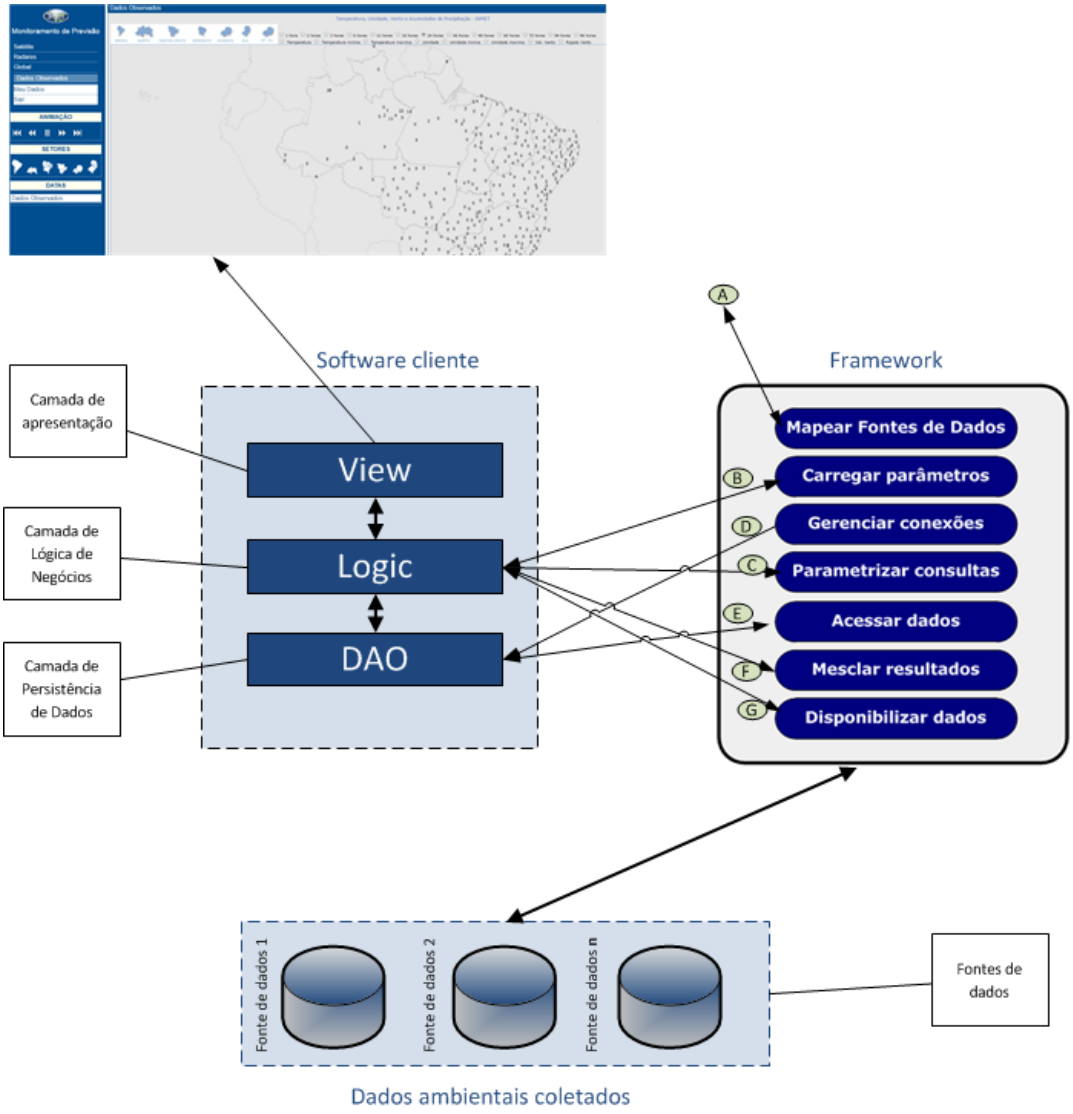
A cada nova fonte de dados incluída no processo de recepção e processamento já em produção na DSA, demanda uma manutenção no código fonte, sendo necessária a implementação de alterações nas rotinas de todas as operações citadas na Tabela 3.1.

O sistema cliente tem como padrão arquitetural o modelo MVC (Model-View-Controller), que divide o sistema em 3 camadas distintas, a camada View, que é responsável pela apresentação dos dados ao usuário, a camada Model, responsável pelos dados e as regras de negócios do sistema e a camada Controller, que gerencia o fluxo das informações entre as outras duas camadas. Este modelo arquitetural segue as melhores práticas no que se refere ao desenvolvimento de software, priorizando, por exemplo, o baixo acoplamento, uma maior coesão e a reusabilidade de código.

A Figura 3.16 demonstra-se o fluxo de execução de cada lógica necessária na arquitetura interna, para a operação do sistema cliente. Iniciando pela seleção dos dados a serem apresentados na interface para

o usuário do sistema, as chamadas referentes às lógicas de negócio e os métodos de persistência dos dados. Este fluxo passa pelas chamadas às funcionalidades do framework que ocorrem obedecendo à lógica definida no sistema cliente.

