5
Especificação e Implementação

5.1.
Objetivo

Este capítulo endereça questões relativas à implementação do servidor WMS: os requisitos funcionais; os diagramas de especificação (casos de uso e de seqüência); e o projeto modular, descrevendo a organização do sistema em módulos e classes.

5.2.
Requisitos e Arquitetura

O servidor WMS deverá aceitar requisições e respondê-las via o protocolo HTTP, implementando os serviços WMS.

O servidor WMS deverá ser extensível para outros Web Services do consórcio OpenGIS. Para uma melhor implementação, a linguagem utilizada deve ser Java pois possui um amplo suporte a Web Services, através da tecnologia J2EE. Como a TerraLib é uma biblioteca em C++, existe uma aparente barreira tecnológica para que o sistema TerraLib (C++) se comunique com o servidor WMS (Java). Para transpor esta barreira foram estudadas duas possibilidades: JNI e CORBA.

O projeto deverá ser constituído de dois subsistemas bem definidos:

- WMSController:
    - Implementa os serviços WMS propriamente ditos, sendo responsável por tratar as requisições e devolver as respostas, tudo via HTTP;
    - Implementado em Java.
- WMSTerraLib:

- Implementa as operações necessárias aos serviços através de extensões à TerraLib.
- Implementado em C++.

A comunicação entre o WMSTerraLib e o WMSController será feita em CORBA, pelos seguintes motivos:

- CORBA permite que o WMSController e o WMSTerraLib fiquem em máquinas diferentes;
- CORBA permite um controle amplo e autônomo sobre os dois subsistemas.

A construção da interface CORBA não foi uma tarefa trivial e foi alvo de estudos, pois a forma de passagem das informações de um lado para outro poderia ser feita de algumas formas:

- Passagem das classes por valor do modelo de negócio de um lado para outro
- Passagem das informações já processadas, ou seja, as respostas das operações do WMS

Foi feita implementação simples da primeira opção (passagem das classes do modelo) e verificou-se um grande gasto de tempo. A construção da resposta às operações do WMS ainda não estava finalizada. Restava ainda gerar esta resposta no lado WMSControler para enviar a resposta ao cliente. Isto também demanda mais um tempo considerável.

Melhores resultados foram obtidos realizando a segunda opção (passagem das informações já processadas). Neste caso, foram criadas classes para implementar as operações do WMS estendendo a TerraLib. Assim, o que seria passado pela interface CORBA seria somente as respostas das operações do WMS, que podem ser resumidas a tipos simples como arranjos de bytes. Assim esta opção foi escolhida para ser usada pelo sistema.

Mesmo com esta escolha, achou-se conveniente manter o WMSControler ema Java ao invés de se fazer todo o sistema em C++ pelos seguintes fatos:

- Java possui um amplo suporte para criação de sistemas Web, principalmente no que se refere a Web Services
- A portabilidade de Java é um fator positivo, uma vez que servidores Web podem ser tanto Unix quanto Windows. Isto é válido mesmo que o WMSTerraLib tenha que, à principio, rodar em Windows pois como se trata de um sistema distribuído usando-se CORBA, os dois módulos podem rodar em máquinas diferentes com diferentes sistemas operacionais

A Figura 11 mostra a arquitetura criada.

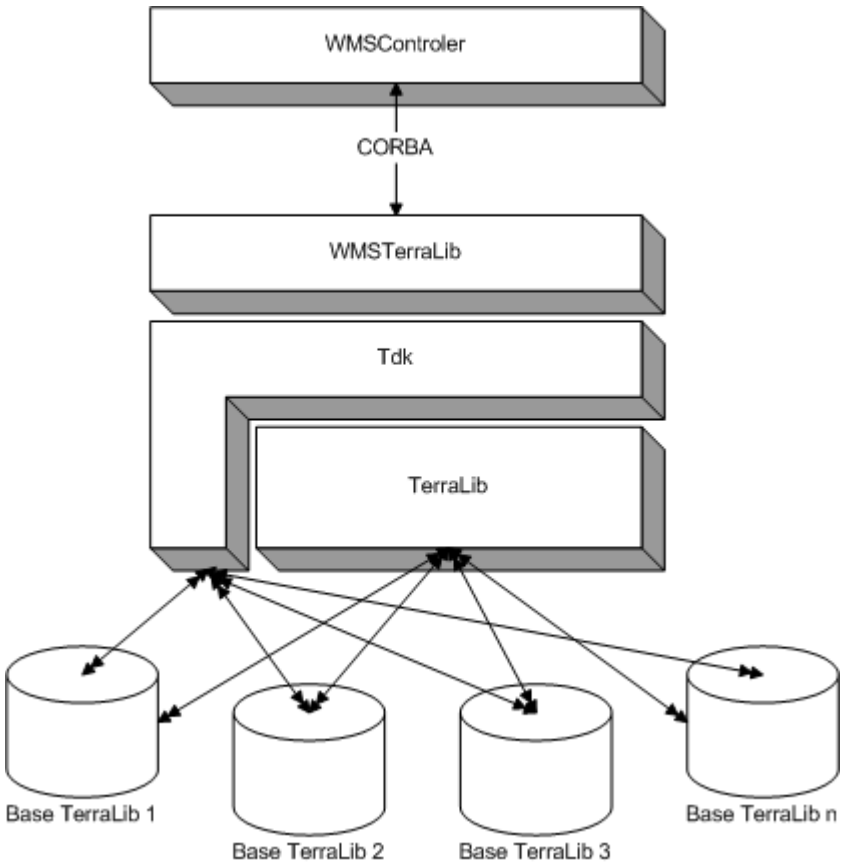

Figura 11 - Arquitetura

Brevemente, as operação GetCapabilities, GetMap e GetFeatureInformation são processadas no TDK/Terralib da seguinte forma.

Uma operação GetCapabilities provoca o envio de arquivos XML de Capabilities, que capturam o mapeamento entre "TeViews" e "TeThemes" e layers, conforme explicado na Seção 3.2.2. Estes arquivos são gerados a partir de metainformações sobre "TeViews" e "TeThemes" mantidas em memória pela TerraLib. Assim, uma requisição GetCapabilities pode ser processada de forma bastante rápida pois apenas provoca a recuperação de dados que já estão em memória.

Uma operação GetMap contém IDs de "TeThemes" e um bounding Box, entre outras informações. As geometrias dos objetos geográficos pertencentes aos "TeThemes" são utilizadas para gerar o mapa, que é então retornado para o cliente. Para criar o mapa, a TerraLib carrega todos os "TeThemes" requisitados, filtra os objetos geográficos que pertencem ao "TeTheme" e que estão na *bounding box* solicitado, insere as geometrias em uma estrutura de dados chamada *canvas* e gera o mapa a partir do canvas[26].

Uma operação GetFeatureInfo contém IDs de "TeThemes" e uma coordenada geográfica. Os atributos dos objetos contidos no "TeThemes" nesta coordenada são retornados para o cliente, como resposta da operação GetFeatureInfo. Para obter estes atributos, a TerraLib realiza consultas ao banco de dados.

5.3. Diagramas

5.3.1. Casos de Uso

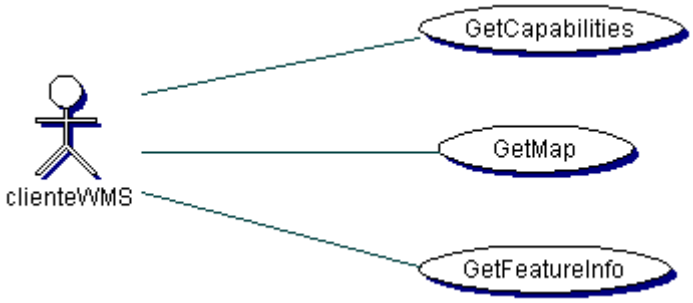

Figura 12 - Casos de Uso

GetCapabilities: permite ao cliente solicitar todas as metainformações sobre a base de dados, tais como o nome dos mapas contidos, projeção, escala, estilo possíveis dos mapas, formatos possíveis dos mapas (GIF, JPEG, PNG), formato das informações descritivas (FeatureInfo). Os parâmetros da operação são recebidos via o HTTP e a resposta deve ser um arquivo XML (text/xml) no mesmo protocolo. Segundo o protocolo WMS, esta operação deve ser implementada obrigatoriamente.

