UERJ

Dissertação de Mestrado em Engenharia de Computação

UMA ARQUITETURA PARA ACESSO REMOTO DA INFORMAÇÃO GEOGRÁFICA BASEADA EM OBJETOS DISTRIBUÍDOS PARA AMBIENTE MARÍTIMO

Autor: Luiz Fernando Yuan Gouvêa

Orientador: João Araújo Ribeiro

Programa de Pós-Graduação em Engenharia de Computação – Área de Concentração em Geomática

Outubro/2002

Faculdade de Engenharia

UMA ARQUITETURA PARA ACESSO REMOTO DA INFORMAÇÃO GEOGRÁFICA BASEADA EM OBJETOS DISTRIBUÍDOS PARA AMBIENTE MARÍTIMO

Luiz Fernando Yuan Gouvêa

Dissertação submetida ao corpo docente da Faculdade de Engenharia da Universidade do Estado do Rio de Janeiro – UERJ, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Computação.

Orientador: João Araújo Ribeiro

Programa de Pós-Graduação em Engenharia de Computação – Área de Concentração em Geomática

Rio de Janeiro
Outubro de 2002

GOUVÊA, LUIZ FERNANDO YUAN

Uma Arquitetura para Acesso Remoto da Informação Geográfica Baseada em Objetos Distribuídos para Ambiente Marítimo [Rio de Janeiro] 2002

v, 109 p. 29,7 cm (FEN/UERJ, M.Sc., Engenharia da Computação – Área de Concentração Geomática, 2002)

Dissertação - Universidade do Estado do Rio de Janeiro – UERJ

1. Geomática
2. Objetos Distribuídos

I. FEN/UERJ II. Título (série)

FOLHA DE JULGAMENTO

Título: Uma Arquitetura para Acesso Remoto da Informação Geográfica Baseada em Objetos Distribuídos para Ambiente Marítimo.

Candidato: Luiz Fernando Yuan Gouvêa

Programa: Pós-Graduação em Engenharia de Computação – Área de Concentração em Geomática

Data da defesa: 22/10/2002

Aprovada por:

_____
João Araújo Ribeiro, Dr, UERJ.

_____
Orlando Bernardo Filho, DSc, UERJ.

_____
Renato Fontoura de Gusmão Cerqueira, DSc,
PUC-Rio.

AGRADECIMENTOS

Agradeço à minha esposa, Cátia, pela sua compreensão constante nas horas extras de trabalho.

Ao meu pequenino filho, Leonardo, que na sua inocência fez com que me empenhasse mais ainda na obtenção dos meus objetivos.

Ao meu orientador João Araújo Ribeiro, pela orientação, cobrança e confiança depositada em mim.

Aos meus chefes de trabalho, diretos e indiretos, que permitiram a utilização de horas de trabalho na obtenção do mestrado.

E, principalmente, a DEUS que é o regente maior de nossas vidas.

Resumo da Dissertação apresentada à FEN/UERJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

UMA ARQUITETURA PARA ACESSO REMOTO DA INFORMAÇÃO GEOGRÁFICA BASEADA EM OBJETOS DISTRIBUÍDOS PARA AMBIENTE MARÍTIMO

Luiz Fernando Yuan Gouvêa

OUTUBRO/2002

Orientador: João Araújo Ribeiro, Dr, UERJ

Programa de Pós-Graduação em Engenharia de Computação - Área de Concentração em Geomática

Sistemas de Informação Geográfica (SIG) são ferramentas computacionais que permitem ao usuário armazenar, analisar e gerenciar dados espaciais referenciados geograficamente. Os primeiros SIGs eram sistemas monolíticos e usavam estruturas de dados proprietárias e fechadas. Com o advento das redes de computadores e das tecnologias de ambiente distribuído, novos requisitos estão sendo reconhecidos como importantes para os futuros SIGs, tais como: suporte a multi-usuários, processamento e gerenciamento de informações distribuídas, transparência e interoperabilidade entre sistemas heterogêneos. Este trabalho descreve uma arquitetura que permite que Sistemas de Informação Geográfica heterogêneos localizem, acessem e processem dados distribuídos pela rede, tirando proveito da natural distribuição do dado geográfico e incorporando a visão de objetos à informação geográfica. O modelo baseia-se na tecnologia de objetos distribuídos e no padrão de dados hidrográficos IHO S-57 a ser utilizado na Marinha do Brasil.

Palavras-chave: Geomática, Sistemas Distribuídos.

Abstract of Dissertation presented to FEN/UERJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

AN ARCHITECTURE FOR REMOTE ACCESS OF GEOGRAPHIC INFORMATION BASED ON DISTRIBUTED OBJECTS FOR MARITIME ENVIRONMENT

Luiz Fernando Yuan Gouvêa

OCTOBER/2002

Advisor: João Araújo Ribeiro, Dr, UERJ

Computer Engineering Program - Field of Geomatic

Geographic Information Systems (GIS) are computational tools allowing users to store, analyze and manage georeferenced spatial data. The first GIS were monolithic systems and handled proprietary and reserved data structures. With advent of the computers networks and distributed environment technologies, new requirements are being recognized as important for the future GIS, like multi-user support, processing and management of distributed information, transparency and interoperability between heterogeneous distributed systems. This work describes an architecture that allows Geographic Information Systems to find, access and process distributed data at network taking advantage of natural distribution of data. The model is based on distributed objects and IHO S-57 hydrographic data standard that will be adopted by Brazilian Navy.

Keywords: Geomatic, Distributed Systems.

Sumário

FIGURAS

CAPÍTULO 1

INTRODUÇÃO

Os Sistemas de Informação Geográfica (SIG) atuais possuem capacidade para armazenar, analisar e manipular dados geográficos comportando diferentes tipos de dados e aplicações, em várias áreas do conhecimento humano. Exemplos são otimização de tráfego, controle cadastral, gerenciamento de serviços de utilidade pública, demografia, cartografia, administração de recursos naturais, monitoramento costeiro, controle de epidemias, planejamento urbano [4].

Os mares e oceanos compõem mais de dois terços da superfície da terra congregando enormes interesses comerciais, ambientais e militares. O saber geográfico marítimo é uma peça chave na defesa de tais interesses tornando os SIGs ferramentas fundamentais no cenário naval de uma nação.

Uma das características marcantes nos SIGs é a sua capacidade de manipular grandes volumes de dados referenciados geograficamente. Os primeiros SIGs armazenavam os dados geográficos em estruturas proprietárias e fechadas. Não possuíam qualquer compromisso em disponibilizar e distribuir os dados para outros sistemas.

Atualmente, segundo Perez et al. [21], com o advento das redes de computadores e as facilidades de gerenciamento de banco de dados, novos requisitos estão sendo reconhecidos como importante para os futuros SIGs, tais como suporte a multi-usuários, processamento de grandes volumes de dados e gerenciamento de informações distribuídas.

Uma das características mais importantes dos futuros SIGs será a capacidade de interconexão entre diferentes sistemas e o compartilhamento de informações de diversas fontes, a fim de não duplicar os dados existentes e de beneficiar-se com as atualizações freqüentes da base de dados. Assim como ocorre em outros sistemas de informações, os SIGs incorporarão a filosofia de sistemas abertos tornando seus dados e funções acessíveis a qualquer outro sistema remotamente.

Nesse sentido, é de fundamental importância a adoção de uma arquitetura apropriada ao suporte de Sistemas de Informação Geográfica distribuídos abertos, considerando tanto as pesquisas e desenvolvimentos realizados na área de SIGs, quanto os aspectos envolvidos na integração e interoperação de sistemas distribuídos heterogêneos.

O mundo atual já apresenta um cenário propício ao desenvolvimento de sistemas distribuídos heterogêneos. A informação e o poder de processamento não mais se limitam ao computador pessoal. Este novo contexto impõe um processo contínuo de inovação e novas tecnologias de ambiente distribuído surgem trazendo novas conquistas na informática. Enquanto isso, os principais Sistemas de Informação Geográfica existentes pouco têm explorado essa área, motivados, talvez, pela falta de padrões e de ferramentas básicas capazes de distribuir a informação geográfica eficientemente.

Neste panorama se insere o presente trabalho que propõe uma arquitetura de distribuição da informação geográfica marítima vetorizada, baseada na tecnologia de objetos distribuídos, utilizando o modelo de objetos definidos no padrão de dados hidrográficos S-57. Esta arquitetura servirá como uma infra-estrutura básica para a implementação de SIGs distribuídos na Marinha do Brasil.

1.1. MOTIVAÇÃO

Atualmente, a quantidade de SIGs instalados no mundo todo está aumentando em todos os campos possíveis variando desde a área de gestão territorial até transportes, logística, prospecção, geologia, arqueologia, etc. Como conseqüência, a disponibilidade de dados geográficos tem sido um dos maiores obstáculos no desenvolvimento e uso de aplicações de SIGs [4].

Tornar a informação disponível remotamente sem torná-la acessível ao usuário, pode provocar o fracasso de um sistema. A capacidade de dois SIGs diferentes trocarem dados independentemente do sistema operacional, da linguagem de programação e da localização dos dados é sem dúvida uma questão fascinante; porém a realidade é outra. Devido às características espaciais multidimensionais da informação geográfica e de seu caráter multidisciplinar, estruturas complexas e não usuais de dados são utilizadas no armazenamento de tal informação, contrastando com uma das principais características que tornam a interoperabilidade possível: a simplicidade.

Outra grande barreira à interoperabilidade dos SIGs é a falta de padronização de dados. Cada fabricante de software possui seu modelo de dados geográficos próprio; e, pior

ainda, cada um criou o seu conceito próprio de dados geográficos, o que torna a tarefa de compartilhar dados praticamente impossível.

Diante desse quadro, algum esforço tem surgido na padronização de dados geográficos. Padrões como STDS (Spatial Data Transfer Standard), SAIF (Spatial Archive and Interchange Format) e S-57 (Transfer Standard for Digital Hydrographic Data – Special Edition 57), mesmo que ainda não aceitos popularmente, vêm ganhando espaço lentamente.

Um outro fator importante diz respeito ao encapsulamento da estrutura interna do banco de dados. Bancos de dados geográficos possuem, em geral, uma estrutura complexa, hierarquizada e com a topologia estruturada. O acesso à informação geográfica por um SIG sem a necessidade de conhecer a estrutura interna do banco de dados torna a tarefa muito mais simples e cria um passo a mais rumo à interoperabilidade.

A interoperabilidade de sistemas é um ramo bastante extenso e sinuoso. Ainda não se tem conhecimento de sistemas totalmente interoperáveis, tanto no que diz respeito aos dados, quanto às funções, métodos e atributos desses sistemas. Porém, algumas características, como vimos, podem tornar um sistema mais ou menos interoperável. Segundo ALMEIDA [1], "Atualmente, a interoperabilidade só é possível sobre o mais curto dos domínios. O esforço para alcançar a interoperabilidade é, portanto, um esforço de estender domínios. A arquitetura atual dos SIGs requer que seus usuários sejam especialistas, que aprendam uma terminologia específica e uma interface dominada por detalhes de implementação. O usuário deve ter a habilidade de decodificar várias siglas e suas características, ou seja, os usuários têm que memorizar a maioria dos metadados".

O OpenGIS (http://www.opengis.org) é provavelmente o mais ambicioso e importante grupo de padronização de SIGs abertos da indústria. O consórcio OpenGIS é uma ampla aliança de órgãos governamentais, institutos de pesquisa, desenvolvedores de software e integradores de sistemas que buscam um grau extremamente alto de interoperabilidade entre os SIGs nos mais variados ramos de atividade. O OpenGIS desenvolve especificações técnicas e encoraja a indústria de software a se adaptar a tais especificações definindo conceitos, requisitos e padrões. O objetivo final é construir uma tecnologia que vai possibilitar ao desenvolvedor de aplicações usar qualquer dado, função ou processo geográfico disponível na rede dentro de um único ambiente e um único fluxo de trabalho integrado. Premido entre o

requisito de simplicidade e a enorme diversidade de dados, aplicações e usuários SIGs, o OpenGIS busca um difícil equilíbrio entre interoperabilidade e praticidade. Como não poderia de ser, a arquitetura OpenGIS não é simples à primeira vista, e mesmo os softwares que se intitulam compatível com OpenGIS conseguiram apenas uma compatibilidade parcial com a especificação.

Há alguns anos surgiram os primeiros SIGs voltados para a navegação marítima. O sucesso desses sistemas foi tão grande que a utilização de um sistema de navegação marítima computadorizada a bordo dos navios é, em certos casos, obrigatório por lei.

O uso de cartas náuticas em papel com um sistema mecânico de plotagem e com uma alta dependência do operador cria um ambiente sujeito a falhas e erros grosseiros. Não existe a possibilidade de distribuir a informação geográfica e nem de uma avaliação espacial detalhada e precisa. Além disso, diferentes versões de cartas são utilizadas provocando variações na análise espacial dos diversos operadores.

Com o surgimento dos SIGs e a disseminação de cartas náuticas em formato eletrônico, tornou-se possível o uso, em tempo real, de um sistema a bordo destinado à navegação e a obtenção de informações geográficas de forma precisa e eficiente. Esses sistemas, denominados ECDIS (Eletronic Chart Display Information System), são hoje uma realidade. Os ECDIS são capazes de traçar rotas, instruir a navegação segura, alarmar perigos e até mesmo funcionar como piloto automático de embarcações mais modernas. Entretanto, os ECDIS atuais trabalham de forma isolada (stand alone) sem nenhum compromisso de disseminar a informação geográfica dentro ou fora do navio. A informação tática (posição, rumo e velocidade dos alvos), extremamente necessária à operação militar, simplesmente não é tratada.

Devido à necessidade, nos navios de guerra, de disseminar a informação entre os diversos compartimentos e até mesmo entre os componentes da frota, tais como navios, aeronaves, submarinos e postos em terra, a adoção de um ECDIS distribuído e com alto nível de interoperabilidade e capaz de tratar a informação tática, seria a solução mais adequada e plausível no ambiente militar.

Resumidamente, pode-se alinhar as seguintes vantagens num ECDIS distribuído para aplicações militares:

- Baixa dependência do operador, possibilitando uma navegação mais segura;
- Disseminação da informação tática (posição, rumo e velocidade do próprio navio e dos demais alvos detectados), geográfica e ambiental (corrente e vento) dentro e fora do navio;
- Distribuição do esforço computacional entre os diversos nós da rede;
- Fácil atualização da base de dados geográfica.

Outra motivação importante é a interoperabilidade com os demais sistemas de bordo existentes tais como o sistema de informações táticas, o sistema de armas e o sistema de navegação.

1.2. OBJETIVOS

Os objetivos principais deste trabalho de pesquisa são:

- Identificar e analisar os fatores críticos que impedem a interoperabilidade nos acessos aos dados espaciais;
- Realizar um levantamento dos principais padrões de formatos neutros de troca de dados espaciais;
- Mostrar o uso da orientação a objetos sobre os dados geográficos.
- Definir “GeoEspaço”, “GeoObjeto”, feição e grupo detalhando o uso do padrão CORBA;

- Apresentar uma arquitetura de distribuição da informação geográfica vetorizada baseada no formato S-57.

1.3. ORGANIZAÇÃO DO TRABALHO

A dissertação está distribuída nos seguintes capítulos:

O capítulo 2 descreve alguns conceitos básicos para melhor compreensão do trabalho, tais como:

- Conceitos básicos de Geomática. São apresentados os conceitos de SIG, Banco de Dados Geográficos, representação espacial da informação geográfica e relacionamento espacial de vizinhança;

- Conceitos de modelagem orientada a objetos e sua aplicação em Geomática;

- Conceito de sistemas distribuídos;

- Levantamento dos principais formatos neutros de troca existentes;

O capítulo 3 descreve as linguagens de definição de interfaces e CORBA.

O capítulo 4 trata da descrição do problema da distribuição da informação geográfica em um ambiente naval bem como os desafios enfrentados na construção de sistemas distribuídos. Abrange também os problemas enfrentados pelo modelo de dados utilizados atualmente e as vantagens de se utilizar a orientação a objetos.

No capítulo 5 é apresentada uma arquitetura de software para distribuição da informação e todas as interfaces necessárias para o acesso de tal informação. É mostrado também como CORBA se ajusta perfeitamente tanto na arquitetura proposta quanto no modelo de dados S-57.

No capítulo 6 estão a conclusão, as contribuições deste trabalho e as perspectivas de trabalhos futuros.

CAPÍTULO 2

CONCEITOS ENVOLVIDOS

Para um melhor entendimento deste trabalho, faz-se necessário a abordagem de determinados conceitos e definições para sedimentar o tema deste trabalho. Este capítulo descreve os pontos importantes relativos à atual tecnologia de Sistemas de Informação Geográfica e Banco de Dados Geográficos. Serão abordados também temas como o paradigma da orientação a objetos, sistemas distribuídos e formatos padrões de armazenamento de dados geográficos.

2.1. SISTEMAS DE INFORMAÇÃO GEOGRÁFICA E BANCO DE DADOS GEOGRÁFICOS

Segundo CÂMARA et al. [4] “Sistemas de Informação Geográfica, ou abreviadamente SIGs, são sistemas de informações construídos especialmente para armazenar, analisar e manipular dados geográficos, ou seja, dados que representam objetos e fenômenos em que a localização geográfica é uma característica inerente e indispensável para tratá-los. Dados geográficos são coletados a partir de diversas fontes e armazenados via de regra nos chamados bancos de dados geográficos”.

Podemos dizer que os Sistemas de Informação Geográfica diferem dos demais tipos de sistemas de informações basicamente por manipular dados que descrevem a geometria, a localização e o relacionamento topológico entre objetos geográficos. Os dados contidos nos bancos de dados geográficos necessitam de estruturas bastante complexas para sua representação e são acessados por vários grupos de usuários, cada um com seu ponto de vista diferente.

CÂMARA [3] considera a existência de três gerações de Sistemas de Informação Geográfica. Na primeira geração, baseada em CAD cartográfico, havia grandes limitações no acesso à base de dados, pois não existia a preocupação de gerar arquivos de dados. A segunda geração, muito usada hoje em dia, é baseada em banco de dados geográficos acoplado a sistemas gerenciadores de banco de dados (SGBD) concebida para ser usado em um ambiente cliente-servidor. Previa-se, para o final da década de 90, a terceira geração com acesso ao banco de dados através de redes locais e remotas com interface WWW (World Wide Web).

Essa terceira geração, conhecida hoje como SIG-WEB, vem aos pouco ganhando espaço e se popularizando. Apesar da grande limitação imposta pelo HTML, novas linguagens que proporcionam maior dinamismo podem ser usadas, tais como: PHP (Personal Home Page), ASP (Active Server Pages), GML (Geographic Markup Language), etc.

As três gerações, como vimos, diferenciam-se basicamente na maneira de acessar o banco de dados. A evolução dos SIGs tem mostrado que os próximos requisitos na cadeia evolutiva são o acesso da informação geográfica via rede e a interoperabilidade de dados. Pode-se prever, com isso, uma nova geração baseada em banco de dados geográficos distribuídos constituído de vários bancos de dados geográficos produzidos por instituições independentes, onde o usuário seria capaz de acessar e cruzar informações de fontes de dados distintas.