Figura 3.16 - Lógica de integração do framework



Fonte: Produção do Autor

A descrição de cada passo é:

– **Passo A:** a configuração dos metadados de todas as fontes de dados a serem utilizadas pelo software em desenvolvimento é a primeira atividade a ser executada pelo desenvolvedor. O arquivo, em formato XML, deverá estar no CLASSPATH do projeto e poderá

ser validado a qualquer momento por um método específico para este fim;

– **Passo B:** na camada de negócios, é invocado o método da classe responsável pelo carregamento dos metadados configurados previamente e a coleção de classes Datasources estará disponibilizada;

– **Passo C:** a parametrização das consultas é implementada através das classes Query e Attribute, conforme as variáveis selecionadas na interface do software;

– **Passo D**: o pool de conexões é disponibilizado para a camada de acesso aos dados e será gerenciado pelo servidor de aplicações sob o qual o sistema cliente está sendo executado;

– **Passo E:** o acesso aos dados é efetuado invocando os métodos implementados no software cliente e as classes Resultset derivadas dessas ações, deverão ser gerenciadas pelo programador e poderão ser efetuadas quantas vezes for necessária, conforme a lógica de negócios implementada;

– **Passo F:** a mescla dos resultados poderá ser invocada a qualquer momento após o acesso aos dados, retornando uma única classe Resultset para a camada de negócios;

– **Passo G:** os métodos de geração de produtos serão invocados conforme as opções de visualização e ou exportação de dados definidas na interface do software, limitados aos formatos previamente implementados no framework.

4 CONCLUSÃO

Neste trabalho propõe-se a construção de um arcabouço (framework) para a integração de bases de dados ambientais através de mapeamentos utilizando metadados, visando aumentar a eficiência, qualidade, simplicidade no uso, transparência nas conexões e reduzir os desperdícios de recursos envolvidos no processo de desenvolvimento de sistemas de informações em operação dentro de um Centro de Missão.

A arquitetura definida possui módulos com funcionalidades para o mapeamento das fontes, acesso e recuperação unificada e disponibilização padronizada dos dados.

A tecnologia de desenvolvimento adotada é a plataforma Java, o que possibilita que o framework interaja com o ambiente de execução (Servidor de Aplicações Java) no qual o sistema cliente está implantado, através do uso das mesmas APIs já disponíveis.

Uma contribuição importante deste trabalho é a comprovação de que uma arquitetura simplificada pode adotar boas práticas essenciais para o desenvolvimento de uma solução de software tais como, baixo acoplamento, baixo esforço de integração, reuso de código, uso de metadados e classes de mapeamento e obter resultados similares aos trabalhos acadêmicos e frameworks de mercado conforme demonstrado na Tabela 2.1 desta dissertação.

Esta solução simplifica a integração do framework ao código fonte do sistema cliente durante o processo de desenvolvimento, substituindo a codificação dos métodos de acesso aos dados por configuração dos metadados e parametrização das consultas.

Com esta abordagem, diferentemente dos frameworks de mercado citados, não é necessária a codificação de uma hierarquia de classes que represente o modelo de cada base de dados a ser consultada, minimizando o acoplamento com o sistema cliente. A implementação das rotinas de acesso aos dados está encapsulada no framework, que

disponibiliza os respectivos métodos para a invocação quando necessário.

Com a adoção do framework proposto para os sistemas, em produção ou em desenvolvimento, será propiciado um ambiente operacional do CMCD mais flexível no que diz respeito à adaptação quando da entrada em operação de uma nova missão espacial que forneça dados ambientais, um novo sistema de processamento ou uma nova fonte de dados externa.

A melhoria do processo de disponibilização dos dados ambientais processados no CMCD propiciará um melhor atendimento à crescente demanda da comunidade científica usuária destes dados, diminuindo, por exemplo, o tempo de resposta entre o horário de coleta e a disponibilização da informação.

As principais limitações observadas durante a execução deste trabalho, estão relacionadas à tecnologia adotada no desenvolvimento, o que atualmente se aplica somente à sistemas Java e a capacidade de escalabilidade da infraestrutura computacional onde o sistema cliente é executado, pois itens como a rede comunicação de dados e a quantidade de memória disponível nos servidores são gargalos que impactam diretamente no processo de integração e disponibilização dos dados ambientais.

Como sugestão para trabalhos futuros propõe-se a adoção e a implementação de Anotações Java no código principal do framework, disponibilizando como uma alternativa ao uso de arquivo XML, para o mapeamento dos metadados, minimizando eventuais erros de edição durante a codificação do sistema cliente.

Com relação aos metadados, outra proposta interessante, é incluir novos atributos que possibilitem que antes do processo de integração dos dados, seja executada uma validação básica dos valores recuperados das fontes de dados através de, por exemplo, definição de valores mínimos e máximos para uma determinada variável ambiental.

A interoperabilidade com sistemas que utilizem outras tecnologias e que também operem fora do ambiente computacional do CMCD através da implementação de um barramento de serviços Web, também é objeto de trabalhos futuros.

Para concluir ressalta-se que a tecnologia aplicada neste trabalho foi adotada em consonância com o padrão definido para o ambiente de software do CMCD, mas que os conceitos e o modelo de arquitetura descritos neste trabalho podem ser empregados em outras tecnologias e também para dados específicos de outras áreas de pesquisa.