GetMap: retorna o mapa propriamente dito. Com esta operação, o cliente do Web Service pode requisitar um mapa em particular. Para isto basta usar as informações que foram retornadas pelo GetCapabilities. Os parâmetros da operação são recebidos via HTTP e a resposta deve ser um arquivo de imagem no formato solicitado, que pode ser um das opções: GIF, PNG ou JPEG. Esta operação também é obrigatória.

GetFeatureInfo: permite ao cliente realizar consultas nos mapas da base de dados. Os parâmetros da operação são recebidos via HTTP e a resposta deve ser um arquivo no formato solicitado que pode ser um dos seguintes: texto, XML ou

GML. Esta operação não é obrigatória segundo o protocolo WMS mas será implementada neste trabalho.

A Figura 13 ilustra como deve ser a comunicação entre o cliente e o servidor WMS. Para cada requisição deve ser retornada uma resposta correspondente, com um formato bem definido.

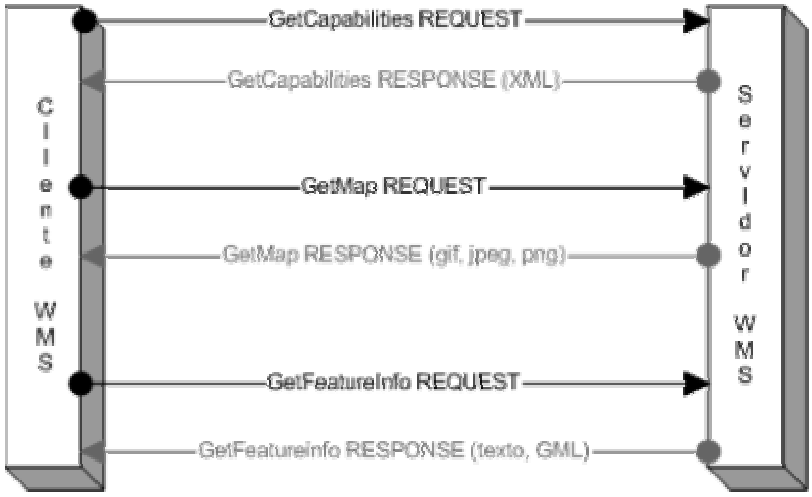

Figura 13 - Modelo do WMS

5.3.2. Diagramas de Seqüência

A primeira operação a ser apresentada é GetCapabilities (Figura 14). Esta operação compõe o arquivo XML *Capabilities*, onde estão dispostas para o cliente as informações sobre os dados do servidor WMS (*layers, bounding box, SRS,* entre outros).

Estas informações devem ser extraídas da árvore de temas da base TerraLib para compor o nodo *layer* do arquivo XML. As outras informações deste arquivo são referentes ao servidor WMS e podem ser extraídas de um arquivo de configuração do sistema como, por exemplo, *OnlineResource*, *KeywordList, Abstract*.

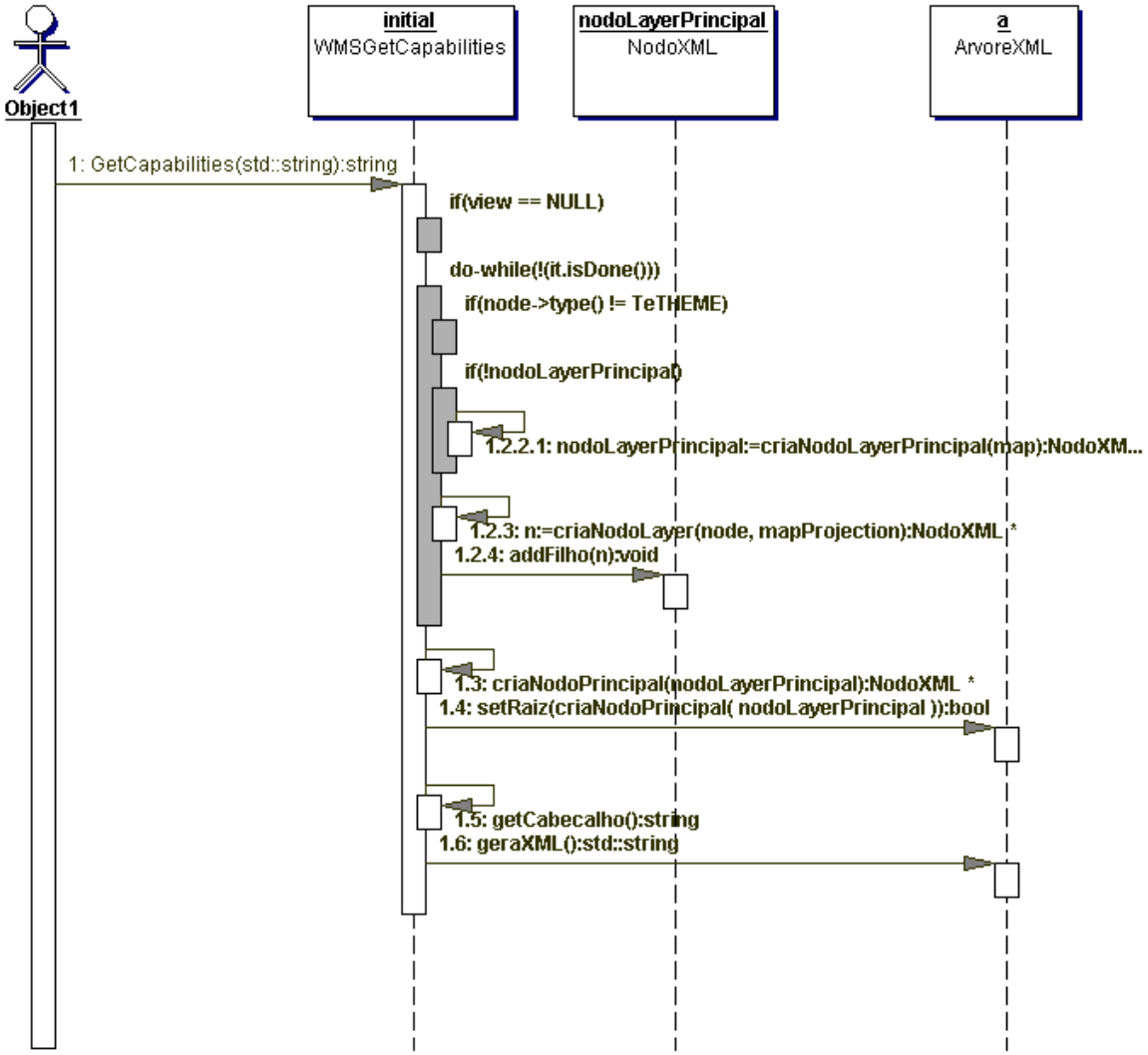

Figura 14 - Diagrama de Seqüência - GetCapabilities

A operação GetMap (Figura 15) é responsável por criar um imagem gráfica a partir da lista de *layers* enviada pelo cliente. Cada *layer* deve ser desenhado em um *canvas* e, a partir deste, deve ser gerado o mapa a ser retornada ao cliente.