Os modelos de representação espaciais dos dados geográficos são inúmeros, porém quase todos derivam de dois tipos básicos: a representação raster (ou por varredura) e a representação vetorial (ou poligonal ou topológica). A principal diferença entre eles é o modelo de dados utilizado para representar os objetos geográficos. A representação vetorial considera o espaço geográfico contínuo, seguindo os postulados da geometria euclidiana, enquanto a representação raster divide o espaço em elementos discretos denominados pixels (Fig. 1).

Todos os modelos de representação do dado geográfico são técnicas usadas para armazenar as características geográficas em um banco de dados. Os objetos espaciais no modelo vetorial são representados por pontos, linhas ou polígonos (ou área) aos quais são associados atributos ou feições dos objetos que representam. Por exemplo, o ponto pode representar local de risco de inundação. A linha pode representar estradas ou rios; já o polígono pode representar feições de área como vegetação, uso da terra, etc (Fig. 2).

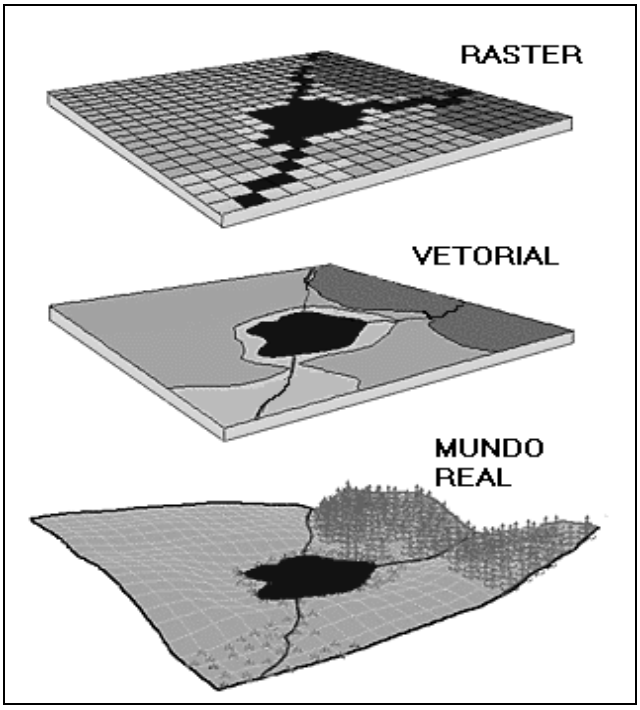


Fig. 1 - Tipos de representação espacial.

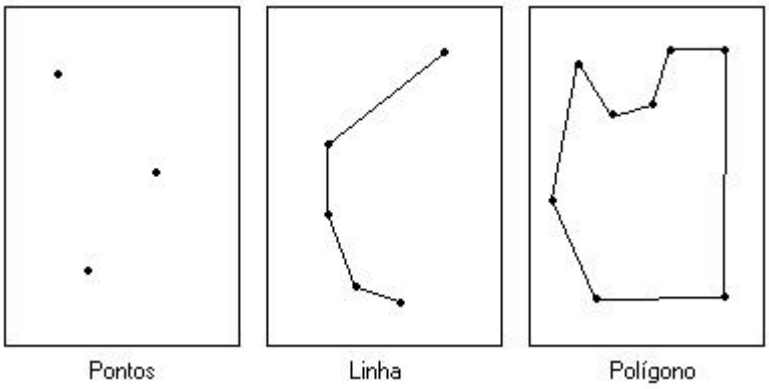


Fig. 2 - Forma de representação dos objetos espaciais no modelo vetorial.

Linhas vetoriais ou arcos consistem de uma seqüência de vértices iniciadas e terminadas por nós. Um nó é definido por um vértice que inicia ou termina um segmento de arco. Um ponto é definido por um par de coordenadas. Um polígono é definido por um conjunto fechado de pares de coordenadas.

Além de armazenar a geometria e as feições, é importante, para uma análise mais apurada dos dados, armazenar também o relacionamento topológico entre os objetos geográficos. O relacionamento topológico de vizinhança é o mais utilizado, porém outros relacionamentos do tipo toca, dentro de, cruza, sobrepõe e disjunto; podem também serem inseridos no banco de dados.

Segundo TEIXEIRA [25], a representação de um polígono, de acordo com a vizinhança, pode ser caracterizado de três formas: polígonos simples, polígonos com dicionário de pontos e polígonos com estrutura topológica explícita.

No polígono simples armazena-se somente as coordenadas X e Y dos vértices. É a maneira mais fácil de representar um polígono. É uma estrutura voltada apenas para a apresentação gráfica. Apesar da vantagem de ser simples, o método tem várias desvantagens:

- As linhas comuns entre dois polígonos têm que ser digitalizada duas vezes, o que aumenta a ocorrência de erros, já que não se pode garantir que as duas linhas ficarão exatamente sobrepostas (Fig. 3);

- Não existe informação sobre polígonos vizinhos;

- Ilhas (polígonos vazados) não são possíveis senão como construções puramente gráficas;

- Difícil verificar se a topologia está correta ou incompleta, ou mesmo se foram criadas situações topologicamente inadmissíveis como “dead ends” e “weird polygon” (Fig. 4).

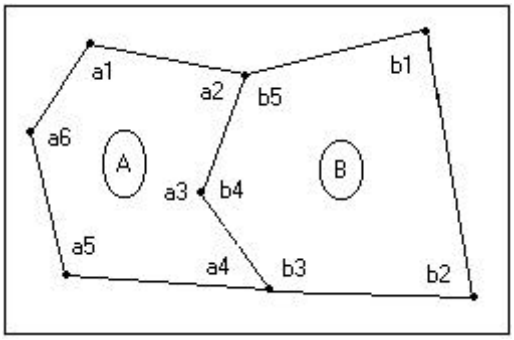


Fig. 3 - Polígonos simples: os pontos a2, a3, a4, b3, b4, b5 são digitalizados duas vezes.

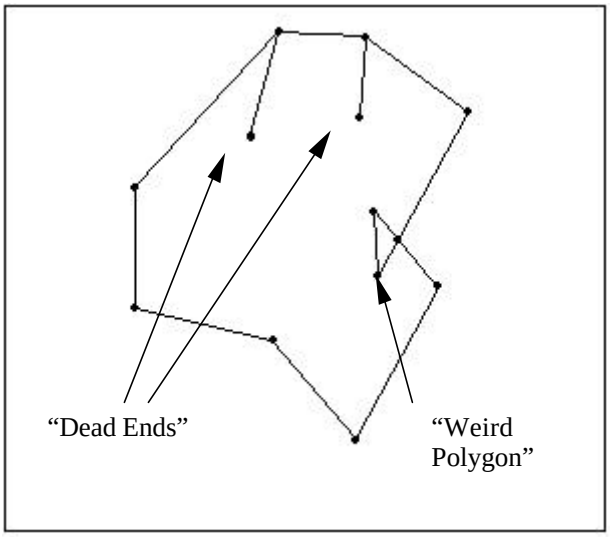
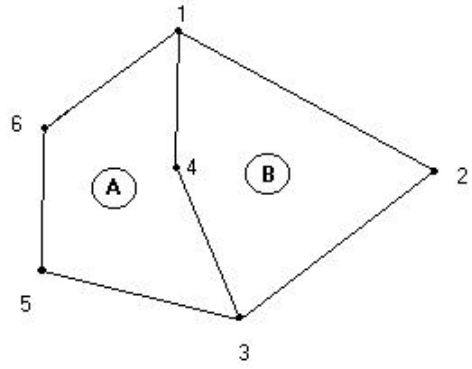


Fig. 4 - Erros na topologia.

No polígono com dicionário de pontos, cada ponto é numerado e referenciado a um dicionário onde são registrados os pontos pertencentes aos polígonos (Fig. 5). Dessa forma as coordenadas dos vértices do polígono ficam armazenadas em uma tabela separada. Embora ainda não resolva o problema de vizinhança, tem como vantagem o fato de que os limites entre polígonos adjacentes serem únicos, ou seja, não existe a redundância. Permanece o problema de ilhas bem como "dead ends" e "weird polygons".



Ponto	Coordenada	Polígonos
1	X1,Y1	A,B
2	X2,Y2	B
3	X3,Y3	A,B
4	X4,Y4	A,B
5	X5,Y5	A
6	X6,Y6	A

Fig. 5 - Polígono com dicionário de pontos.

Nos polígonos com estrutura topológica explícita os relacionamentos topológicos de ilhas e vizinhança ficam armazenados no banco de dados. Essa estrutura topológica só pode ser resolvida de duas maneiras: durante a entrada de dados ou com o uso de programas específicos que criam a topologia a partir de um conjunto de linhas. No primeiro caso a tarefa recai sobre o operador e no segundo caso sobre o computador. De qualquer maneira, nesse tipo de estrutura, há um aumento significativo de dados a serem armazenados, pois, além do dicionário de pontos, existe a estrutura topológica que deve ser inserida no banco de dados (Fig. 6).

Arco	Nós	Polígono Direito	Polígono Esquerdo
P	3,5,6,1	A	0
Q	1,2,3	B	0
R	3,4,1	B	A

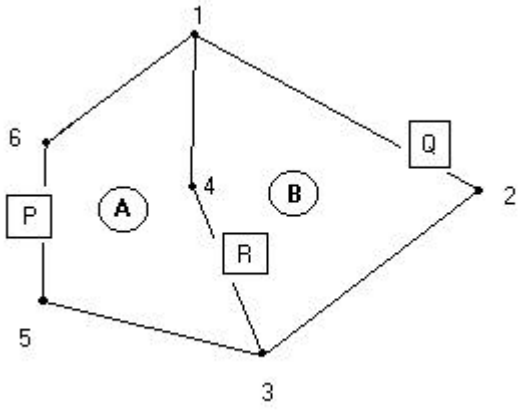


Fig. 6 - Polígono de estrutura topológica explícita.

Além dos atributos espaciais (ponto, linha ou polígono), o objeto geográfico pode possuir inúmeros atributos não-espaciais. Por exemplo, o objeto geográfico que representa o município do Rio de Janeiro pode ser definido pelo atributo espacial do tipo polígono que define o seu limite territorial e pelos atributos não-espaciais nome, população, renda per capita, etc.

2.2. ORIENTAÇÃO A OBJETOS E UML (Unified Modeling Language)

O grande desenvolvimento da indústria de software tem fomentado questões relacionadas ao desenvolvimento e manutenção de software. Tem crescido o interesse por novas abordagens de desenvolvimento de software que permitam que o código seja facilmente compreendido e que o processo de expansão e alteração de sistemas se torne cada vez mais simples. A tecnologia da orientação a objetos (OO) emergiu nesse meio como um dos paradigmas mais promissores na solução desses problemas.

O paradigma da orientação a objetos surgiu da linguagem de programação Simula 67, porém somente no final da década de 80 ganhou grande aceitação no mercado. A orientação a objetos permite uma representação natural dos objetos do mundo real incluindo seus atributos, relacionamentos e comportamentos. É, portanto, mais próximo da maneira de pensar do usuário final. Uma aplicação orientada a objetos consiste de um conjunto de objetos com seus próprios estados privados e que interagem entre si. Sistemas orientados a objetos são mais fáceis de manter devido a sua característica modular. Uma alteração realizada no código de um objeto não afeta os outros objetos no sistema. O paradigma da orientação a objetos elimina a necessidade de compartilhar áreas de dados, reduzindo então o acoplamento do sistema. Promove, também, a reusabilidade: objetos são autocontidos e podem ser usados em outras aplicações suficientemente similares. A orientação a objetos permite uma fácil distribuição e paralelismo no processamento.

A grande diferença entre o paradigma da orientação a objetos e os modelos de dados anteriores, como o relacional ou semântico, é a capacidade de representar o componente funcional (comportamento) das entidades (objetos), não somente a sua estrutura. O paradigma OO se adapta melhor a aplicações não tradicionais como CAD, SIG e CASE, segundo MITROVIC [16].

UML [19] é a sucessora da onda de métodos de análise e projeto orientado a objetos que surgiu no final dos anos 80 que unificou os três métodos mais populares da época: do Booch [2], do Rumbaugh [23] e do Jacobson [12]. Atualmente é um padrão da OMG (vide seção 3.1.). UML é composto por um conjunto de diagramas e por uma metodologia unificada que procura facilitar o processo de desenvolvimentos de aplicações orientadas a objetos através de uma notação gráfica.

Apresentaremos a seguir alguns conceitos básicos de UML (abstração, encapsulamento, associação e agregação) e de OO (objetos, classes, herança, polimorfismo, métodos e atributos).

2.2.1. ABSTRAÇÃO

O objetivo da abstração é identificar o comportamento importante de objetos para posterior implementação. RUMBAUGH [23] define abstração como uma análise seletiva de certos aspectos de um problema isolando apenas os aspectos importantes. BOOCH [2] define abstração como a principal característica de um objeto que o distingue dos demais.

A abstração, como vimos, é definida tanto como um processo tanto como uma entidade. Como um processo denota a extração dos detalhes essenciais de um item ou grupo de itens ignorando os detalhes desnecessários. Dessa maneira é possível definir um objeto focando a atenção nas questões importantes e não se preocupando em detalhes desnecessários. Como entidade, a abstração denota uma representação resumida de uma entidade do mundo real.

A abstração é uma ferramenta importante na análise orientada a objetos. O analista foca a atenção apenas nos aspectos necessários para a estruturação do problema. Abandonando os detalhes não essenciais, principalmente os detalhes de implementação, é possível ter uma visão macro do problema.

Os detalhes importantes dependem fundamentalmente dos aspectos que se deseja modelar das entidades do mundo real. Um biólogo e um geólogo, por exemplo, enxergam o mundo real de maneiras diferentes. Cada um considera os detalhes importantes de acordo com

a sua perspectiva. Devido a sua característica multidisciplinar, o conceito de abstração é de suma importância para a geomática.

A engenharia de software trata dois diferentes tipos de abstração: de função e de dados. Na abstração de função a atenção é concentrada na interface da função, ignorando a sua implementação. A abstração de dados estendeu o conceito da abstração de função onde ambos, dados e implementação, não são de interesse do usuário. A implementação de uma pilha de dados, por exemplo, pode ser feita através de funções de empilhamento e desempilhamento ("pop" e "push") escondendo a estrutura de dados da pilha e a implementação das funções. A vantagem deste tipo de abstração é que tanto a implementação das operações quanto a estrutura de dados da pilha pode ser alterada sem afetar os usuários.

2.2.2. ENCAPSULAMENTO[1]

Encapsulamento significa construir uma cápsula, neste caso uma barreira conceitual em torno de uma entidade. Segundo BOOCH [2], "Encapsulamento é o processo de divisão dos elementos de uma abstração que constituem sua estrutura e comportamento; encapsulamento serve para separar a interface contratual de uma abstração e sua implementação". RUMBAUGH [23], na mesma linha, diz que encapsulamento consiste na separação dos aspectos externos de um objeto, acessível por outros objetos, dos detalhes internos de implementação que ficam ocultos dos demais objetos.

O encapsulamento permite uma maior independência entre objetos. Os detalhes internos de um objeto podem ser modificados sem que isso afete as aplicações que utilizam o mesmo. Qualquer alteração no código, por exemplo, para eliminar um erro ou aumentar o desempenho, fica restrito ao objeto que contém o código. Isso permite que mudanças possam ser realizadas de modo seguro facilitando a evolução e manutenção do software.

[1] Existe um outro conceito chamado de Ocultação de Informação que, para alguns autores, significa o mesmo que Encapsulamento. Este trabalho não faz distinção entre eles.

2.2.3. OBJETOS E CLASSES

Objeto é uma abstração de software que modela todos os aspectos relevantes das entidades físicas ou conceituais que encontramos no universo à nossa volta. O chefe de uma empresa pode ver os empregados, os edifícios, os departamentos e os documentos como objetos. Um engenheiro pode ver os pneus, as portas, o motor, a velocidade máxima e o nível de combustível de um automóvel como objetos. Átomos, moléculas, volumes e temperaturas podem ser objetos que um químico considerou na criação de uma simulação de reação química orientada a objetos. Finalmente um engenheiro de software pode ver pilhas, filas, janelas e botões como objetos.

Em geomática, as possibilidades de identificação de objetos são enormes. Dependendo da especialidade e do objetivo em questão, a estrutura e a composição da superfície da terra, do subsolo, da atmosfera, dos mares, das florestas, de um desastre ecológico, de um abalo sísmico, etc; força o especialista a considerar uma vasta quantidade de objetos necessários à modelagem do fenômeno de interesse.

Objetos possuem um estado. O estado de um objeto é uma condição ou conjunto de circunstâncias que descreve o objeto. O estado de uma conta bancária, por exemplo, poderia incluir o saldo da conta. O estado de um relógio poderia incluir a hora atual. O estado de uma lâmpada elétrica poderia ser “ligado” ou “desligado”. O estado de um rio poderia ser o nome, a vazão e o grau de poluição do rio.

Para objetos complexos como seres humanos e entidades geográficas, uma completa descrição do estado pode ser bastante extensa. Felizmente quando utilizamos objetos para modelar o mundo real, restringimos os possíveis estados para somente aquele que é relevante para o nosso modelo (abstração).

A comunicação entre objetos é feita através de métodos ou serviços. É através dos métodos que se define o comportamento do objeto. Os objetos em geral possuem dois métodos reservados: o método construtor e o método destrutor. O construtor é chamado sempre que o objeto é criado. O destrutor é chamado sempre que o objeto é destruído.

Os atributos armazenam o estado do objeto. Em geral o usuário não possui acesso direto aos atributos do objeto. Neste caso, para o usuário conhecer o valor de um atributo deve-se recorrer aos métodos (Fig. 7).

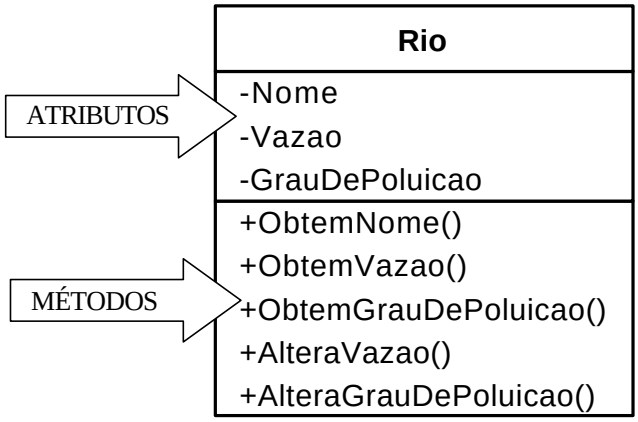


Fig. 7 - Representação gráfica da classe "Rio" em UML (Unified Modeling Language).

Classe é a descrição de um conjunto de objetos com estruturas, comportamento e relacionamento similares. Classe é um padrão, um esboço ou protótipo que descreve os atributos e métodos comuns de todos os objetos de um certo tipo. Classes servem como fôrmas para criação dos objetos. A modelagem de uma cidade, por exemplo, poderia incluir as classes “Residência”, “Rua” e “Terreno”. Uma casa situada na rua José da Silva, n° 32; é um objeto da classe “Residência”. As classes definem os métodos e atributos relativos aos objetos. A classe “Residência” pode possuir o atributo “Quantidade De Quartos” que é específico para todas as residências. A classe “Rua” pode possuir o método “Obtém Terrenos” que informa todos os terrenos situados na rua.