REFERÊNCIAS BIBLIOGRÁFICAS

ALVES, R C V. **Metadados como elementos do processo de catalogação**. 2010. 132 f. Tese (doutorado) - Universidade Estadual Paulista, Faculdade de Filosofia e Ciências, 2010. Disponível em: <http://hdl.handle.net/11449/103361>

BAUER, C.; KING, G. **Java persistence com hibernate.** 1. ed, Rio de Janeiro Editora Ciência Moderna Ltda,2007 Brasil

BOSCH, J. **Design and use of software architecture**: adopting and evolving a product line architecture. [S.l.]: Addison-Wesley, 2000.

DATE, C. J. **Introdução a sistemas de bancos de dados**. 8. ed., Rio de Janeiro: Campus, 2004.

Documentação OpenJPA disponível em http://openjpa.apache.org/. Acesso em 01/08/2016.

Documentação Hibernate disponível em http://hibernate.org/orm/. Acesso dia 01/08/2016.

Documentação EclipseLink disponível em http://www.eclipse.org/eclipselink/. Acesso em 01/08/2016.

FOWLER, M.; SCOTT, K. **UML Essencial**. [S.l.]: Bookman, 2000. ISBN 8573077298.

FOWLER, M. **Padrões de arquitetura de aplicações corporativas**. São Paulo, SP: Artmed, 2008.

FOWLER, M. **Patterns of enterprise application architecture**. Boston, MA, USA Addison-Wesley Longman Publishing Co., Inc., 2002.

FOWLER, M. Inversion of control. http://martinfowler.com/bliki/InversionOfControl.html, 2005. Acesso em: 01/08/2016

FRANTZ, R. Z.; QUINTERO, A M R; CORCHUELO R. A domain-specific language to design enterprise application integration solutions. **International Journal of Cooperative Information Systems**, v. 20, n. 02, p. 143/176, 2011.

FREEMAN, E.; FREEMAN, E.; SIERRA, K.; BATES, B. **Padrões de projeto**. [S.l.]: Alta Books Rio de Janeiro, 2005.

GUERRA, E. **Design patterns com java**: projeto orientado a objetos guiado por padrões. [S.l.]: Editora Casa do Código, 2014.

Guerra M. E. **A conceptual model for metadata-based frameworks.** 2010. Tese (Doutorado em Informática) - Instituto de Tecnologia Aeroespacial - ITA, São José dos Campos, 2010.

HOLMS G., **BalanceLine4j framework overview.** 2011. Disponível em http://pt.slideshare.net/GilbertoHolms/balanceline4j-framework-overview. Acesso em: 01/08/2016.

LARMAN, G; **Utilizando UML e padrões** – uma introdução à análise e ao projeto orientado a objetos. Porto Alegre, Bookman, 2000.

MARKIEWICZ, M. E.; DE LUCENA, C. J. P. Object oriented framework development. **Crossroads**, v 7, p 3–9, July 2001

MEHTA, H.; KANUNGO, P.; CHANDWANI, M. Generic data access and integration service for distributed computing environment. **International Journal of Grid Computing & Applications (IJGCA),** Australia, v. 1, n. 1, p. 14-21, 2010.

NAVATHE,B.S; ELMASRI, R. **Fundamentals of database systems**. 6. ed. Addison-Wesley, 2010.

NUAIMI, K. A. et al. A partial replication load balancing algorithm for distributed data as a service (daas). In: INTERNATIONAL CONFERENCE ON IEEE HIGH PERFORMANCE COMPUTING AND SIMULATION (HPCS), 2013, Helsinki, Finland. **Proceedings…** p. 35–40.

PASCUAL, M, F **XGIS FLEX**: Um Framework Livre para o desenvolvimento de Sistemas de Informações Geográficas para a Web. 2013. Dissertação (Mestrado em Geociências) - Universidade de Brasília-UNB, Brasília, 2013.

SILBERSCHATZ, A.; KORTH, H.; SUDARSHAN, S. **Sistemas de banco de dados**. 1 ed. São Paulo; Campus, 2006.

ROMBALDO JR, C A, **Proposta de um framework de persistência de objetos em bases de dados objeto-relacional**, 2012. Dissertação (Mestrado em Sistemas Digitais) - Universidade de São Paulo - USP, 2012.

SALDANHA, H. M. C. **Utilizando anotações em linguagens orientadas a objetos para suporte à programação orientada a componentes**, 2010. Dissertação (Mestrado em Informática) - Pontifícia Universidade Católica - PUC, Rio de Janeiro, 2010.

SANTOS, M. A. F.; MATTIELO-FRANCISCO, M. F. M.; YAMAGUTI, W. O sistema nacional de dados ambientais e a coleta de dados por satélite. In: SIMPÓSIO BRASILEIRO DE SENSORIAMENTO REMOTO, 16. (SBSR)., 2013,Foz do Iguaçu. **Anais**... São José dos Campos: INPE, 2013. p. 9116–9123. ISBN 978-85-17-00066-9. Disponível em: <http://urlib.net/3ERPFQRTRW34M/3E7GKDH>.