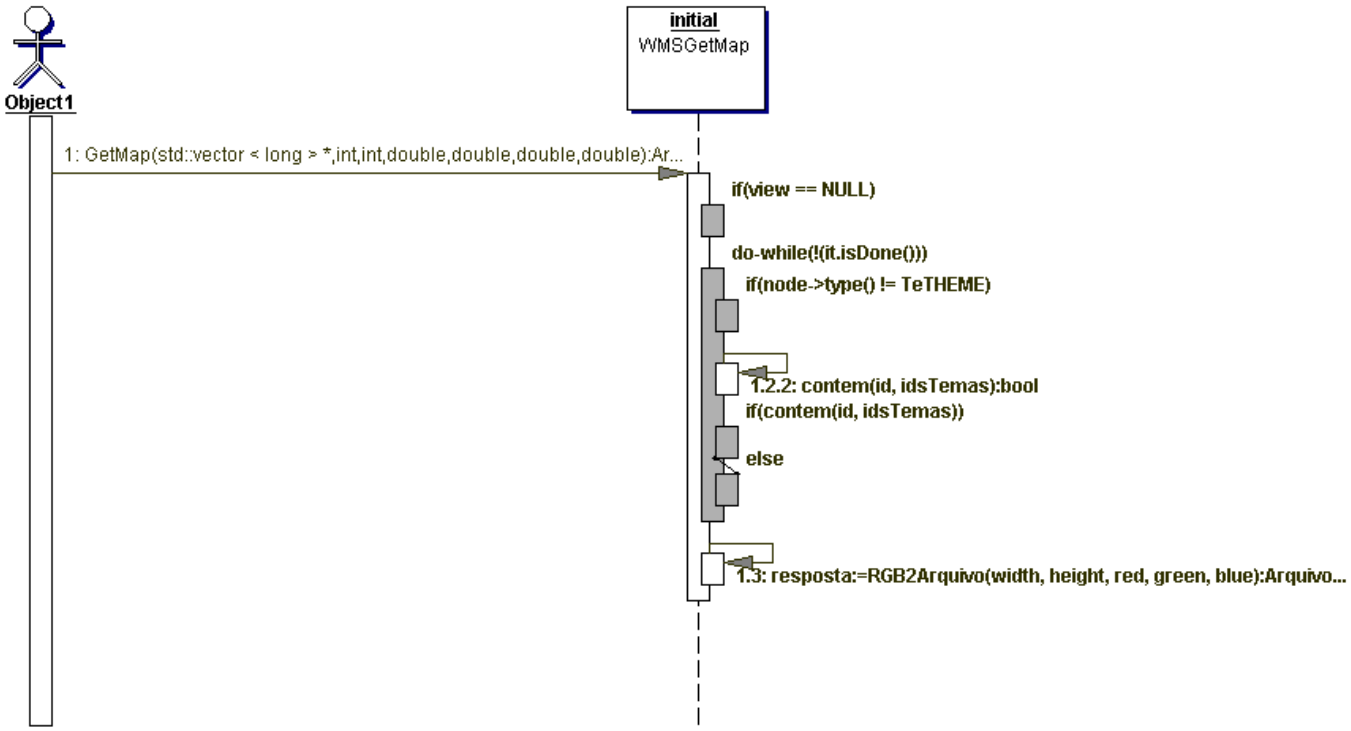

Figura 15 - Diagrama de Seqüência - GetMap

O GetFeatureInfo (Figura 16) implementa a consulta aos atributos do mapa. Dada a lista de *layes* e as coordenadas de mundo, os atributos dos objetos situados nestas coordenadas devem ser retornados.

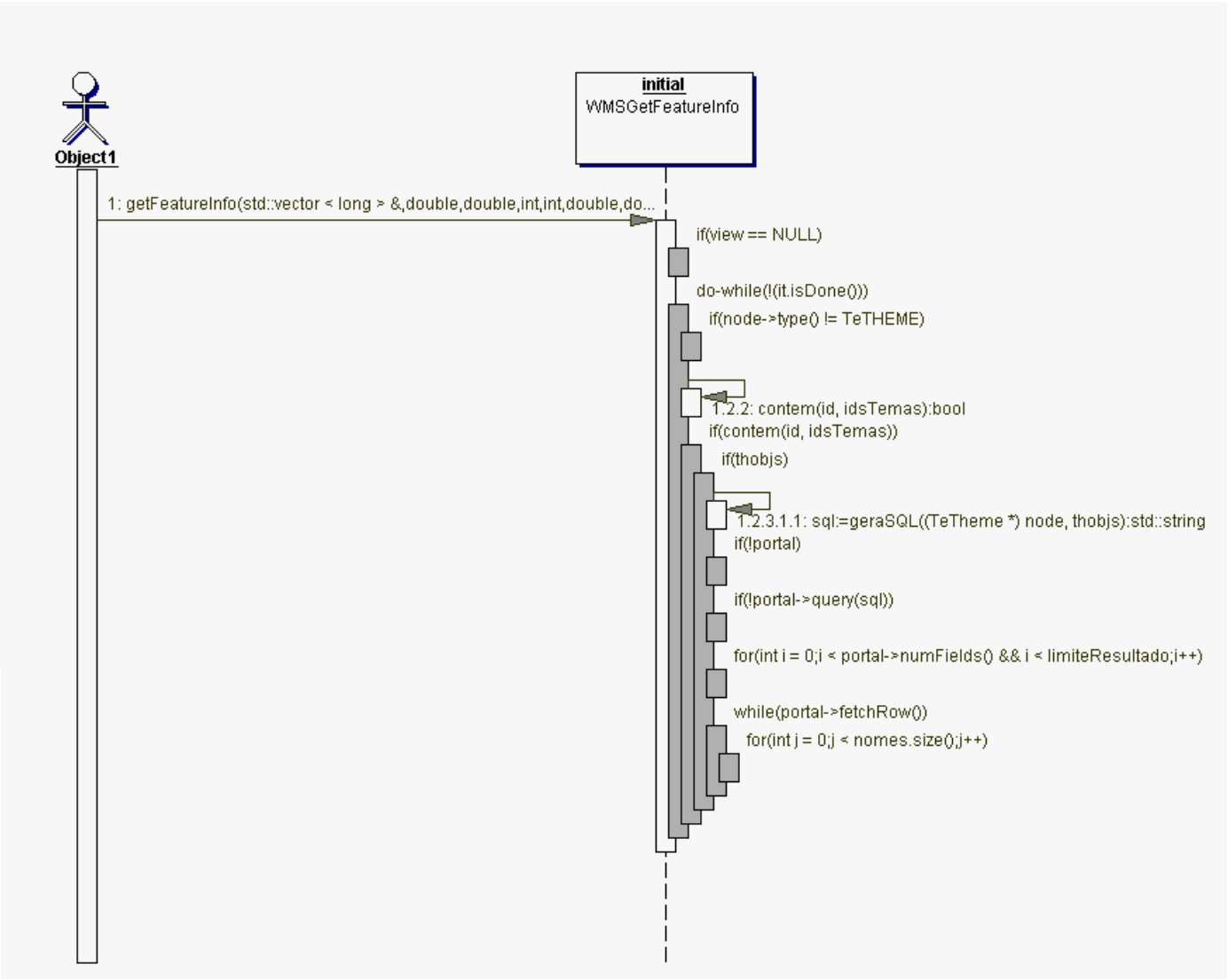

Figura 16 - Diagrama de Seqüência - GetFeatureInfo

5.4. Projeto Modular

5.4.1. Diagrama de Pacotes

Como disposto na Figura 11, o sistema é constituído de dois subsistemas que, por sua vez, estão organizados em pacotes (Figura 17).

O módulo WMSTerraLib é organizado em apenas um pacote, que leva o mesmo nome do módulo.

O WMSController é implementado por três pacotes:

- wms: recebe as requisições e escreve e envia as respostas;
- OGCWebService: implementa um *framework* para os Web Services OpenGIS, sendo instanciado para o WMS; e
- wmsCORBA: implementa as classes da IDL, visando estabelecer comunicação com o WMSTerraLib, repassando os parâmetros das operações e recebendo as devidas respostas.

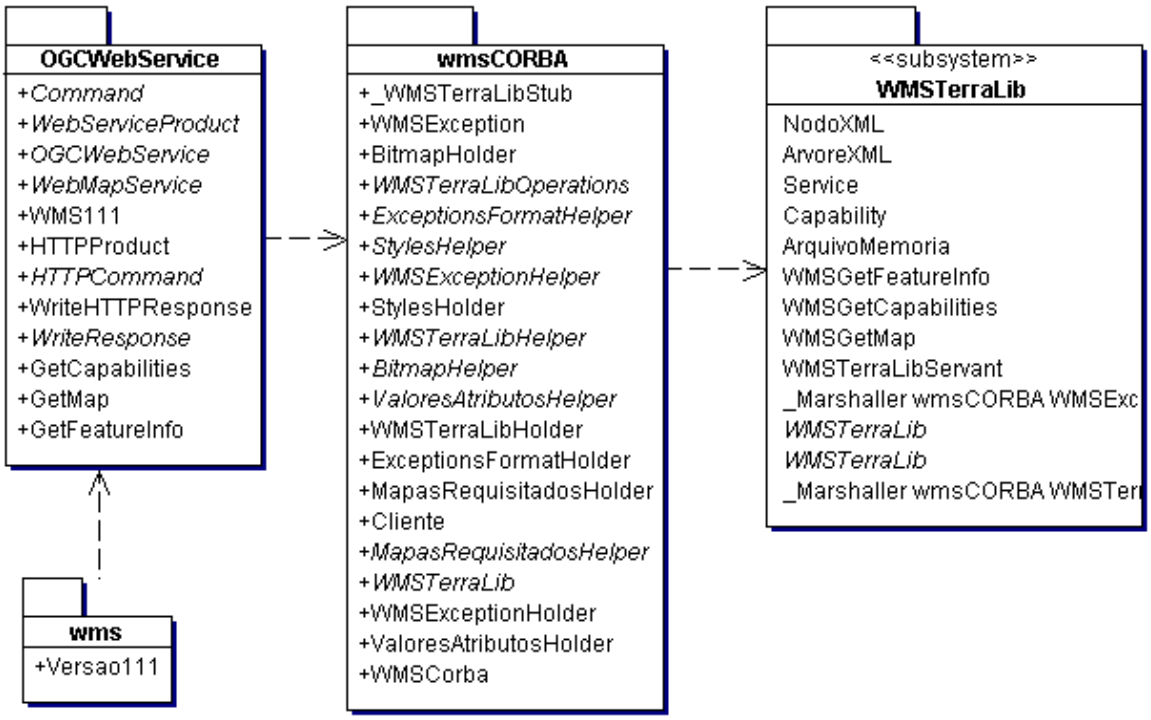

Figura 17 - Pacotes

5.4.2. Diagramas de Classes

O pacote wms (Figura 18) visa implementar o controle das requisições e respostas HTTP para o protocolo WMS. Ele é composto por uma única classe ("Versao111"), que estende a classe servlet.