2.2.4. HERANÇA

Herança de classes é um relacionamento entre classes onde uma classe é pai de outra classe. A classe pai é chamada de classe base ou superclasse. Herança de classes pode ser definida como o processo onde um objeto adquire as características de outro objeto. A classe “Residência”, por exemplo, poderia herdar as características da classe “Construção”.

“Residência” é uma especialização de “Construção”. “Construção” é uma generalização de “Residência”.

Na Fig. 8 vemos que “Residência” é uma “Construção”. Além disso, tanto “Residência Urbana” quanto “Residência Rural” são tipos de “Residência”. A propriedade transitiva da herança implica que todos os atributos e métodos da superclasse são passados não somente para as subclasses imediatamente inferiores como também às sub-subclasses e assim por diante. Por exemplo, os atributos “Proprietário” e “Endereço” são herdados pela subclasse “Residência” como também transitivamente pelas subclasses “Residência Urbana” e “Residência Rural”.

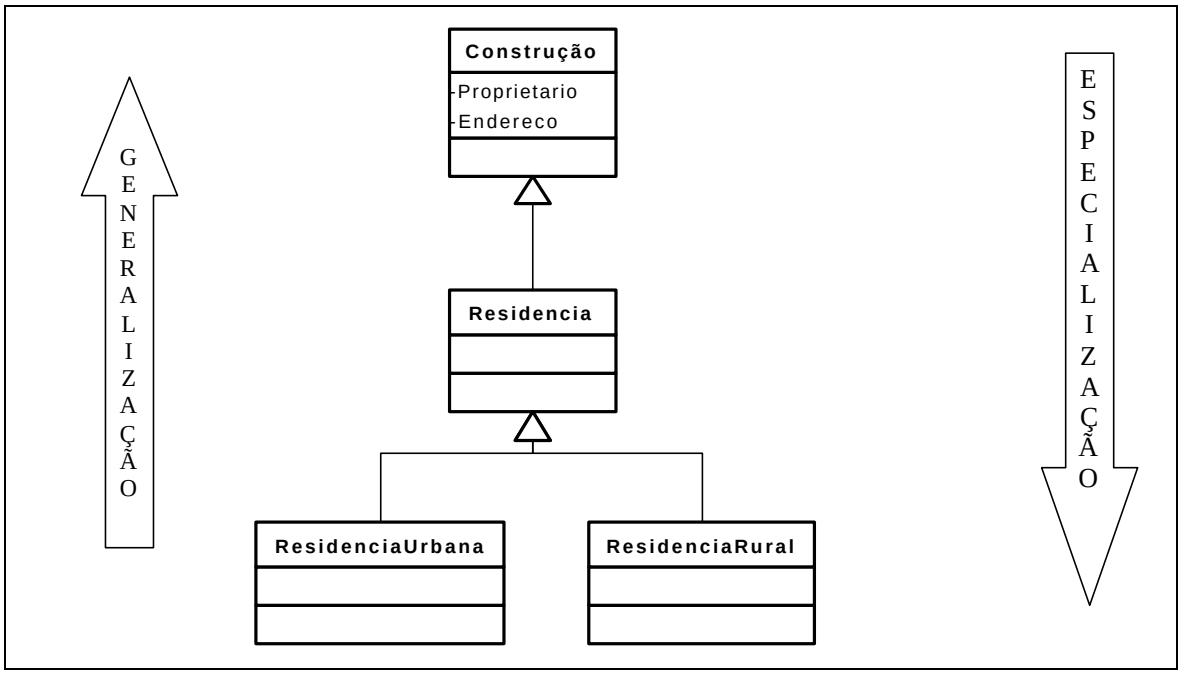


Fig. 8 - Herança em UML, adaptada de EGENHOFER [6].

O conceito de herança oferece os seguintes benefícios:

- Subclasses fornecem um comportamento especializado em relação aos elementos comuns pertencentes à superclasse. Através do uso de herança, programadores podem reusar o código escrito na superclasse muitas vezes;

- Programadores podem implementar superclasses denominadas classes abstratas que definem métodos genéricos. A classe abstrata pode apenas declarar ou

parcialmente implementar o método, deixando parte da classe indefinida e não implementada. Outros programadores podem preencher os detalhes nas subclasses especializadas.

Na orientação a objetos é comum várias classes herdarem as características de uma única superclasse. Este tipo de relacionamento é chamado de herança simples. Porém, esse tipo de hierarquia freqüentemente falha quando aplicada às entidades do mundo real. Na modelagem podem surgir casos em que um objeto geográfico necessite herdar as características de vários outros objetos. Esse tipo de relacionamento é denominado de herança múltipla. Essa estrutura, porém, não é hierárquica, porque, em termos de uma relação pai-filho, um filho pode ter muitos pais.

O exemplo a seguir mostra como a herança múltipla combina duas hierarquias distintas (Fig. 9). A primeira hierarquia corresponde a separação de “Via de Transporte” em via “Artificial” e via “Natural”. “Estrada” e “Canal” herdam de via “Artificial”. “Rio Navegável” é uma via de transporte “Natural”. “Corpo D’água” com “Rio”, “Canal” e “Lago”; formam a segunda hierarquia, no qual dois rios são distinguidos: “Rio Navegável” e “Rio Não Navegável”. Essas hierarquias não podem ser comparadas entre si, porque um corpo d’água não é necessariamente uma via de transporte e, vice-versa, nem toda via de transporte é um corpo d’água. Entretanto as duas hierarquias compartilham subclasses comuns, porque “Canal” é tanto “Corpo D’água” como via de transporte ”Artificial”. “Rio Navegável” é tanto “Rio” como via de transporte “Natural”. As demais classes, neste esquema, possuem somente herança simples.

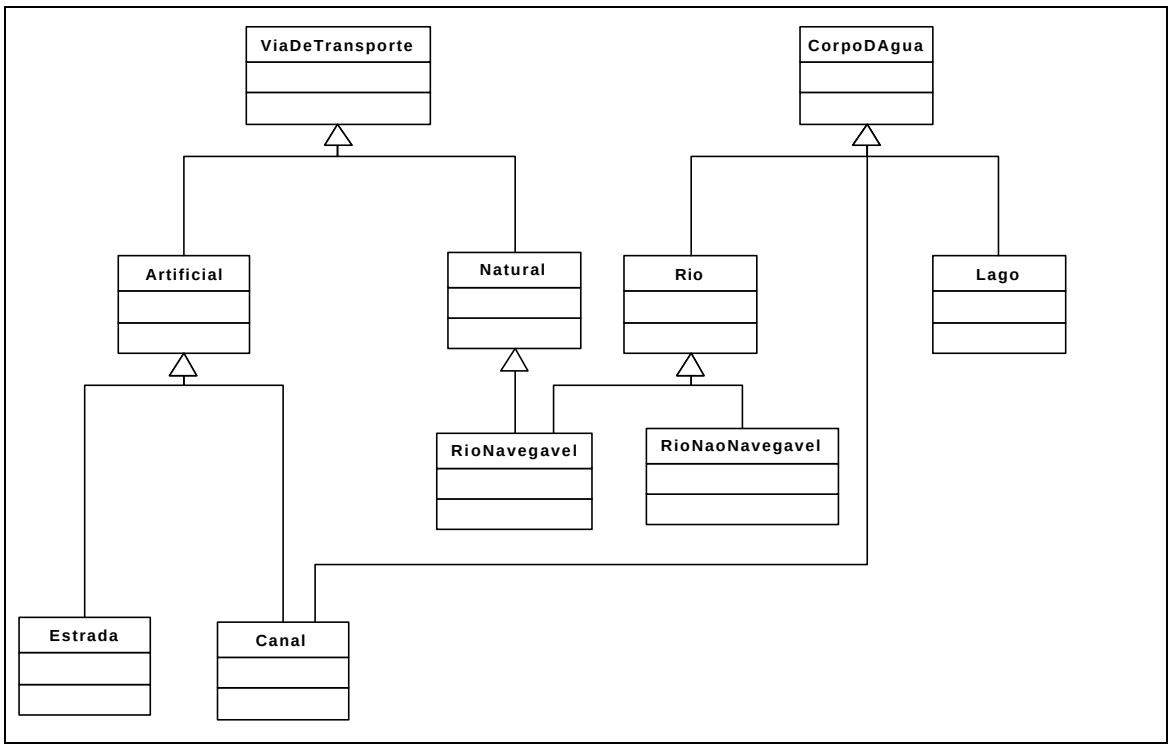


Fig. 9 - Herança múltipla do objeto “Canal” e do objeto “Rio Navegável”, adaptada de EGENHOFER [6].

Um outro aspecto importante diz respeito à herança de interfaces. Interfaces são, em linhas gerais, declarações de um conjunto de métodos que não possuem informação de implementação. Sua implementação deve ser fornecida através de uma classe. Uma mesma classe pode implementar várias interfaces. A herança de interfaces não deriva da herança das classes que a implementam. Herança de classes e herança de interfaces não se misturam.

2.2.5. POLIMORFISMO

Polimorfismo é um conceito no qual o nome de um de um método pode referir-se a objetos de diferentes classes que possuem um ancestral comum. Os objetos, ao serem acionados por este nome, respondem de maneira diferente, dependendo da classe a qual pertençam.

Polimorfismo significa várias formas, ou seja, a habilidade de um objeto se referir a várias classes diferentes. Por exemplo, os objetos geográficos poligonais do tipo triângulo e

do tipo retângulo podem possuir métodos próprios para cálculo de sua área (Fig. 10). Note que dependendo do tipo do objeto, o método correspondente de cálculo de área é chamado.

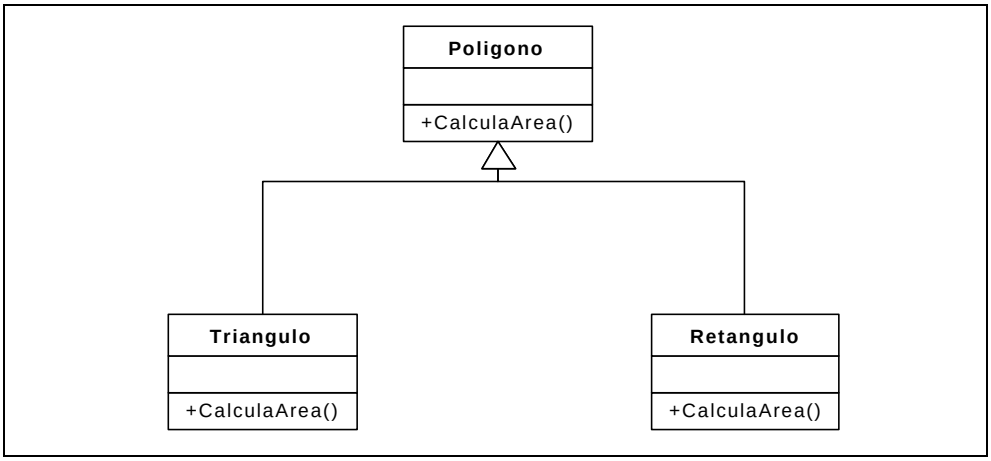


Fig. 10 – Polimorfismo.

2.2.6. ASSOCIAÇÃO

A associação relaciona dois ou mais objetos independentes. A associação é considerada um conjunto de objetos e os objetos envolvidos no relacionamento são chamados membros. Portanto esse relacionamento é dito uma relação do tipo membro-de. A associação é sempre relativa a dois ou mais objetos que possuem existência própria. A figura abaixo mostra a associação entre duas classes. Note a multiplicidade da associação onde uma loja vende várias (representado pelo asterisco) mercadorias.

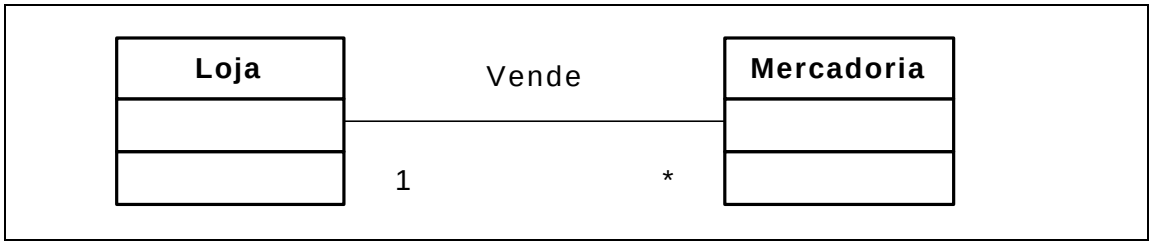


Fig. 11 - Associação simples.

Um exemplo de associação em geomática é o relacionamento topológico de vizinhança que associa um objeto geográfico com os objetos vizinhos. Um objeto geográfico pode ter nenhum, um ou vários objetos vizinhos (Fig. 12).

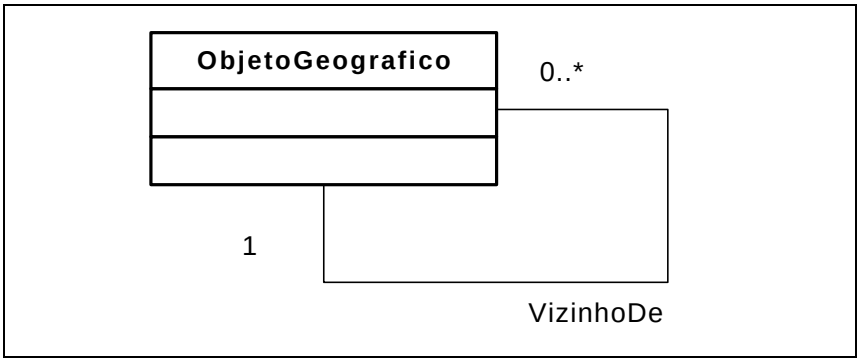


Fig. 12 - Associação de vizinhança entre objetos geográficos.

2.2.7. AGREGAÇÃO

Na orientação a objetos é possível que um objeto seja composto de vários outros objetos. O relacionamento de composição entre objetos é denominado de agregação. A agregação é uma abstração similar à associação, a qual modela objetos compostos. Vários objetos podem ser combinados para formar um objeto semanticamente de mais alta ordem, chamado de objeto agregante ou objeto composto, onde cada parte tem sua própria funcionalidade.

A relação formada pela agregação é freqüentemente chamada de relação parte-de posto que os objetos agregados são partes do objeto agregante. O relacionamento inverso de parte-de é o consiste-de visto que o agregante consiste de vários objetos agregados (Fig. 13).

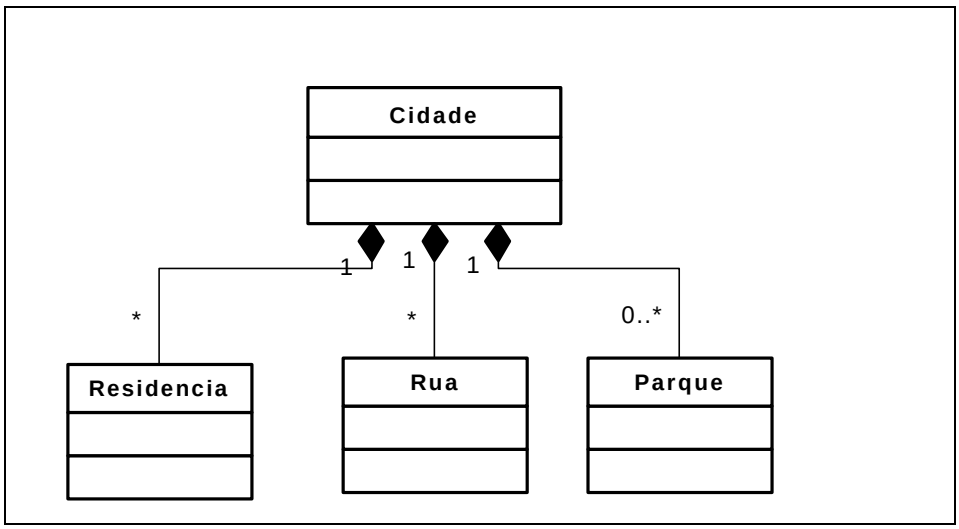


Fig. 13 - Agregação. Uma Cidade consiste de Residências, Ruas e, se existir, Parques.

2.3. SISTEMAS DISTRIBUÍDOS

Segundo TANENBAUM [24], existem inúmeras definições para sistemas distribuídos na literatura, nenhuma completamente satisfatória. Para o propósito deste trabalho é suficiente seguinte definição: “Sistema distribuído é uma coleção de computadores independentes que aparentam, para o usuário do sistema, como um único computador”.

Por essa definição podemos inferir que uma das principais características dos sistemas distribuídos é a transparência, ou seja, a capacidade do sistema de ocultar toda a sua estrutura e complexidade do usuário.

Uma segunda característica é a cooperação. O resultado final de um processo é fruto da cooperação de vários computadores que interagem entre si de forma integrada. Sistemas distribuídos não são apenas sistemas de rede. Não basta somente conectar várias máquinas com capacidade de troca de arquivos. Sistemas distribuídos são uma extensão dos sistemas de rede onde a distribuição da capacidade de processamento é repartida entre vários nós da rede de forma cooperativa.

Como vimos, um ambiente de computação distribuída não é apenas um simples recurso de comunicação entre os vários nós da rede. Um sistema distribuído deve oferecer para o usuário uma capacidade de processamento e de recursos que vão muito além daqueles

fornecidos pelo seu computador pessoal. Esses recursos permitem a utilização mais racional e eficiente dos serviços oferecidos pelo sistema.

A seguir são apresentadas as vantagens de sistemas distribuídos sobre sistemas centralizados e as vantagens sobre PCs independentes, segundo TANENBAUM [24].

2.3.1. VANTAGENS DE SISTEMAS DISTRIBUÍDOS SOBRE SISTEMAS CENTRALIZADOS

A primeira vantagem é econômica. Conectar vários computadores que possuem um bom poder de processamento cada um é mais barato do que adquirir um simples e único supercomputador. Como resultado obtemos uma melhor relação custo-benefício utilizando vários computadores baratos em um sistema do que um único e grande sistema centralizado.

A segunda vantagem é a velocidade de processamento. Um conjunto de microprocessadores pode não somente igualar a velocidade de processamento de um mainframe como até obter um desempenho jamais obtido com um mainframe de última geração.

A terceira vantagem diz respeito a algumas aplicações que são inerentemente distribuídas. Uma cadeia de supermercados, por exemplo, pode possuir várias lojas cada uma com autonomia de adquirir produtos produzidos localmente, fazer vendas e tomar decisões locais. É, portanto mais fácil cada loja realizar seu controle de estoque em um computador local ao invés de uma única máquina centralizando todos os dados no escritório matriz. Todavia, o gerente geral pode desejar realizar uma consulta, de tempos em tempos, para obter a quantidade de determinada mercadoria existente no estoque. Uma maneira obter o resultado é fazer com que o sistema completo pareça um único e simples computador para as aplicações, mas implementado de forma descentralizada.