SOMMERVILLE, I. **Engenharia de software**. [S.l]: Pearson Education, Inc, 2012.

TAYLOR, R. N.; MEDVIDOVIC, N.; DASHOFY, E. M. **Software architecture**: foundations, theory, and practice. [S.l.]: Wiley Publishing, 2009.

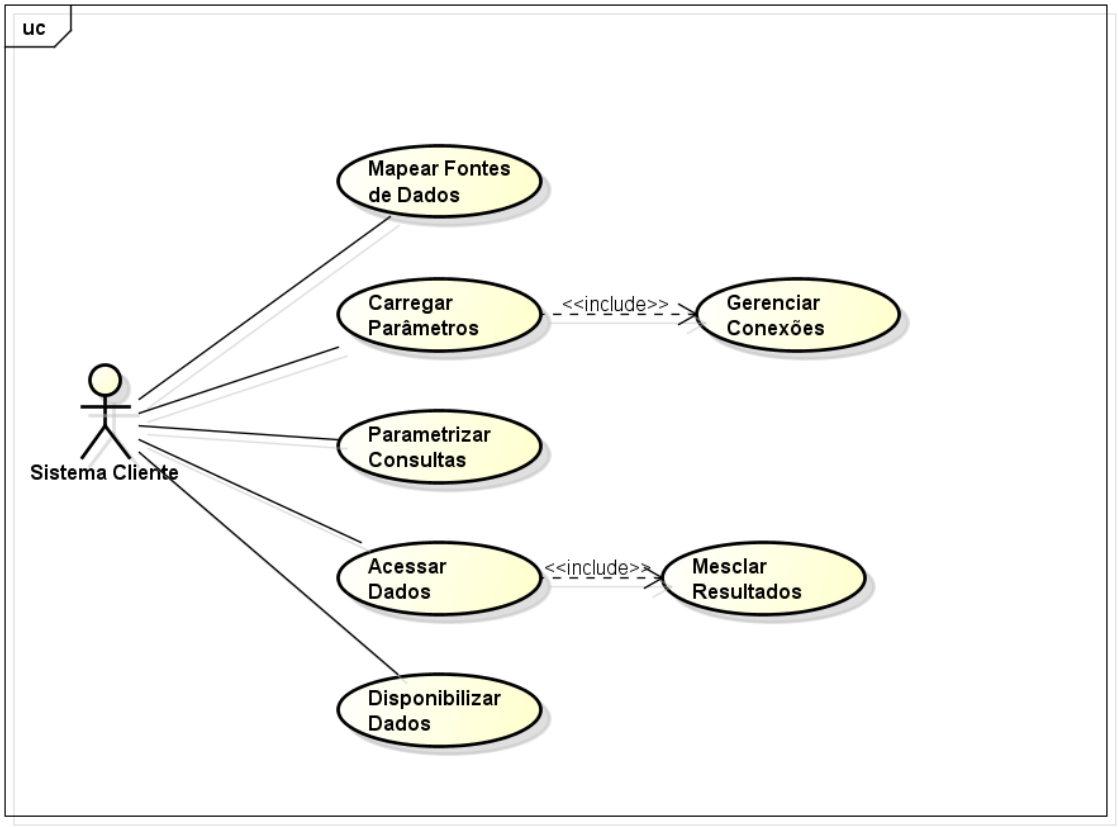
YAMAGUTI, W. et al. O Sistema Brasileiro de Coleta de Dados Ambientais:Status e planos futuros In: SIMPÓSIO BRASILEIRO DE SENSORIAMENTO REMOTO, 14. (SBSR), 2009, Natal. **Anais...** São José dos Campos: INPE, 2009. p. 1633-1640. DVD, On-line. ISBN 978-

85-17-00044-7. (INPE-16070-PRE/10679). Disponível em: <http://urlib.net/dpi.inpe.br/sbsr@80/2008/11.17.21.20.46>.

APENDICE A – Artefatos referentes à implementação do Framework

A.1 – Diagrama de Caso de uso Geral

Figura A.1 – Diagrama de caso de uso Geral



A.2 - Código fonte das classes do pacote Model

A.2.1 – Classe Attribute

```
package br.inpe.frame.model;

public class Attribute {
        private String name;
        private Double minvalue;
        private Double equalvalue;
        private Double maxvalue;

        public Attribute(){
```

```
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public Double getMinvalue() {
		return minvalue;
	}

	public void setMinvalue(Double minvalue) {
		this.minvalue = minvalue;
	}

	public Double getEqualvalue() {
		return equalvalue;
	}

	public void setEqualvalue(Double equalvalue) {
		this.equalvalue = equalvalue;
	}

	public Double getMaxvalue() {
		return maxvalue;
	}

	public void setMaxvalue(Double maxvalue) {
		this.maxvalue = maxvalue;
	}

}
```

A.2.2 – Classe Column

```
package br.inpe.frame.model;

import java.util.List;

public class Column {
	private String name;
	private List<Value> values;
```

```
	public Column(){

	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public List<Value> getValues() {
		return values;
	}
	public void setValues(List<Value> values) {
		this.values = values;
	}


}
```