A classe “Versao111” recebe as requisições e as encaminha para a instância do *framework* OGCWebService (Figura 19).

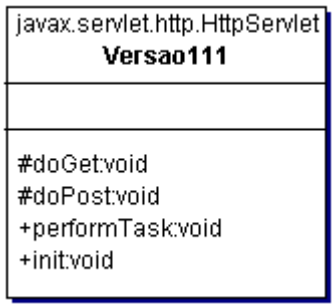

Figura 18 - Pacote wms

A missão do OGCWebService é atender aos Web Services definidos pelo OpenGIS. Ele proporciona, através do padrão *Factory Method* [28], uma metodologia para cada Web Service implemente suas operações. O controle destas operações é realizado usando o padrão *Command* [28], o que permite tratar todas as operações dos Web Services de forma homogênea. A forma de retornar o resultado destas operações é tratado de forma genérica, usando o padrão *Abstract Factory* [28] e o *Factory Method*.

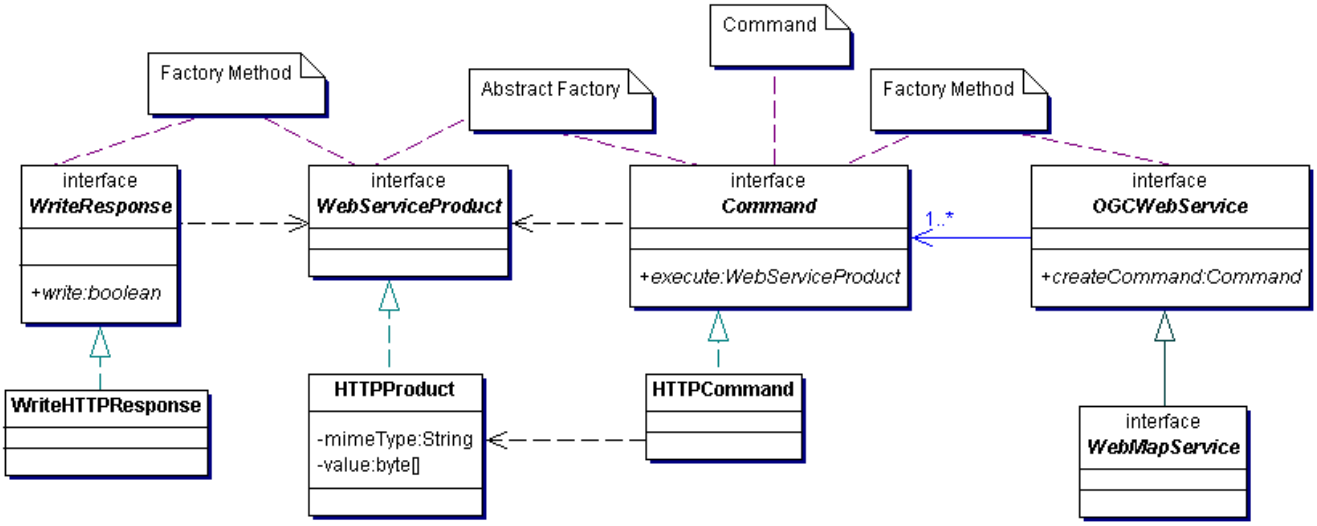

Figura 19 - *Framework* OGCWebService

A instância do *framework* para os serviços WMS é mostrada na Figura 20. A classe "WMS111" gera cada uma de suas operações usando as classes "GetCapabilities", "GetMap", "GetFeatureInfo".

Estas operações geram suas respostas na forma da classe "HTTPProduct" e as insere num pacote do protocolo HTTP através da classe "WriteHTTPResponse". Assim a classe "Versao111" (Figura 18) encaminha a resposta para o cliente.

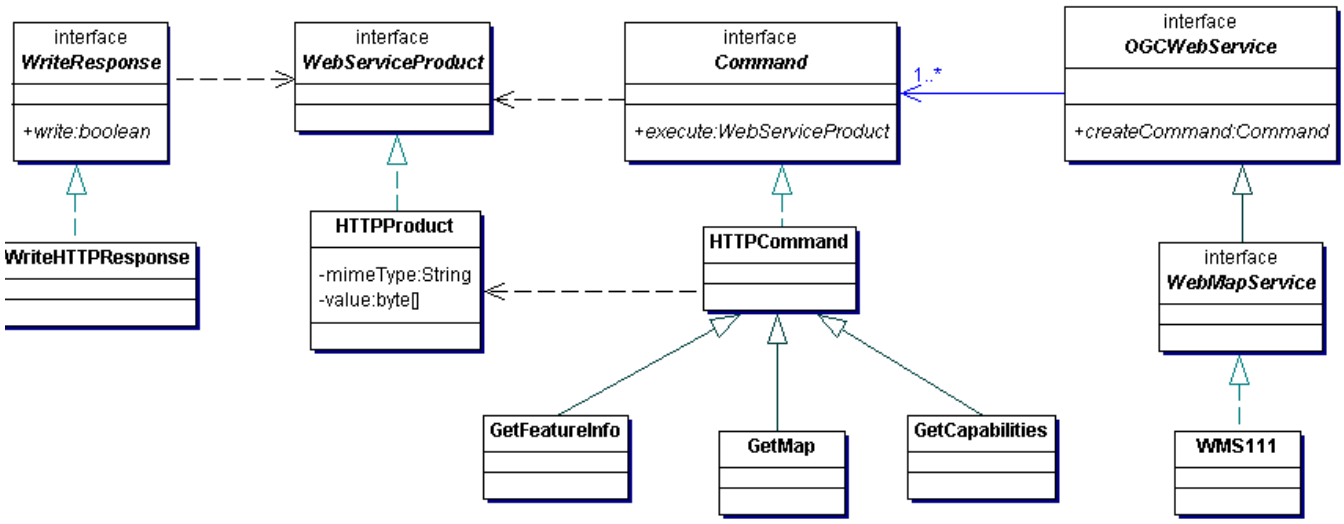

Figura 20 - Instância do *Framework* para o WMS

As classes "GetCapabilities", "GetMap", "GetFeatureInfo" acessam uma fachada (do padrão *Façade* [28]), chamada "WMSCorba", contida no pacote "wmsCORBA". Esta fachada encapsula o acesso via CORBA às funções que estendem a TerraLib. A Figura 21 mostra o acesso a esta fachada.