A quarta vantagem diz respeito à confiança. Distribuindo a carga entre várias máquinas, uma simples falha paralisará no máximo uma máquina, deixando o resto do sistema intacto. O sistema terá uma perda de desempenho, porém continuará funcionando.

Por último, sistemas distribuídos permitem o crescimento incremental à medida que novas máquinas são incorporadas ao sistema. Nos sistemas centralizados, quando o

mainframe não suporta mais a carga de processamento, a solução é trocar o mainframe por outro mais veloz. O ambiente distribuído permite que novas máquinas sejam adicionadas ao sistema, aumentando gradualmente a capacidade total do sistema. Todas essas vantagens estão sumariadas na figura abaixo.

Economia	Microprocessadores oferecem uma melhor relação preço/desempenho do que a oferecida pelos mainframes
Velocidade	Um sistema distribuído pode ter um poder de processamento total maior que qualquer mainframe
Distribuição inerente	Algumas aplicações envolvem máquinas distribuídas fisicamente
Confiabilidade	Se uma máquina falha, o sistema como um todo continua funcionando
Crescimento incremental	O poder computacional pode crescer em pequenos incrementos

Fig. 14 - Vantagens de sistemas distribuídos sobre sistemas centralizados segundo TANENBAUM [24].

2.3.2. VANTAGENS DE SISTEMAS DISTRIBUÍDOS SOBRE COMPUTADORES INDEPENDENTES

Compartilhamento de dados é essencial para determinadas aplicações. Visto que computadores independentes não possuem capacidade de compartilhar dados, uma cópia da base de dados tem que ser fornecida para todos os usuários. Num sistema de reserva de passagem aérea, por exemplo, fornecer a cada agente de viagem uma cópia da base de dados não funciona, pois ninguém saberia quais assentos foram reservados pelas demais agências.

Compartilhamento envolve mais do que dados. Periféricos caros, tais como impressoras laser coloridas, plotters etc, também são candidatos.

Melhorar a comunicação pessoa a pessoa é a terceira razão para conectar grupos de pessoas em um sistema distribuído. Correio eletrônico é mais rápido e eficiente do que o correio convencional, não requer que as duas pessoas estejam disponíveis ao mesmo tempo,

como o telefone, e, diferentemente do FAX, produz documentos que podem ser editados, armazenados em um computador e manipulados por editores de texto.

Finalmente, sistemas distribuídos são mais flexíveis visto que uma rede pode possuir diferentes tipos de computadores com capacidades distintas de processamento. Uma tarefa então pode ser executada na máquina mais apropriada da rede. Dessa maneira a carga de trabalho pode ser espalhada entre os computadores mais eficazmente. A figura abaixo sumaria esses pontos.

Compartilhamento de dados	Permite vários usuários acessar uma base de dados comum
Compartilhamento de dispositivos	Permite vários usuários ter acesso a periféricos caros
Comunicação	Torna a comunicação pessoa a pessoa muito mais simples
Flexibilidade	Espalha a carga de trabalho sobre as máquinas disponíveis ao longo da rede.

Fig. 15 - Vantagens de sistemas distribuídos sobre computadores isolados segundo TANENBAUM [24].

2.3.3. DESVANTAGENS DE SISTEMAS DISTRIBUÍDOS

Apesar das inúmeras vantagens, os sistemas distribuídos possuem algumas desvantagens. A pior desvantagem é o software. Com o estado tecnológico atual, não existe experiência estabelecida no projeto, implementação e uso de software distribuído. Quanto mais pesquisa for feita, o problema diminuirá, mas no momento ele não pode ser subestimado, segundo TANENBAUM [24].

O segundo problema potencial é devido à rede de comunicação. A rede pode perder mensagens ou ficar saturada, sendo necessário trocar a rede de comunicação por uma mais confiável ou mais veloz (fibra ótica, por exemplo). Uma vez que sistemas distribuídos possuem uma alta dependência da rede, qualquer problema na mesma pode neutralizar a maioria das vantagens fornecida pela distribuição.

Segurança é o terceiro problema. O compartilhamento de dados proporciona uma conveniente e fácil maneira de disseminar a informação, mas para dados que necessitam ser mantidos secretos de qualquer maneira, é muitas vezes preferido coloca-los em um computador isolado de qualquer conexão de rede. As desvantagens de sistemas distribuídos estão resumidas na figura abaixo.

Software	Não existem muitos softwares, no momento, para sistemas distribuídos
Rede	A rede pode saturar ou ocorrer outros problemas
Segurança	Fácil acesso também se aplica aos dados secretos

Fig. 16 - Desvantagens de sistemas distribuídos segundo TANENBAUM [24].

É possível desenvolver sistemas de maneira distribuída, mas isso não significa que sempre é a melhor solução. Fortes razões devem existir a fim de obter vantagens na implementação de uma aplicação num ambiente distribuído.

2.4. PADRONIZAÇÃO DE FORMATOS DE DADOS GEOGRÁFICOS

Nas duas últimas décadas a tecnologia de Sistemas de Informação Geográfica tem sofrido um acelerado crescimento. Os SIGs se beneficiaram muito com a popularização das estações de trabalho gráficas e com o advento dos computadores pessoais. Tudo isso contribuiu para o surgimento de um grande número de SIGs no mercado, cada um com uma solução própria e com o seu modelo particular de dados. A produção de dados cartográficos cresceu assustadoramente seguida de altíssimos investimentos. Infelizmente cada agência cartográfica adotava um modelo próprio e, não raro, dentro de uma mesma agência, encontravam-se dados nos mais variados formatos. Diversidade de hardware, software e a falta de uma especificação de dados comum, inibiam a troca de dados entre sistemas. A falta de um padrão resultou em duplicidade de esforços e conjuntos incompatíveis de dados.

A transferência de dados entre os SIGs tornou-se então um grande problema. Cada fabricante de SIG procurou contornar o problema incorporando ao seu produto um tradutor que convertesse os dados do outro fabricante para o seu formato interno (Fig. 17). Esta solução tornou-se inviável à medida que novos formatos surgiam, pois o problema da conversão de dados crescia exponencialmente (Fig. 18).

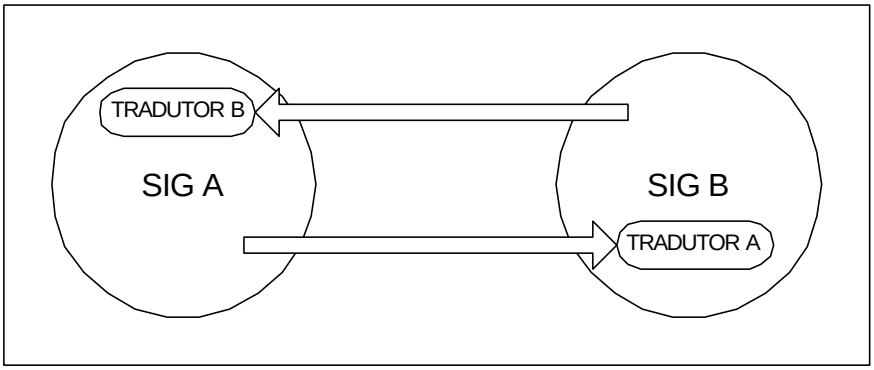


Fig. 17 - Dois SIGs e dois tradutores.

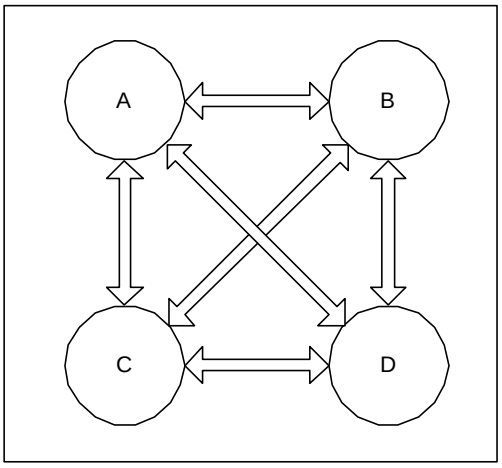


Fig. 18 - Quatro SIGs e doze tradutores.

Além do problema da quantidade de tradutores, existe freqüentemente, no processo de tradução, perda de informação devido a grande diferença no modelo conceitual dos formatos utilizados e geralmente no processo reverso não se obtinha o dado original. A tradução buscava realizar apenas a transformação estrutural dos dados, o aspecto semântico do dado, simplesmente, não é utilizado.

A solução natural para este problema seria a adoção de um formato padrão de troca de dados onde cada SIG pudesse converter desse formato para o seu formato interno e vice-versa (Fig. 19). A neutralidade é uma característica essencial do formato de troca, isto é, não ser voltado para nenhum tipo particular de SIG. O formato neutro deve atender a vários requisitos para permitir uma boa troca de dados, tais como: acurácia, completude e qualidade dos dados.

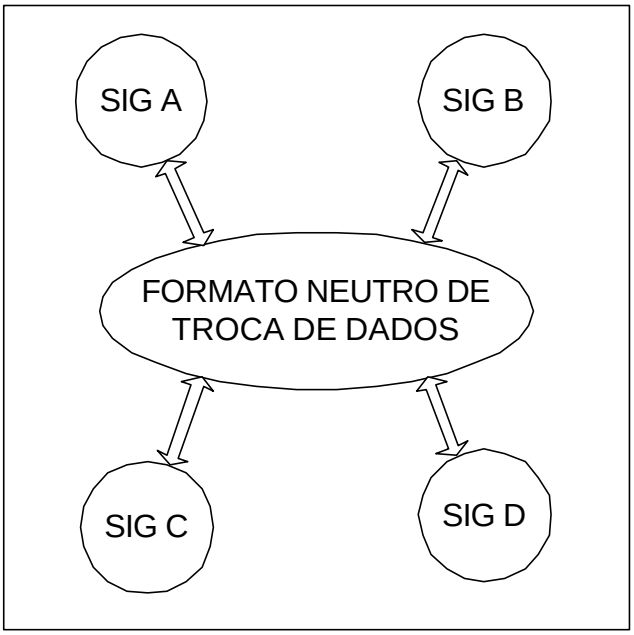


Fig. 19 - Formato neutro de troca de dados entre SIGs.

A adoção de um único formato neutro de troca por todos seria a solução para a interoperabilidade de dados entre os diversos SIGs. Paradoxalmente, começaram a surgir vários formatos neutros de troca no mundo. A lista a seguir, sem a intenção de ser completa, fornece uma boa idéia dos padrões neutros de troca existentes atualmente.

- Amtliches Topographisch-Kartographisches Informationssystem (ATKIS);
- BS 7666: Spatial Data Sets for Geographical Referencing;
- Catalogue Interoperability Protocol (CIP);
- Content Standard for Digital Geospatial Metadata (CSDGM);
- Data Interchange Standard for Cadastral Mapping (DFT);
- Digital and Electronic Maps Transfer Standard (DEMTS);
- Digital Geographic Information Exchange Standards (DIGEST);

- Echange de Données Informatisées Géographiques (EDIGéO);
- Formato de Intercâmbio de Dados Geográficos no Brasil (GEOBR);
- Geographic Data File (GDF);
- Geographic Tag Image File Format (GeoTIFF);
- Geography Markup Language (GML);
- IHO Transfer Standard for Digital Hydrographic Data (IHO S-57);
- INTERLIS Data Exchange Mechanism for LIS;
- International Terrestrial Reference Frame (ITRF);
- ISCGM Global Map;
- ISO 6709: Standard representation of latitude, longitude and altitude;
- ISO 8211: Specification for a data descriptive file for information interchange;
- ISO TC211: Geographic information/Geomatics;
- Israel Exchange Format (IEF91);
- JHS 117-119: EDI-based information services for geographic data;
- National Standard for the Exchange of Digital Geo-referenced Information (NES);
- Netherlands Transfer Standard for Geographic Information (NEN 1878);
- NGDF Discovery Metadata Guidelines;
- Neutral Transfer Format (NTF);
- Norma de Intercambio de Cartografia Catastral (NICCa);
- Norma de Transferencia de Informacion Geografica (NOTIGEO);
- Open Geodata Interoperability Specification (OGIS);
- ON-A2260/1: Interface for Digital Exchange of Geographic-Geometric Data;
- Receiver Independent Exchange Format (RINEX);
- Samordnet Opplegg for Stedfestet Informasjon (SOSI);
- Spatial Archive and Interchange Format (SAIF);
- Spatial Data Transfer Standard (SDTS);
- Standard Procedure and Data Format for Digital Mapping (SPDFDM);
- Vector Product Format (VPF).

A grande quantidade de formatos corrompe o objetivo final que é a adoção pelos SIGs de um único formato neutro de troca, pois, dessa maneira, a cada novo formato um novo

tradutor é necessário. Felizmente alguns formatos vêm, aos poucos, se destacando dos demais, tais como: SAIF, STDS, IHO S-57, OGIS e GEOTiff.

Existem formatos bastante genéricos como o OGIS, o SAIF e o STDS que procuram padronizar qualquer tipo de formato de dados geográficos. Outros padronizam apenas um tipo como o GEOTiff que é voltado para imagens raster. Alguns formatos são voltados para um tipo de aplicação específica como o IHO S-57 que possui representação vetorial voltada para ambiente marítimo.

Em geral, os formatos neutros, na representação vetorial, são baseados no modelo orientado a objetos. O modelo conceitual de dados desses formatos congrega conceitos de identidade do objeto geográfico, herança, agregação, associação, etc. Não possuem especificados os métodos dos objetos. A base de dados vetorial só armazena os atributos e os relacionamentos entre os objetos tal como o IHO S-57.

CAPÍTULO 3

COMMON OBJECT REQUEST BROKER ARCHITECTURE (CORBA)

Neste Capítulo, será apresentada uma pequena visão retrospectiva da evolução da tecnologia de ambiente distribuído detalhando, em seguida, CORBA e seus componentes.

3.1. HISTÓRICO

Antes do surgimento dos computadores pessoais, a tecnologia da informação era dominada por mainframes e minicomputadores que tinham um alto custo e controlavam um grande número de terminais sem poder de processamento (terminais burros). Todo o processamento era concentrado no computador central. Os softwares até então eram monolíticos, isto é, contidos em uma única aplicação.

Na medida que a informática ficou voltada para os computadores pessoais, o modelo de terminais burros começou a ser substituído pelas LANs (Local Area Network). Pela primeira vez, tarefas básicas como compartilhar arquivos e periféricos se tornaram possíveis. Grupos de trabalhos cooperativos se formaram. Logo surgiu a tecnologia Cliente/Servidor, modelo de gerenciamento onde um comutador oferece um serviço e outro que requer o serviço.

A arquitetura de sistemas distribuídos surgiu com a evolução da tecnologia Cliente/Servidor. Novas tecnologias foram desenvolvidas para esconder a complexidade da comunicação, tais como os middlewares[2] e as linguagens de definição de interface (IDL). Os principais middlewares do mercado são:

- RPC – Remote Procedure Call
- DCE – Distributed Computing Environment
- CORBA – Common Object Request Broker Architecture
- DCOM – Distributed Component Object Model
- Java/RMI – Java/Remote Method Invocation

[2] Middleware é um componente de software que reside entre as aplicações e o sistema operacional que oculta a complexidade da comunicação entre processos em uma rede.

A IDL é uma linguagem declarativa intermediária usada para definir a interface entre o cliente e o servidor em um sistema distribuído. Basicamente uma interface define um conjunto de métodos ou funções relacionadas. Cada IDL possui compiladores que geram arquivos na linguagem destino desejada contendo funções que ocultam a complexidade da comunicação remota, incluindo, dessa forma, uma camada extra, denominada Stub, entre a aplicação e a rede. Um compilador IDL compila a especificação da interface contida no arquivo de entrada IDL gerando o código fonte do Stub (por exemplo: C++ ou Java) que implementa os detalhes de comunicação de baixo nível da interface (Fig. 20).

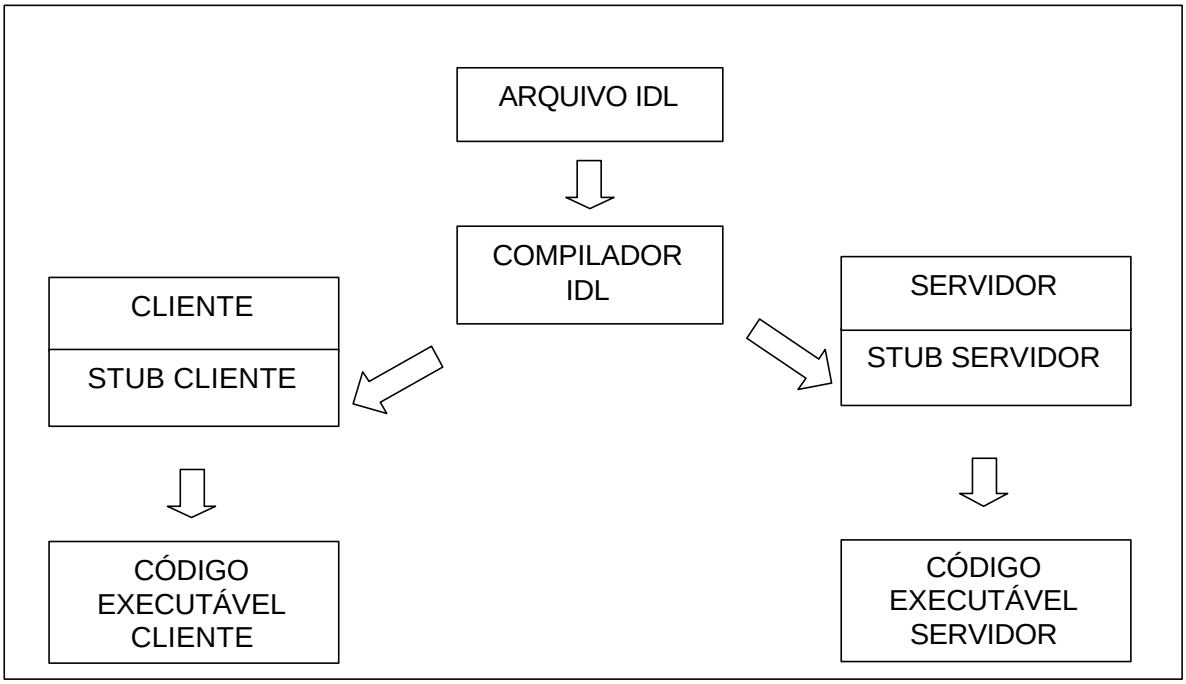


Fig. 20 - Linguagem de definição de interface.

Além de ocultar os detalhes de implementação, a linguagem de definição de interface permite a utilização de plataformas heterogêneas e linguagens de programação distintas no cliente e no servidor, ajudando a interoperabilidade de sistemas distribuídos. Esses benefícios, dentre outros, possibilitaram o sucesso dos middlewares e dos sistemas distribuídos.

Em 1989, um grupo de oito empresas (3Com, American Airlines, Canon, Data General, HP, Philips, Sun e Unisys) fundou o Object Management Group (OMG) a fim de

prover uma estrutura comum para o desenvolvimento de aplicações orientadas a objetos. Hoje é um dos maiores grupos de padronização contando com mais de 800 filiados no mundo todo, incluindo a Microsoft, que opera apenas como observadora. A função do OMG é gerar especificações técnicas para a indústria de software que tornam a interoperabilidade entre objetos possível. O OMG não produz software.