A.2.3 – Classe Datasource

```
package br.inpe.frame.model;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlSchemaType;
import javax.xml.bind.annotation.XmlType;
import javax.xml.bind.annotation.adapters.CollapsedStringAdapter;
import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;


@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "name",
    "url",
    "driver",
    "user",
    "passwd",
    "table"
})
@XmlRootElement(name = "datasource")
public class Datasource {
```

```
@XmlElement(required = true)
@XmlJavaTypeAdapter(CollapsedStringAdapter.class)
@XmlSchemaType(name = "NCName")
protected String name;
@XmlElement(required = true)
@XmlJavaTypeAdapter(CollapsedStringAdapter.class)
@XmlSchemaType(name = "NCName")
protected String url;
@XmlElement(required = true)
@XmlJavaTypeAdapter(CollapsedStringAdapter.class)
@XmlSchemaType(name = "NCName")
protected String driver;
@XmlElement(required = true)
@XmlJavaTypeAdapter(CollapsedStringAdapter.class)
@XmlSchemaType(name = "NCName")
protected String user;
@XmlElement(required = true)
@XmlJavaTypeAdapter(CollapsedStringAdapter.class)
@XmlSchemaType(name = "NCName")
protected String passwd;
@XmlElement(required = true)
protected List<Table> table;

public String getName() {
    return name;
}

public void setName(String value) {
    this.name = value;
}

public String getUrl() {
    return url;
}

public void setUrl(String value) {
    this.url = value;
}

public String getDriver() {
    return driver;
}

public void setDriver(String value) {
    this.driver = value;
```

```
    }

    public String getUser() {
        return user;
    }

    public void setUser(String value) {
        this.user = value;
    }

    public String getPasswd() {
        return passwd;
    }

    public void setPasswd(String value) {
        this.passwd = value;
    }

    public List<Table> getTable() {
        if (table == null) {
            table = new ArrayList<Table>();
        }
        return this.table;
    }

}
```

A.2.4 – Classe Datasources

```
package br.inpe.frame.model;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;


@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "datasource"
})
@XmlRootElement(name = "datasources")
```

```
public class Datasources {

    @XmlElement(required = true)
    protected List<Datasource> datasource;

    public List<Datasource> getDatasource() {
        if (datasource == null) {
            datasource = new ArrayList<Datasource>();
        }
        return this.datasource;
    }

}
```

A.2.5 – Classe Field

```
package br.inpe.frame.model;

import java.math.BigInteger;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlSchemaType;
import javax.xml.bind.annotation.XmlType;
import javax.xml.bind.annotation.adapters.CollapsedStringAdapter;
import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;


@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "fullname",
    "dataname",
    "frequency",
    "type",
    "source"
})
@XmlRootElement(name = "field")
public class Field {

    @XmlElement(required = true)
    protected String fullname;
    @XmlElement(required = true)
```

```
@XmlJavaTypeAdapter(CollapsedStringAdapter.class)
@XmlSchemaType(name = "NCName")
protected String dataname;
@XmlElement(required = true)
protected BigInteger frequency;
@XmlElement(required = true)
@XmlJavaTypeAdapter(CollapsedStringAdapter.class)
@XmlSchemaType(name = "NCName")
protected String type;
@XmlElement(required = true)
@XmlJavaTypeAdapter(CollapsedStringAdapter.class)
@XmlSchemaType(name = "NCName")
protected String source;

public String getFullname() {
    return fullname;
}

public void setFullname(String value) {
    this.fullname = value;
}

public String getDataname() {
    return dataname;
}

public void setDataname(String value) {
    this.dataname = value;
}

public BigInteger getFrequency() {
    return frequency;
}

public void setFrequency(BigInteger value) {
    this.frequency = value;
}

public String getType() {
    return type;
}

public void setType(String value) {
    this.type = value;
}
```

```
    public String getSource() {
        return source;
    }

    public void setSource(String value) {
        this.source = value;
    }

}
```