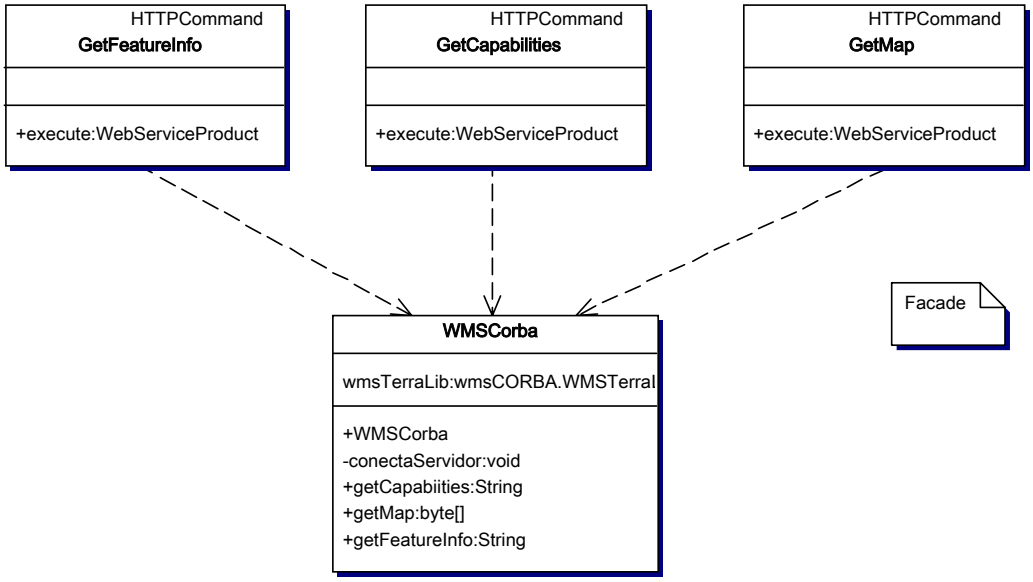

Figura 21 - Fachada de Acesso a wmsCORBA

O pacote "wmsCORBA" (Figura 22) proporciona acesso via CORBA às funcionalidades da TerraLib que implementam as operações do protocolo WMS. Este pacote oferece a classe fachada de acesso "WMSCorba".

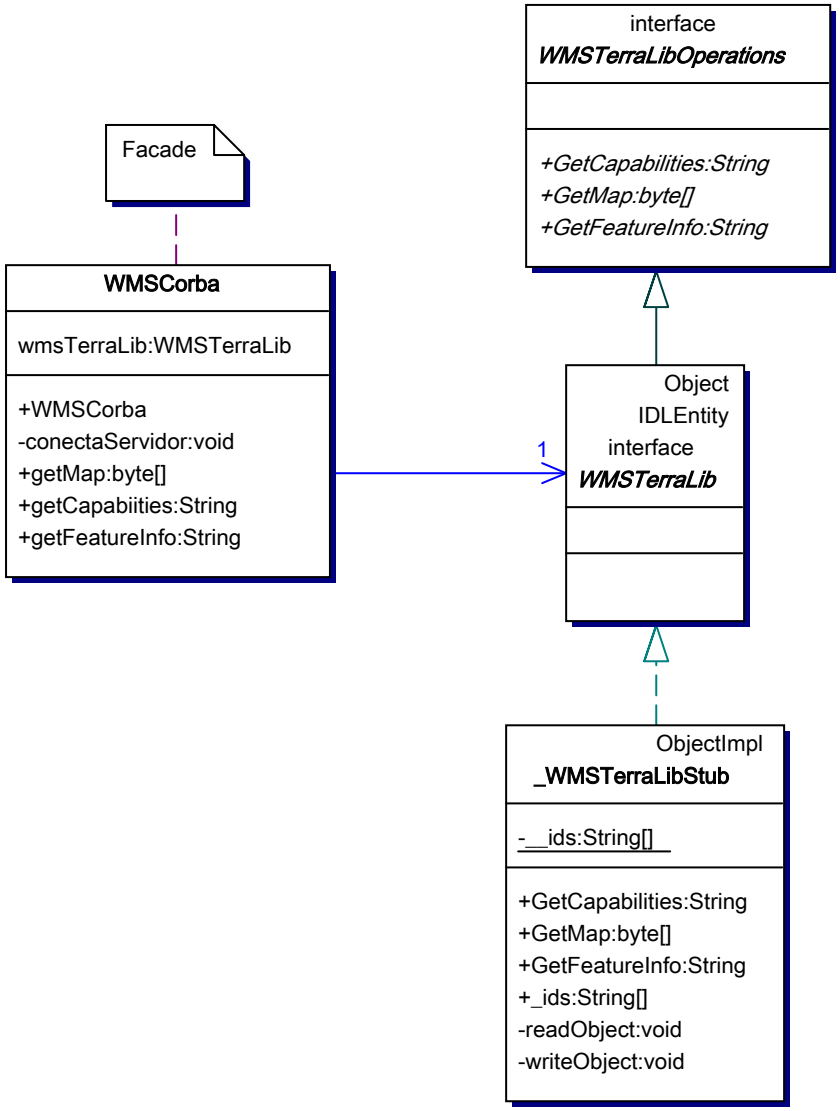

Figura 22 - Pacote wmsCORBA

Para tratar as exceções que podem ocorrer no nível do modelo dos dados, foram criadas classes que descrevem a exceção gerada (Figura 23). A classe principal é "WMSException", que contém as informações sobre a exceção; as classes com sufixo "*Holder*" e "*Helper*" oferecem as funcionalidades necessárias à implementação em CORBA.

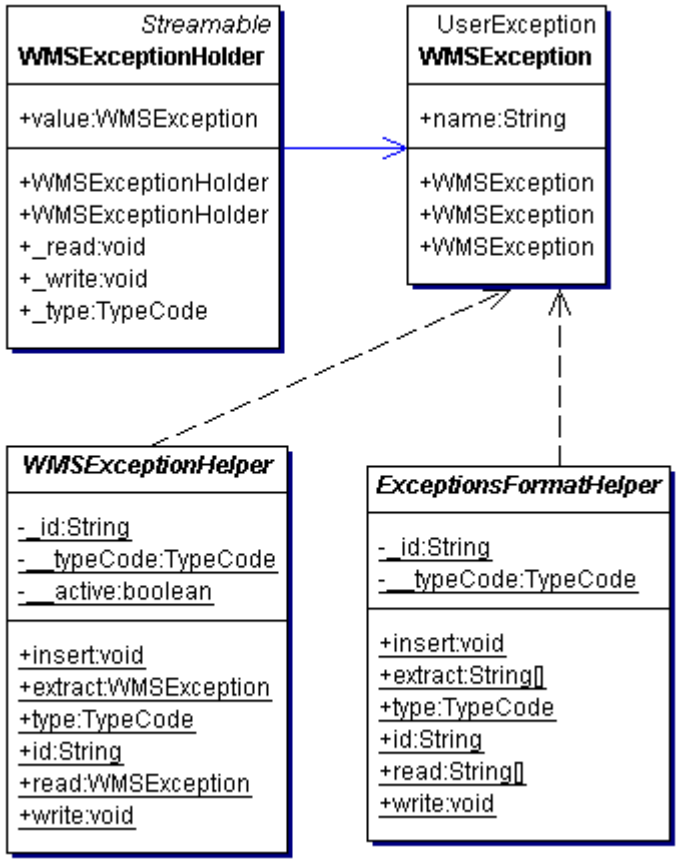

Figura 23 - wmsCORBA - Exceções

A Figura 24 mostra a extensão da TerraLib para suportar o núcleo das operações do protocolo WMS. A instância da classe "WMSTerraLibServant" é o objeto CORBA principal do subsistema WMSTerraLib. É ele quem retorna, através do pacote "wmsCORBA", os arquivos texto e as imagens gráficas para que o pacote "wmsSERVER" os encaminhe para os devidos clientes.

Para implementar cada operação do protocolo WMS, foi criada uma classe que acessa o banco de dados, recebe os parâmetros e retorna a resposta da operação. Estas classes são "WMSGetCapabilities", "WMSGetMap" e "WMSGetFeatureInfo".

A classe "WMSGetCapabilities" é responsável por criar o arquivo XML Capabilities. A classe "WMSGetMap" cria um arquivo de imagem em memória, contendo o mapa requisitado. Finalmente, a classe "WMSGetFeatureInfo" gera o arquivo texto com os atributos dos *layers* requisitados.