Em 1991, o OMG lançou o CORBA 1.0, a primeira versão da arquitetura CORBA que especifica um modelo onde os métodos de um objeto podiam ser invocados remotamente. Esse mecanismo é implementado através de uma camada intermediária chamada ORB (Object Request Broker), localizada entre o objeto e o sistema operacional acrescido de algumas funções de comunicação de rede. Logo começaram a surgir as primeiras implementações ORBs no mercado. Nesta primeira versão, o OMG não deixou claro como objetos de diferentes implementações ORBs poderiam se comunicar. Dessa forma, apenas os objetos criados sobre a mesma implementação podiam interagir.

A versão CORBA 2.0, em 1994, trouxe um grande avanço sobre o seu antecessor com a definição do Internet Inter-ORB Protocoll (IIOP), um protocolo de transferência de dados padronizado para a comunicação entre diferentes ORBs, tornando-se a solução definitiva para interoperabilidade entre objetos. O IIOP permite a comunicação entre objetos que executam sobre diferentes ORBs, independentemente do fornecedor, do sistema operacional, da linguagem de programação e da plataforma utilizada. Desde então, várias versões intermediárias da norma têm sido lançadas. A mais recente é o CORBA 2.4, lançada no final de 2000, que permite a interoperabilidade com outros sistemas distribuídos, suporte de interação assíncrona entre objetos, suporte de mobilidade de objetos, suporte a tempo real, etc.

A IDL do OMG (ou IDL CORBA) é uma linguagem neutra e puramente declarativa. Isto significa que ela não depende de sistema operacional e de linguagem de programação e, além disso, não fornece nenhum detalhe sobre a implementação dos objetos, ou seja, não possui nenhum mecanismo de construção de estruturas algorítmicas. A sintaxe da IDL CORBA é semelhante a C++, embora use palavras-chave diferentes. Permite a declaração de constantes, tipos de dados, operações, interfaces e exceções. A IDL suporta ainda herança múltipla e definição de espaços de nomes. É bastante completa e concisa. A linguagem inteira é descrita no capítulo 3 da especificação CORBA [18] que pode ser obtida em http://www.omg.org.

3.2. OBJECT MANAGEMENT ARCHITECTURE

O OMG definiu também uma arquitetura mais genérica denominada Object Management Architecture (OMA) sendo o ORB um de seus componentes. Os principais componentes do OMA são o ORB, os Serviços CORBA (Common Object Services), as Facilidades CORBA (Common Facilities) e os Objetos de Aplicações (Fig. 21).

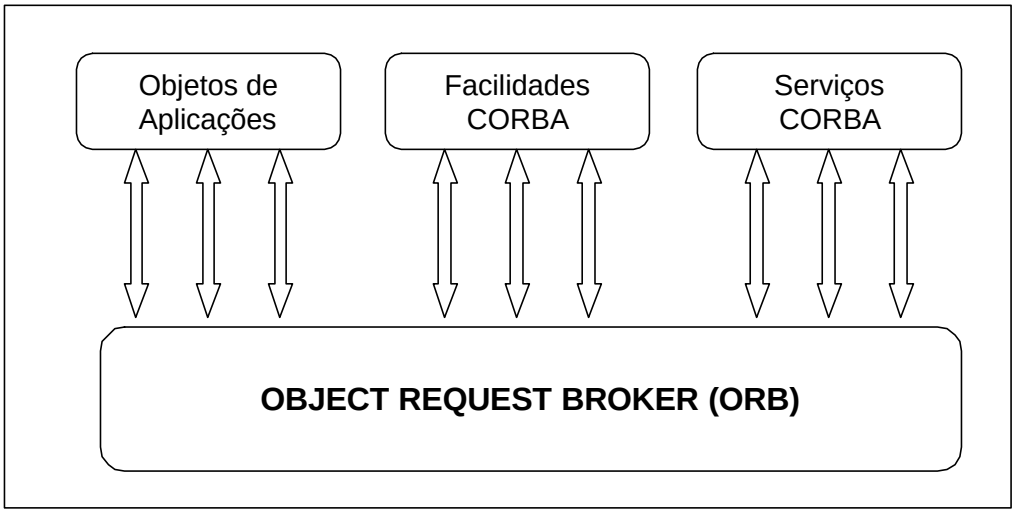


Fig. 21 - Modelo de referência OMA.

3.2.1. OBJETOS DE APLICAÇÕES

São os objetos da aplicação que farão a comunicação entre si. Representam o topo da hierarquia OMA. São objetos voltados a uma aplicação específica e não necessitam de nenhum tipo de padronização. Essa categoria identifica objetos que não são afetados pelos esforços de padronização do OMG. A literatura CORBA divide os Objetos de Aplicações em dois tipos básicos: o objeto cliente, que requisita o serviço; e o objeto servidor, mais conhecido como implementação do objeto (Object Implementation), que fornece o serviço através de seus métodos.

3.2.2. ORB

O ORB fornece a infraestrutura básica de chamada a procedimentos remotos. É o componente mais importante da arquitetura OMA, pois é através dele que os objetos se

comunicam em um ambiente distribuído e heterogêneo. O ORB é definido pela especificação CORBA e não pela especificação OMA.

A Fig. 22 mostra a invocação de um método remoto via ORB. O ORB é responsável pela localização do objeto, pelo envio de parâmetros de entrada e pelo retorno dos parâmetros de saída. A interface que o cliente invoca informa apenas quais os métodos que são oferecidos pela implementação, ela não possui nenhuma dependência sobre a localização do objeto, a linguagem de programação utilizada, o sistema operacional e tipo de máquina usada no lado do servidor. A implementação do objeto, por sua vez, pode estar no mesmo processo do cliente, em processos diferentes na mesma máquina e até em máquinas diferentes, pois o ORB esconde toda a complexidade da comunicação, local ou em rede, entre o cliente e servidor.

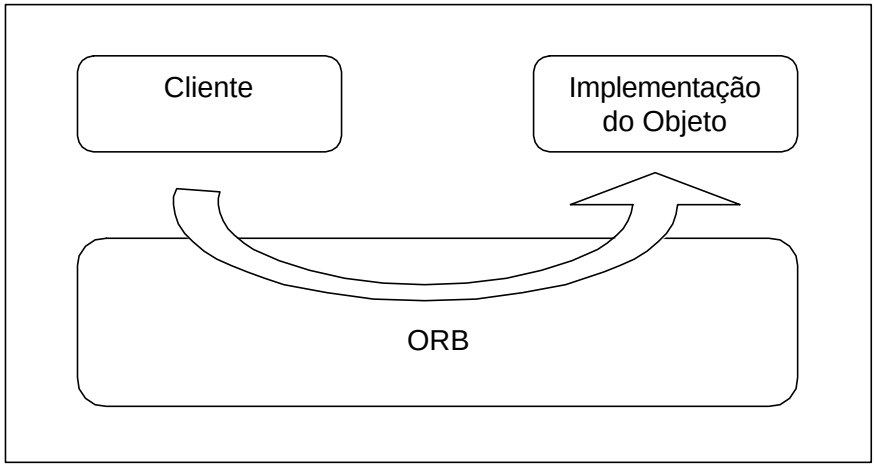


Fig. 22 - Requisição via ORB.

O cliente faz uma requisição a um objeto específico usando uma referência do objeto. A referência do objeto desejada pode ser obtida, por exemplo, através do serviço de nomes do CORBA. De posse dessa referência, é possível então invocar um método da implementação do objeto. Os parâmetros de entrada são empacotados (marshal) pelo ORB, no lado do cliente e desempacotados (unmarshal) no lado da implementação. O método do objeto é então invocado. Este objeto ao final empacota de volta os parâmetros de saída para o cliente. Os processos de empacotamento e desempacotamento geram dados formatados de maneira independente da plataforma e da ordenação de bytes do processador. Isso permite, por

exemplo, a um cliente rodando em uma máquina Macintosh invocar métodos de um objeto rodando em UNIX e vice-versa. A norma CORBA 2.0 também especificou um formato padrão para referência do objeto denominada Interoperable Object Reference (IOR).

Na Fig. 23 é explicitada a estrutura interna de um ORB com suas interfaces com o cliente e com a implementação do objeto. A seguir é apresentada uma breve descrição de seus componentes:

- Client Stub – é usado pelo objeto cliente para empacotar e desempacotar os parâmetros e fazer a chamada ao objeto remoto. Um cliente deve ter um Stub para cada interface utilizada no servidor;

- Skeleton – É o complemento do Stub do lado do servidor. Tanto o Stub quanto o Skeleton são criados usando a linguagem de definição de interface CORBA IDL;

- Dynamic Invocation Interface (DII) – É usado quando o Stub de uma interface não está disponível ao cliente em tempo de execução. Em essência, o DII constrói, em tempo de execução, o Stub requerido a partir das informações obtidas do repositório de interfaces. O DII consiste de várias funções usadas para localizar os meta-dados que definem a interface, para gerar uma lista de parâmetros, para executar a chamada remota e para obter de volta os resultados;

- Dynamic Skeleton Interface (DSI) – Permite que uma implementação de objeto receba invocações CORBA sem saber, em tempo de compilação, qual interface IDL ele implementa. É análogo ao DII no lado do servidor. Os clientes por sua vez não necessitam saber se o servidor implementa um esqueleto estático ou dinâmico;

- Interface Repository – O repositório de interfaces é uma base de dados que possui as informações sobre as interfaces IDL que podem ser acessadas em tempo de execução pelo cliente;

- Implementation Repository – O repositório de implementações permite ao ORB ativar dinamicamente um objeto sempre que um de seus métodos for invocado. Dessa maneira o objeto não precisa estar ativo em memória para poder ser invocado;

- ORB Interface – Consiste de algumas funções locais que podem ser de interesse para uma aplicação. Por exemplo, CORBA fornece uma função que converte uma referência ao objeto em string e vice-versa. Essa função pode ser muito útil quando se necessita armazenar referências de objetos;

- Object Adapter – É uma interface entre o núcleo do ORB e os vários tipos de implementações de objetos. Serviços fornecidos pelo Object Adapter incluem: geração e interpretação de referências de objetos, ativação e desativação de servidores, mapeamento das requisições para os objetos correspondentes e colaboração com o DSI na invocação dinâmica do servidor. Após a versão CORBA 2.2 passou a ser conhecido como Portable Object Adapter (POA).

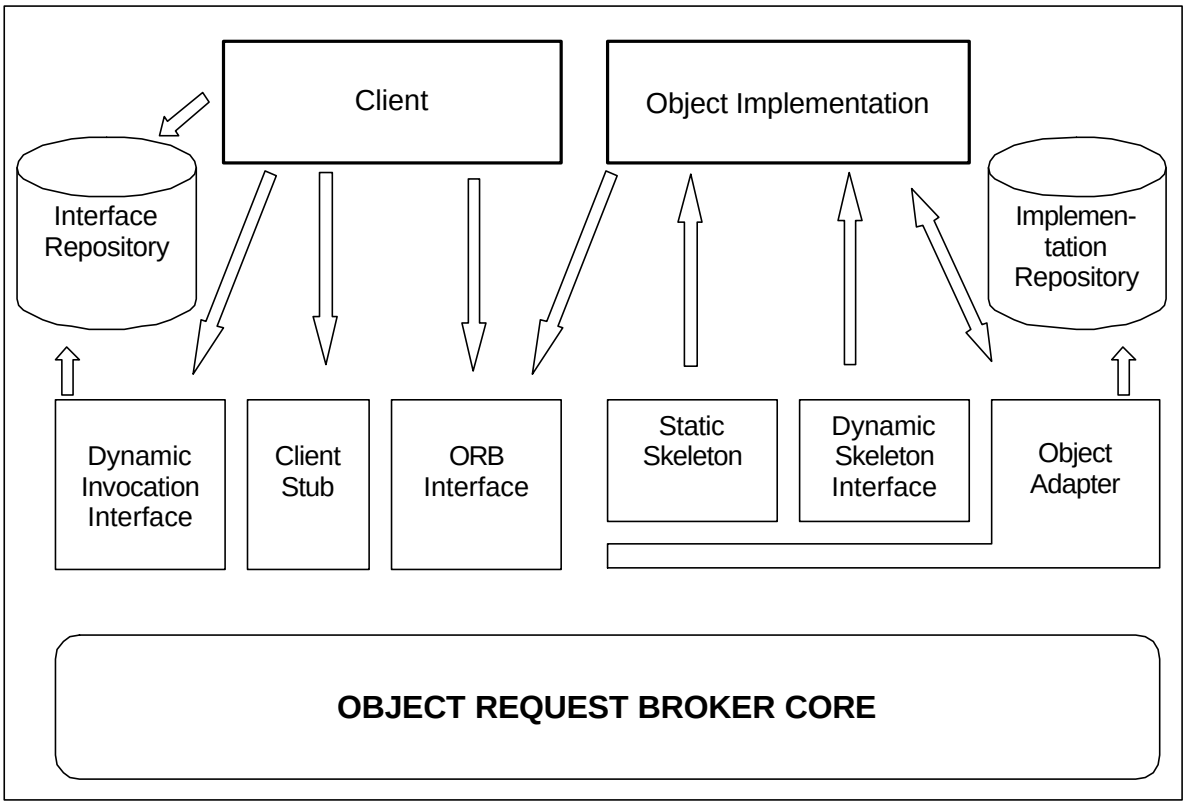


Fig. 23 – Estrutura de um ORB.

3.2.3. SERVIÇOS CORBA (*OBJECT SERVICES*)

Os serviços CORBA fornecem funções genéricas, ao nível de sistema, que complementam a funcionalidade do ORB e que são agrupados em componentes. São serviços que não são voltados para um tipo específico de aplicação. Atualmente a OMG oferece 16 serviços CORBA: Serviço de Nomes, Eventos, Ciclo de Vida, Persistência, Transação, Concorrência, Relacionamentos, Externalização, Consulta, Licenciamento, Propriedades, Tempo, Trader, Segurança, Coleções e Gerenciamento de Versão.

O Serviço de Nomes permite atribuir ao objeto servidor um nome qualquer, permitindo, dessa forma, que o objeto cliente possa localizar o servidor conhecendo-se apenas o seu nome. O serviço de Transações assegura as propriedades de atomicidade, consistência, isolamento e durabilidade na comunicação ORB. É baseada no protocolo commit de duas fases. Uma descrição mais detalhada de todos os serviços CORBA pode ser obtida em ORFALI [20].

3.2.4. FACILIDADES CORBA (*COMMON FACILITIES*)

São serviços adicionais que simplificam o desenvolvimento de aplicações distribuídas. As facilidades CORBA são um conjunto de objetos pré-definidos orientados ao usuário final que não são considerados tão fundamentais como os serviços CORBA. Por exemplo, um gerenciador de sistema ou um correio eletrônico poderiam ser classificados como serviços das facilidades CORBA.

As facilidades CORBA são coleções de componentes que oferecem serviços diretamente aos objetos da aplicação. Note que a diferença marcante entre os serviços e as facilidades está no nível em que cada um se aplica: os serviços estão intimamente relacionados ao ORB, enquanto as facilidades estão intimamente relacionadas aos objetos da aplicação.

As facilidades CORBA são agrupadas em duas categorias:

- Facilidades Horizontais – podem ser usados pela maioria dos sistemas como os serviços de gerenciamento de sistemas, de gerenciamento de informações, de gerenciamento de tarefas e os serviços de interface com o usuário;

- Facilidades Verticais – são voltadas a aplicações específicas como correio eletrônico, simulações distribuídas, exploração de óleo e gás, telecomunicações, saúde, etc.

CAPÍTULO 4

OBJETOS GEOGRÁFICOS DISTRIBUÍDOS

Este capítulo possui três objetivos principais. O primeiro busca descrever o problema da necessidade de distribuição da informação geográfica no âmbito naval. O segundo busca relacionar os conceitos que envolvem a informação geográfica e sistemas distribuídos. O terceiro, procura mostrar que a visão de objetos pode ser naturalmente associada ao dado geográfico.

4.1. O AMBIENTE NAVAL

Cerca de 72% da superfície da terra é coberta por água. Os mares e oceanos são os principais meios de transportes que agregam 83% do transporte internacional mundial que percorrem milhares de rotas marítimas diferentes. Não há país que não identifique interesses no mar, resultantes de anseios, necessidades, possibilidades e cultura de um povo. Interesses esses que se relacionam de alguma forma com a navegação, o transporte aquaviário, a pesca, a extração de petróleo, as indústrias afins, a população que o integra, os limites políticos e o meio ambiente. Como os mares têm sido caminho natural de forças antagônicas durante as crises, torna-se fundamental dispor-se de um poder naval capaz de assegurar um grau de dissuasão compatível com o vulto dos interesses em jogo [15]. Nesse contexto, é de fundamental importância prover ao poder naval o acesso a vários tipos de informações necessárias ao efetivo exercício da soberania nacional.

A superfície do planeta é o palco onde se desenvolvem os choques bélicos e muitas vezes o domínio de uma fração desta é a causa do próprio conflito. A geografia é um saber estratégico estreitamente ligado a práticas políticas e militares e são tais práticas que exigem um conjunto articulado de informações extremamente variadas. A produção de um mapa, isto é, a conversão de uma realidade mal conhecida em uma representação abstrata, confiável, eficaz é uma operação difícil, longa e onerosa e serve como um instrumento de poder sobre este espaço e as pessoas que ali vivem. Julga-se que após a confecção de mapas relativamente precisos para todos os países e regiões, os militares não têm mais a necessidade de recorrer ao conhecimento geográfico. Nada é mais falso porque “as coisas” se transformam rapidamente: se a topografia evolui muito lentamente, o traçado das vias de circulação, as formas do habitat, as instalações industriais se modificam em um ritmo bem mais rápido e é preciso levar em consideração essas transformações para estabelecer as táticas e estratégias militares [14].

A esquadra brasileira é composta de cerca de trinta navios, sete submarinos, setenta helicópteros e alguns aviões. Quando envolvidos em qualquer operação, os navios, os submarinos, os helicópteros e aviões são agrupados e organizados visando o atendimento dos requisitos da missão estabelecida, quando, então, o conjunto recebe a denominação de Força-Tarefa. A Força-Tarefa tem capacidade de comunicação de voz e dados entre seus integrantes e com postos em terra como agências ou missões terrestres (Fig. 24).