A.2.6 – Classe ObjectFactory

```
package br.inpe.frame.model;

import java.math.BigInteger;
import javax.xml.bind.JAXBElement;
import javax.xml.bind.annotation.XmlElementDecl;
import javax.xml.bind.annotation.XmlRegistry;
import javax.xml.bind.annotation.adapters.CollapsedStringAdapter;
import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;
import javax.xml.namespace.QName;


@XmlRegistry
public class ObjectFactory {

    private final static QName _Passwd_QNAME = new QName("", "passwd");
    private final static QName _Dataname_QNAME = new QName("",
"dataname");
    private final static QName _Source_QNAME = new QName("", "source");
    private final static QName _Name_QNAME = new QName("", "name");
    private final static QName _Driver_QNAME = new QName("", "driver");
    private final static QName _Frequency_QNAME = new QName("",
"frequency");
    private final static QName _Type_QNAME = new QName("", "type");
    private final static QName _User_QNAME = new QName("", "user");
    private final static QName _Url_QNAME = new QName("", "url");
    private final static QName _Fullname_QNAME = new QName("",
"fullname");


    public ObjectFactory() {
    }

    public Field createField() {
```

```
        return new Field();
    }

    public Datasources createDatasources() {
        return new Datasources();
    }

    public Datasource createDatasource() {
        return new Datasource();
    }

    public Table createTable() {
        return new Table();
    }

    @XmlElementDecl(namespace = "", name = "passwd")
    @XmlJavaTypeAdapter(CollapsedStringAdapter.class)
    public JAXBElement<String> createPasswd(String value) {
        return new JAXBElement<String>(_Passwd_QNAME, String.class, null,
value);
    }

    @XmlElementDecl(namespace = "", name = "dataname")
    @XmlJavaTypeAdapter(CollapsedStringAdapter.class)
    public JAXBElement<String> createDataname(String value) {
        return new JAXBElement<String>(_Dataname_QNAME, String.class,
null, value);
    }

    @XmlElementDecl(namespace = "", name = "source")
    @XmlJavaTypeAdapter(CollapsedStringAdapter.class)
    public JAXBElement<String> createSource(String value) {
        return new JAXBElement<String>(_Source_QNAME, String.class, null,
value);
    }

    @XmlElementDecl(namespace = "", name = "name")
    @XmlJavaTypeAdapter(CollapsedStringAdapter.class)
    public JAXBElement<String> createName(String value) {
        return new JAXBElement<String>(_Name_QNAME, String.class, null,
value);
    }

    @XmlElementDecl(namespace = "", name = "driver")
    @XmlJavaTypeAdapter(CollapsedStringAdapter.class)
    public JAXBElement<String> createDriver(String value) {
```

```
        return new JAXBElement<String>(_Driver_QNAME, String.class, null,
value);
    }

    @XmlElementDecl(namespace = "", name = "frequency")
    public JAXBElement<BigInteger> createFrequency(BigInteger value) {
        return new JAXBElement<BigInteger>(_Frequency_QNAME,
BigInteger.class, null, value);
    }

    @XmlElementDecl(namespace = "", name = "type")
    @XmlJavaTypeAdapter(CollapsedStringAdapter.class)
    public JAXBElement<String> createType(String value) {
        return new JAXBElement<String>(_Type_QNAME, String.class, null,
value);
    }

    @XmlElementDecl(namespace = "", name = "user")
    @XmlJavaTypeAdapter(CollapsedStringAdapter.class)
    public JAXBElement<String> createUser(String value) {
        return new JAXBElement<String>(_User_QNAME, String.class, null,
value);
    }

    @XmlElementDecl(namespace = "", name = "url")
    @XmlJavaTypeAdapter(CollapsedStringAdapter.class)
    public JAXBElement<String> createUrl(String value) {
        return new JAXBElement<String>(_Url_QNAME, String.class, null,
value);
    }

    @XmlElementDecl(namespace = "", name = "fullname")
    public JAXBElement<String> createFullname(String value) {
        return new JAXBElement<String>(_Fullname_QNAME, String.class,
null, value);
    }

}
```

A.2.7 – Classe Query

```
package br.inpe.frame.model;

import java.util.List;
```

```
public class Query {
	private String name;
	private String sourcetable;
	private String initialdatetime;
	private String finaldatetime;
	private List<Attribute> attributes;

	public Query(){

	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getSourcetable() {
		return sourcetable;
	}

	public void setSourcetable(String sourcetable) {
		this.sourcetable = sourcetable;
	}

	public String getInitialdatetime() {
		return initialdatetime;
	}

	public void setInitialdatetime(String initialdatetime) {
		this.initialdatetime = initialdatetime;
	}

	public String getFinaldatetime() {
		return finaldatetime;
	}

	public void setFinaldatetime(String finaldatetime) {
		this.finaldatetime = finaldatetime;
	}

	public List<Attribute> getAttributes() {
		return attributes;
	}
```

```
	public void setAttributes(List<Attribute> attributes) {
		this.attributes = attributes;
	}



}
```

A.2.8 – Classe Resultset

```
package br.inpe.frame.model;

import java.util.List;

public class Resultset {
	private String name;
	private List<Column> columns;
	public Resultset(){

	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public List<Column> getColumns() {
		return columns;
	}
	public void setColumns(List<Column> columns) {
		this.columns = columns;
	}

}
```

A.2.9 – Classe Table

```
package br.inpe.frame.model;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
```

```
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlSchemaType;
import javax.xml.bind.annotation.XmlType;
import javax.xml.bind.annotation.adapters.CollapsedStringAdapter;
import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;


@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "field"
})
@XmlRootElement(name = "table")
public class Table {

    @XmlElement(required = true)
    protected List<Field> field;
    @XmlAttribute(name = "name", required = true)
    @XmlJavaTypeAdapter(CollapsedStringAdapter.class)
    @XmlSchemaType(name = "NCName")
    protected String name;

    public List<Field> getField() {
        if (field == null) {
            field = new ArrayList<Field>();
        }
        return this.field;
    }

    public String getName() {
        return name;
    }

    public void setName(String value) {
        this.name = value;
    }

}
```