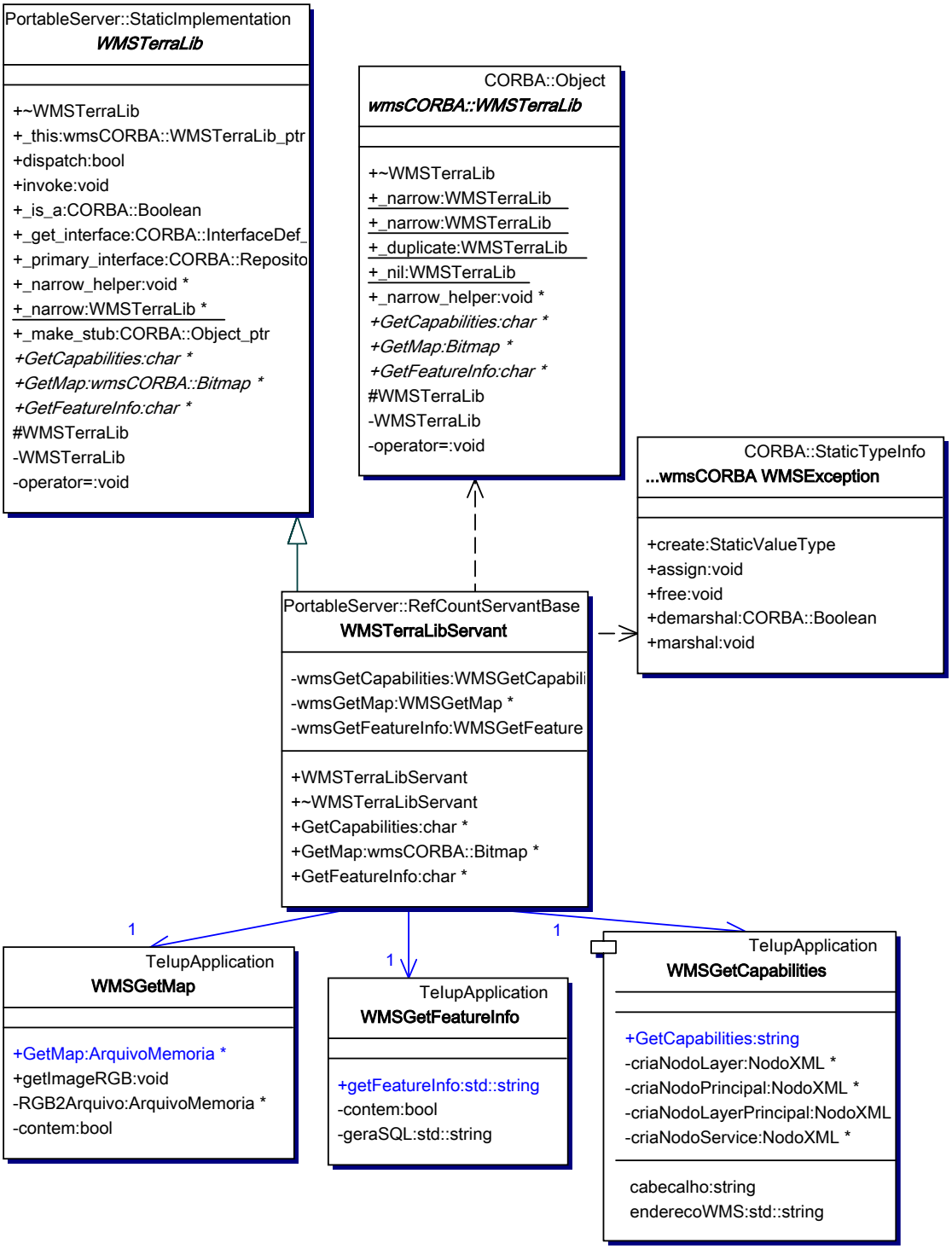

Figura 24 - Pacote WMSTerraLib - controle CORBA

A Figura 25,contém as classes auxiliares e a classe principal "WMSGetCapabilites", responsáveis pela criação do XML Capabilities. As classes auxiliares servem para dividir o processo de criação do XML em blocos e

depois uni-los para gerar o arquivo principal, a partir de uma árvore ("ArvoreXML").

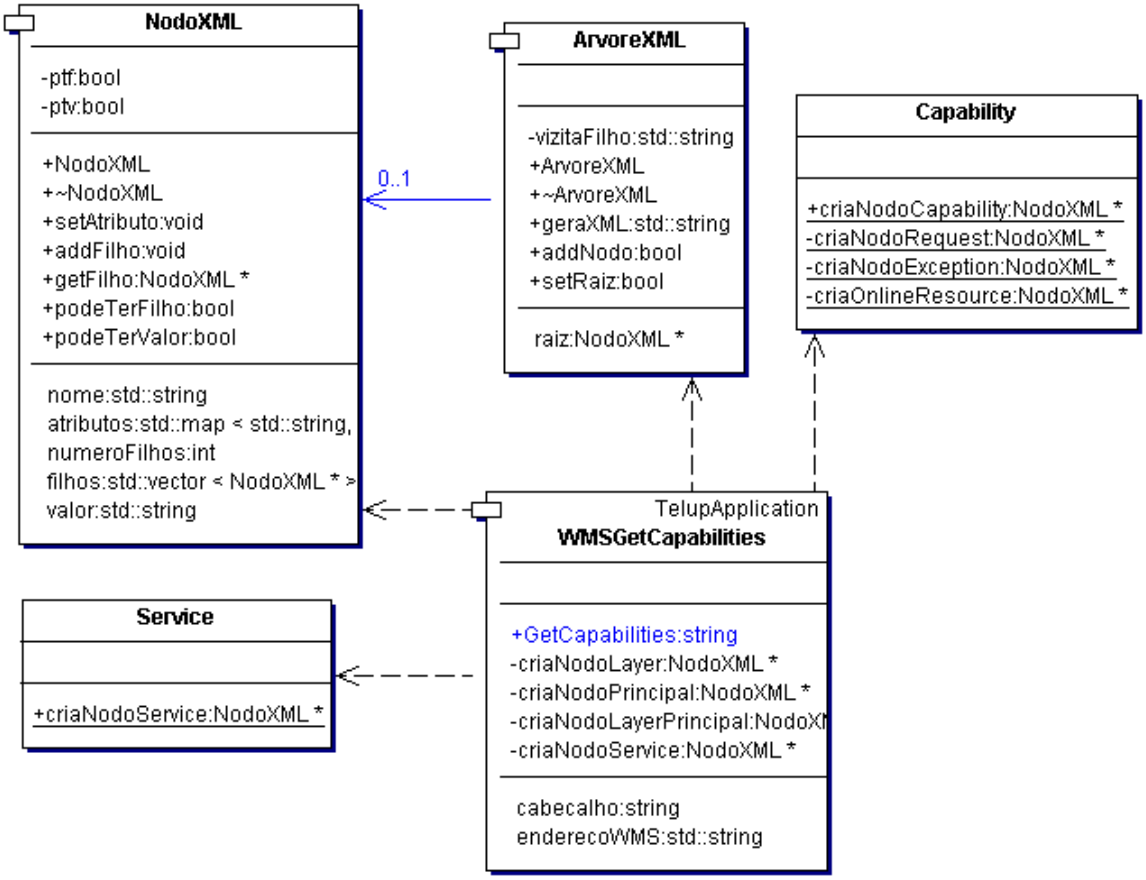

Figura 25 - Pacote WMSTerraLib - GetCapabilities

A operação GetMap (Figura 26) foi dividida em duas etapas: geração do mapa num *canvas* e geração de um arquivo gráfico em memória a partir deste *canvas*. Estas duas etapas estão implementadas na classe "WMSGetMap".

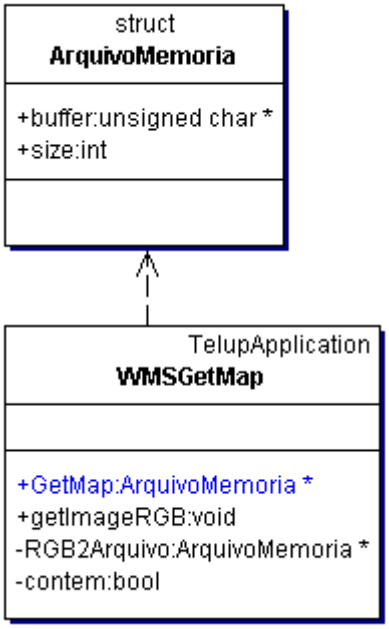

Figura 26 - Pacote WMSTerraLib - GetMap

A operação GetFeatureInfo (Figura 27) também possui duas etapas bem definidas: consulta às informações sobre os *layers* contidas na base e agrupamento das informações em um arquivo. Para implementar estas funcionalidades foi criada a classe "WMSGetFeatureInfo".

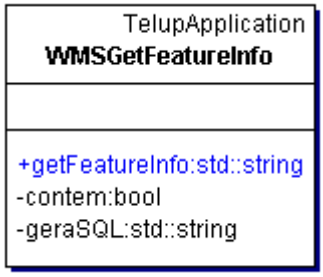

Figura 27 - Pacote WMSTerraLib - GetFeatureInfo

5.5. Configurações do Sistema

O módulo WMSController baseia-se em Java servlets. Os testes realizados usaram o Apache Tomcat 4.1.28.