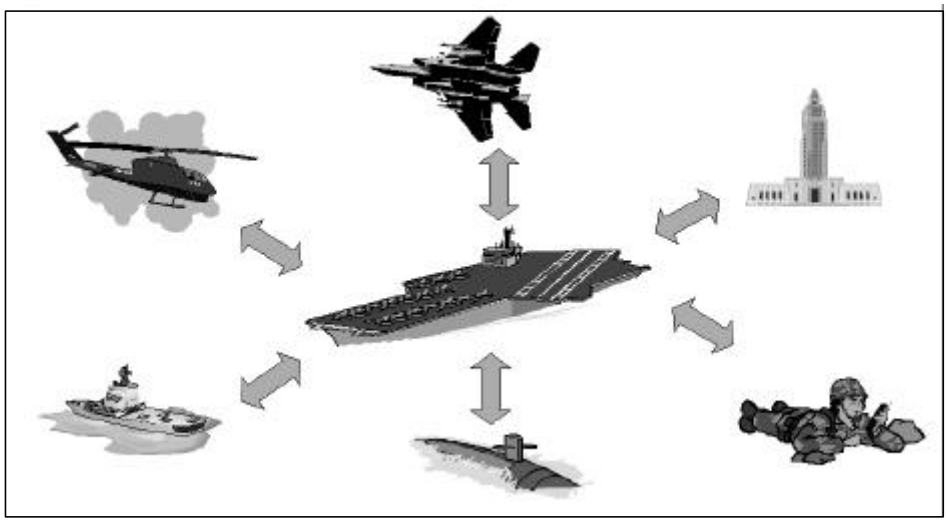


Fig. 24 - Ambiente Naval.

A comunicação digital de dados entre computadores permite a troca mais eficiente da informação do que a permitida pela linguagem humana. A comunicação de dados é feita através de data-links. Por um data-link pode trafegar uma série de mensagens padronizadas que permitem até a comunicação com forças de países aliados a fim de se apoiarem mutuamente numa operação em conjunto. A troca destas informações permite uma coordenação contínua e harmoniosa entre o controle tático e sistemas de defesas táticas envolvidas. Isto minimiza a interferência mútua e aumenta significativamente a efetividade do sistema em operações conjuntas.

Partindo para uma visão interna ao navio o panorama não muda. Existe a necessidade de disseminar a informação para diversos compartimentos do mesmo. A diferença está no meio físico e no protocolo utilizado (TCP/IP). As diversas máquinas do navio são conectadas através de um hub simples. A Fig. 25 mostra os sistemas de bordo que se beneficiam com a informação geográfica distribuída.

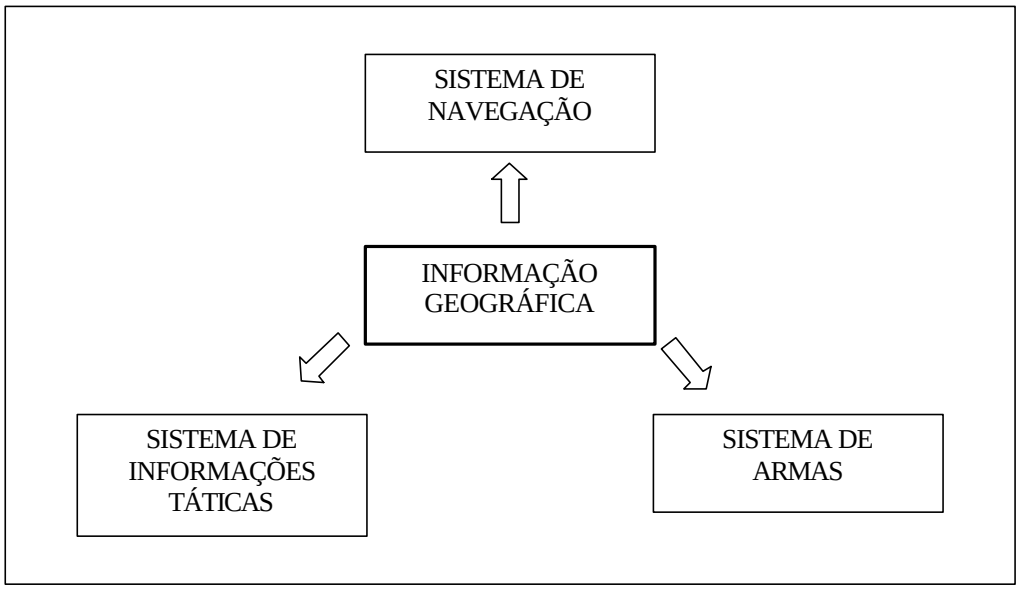


Fig. 25 - Distribuição da informação geográfica.

Por trás de toda essa estrutura de comunicações de dados ainda não existe um meio de disseminar a informação geográfica tanto dentro quanto fora do navio. A rede de comunicações de dados é utilizada somente para o tráfego de informações táticas e de controle. Falta ainda a definição de uma arquitetura apropriada de software capaz de distribuir a informação geográfica para todos as unidades das forças aliadas e que permita também a integração com outros sistemas navais de bordo, tais como o sistema tático e o sistema de armas.

Com a evolução atual dos SIGs e modernas técnicas de software distribuído já é possível distribuir de uma forma racional a informação geográfica num ambiente bastante diversificado e heterogêneo, como veremos nas próximas seções.

4.2. INFORMAÇÃO GEOGRÁFICA DISTRIBUÍDA

Tem sido mostrado que os bancos de dados distribuídos possuem várias vantagens sobre os centralizados [13, 20] e isso fica ainda mais evidente quando se trata da informação geográfica. Abordaremos a seguir essas vantagens [13].

A informação geográfica em si é inerentemente distribuída. Não existe um órgão único central responsável pela produção e distribuição da informação geográfica. As organizações cartográficas que fazem a manutenção do dado geográfico estão distribuídas por todo o mundo. Em geral, o levantamento cartográfico é realizado em unidades administrativas como estados, municípios, blocos, etc. Várias organizações realizam a coleta de dados local e trocam dados com outros órgãos.

As consultas em um SIG são em geral realizadas na base de dados local. A fração de consultas que acessam dados remotos não é grande suficiente para justificar o armazenamento de todos os dados em cada localidade.

A adição de novos distribuidores de informação geográfica é mais fácil em sistemas descentralizados de gerenciamento de dados. Novos fornecedores podem facilmente e a qualquer momento se conectar à rede de distribuição de dados disponibilizando seus dados.

A informação geográfica pode conter grandes quantidades de dados. Para manter bases de dados monolíticas, todos os dados precisam ser copiados para todas as localidades, o que representa uma fenomenal quantia de dados transmitidos e enormes capacidades de armazenamento em todas as localidades.

Os custos de atualização de dados geográficos são grandes. Atualizações periódicas em todas as localidades que possuem uma cópia da base de dados é praticamente inviável. Uma alteração qualquer no dado geográfico necessita ser propagado para todas as localidades, gerando questões de acesso a todas as cópias do dado, o que pode provocar inconsistência entre bases de dados diferentes. Uma atualização em um sistema distribuído é muito mais simples e é realizada com menor tráfego de dados.

Como vimos, existem várias vantagens em distribuir a informação geográfica em ambientes onde cada localidade possui a sua base de dados própria. Contudo, para a modelagem e implementação de sistemas distribuídos faz-se necessário uma tecnologia mais complexa do que a exigida nos sistemas centralizados, como controle e armazenamento da distribuição e localização dos dados, bem como a garantia da consistência dos mesmos [21]. A implantação de um sistema distribuído requer, portanto, que vários desafios sejam antes vencidos. Veremos a seguir alguns desses desafios.

Existe heterogeneidade de vários níveis em um ambiente distribuído. Diversidade de protocolo de rede, hardware, sistema operacional, linguagem de programação e programadores são os mais evidenciados. A utilização de um middleware apropriado permite lidar com a heterogeneidade do sistema. O middleware fornece uma abstração para a programação de aplicações em rede, ocultado a complexidade dos vários níveis do sistema e proporcionando uma ferramenta única e padronizada de programação. O uso desta tecnologia permite que cada localidade utilize seu próprio software bem como a estrutura e modelos de dados próprios de modo transparente para o sistema como um todo. O desacoplamento é também garantido, pois componentes poderão mudar a qualquer momento mantendo apenas intacta a interface com o resto do sistema.

A abertura é outro desafio na construção de sistemas distribuídos. Um sistema aberto usa mecanismos de comunicação padronizados que permite a integração de componentes de software e hardware de várias origens. Faz-se necessário também a publicação da interface de acesso aos dados compartilhados do sistema. O modelo de sistemas abertos possibilita e favorece o compartilhamento de dados, recursos e ferramentas entre diferentes usuários e aplicações. Na área de SIG, ainda dominada por sistemas fechados, significa mudar aplicações, estruturas e dados internos por uma arquitetura aberta, interoperável e que interconecte diferentes SIGs a uma base de dados geográfica distribuída [21].

A transparência é outro requisito importante na implementação de sistemas distribuídos. Transparência é a capacidade de ocultar a distribuição da informação do usuário e do programador de maneira a visualizar o sistema como um todo em vez de uma coleção de componentes independentes. A transparência pode se refletir em vários níveis:

- No acesso - permite o acesso a recursos locais ou remotos utilizando as mesmas operações;

- Na localização - permite o acesso aos recursos sem ter conhecimento de sua localização;

- Na concorrência - permite que vários processos operem concorrentemente sobre os recursos compartilhados sem que haja interferência entre eles;

- Na replicação - permite a utilização de múltiplas instâncias de recursos a fim de melhorar a confiabilidade e o desempenho sem que os usuários e programadores tenham conhecimento;

- Na falha - permite a recuperação de falhas de hardware ou software. Exemplo: reenvio de pedido a outro servidor;

- No crescimento incremental - permite a expansibilidade em escala de um sistema sem alterar a estrutura ou algoritmos das aplicações.

Novamente a escolha do middleware adequado vai permitir um alto grau de transparência do sistema.

4.3. INFORMAÇÃO GEOGRÁFICA E A ORIENTAÇÃO A OBJETOS

Os sistemas de informações atuais oferecem muito pouco em termos da semântica dos dados sendo processados. Basicamente eles foram projetados para lidar com dados e não com a informação, pois a semântica é difícil de ser mapeada durante a entrada de dados em um modelo relacional [13]. Além disso, os SIGs raramente foram projetados para aproveitar a distribuição inerente dos dados geográficos. SIGs distribuídos oferecem várias vantagens comparados aos sistemas monolíticos, tirando proveito da distribuição tanto do processamento como do gerenciamento do dado espacial. A orientação a objetos surge como uma boa candidata na solução destes problemas, tanto na questão da semântica dos dados quanto na disseminação e distribuição da informação geográfica.

O paradigma usual de gerenciamento da informação de propósitos gerais é o modelo relacional. O sucesso comercial de sistemas baseados em tal modelo também se refletiu nos SIGs que logo o adotaram como a estrutura de dados básica para o armazenamento da informação geográfica. Dessa forma foi possível liberar os projetistas da tarefa de construir e manter uma substancial e complexa parte do sistema deixando a cargo dos Sistemas Gerenciadores de Banco de Dados (SGBD). Os SGBDs tornaram-se uma ferramenta poderosa no gerenciamento de dados controlando o acesso concorrente aos dados feito por múltiplos usuários, prevenindo a perda acidental de dados, e promovendo acessos restritos e seguros aos dados [6].

O modelo relacional atende satisfatoriamente a grande maioria dos sistemas de informações tradicionais, porém a maior parte dos SIGs que implementaram o modelo relacional tipicamente adotaram uma arquitetura híbrida onde os atributos espaciais e os atributos não-espaciais são armazenados e gerenciados em estruturas independentes. Tais sistemas mantêm os dados não-espaciais armazenados em bancos de dados convencionais e gerenciados por um SGBD, enquanto que os dados espaciais são armazenados em arquivos convencionais com estrutura de dados própria.

MITROVIC [16] justifica a utilização do modelo híbrido afirmando que trabalhos iniciais de implementar banco de dados espaciais em um modelo puramente relacional mostraram que tal abordagem, embora possível, é insatisfatório devido ao baixo desempenho. Como o modelo relacional não é adequado para representar dados espaciais, a consulta espacial relativa a um objeto é propagada sobre vários relacionamentos e, além disso, muitas operações de junção são necessárias a fim de recriar a estrutura espacial do objeto.

No início da década de 90 foi lançado o primeiro SIG utilizando o modelo híbrido: o ARC/INFO da ESRI. Devido ao grande sucesso comercial do ARC/INFO, muitos SIGs que se seguiram também adotaram o modelo híbrido em sua estrutura. Atualmente a grande maioria dos SIGs utilizam o modelo híbrido como forma de armazenar a informação geográfica, dentre eles o ARC/VIEW, o SPRING e o MGE.

Segundo CÂMARA [3], a principal vantagem dessa estrutura é poder utilizar SGBDs relacionais comerciais, porém os requisitos de integridade, concorrência, gerência de transações e otimização de consultas ficam prejudicados já que parte da consulta fica de fora

do SGBD. Faz-se necessário a construção de subsistemas, externos ao SGBD, para processar a consulta espacial do dado geográfico de acordo com a arquitetura de armazenamento de cada fabricante.

KUMAR [13] argumenta que a abordagem híbrida não possui um gerenciamento rigoroso e nem um controle adequado da integridade e segurança dos dados. Além disso, o acesso de múltiplos usuários e o gerenciamento de concorrência não é disciplinado. Também, e talvez o mais importante, o modelo híbrido não explora a natural distribuição dos dados geográficos. Se benefícios podem ser extraídos da tecnologia de banco de dados distribuídos, então os SIGs devem ser desenvolvidos para lidar com a complexidade de gerenciamento banco de dados distribuídos sem ser impedido por uma abordagem que armazena parte dos dados fora do SGBD, inacessível a qualquer função externa. Em tais sistemas híbridos, a otimização das consultas é uma questão muito complicada já que as mesmas têm que ser particionadas em componentes espaciais e não espaciais antes de qualquer avaliação.

A abordagem orientada a objetos tem sido promovida nas áreas onde as tecnologias puramente relacionais e híbridas não atendam adequadamente aos requisitos do sistema. Existe uma forte tendência do uso da orientação a objetos em aplicações geográficas [6,13,21] e outras aplicações não-convecionais que possuem um modelo de dados bastante complexo, tais como CAD/CAM, CASE e multimídia.

Além de poder representar estruturas de dados complexas, a orientação a objetos permite especificar o comportamento dos objetos geográficos e possui algumas características que são úteis para aumentar a compatibilidade de informações, permitindo um nível maior de interoperabilidade entre sistemas [8].

Segundo PIRES [22], a tecnologia da orientação a objetos é apontada como um dos principais requisitos das aplicações modernas, pois é bastante apropriada tanto no processamento distribuído quanto na arquitetura cliente-servidor, já que os métodos de um objeto podem ser vistos como serviços oferecidos pelo mesmo. Adicionalmente, a utilização de objetos em um ambiente distribuído atende requisitos de desacoplamento e heterogeneidade. A heterogeneidade é obtida pelo fato que a interação entre objetos depender somente da interface externa e não da implementação interna do objeto. O desacoplamento é

preservado porque os objetos operam de maneira independente e transparente, contanto que as interfaces permaneçam inalteradas.

O processo de consultas para um sistema distribuído e orientado a objetos é completamente diferente das consultas utilizadas em sistemas convencionais de banco de dados. Existem duas maneiras nas quais as consultas podem ser processadas. A abordagem mais simples é coletar, dos seus respectivos locais, todos os objetos envolvidos, e daí proceder a consulta. A melhor abordagem é enviar algumas informações adicionais junto com o dado requerido para o local remoto que poderá selecionar um conjunto menor de objetos envolvidos na consulta e realizar a consulta sobre esse pequeno conjunto. Este é a base da computação cooperativa onde diferente segmentos da solução são gerados de uma maneira distribuída. A estrutura cooperativa permite uma avaliação paralela das consultas.

CAPÍTULO 5

ARQUITETURA PARA DISTRIBUIÇÃO DA INFORMAÇÃO GEOGRÁFICA BASEADA NO S-57

Este capítulo visa construir uma arquitetura de software que possibilite a distribuição da informação geográfica de forma aberta e transparente. Dessa forma o capítulo transcorrerá sobre:

- A melhor tecnologia de ambiente distribuído a ser empregada no modelo;
- A arquitetura do sistema e o modelo S-57;
- As interfaces de acesso remoto aos objetos.

5.1. O *MIDDLEWARE*

No capítulo 3 foram relacionados alguns middlewares no mercado de sistemas distribuídos. A adoção de um ou de outro depende em grande parte da arquitetura desejada para o sistema. Nesse sentido CORBA é o que melhor se adapta ao nosso projeto, visto que a tecnologia de objetos distribuídos se ajusta perfeitamente a um ambiente de programação orientado a objetos. Em contrapartida, RPC e DCE foram projetados para ambiente procedural de difícil utilização em sistemas OO. Além disso, a interface RPC não é padronizada e a sua utilização atrela o sistema ao fabricante do RPC utilizado. CORBA, por sua vez, oferece várias características de ambientes OO, tais como:

- Encapsulamento de dados e funções que manipulam dados internos ao objeto garantindo assim a ocultação de dados, pois a única maneira de acessar os dados do objeto é através dos métodos públicos do objeto declarados na interface;

- Herança de interfaces. É o mecanismo que permite a especialização ou a generalização de interfaces da mesma maneira que a programação OO permite com as classes;

- Polimorfismo, que é a habilidade de tratar uma invocação a um método de maneira diferente dependendo do tipo de objeto que foi invocado.

Componentes DCOM não são objetos clássicos [20]. Não é possível estender a capacidade de uma classe DCOM via herança. Uma classe DCOM é na realidade apenas um

mecanismo de encapsulamento. DCOM permite a construção de classes complexas usando mecanismos de delegação. Por exemplo, DCOM fornece um mecanismo chamado agregação que permite que um objeto ofereça múltiplas interfaces. Usando agregação é possível um componente mais externo representar a interface dos componentes internos. Na melhor das hipóteses, é um mecanismo muito pobre de herança. DCOM é uma especificação proprietária da Microsoft e, como RPC, torna o sistema dependente do fabricante. DCOM também é fortemente direcionado para ambiente Windows prejudicando a utilização do sistema em ambientes heterogêneos.

Apesar da grande facilidade de programação em JAVA/RMI, o protocolo RMI não é estendido para as demais linguagens de programação. Um servidor escrito em JAVA/RMI obriga que os clientes sejam escritos na mesma linguagem. Embora a JavaSoft tenha desenvolvido um novo pacote para plataforma Java 2 denominado "RMI-IIOP" (RMI sobre IIOP), o protocolo básico continua sendo CORBA (IIOP). A JavaSoft incorporou apenas um recurso na linguagem Java que permite a utilização de objetos CORBA sem a utilização de pacotes externos de programação. Dessa maneira o RMI-IIOP e CORBA se confundem já que ambos usam o mesmo protocolo.

5.2. O MODELO CONCEITUAL DE DADOS S-57

S-57 [9,10] é um formato padrão de armazenamento em meio magnético de cartas náuticas eletrônicas criado pela IHO (International Hydrographic Organization) utilizado para troca de dados entre os SIGs. Concebido originalmente para padronizar qualquer tipo de dados utilizados em hidrografia, só está definido, atualmente, o perfil para cartas náuticas eletrônicas vetorizadas, conhecido como perfil ENC (Eletronic Navigation Chart). É um dos raros padrões internacionais de troca de dados adotado pela maioria das autoridades cujo interesse tem aumentado nos últimos anos.

O S-57 é um formato complexo que envolve um modelo de dados específico com hierarquia de objetos, compartilhamento de geometrias com estrutura arco-nó, vasto número de atributos, etc. É encapsulado de acordo com a linguagem descritiva de dados ISO8211. O perfil ENC do S-57 assume uma abordagem incremental a fim de manter o sistema ECDIS a bordo sempre atualizado. Requer uma habilidade adicional do sistema para identificar, a partir

do arquivo original e dos arquivos de atualização, as alterações realizadas nas feições e na topologia base.