A.2.10 – Classe Value

```
package br.inpe.frame.model;

public class Value {
```

```
	private String datetime;
	private Double value;
	public Value(String datetime, Double value) {
		super();
		this.datetime = datetime;
		this.value = value;
	}
	public String getDatetime() {
		return datetime;
	}
	public void setDatetime(String datetime) {
		this.datetime = datetime;
	}
	public Double getValue() {
		return value;
	}
	public void setValue(Double value) {
		this.value = value;
	}
}
```

A.3 – Código fonte da classe ConfigController

Classe que implementa o método responsável pela carga dos parâmetros definidos no arquivo de configuração de metadados.

```
package br.inpe.frame.controller;

import java.io.FileInputStream;
import java.io.IOException;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;

import br.inpe.frame.model.Datasources;

public class ConfigController {

	public ConfigController(){

	}
	public Datasources CarregarParams(){
		Datasources dts = null;
		JAXBContext context;
```

```
		try {
			context = JAXBContext.newInstance("br.inpe.frame.main");
			Unmarshaller unmarshaller = context.createUnmarshaller();
			dts = (Datasources) unmarshaller.unmarshal(new
FileInputStream("cdam-config.xml"));

		} catch (JAXBException | IOException e) {
			e.printStackTrace();
		}

		return dts;
	}
	public void MapearFontes(){

	}
	public void GerenciarConexoes(){

	}
}
```

A.4 – Diagrama de sequência do caso de uso Carregar Parâmetros

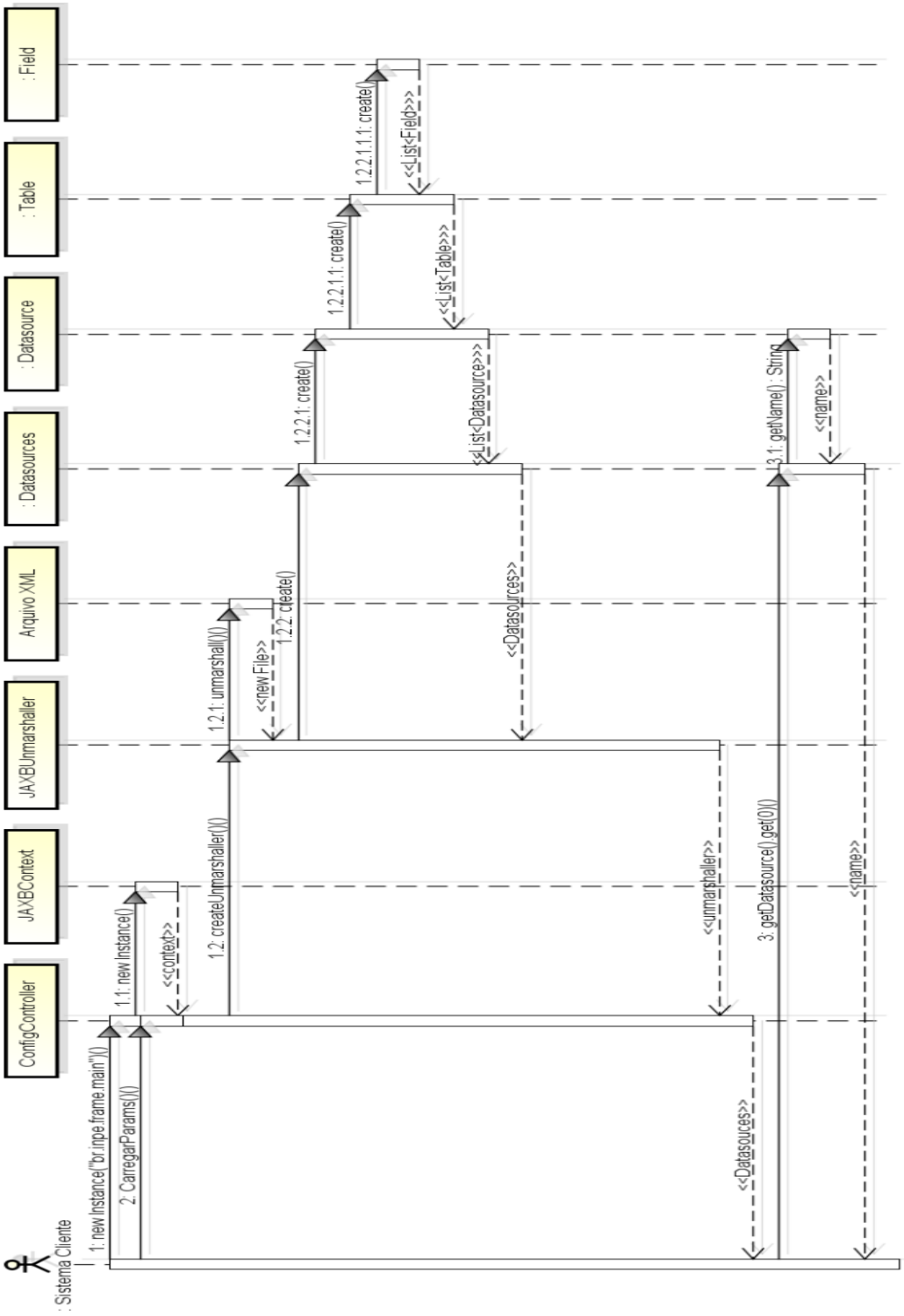
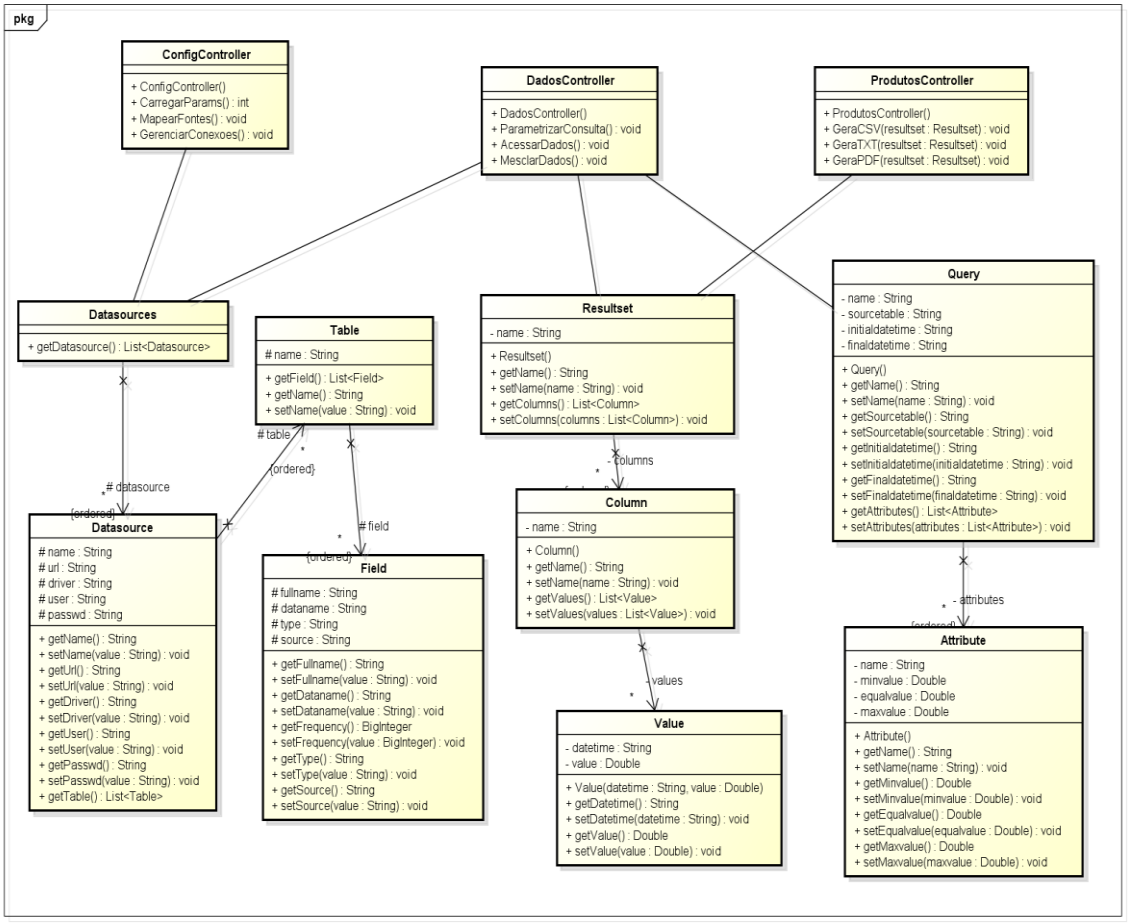