O módulo WMSTerraLib é um pouco mais complexo que o anterior e exige algumas observações.

Como o sistema é baseado em CORBA, é necessário utilizar funcionalidades de um ORB. O ORB escolhido foi o MICO 2.3.7 pois permite gerar uma versão ESTÁTICA da sua biblioteca, utilizando o compilador Microsoft Visual C++ versão 6. O uso deste compilador é imposto pelo uso do TDK/TerraLib.

As outras bibliotecas necessárias para compilar o projeto são:

- Terralib.lib (*)
- Ijl15.lib
- Zlibstat.lib
- Tiff.lib
- Teiupapplication.lib (*)
- Im.lib versão 3(*)
- Iup.lib (*)
- Iupcontrols.lib (*)
- Iupecontrols.lib (*)
- Iupgl.lib (*)
- Cd.lib (*)
- Cdgdiplus.lib (*)
- Cdiup.lib (*)
- Xy.lib (*)

(*) bibliotecas de responsabilidade do TecGraf/PUC-Rio.

É imprescindível que todas as bibliotecas estejam na versão Release Multithreaded.

A STL usada no projeto também deve ser comum. Todas as libs usadas nesta versão do software usam a STLPort versão 4.5.3.

5.6. Utilização do Sistema

O procedimento abaixo indica com inicializar e executar o subsistema WMSTerralib (que deve ser iniciado primeiro):

- Obtenha e instale os arquivos WMSTerraLib.exe, temrsid.dll e ijl15.dll.
- Edite o arquivo base.dat, inserindo o caminho e o nome do arquivo que contém a base de dados TerraLib no formato do sistema ACCESS (por exemplo, c:\bases\brasil.mdb).
- Execute WMSTerraLib.exe

O procedimento abaixo indica como instalar o wmsSERVER no Apache Tomcat 4.1.28:

- Descompacte o arquivo zip no diretório c:\wmsTerralib
- Crie o contexto do cliente no arquivo server.xml do Tomcat (localizado em $TOMCAT/conf/):

```
<!-- TERRALIB WMS-server -->
<Context path="/wmsterralib"
        docBase="c:/ wmsTerralib"
        crossContext="false"
        debug="0"
        reloadable="true" >
</Context>
```

- Inicie o Tomcat

Para acessar o servidor implementado, naturalmente é necessário um cliente WMS. Atualmente, muitos softwares de GIS implementam um cliente WMS, diretamente ou através de plug-ins.

O cliente usado para testes foi implementado pela Intergraph e encontra-se online no endereço www.wmsviewer.com/main.asp. Para utilizá-lo:

- registre o servidor através de sua URL, por exemplo http://menzel.tecgraf.puc-rio.br:8080/marcWMS/wms.
- uma vez registrada a URL, o cliente chamará o serviço GetCapabilities do servidor.

5.7. Medidas de Desempenho

O desempenho da implementação como um todo e das etapas principais de processamento foi avaliado através de uma série de experimentos (Figura 28).

Antes de prosseguir, sugerimos ao leitor relembrar como os serviços WMS são processados na TerraLib (ver final da Seção 5.2).

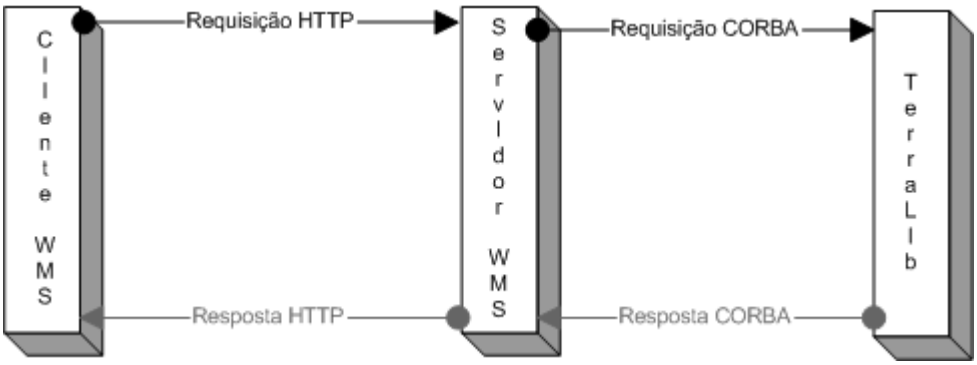

Figura 28 - Etapas do Processo de Medição

Mais precisamente, os experimentos concentram-se em medir (os tempos são em milisegundos):

- *tempo total de processamento* de uma requisição, ou seja, tempo gasto na requisição HTTP, requisição CORBA, processamento na TerraLib e respostas CORBA e HTTP;
- *tempo de processamento na TerraLib,* incluindo o tempo de processamento no TDK;
- *tempo da resposta CORBA;*
- *tempo da resposta HTTP*.

Todos os experimentos utilizaram 11 *layers* de um banco de dados Terralib, armazenado em MS ACCESS. A Tabela 5 apresenta dados sobres o armazenamento dos layers.

Tabela 5 - *Layers* utilizados nos experimentos.

Nome	Descrição	Tabela Interna	Número de Linhas [1]	Tamanho [2] (KB)
Limite Municipal	Poligonos	LimMunicipal	1	112
PEZR-2em vigor	Polígonos	PlanEspZoneRuidoNivel2	1	36
PEZR-1em vigor	Polígonos	PlanEspZoneRuidoNivel1	1	36
Curvas de Nível	Linhas	Cnivel	376	660
Uso do Solo	Polígonos	UsoSolo	762	452
Favelas	Polígonos	Favelas	7	36
Setores Censitários	Polígonos	SetCensit00	1230	820
Densidade Demográfica	Polígonos	SetCensit00	1230	820
Bairros	Polígonos	Bairros	57	328
Quadras	Linhas	Quadras	18172	10247
Eixos de Logradouros	Linhas	Eixos_Logradouros	3996	2792
Corpos de Água	Polígonos	CorposDagua	98	356
Equipamentos	Pontos	Equipamentos	30	24
Pontos de Monitoramento	Pontos	PtosMonitorados	10	24
Hidrografia	Linhas	Hidrografia	420	252
Plano de Proteção	Polígonos	PlanoProtecao	14	36
Curva de Ruído 2-Ano 2000	Polígonos	CurvaRuido2	1	44
Curva de Ruído 1-Ano 2000	Polígonos	CurvaRuido1	1	36
Área Patrimonial	Polígonos	Apatrimonial	5	52
Instalações do Aeroporto	Linhas	Inst_Aeroporto	9392	2894

Notas:

(1) O número de linhas de cada tabela indica o número de geometrias distintas do *layer*.

(2) O tamanho de um layer L refere-se a uma estimativa do número de bytes ocupado pela tabela T[L] do ACCESS que armazena as geometrias de L. Este dado é uma boa métrica do esforço necessário para recuperar L do banco de dados em ACCESS. O espaço ocupado por T[L] não é diretamente disponível pelo ACCESS. Ele foi obtido exportando-se T[L] para um novo banco de dados vazio e subtraindo-se do total do novo

banco o espaço ocupado por um banco de dados ACCESS totalmente vazio.

O primeiro experimento caracterizou-se pelo cliente e servidor rodarem na mesma máquina e envolveu requisições de 11 *layers,* que são combinados em um mapa de 640 por 480 *pixels*, gerado como uma imagem JPEG. Os resultados estão apresentados na Tabela 6.

Tabela 6 - Resultado do Experimento 1.

	GetCapabilities		GetMap		GetFeatureInfo	
Tempo total de proc.	12,30		8351,30		31,60	
Tempo de proc. na TerraLib	3,20	26,02%	8334,20	99,80%	18,70	59,18%
Tempo da resposta CORBA	4,50	10,57%	7,10	0,09%	3,20	10,13%
Tempo da resposta HTTP	4,60	37,40%	10,00	0,12%	9,70	30,70%

O segundo experimento caracterizou-se também pelo cliente e servidor rodarem na mesma máquina e envolveu requisições de 11 *layers,* que são combinados em um mapa de 320 por 240 pixels, também gerado como uma imagem JPEG. Esta mudança, em relação em primeiro experimento, implica que nem todos os elementos dos *layers* precisam ser exibidos. Os resultados estão apresentados na

Tabela 7.