Apesar de o S-57 ser concebido para troca de dados entre SIGs, essa transferência de dados entre formatos diferentes causa diversos problemas de perda de informações motivada pelas diferenças semânticas, sintáticas e esquemáticas entre os formatos utilizados [8]. Para solucionar este problema, a arquitetura de software utilizada neste trabalho utilizará a mesma estrutura de objetos definidos no S-57 evitando com isso a necessidade de transferência de dados entre formatos diferentes.

A arquitetura S-57 define objeto S-57 (GeoObjeto) como uma entidade que pode representar uma feição (objeto com atributos não-espacias) ou um objeto do tipo espacial (nós e arestas). Um objeto espacial pode descrever para a feição três tipos de geometrias diferentes como ponto, linha ou área. Uma carta náutica pode ser definida como um conjunto de geo-objetos e metadados inter-relacionados. Cada objeto S-57 possui identificação única dentro de uma mesma carta náutica. No apêndice A encontra-se a lista completa de feições definidas pela IHO para o S-57.

A arquitetura de distribuição da informação geográfica que será descrita neste trabalho tem como ponto de partida o modelo conceitual de dados definido no S-57. A figura abaixo mostra o primeiro nível de detalhamento do S-57. Os atributos das classes foram omitidos a fim de facilitar o entendimento. Em IHO [10], são apresentados todos os atributos das classes com uma descrição detalhada de cada um.

Como o S-57 define apenas um modelo de dados, os métodos das classes são definidos pelas aplicações que utilizam o S-57.

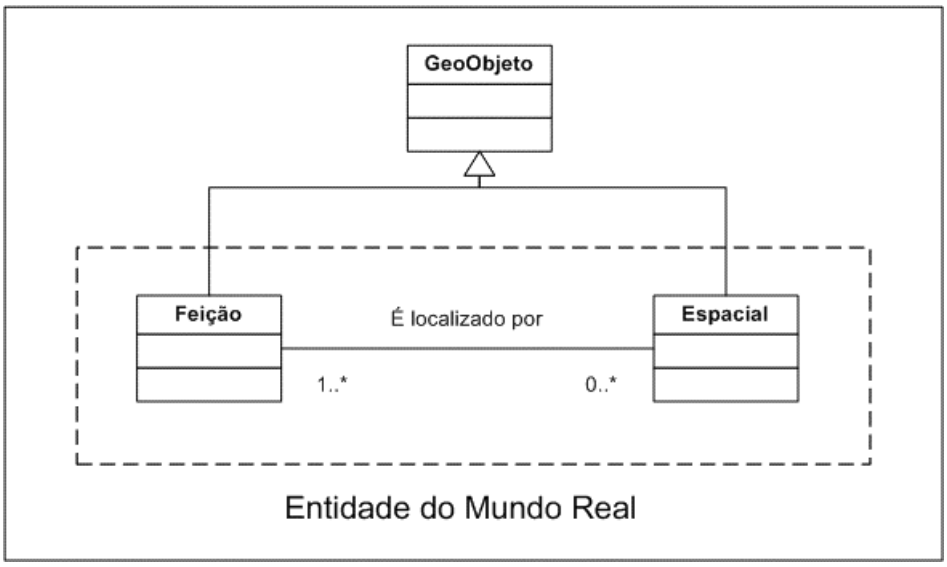


Fig. 26 - Modelo conceitual S-57 nível 1.

A classe “Feição” representa os dados não espaciais de uma entidade do mundo real. A classe “Espacial” representa os dados espaciais. O conjunto de feição e o seu respectivo dado espacial representam uma entidade geográfica do mundo real. Uma feição pode estar associada a zero ou mais dados espaciais, que podem ser um nó ou um conjunto de arestas formando uma linha (polígono aberto) ou uma área (polígono fechado). O dado espacial necessariamente deve possuir pelo menos uma feição associada. Como o S-57 proíbe a duplicação de geometrias, um mesmo dado espacial pode estar associado a várias feições como é o caso, por exemplo, de dois polígonos adjacentes que compartilham uma mesma aresta. A aresta comum estará associada às duas feições e é dessa maneira que é representado o relacionamento topológico de vizinhança, pois dois polígonos vizinhos compartilham a aresta comum.

Como existem feições que representam dados relativos à carta náutica em geral, tais como datum, declinação magnética, limites da carta, agência produtora, etc., é permitido que tais feições não possuam dados espaciais relacionados.

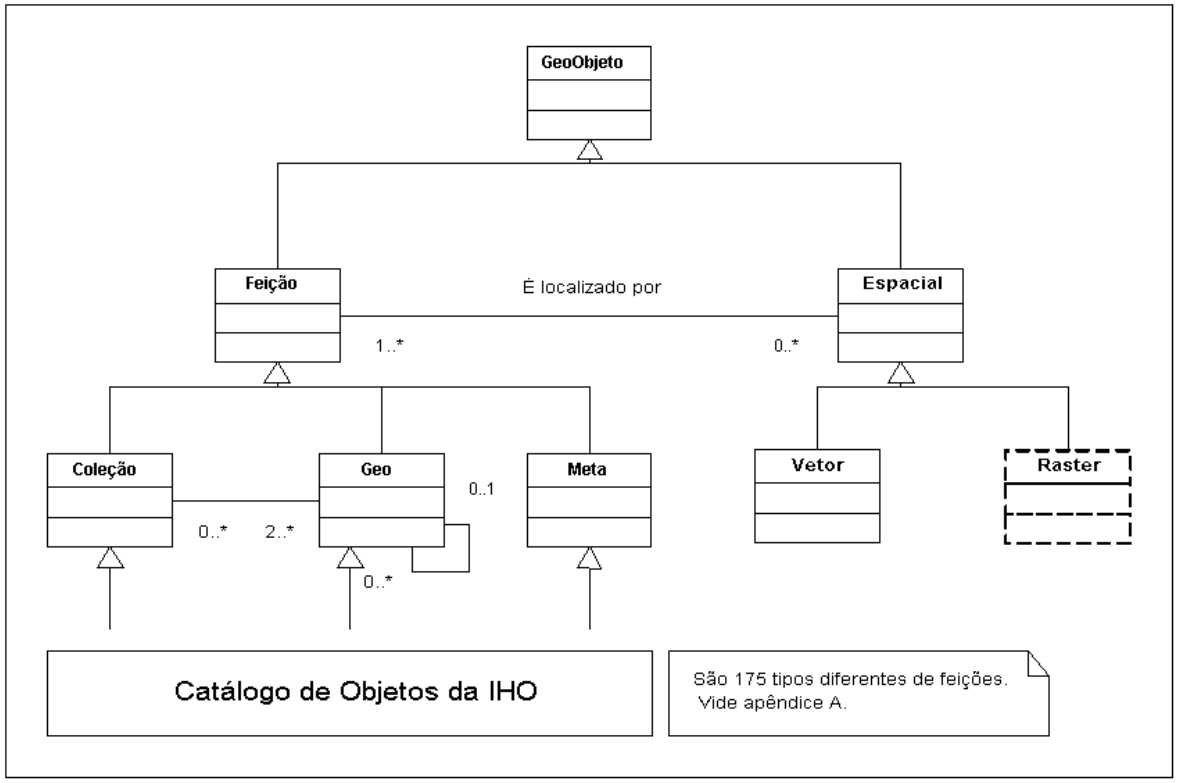


Fig. 27 - Modelo conceitual S-57 nível 2.

Para facilitar a troca de dados não espaciais da entidade do mundo real, o modelo define três tipos de feições (Fig. 27):

- "Geo" - feição que armazena as características descritivas da entidade do mundo real. Os principais atributos não espaciais encontram-se nesse objeto. Representa uma entidade geográfica do mundo real tais como Rio, Ponte, Cais, Bóia, Farol, Etc. Um objeto dessa classe pode estar associado a outros objetos do tipo Geo;

- "Coleção" - feição que descreve o relacionamento entre objetos. Agrupa objetos segundo três tipos de relacionamentos: agregação, associação e empilhamento de objetos;

- "Meta" - feição que contém informação sobre outros objetos. Define a acurácia, escala, datum, qualidade do dado, etc.

Os sub-tipos dessas feições formam um conjunto de 175 tipos diferentes, tais como: bóia, canal, linha de costa, ponte, ancoradouro, doca, etc. Para cada tipo diferente de geo-

objeto, o S-57 define um conjunto diferente de atributos. Por exemplo, a bóia de águas seguras (BOYSAW) possui os seguintes atributos:

BOYSHP – formato da bóia: cônica, esférica, etc;
COLOUR – cor da bóia;
COLPAT – padrão de cores: listras horizontais, listras verticais, etc;
CONRAD – refletância radar da bóia;
DATEND – data final da presença da bóia;
DATSTA – data inicial da presença da bóia;
INFORM – informação textual sobre a bóia;
MARSYS – sistema de sinalização internacional utilizada pela bóia;
NATCON – natureza do material empregado na construção da bóia;
NINFOM - informação textual sobre a bóia no idioma origem;
NOBJNM – nome do objeto no idioma origem;
NTXTDS – descrição textual no idioma origem;
OBJNAM – nome único do objeto;
PEREND – data final para bóias sazonais;
PERSTA – data inicial para bóias sazonais;
PICREP – indica se existe representação pictórica da bóia (foto ou desenho);
RECDAT – data da edição cartográfica da bóia;
RECIND – método de codificação e entrada de dados da bóia na carta;
SCAMAX – escala de apresentação máxima em que a bóia deve ser usada;
SCAMIN - escala de apresentação mínima em que a bóia deve ser usada;
SORDAT – data do levantamento cartográfico;
SORIND – informação sobre a fonte de levantamento cartográfico da bóia;
STATUS – permanente, temporário, periódico, iluminado, sincronizado, etc;
TXTDSC - descrição textual em inglês;
VERACC – acurácia vertical;
VERLEN – altura da bóia.

O S-57 define que nem todos os atributos são obrigatórios. Uma descrição detalhada de todos os atributos, obrigatórios ou não, dos geo-objetos é apresentada em IHO [10].

Atualmente, o S-57, só define o dado espacial do tipo vetorial. O modelo raster foi deixado em aberto para uma posterior definição.

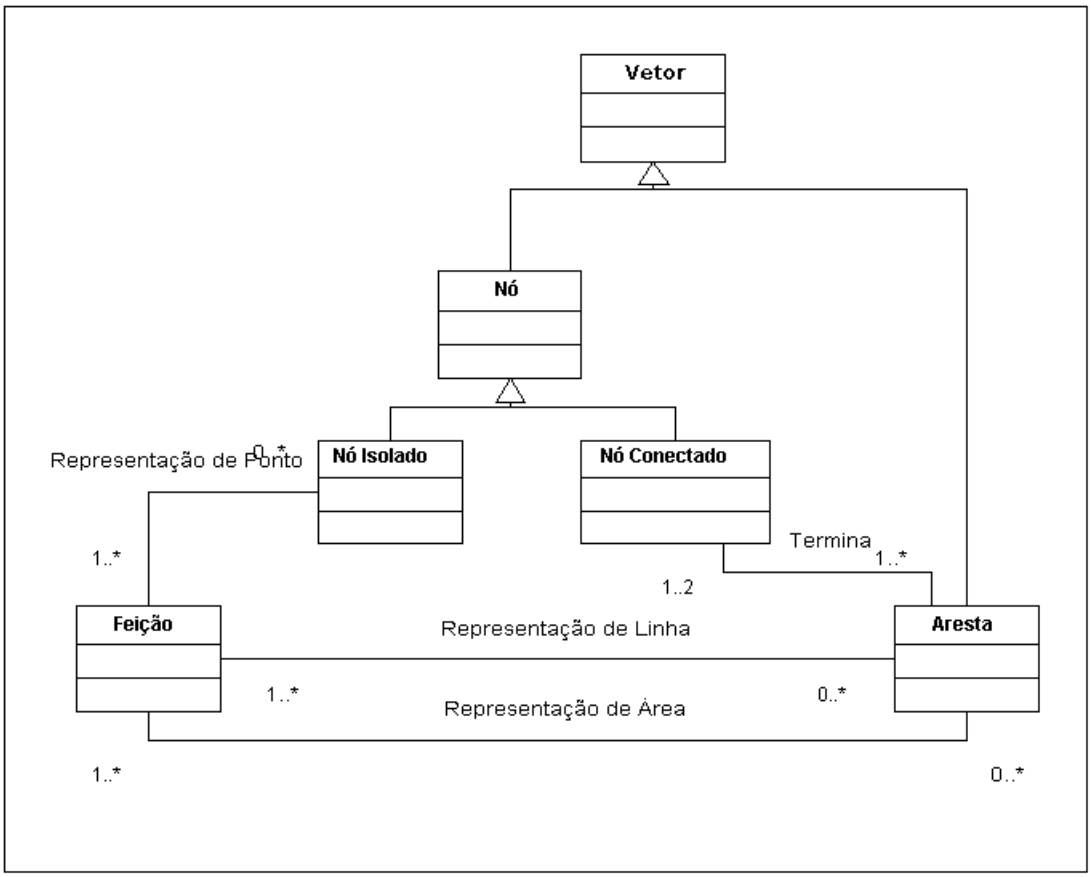


Fig. 28 - Modelo conceitual S-57 nível 3.

A Fig. 28 mostra o terceiro nível de detalhamento do modelo conceitual S-57. “Aresta” é um conjunto de segmentos de reta conectados. “Nó Conectado” une uma ou mais arestas. Uma aresta inicia e termina em um (aresta fechada) ou dois nós conectados. O relacionamento “É localizado por” da Fig. 26, foi desmembrado nas representações de ponto, linha ou área. A feição representada por um ponto está associada ao nó isolado. A feição do tipo linha está associada ao conjunto de arestas formando um polígono aberto. A feição área forma um polígono fechado de arestas. Todos os tipos de representações possíveis de uma feição, permitidas no S-57, são apresentados no apêndice B.

5.3. ARQUITETURA PROPOSTA

5.3.1. VISÃO GERAL

A figura abaixo mostra a arquitetura básica de hardware do sistema de um integrante da frota (um navio, por exemplo). O equipamento de link de dados permite a troca de dados com os demais participantes. A quantidade de servidores S-57 pode variar muito. Um integrante pode possuir vários servidores ou, até mesmo, não possuir servidor. O requisito mínimo é que pelo menos um participante da frota possua um servidor.

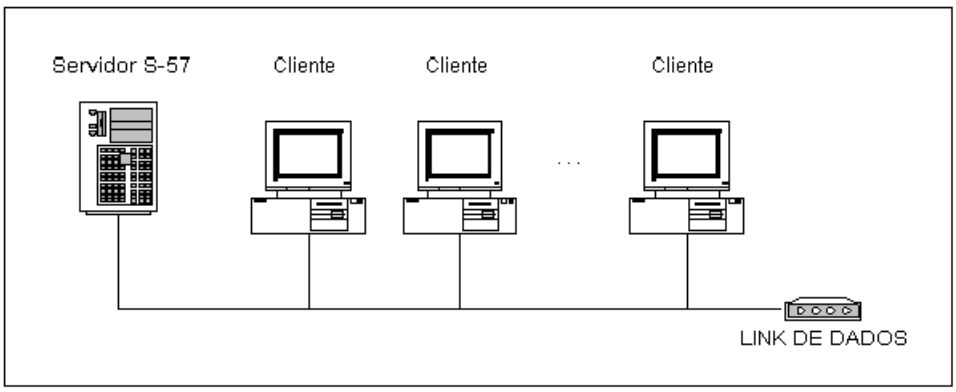


Fig. 29 - Arquitetura de hardware.

Os clientes podem ser qualquer computador pertencente à rede da frota ou outros sistemas navais tais como o sistema de navegação, o sistema de informações táticas ou o sistema de armas. Embora fisicamente separadas, as redes de todos os participantes comportam-se como se fosse uma única e grande rede lógica de dados capaz de acomodar vários servidores e vários clientes logicamente interligados.

5.3.2. ARQUITETURA DE SOFTWARE

A figura a seguir mostra a arquitetura de software do sistema idealizada para suportar a distribuição dos dados geográficos. Os objetos gerente, cliente e servidor de cartas podem estar localizados em máquinas diferentes. O objeto carta e as feições dos objetos geográficos são instanciados na mesma máquina do servidor de cartas a fim de diminuir o tráfego de rede do sistema.

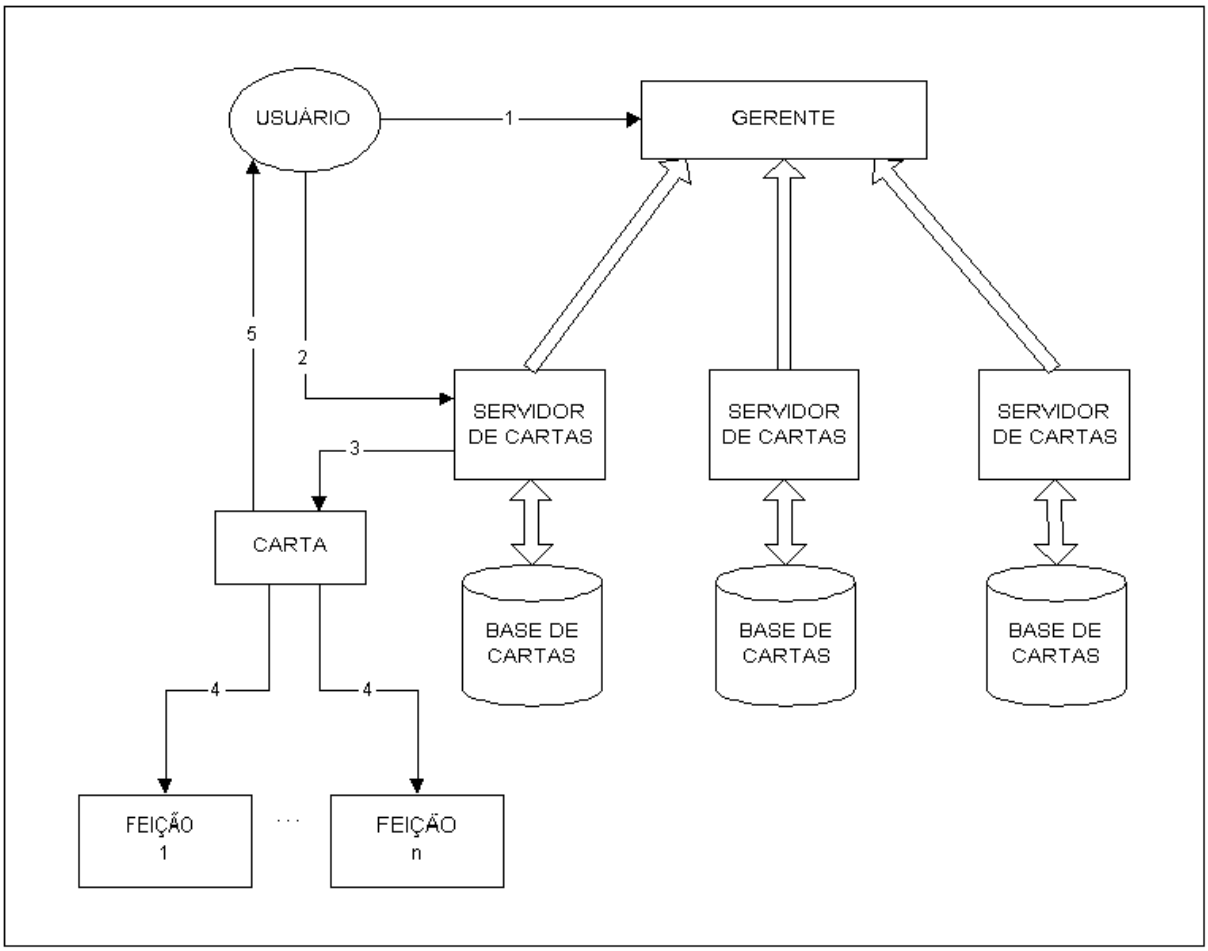


Fig. 30 - Arquitetura de software.