Figura A.2 - Diagrama de sequência (Carregar Parâmetros)

A.5 – Diagrama de classes Principal

Figura A.3 – Diagrama de classes Principal



A.6 – Código XML do arquivo de configuração dos metadados

```
<datasources>
        <datasource>
                <name>dsSismetPG</name>
                <url>jdbc:postgresql://150.163.133.25/sismet</url>
                <driver>org.postgresql.Driver</driver>
                <user>nome_usuario</user>
```

```
        <passwd>senha</passwd>
        <table name="tbl_dadosmet">
            <field>
                <fullname>Temperatura do AR</fullname>
                <dataname>tempar</dataname>
                <frequency>30</frequency>
                <type>Float</type>
                <source>Satelite</source>
            </field>
            <field>
                <fullname>Pluviometro</fullname>
                <dataname>pluvio</dataname>
                <frequency>30</frequency>
                <type>Float</type>
                <source>Satelite</source>
            </field>
        </table>
</datasource>
<datasource>
        <name>dsBDpcd</name>
        <url>jdbc:mysql://150.163.132.12:3306/bdpcd</url>
        <driver>com.mysql.jdbc.Driver</driver>
        <user>nome_usuario</user>
        <passwd>senha</passwd>
        <table name="dadostemp">
```

```
                <field>
                    <fullname>Temperatura do AR</fullname>
                    <dataname>temperatura</dataname>
                    <frequency>30</frequency>
                    <type>Float</type>
                    <source>satelite</source>
                </field>
            </table>
            <table name="dadospluvio">
                <field>
                    <fullname>Pluviometro</fullname>
                    <dataname>pluvio</dataname>
                    <frequency>30</frequency>
                    <type>Float</type>
                    <source>satelite</source>
                </field>
            </table>
        </datasource>
</datasources>
```