Tabela 7 - Resultado do Experimento 2

	GetCapabilities		GetMap		GetFeatureInfo	
Tempo total de proc.	11,00		8278,90		85,80	
Tempo de proc. na TerraLib	3,20	29,09%	8272,10	99,92%	70,20	81,82%
Tempo da resposta CORBA	2,10	19,09%	1,00	0,01%	4,90	5,71%
Tempo da resposta HTTP	5,70	51,82%	5,80	0,07%	10,70	12,47%

O terceiro experimento caracterizou-se pelo cliente e servidor rodarem em máquinas diferentes, embora na mesma intranet, e envolveu requisições de 11 *layers,* que são combinados em um mapa de 640 por 480 *pixels,* também gerado como uma imagem JPEG. Os resultados estão apresentados na Tabela 8.

Tabela 8 - Resultado do Experimento 3

	GetCapabilities		GetMap		GetFeatureInfo	
Tempo total de proc.	19,0		8389,4		34,0	
Tempo de proc. na TerraLib	5,2	27,37%	8379,6	99,88%	18,7	55,00%
Tempo da resposta CORBA	4,5	23,68%	3,1	0,04%	6,4	18,82%
Tempo da resposta HTTP	9,3	48,95%	6,7	0,08%	8,9	26,18%

O quarto experimento caracterizou-se pelo cliente e servidor rodarem em máquinas diferentes, embora na mesma intranet, e envolveu requisições de 11 *layers,* que são combinados em um mapa de 320 por 240 pixels, também gerado como uma imagem JPEG. Os resultados estão apresentados na Tabela 9.

Tabela 9 - Resultado do Experimento 4

	GetCapabilities		GetMap		GetFeatureInfo	
Tempo total de proc.	17		8205		84	
Tempo de proc. na TerraLib	3,2	18,82%	8192,3	99,85%	64,1	76,31%
Tempo da resposta CORBA	6,1	35,88%	2,7	0,03%	7,6	9,05%
Tempo da resposta HTTP	7,7	45,29%	10	0,12%	12,3	14,64%

O tamanho total, em bytes, de cada resposta é o seguinte:

- GetCapabilities: 7284 bytes em todos os experimentos
- GetMap: 109547 bytes nos experimentos 1 e 3 e 22584 bytes nos experimentos 2 e 4
- GetFeatureInfo: 137 bytes em 1 e 3 e 83 bytes nos experimentos 2 e 4

Os mapas resultantes das requisições estão na Figura 29 e na Figura 30.

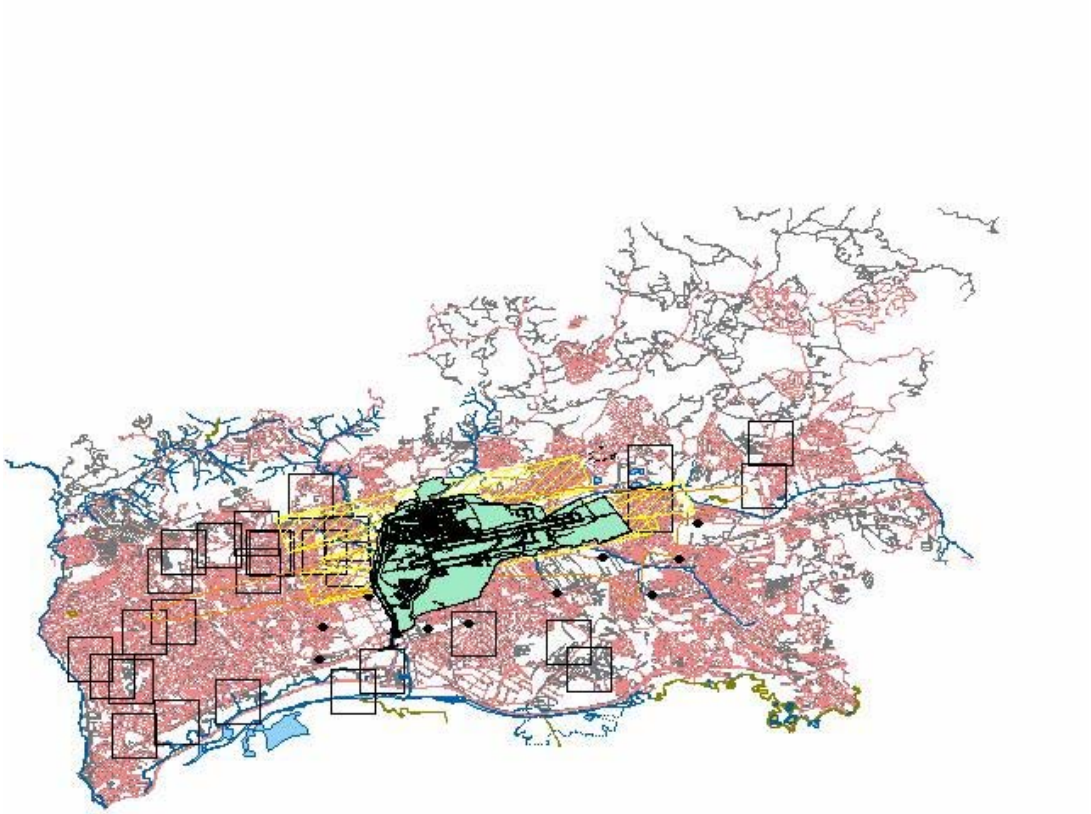

Figura 29 - Mapa Resultante dos Experimentos 1 e 3.

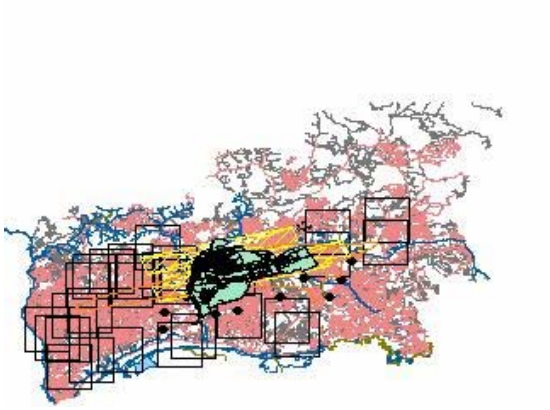

Figura 30 - Mapa Resultante dos Experimentos 2 e 4.

Podemos concluir que, em GetCapabilities, a maior parte do tempo é gasta para transmitir os dados via CORBA e através do protocolo HTTP.

Em GetMap, fica claro em todos os casos que o custo maior está na geração do mapa. Este processamento é feito através de funções implementas usando Tdk/TerraLib.

Análogo ao ocorrido em GetMap, em GetFeatureInfo a maior parcela do tempo foi gasta no lado do servidor C++/CORBA, onde as funções são implementadas usando Tdk/TerraLib.

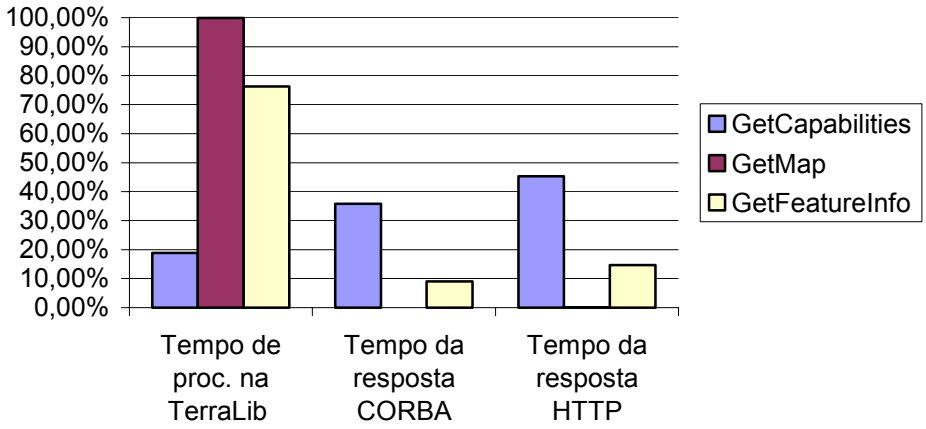

Figura 31 - Fatias de Tempo Médias

Como mostrado na Figura 31, a maior parte do tempo é destinada ao processamento na TerraLib. O tempo gasto com HTTP e CORBA é mínimo se comparado ao tempo gasto pela TerraLib.

O tempo de processamento na TerraLib pode ser diminuído se for o *canvas* puder ter uma política de "ligar e desligar temas" e se houver um cache de *canvas*. Uma outra forma de melhorar o desempenho é adotando a política de multi-resolução das informações, uma vez que nem todos os pontos geográficos são sempre visualizados em qualquer que seja a resolução [29].