Toda a comunicação entre objetos é feita via CORBA. Um cliente solicita que o gerente localize um servidor de cartas que forneça a carta náutica desejada. O gerente responde retornando a referência CORBA do servidor. O cliente então acessa o servidor de cartas solicitando a abertura da carta desejada que instancia um objeto carta que, por sua vez, lê o arquivo S-57 e instancia as respectivas feicões. O servidor de cartas retorna a referência

CORBA da carta para o cliente, permitindo então que o cliente possa acessá-la remotamente através de seus métodos.

O sistema poderá possuir tantos servidores de cartas quanto necessários. É permitida a existência de cartas duplicadas desde que em servidores distintos.

5.3.2.1 FEIÇÃO

"Feição" neste contexto representa uma feição S-57 apresentada na Fig. 26 acrescido de métodos de acesso já que o S-57 armazena apenas os atributos das classes. Apesar de o geo-objeto representar tanto uma feição como um objeto espacial, o acesso aos dados do geo-objeto se dá exclusivamente através da feição, isto é, o cliente não possui acesso direto aos dados espaciais senão por meio da feição associada ao objeto espacial. Através de uma interface IDL (ou interface CORBA), padroniza-se o acesso aos objetos. A interface IDL da classe "Feição" é apresentada a seguir:

```
// feature.idl

module Feature
{
        enum spatial_representation {REP_NONE, REP_POINT, REP_LINE,
    REP_AREA};
        enum match_attrib_status {MAS_ERRO, MAS_TRUE, MAS_FALSE,
MAS_NOT_EXIST};
        enum atrib_type {ATT_ALPHANUMERIC, ATT_NUMERIC};

        struct atrib_value
        {
                string value;
                atrib_type type;
        };

        struct point
        {
```

```
            double latitude;
            double longitude;
        };

        struct spatial_data
        {
            spatial_representation representation;
            sequence <point> node;
        };

        interface s57Feature
        {
            string getName( );
            string getId( );
            string getType( );
            boolean getAttribute( in string attrib , out atrib_value value);
            spatial_data getSpatialData( );
            boolean matchName( in string name );
            boolean matchId( in string Id );
            boolean matchType( in string type );
            spatial_representation getSpatialRepresentation( );
            boolean Neighbor( in s57Feature feature );
        };
};
```

Descrição dos métodos:

- getName, getId, getType - toda feição possui um nome, um identificador S-57[3] e um tipo definido no S-57 (apêndice A). Esses métodos permitem ao cliente consultar os valores desses atributos;

[3] Um identificador S-57 é uma string gerada pelas agências cartográficas que identifica um objeto geográfico. Cada entidade geográfica no mundo possui uma identidade única.

- getAttribute - retorna o valor de um atributo qualquer da feição. Caso o atributo em questão não exista na feição o código de erro ATS_NOT_EXIST é retornado;

- getSpatialData - retorna o dado espacial associada à feição assim como a sua representação (nenhuma, ponto, linha ou área);

- matchName, matchId, matchType - retornam um valor booleano que indica se o nome, o identificador ou o tipo fornecido são correspondentes aos atributos da feição. O "GeoEspaço" utiliza esses métodos para localizar e agrupar feições;

- GetSpatialRepresentation - retorna a representação espacial da feição;

- Neighbor - verifica a vizinhança entre as feições.

5.3.2.2 CARTA

A Fig. 31 mostra o diagrama de classes da carta náutica. A classe "Feição" representa a feição de um geo-objeto S-57, ou seja, um objeto geográfico qualquer. As demais classes são descritas a seguir. Ressalta-se aqui a herança múltipla da interface Grupo.

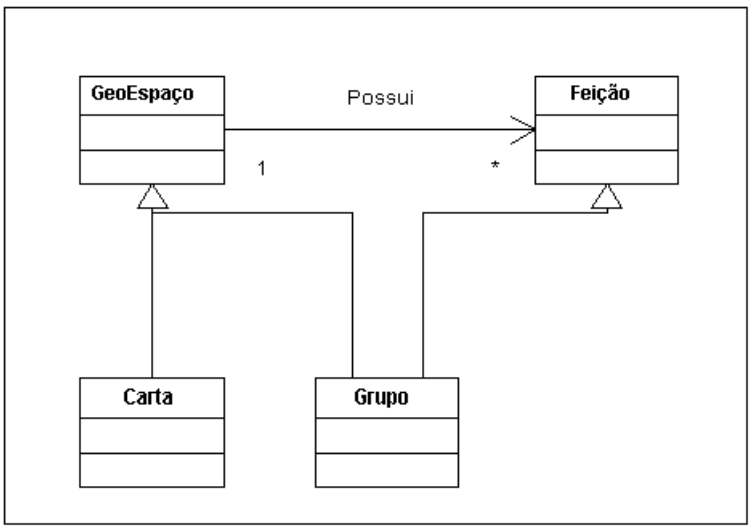


Fig. 31 - Relacionamento entre classes da carta náutica

"GeoEspaço" é uma classe abstrata que representa um espaço geográfico de duas dimensões onde são inseridos as feições dos "GeoObjetos". É um espaço retangular delimitado por duas latitudes e duas longitudes e tem como principal característica a sua capacidade de agrupar várias feições distintas e possui uma lista interna que armazena as referências CORBA das feições que estão contidos no "GeoEspaço". O "GeoEspaço" fornece uma série de métodos que permitem a realização de consultas baseadas no relacionamento espacial de vizinhança[4] das feições, assim como consultas baseadas em atributos das feições, tais como o tipo do "GeoObjeto" e a sua representação (ponto, linha ou área). O Cliente pode criar "iteradores" para percorrer a lista de "GeoObjetos" contidos no "GeoEspaço". A interface IDL do "GeoEspaço" é apresentada a seguir.

```
. . .

    interface featureIterator
    {
            Feature::s57Feature getFirst( );
            Feature::s57Feature getNext( );
            void destroy();
    };

    interface s57GeoSpace
    {
            s57Group getFeatureNeighborhood( in Feature::s57Feature feature );
            s57Group getFeatureNeighborhoodByName( in string name );
            s57Group getFeatureNeighborhoodById( in string id );
            Feature::s57Feature getFeatureByName( in string name );
            Feature::s57Feature getFeatureById( in string id );
            s57Group getFeatureByType( in string type);
```

[4] O S-57 armazena obrigatoriamente apenas o relacionamento de vizinhança entre os geo-objetos. A orientação a objetos permite, porém, a implementação de métodos de consultas baseados em outros tipos de relacionamentos não existentes na base de dados, tais como "*contém", "está contido", "interseção"*, etc; para isso basta inserir no método o algoritmo apropriado capaz de identificar esses relacionamentos na base de dados S-57.

```
        s57Group getPointFeature( );
        s57Group getLineFeature( );
        s57Group getAreaFeature( );
        featureIterator createFeatureIterator ( );
    };
. . .
```

Descrição dos métodos:

- getFeatureNeighborhood - retorna um grupo (o conceito de grupo será visto mais adiante) contendo todas as feições representadas por uma área que fazem vizinhança com a feição fornecida. A feição fornecida é identificada através da sua referência CORBA;

- getFeatureNeighborhoodByName - retorna um grupo contendo todas as feições representadas por uma área que fazem vizinhança com a feição fornecida. A feição fornecida é identificada através de seu nome;

- getFeatureNeighborhoodById - retorna um grupo contendo todas as feições representadas por uma área que fazem vizinhança com a feição fornecida. A feição fornecida é identificada através de seu identificador S-57;

- getFeatureByName, getFeatureById - retorna a referência CORBA de uma feição dado o seu nome ou o seu identificador S-57;

- getFeatureByType - retorna um grupo contendo todas as feições de um determinado tipo. Vide apêndice A;

- getPointFeature, getLineFeature e getAreaFeature - retornam um grupo contendo as feições do tipo ponto, linha e área respectivamente;

- createFeatureIterator - usados para criar iteradores que tornam possível ao cliente acessar a lista de feições do “GeoEspaço”.

A classe “Carta” é uma especialização da classe “GeoEspaço” e representa um objeto que agrupa todas as feições pertencentes à carta náutica. Assim que ele é criado, através do método openChart da interface s57Server, instancia em memória local todos os “GeoObjetos” da carta. A “Carta”, diferentemente do “Grupo”, não permite que sejam inseridas e nem removidas feições. Sempre que é aberta, a referência CORBA da “Carta” é retornada para o cliente e representa o ponto inicial de comunicação do cliente com os objetos remotos. A “Carta” está associada à interface CORBA do “GeoEspaço”.

“Grupo” é uma classe de objetos que permite ao cliente agrupar feições segundo determinada condição, gerando um subconjunto menor de objetos geográficos; como, por exemplo, agrupar todos os faróis contidos na carta náutica 1500 (Baía de Guanabara). O “Grupo”, assim como a “Carta”, é uma especialização do “GeoEspaço” e permite delimitar o escopo das consultas ao subconjunto definido no grupo e possui os mesmos métodos de consultas que a “Carta”. A “Carta”, em essência, é um tipo particular de “Grupo” que engloba todos os objetos geográficos definidos naquele arquivo da base de dados (Fig. 32).

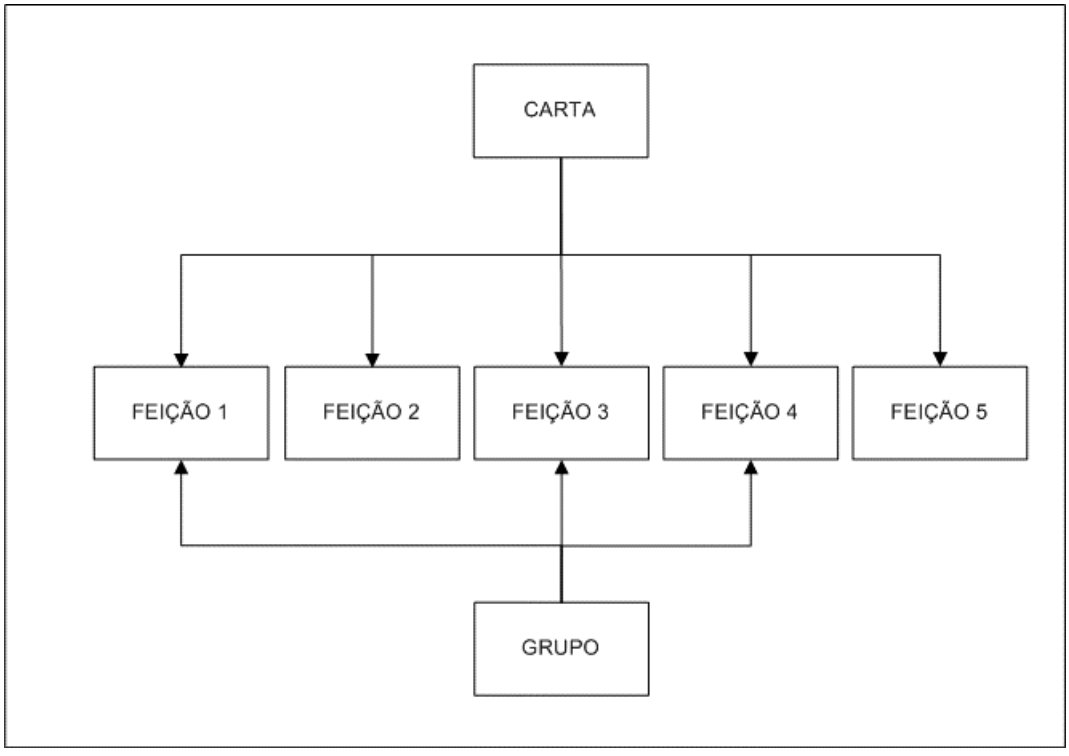


Fig. 32 - O grupo está associado às feições 1,3 e 4.

O usuário pode criar e destruir quantos grupos desejar através dos métodos createGroup e destroyGroup da classe “Servidor de Cartas”. Um mesmo objeto geográfico pode pertencer a vários grupos simultaneamente. Como o “Grupo” herda as operações tanto da classe “GeoEspaço” quanto da classe “Feição”, o “Grupo” tem tanto a capacidade de possuir várias feições quanto de ser um elemento de outro grupo, permitindo, dessa maneira, a configuração de uma hierarquia de grupos (Fig. 33).

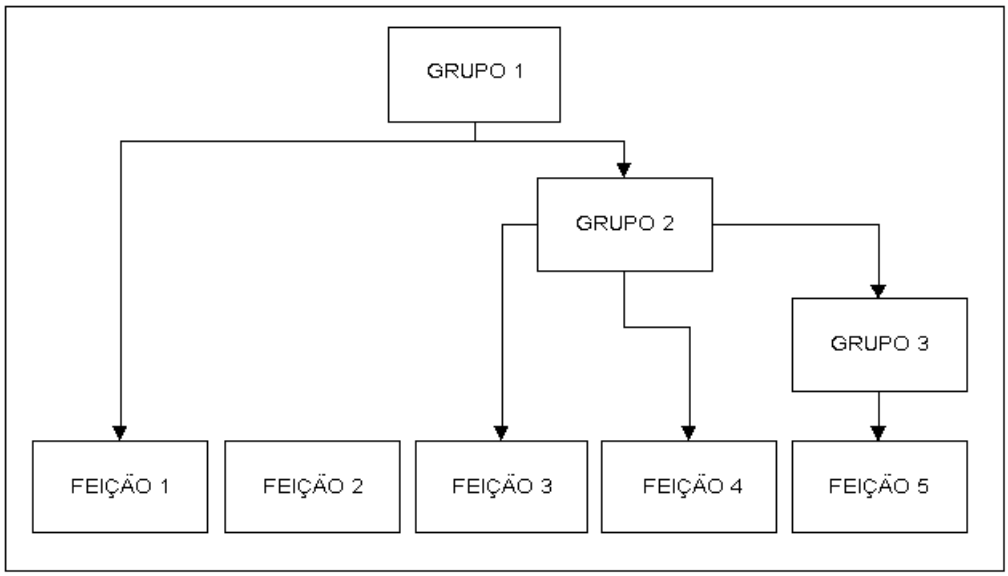


Fig. 33 - Hierarquia de grupos.

A invocação de um método do grupo percorre hierarquicamente os subgrupos até atingir as feições. Na Fig. 33, uma chamada a um método do grupo 1 é disseminada à feição 1 e ao grupo 2 que por sua vez dissemina para seus filhos e assim sucessivamente. A feição 2 não é invocada nessa hierarquia. Por exemplo, vamos supor que desejamos obter a referência CORBA do geo-objeto “Farol de Cabo Frio”, a invocação do método getFeatureByName do grupo 1, passando como parâmetro o nome do geo-objeto, termina atingindo o método matchName da feição 1,3,4 e 5. Caso um destes seja a feição desejada, ele responderá com sua referência CORBA para o nível superior da hierarquia até atingir o cliente que requisitou a informação. Note que para o grupo 1, o grupo 2 é uma feição como outra qualquer e da mesma maneira, o grupo 3 é uma feição do grupo 2.

Uma outra maneira de se criar grupos é através de métodos de consulta espacial. Os resultados das consultas realizadas sobre o objeto mapa ou sobre outro grupo qualquer são sempre referências CORBA de um novo grupo. Sempre que um cliente chama um método de consulta, um grupo é criado contendo as feições que satisfazem a consulta e a referência CORBA do grupo é retornada para o cliente. Um exemplo de consulta sobre um relacionamento de vizinhança pode ser usado para responder a seguinte consulta:

"Quais os estados brasileiros que fazem divisa com o Rio de Janeiro?".

O cliente recebe como resposta a referência CORBA do grupo contendo as feições de nome "São Paulo", "Minas Gerais" e "Espírito Santo".

Os grupos também podem ser vistos como conjuntos finitos cujos elementos são as feições nele contidas (Fig. 34). Com isso é possível realizar operações básicas sobre conjuntos, tais como: união, interseção, diferença e pertinência.

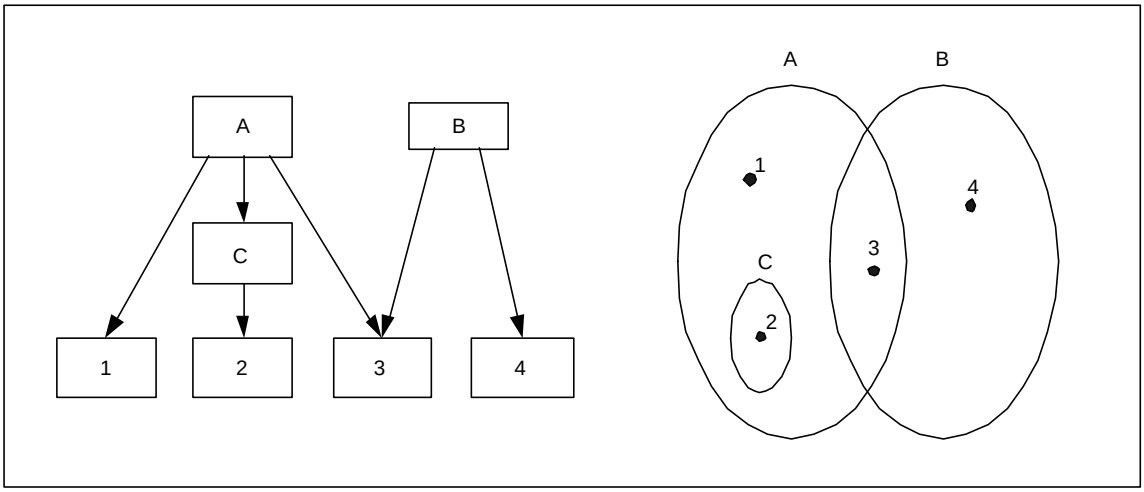


Fig. 34 - Diagramas de Venn para grupos.

A IDL do grupo, correspondente à Fig. 31, fica:

```
. . .
        interface s57Group : s57GeoSpace, Feature::s57Feature
        {
```

```
        void insertFeature( in Feature::s57Feature feature );
        void removeFeature( in Feature::s57Feature feature );
        void groupUnion( in s57Group group );
        void groupIntersect( in s57Group group );
        void groupDifference( in s57Group group );
        boolean isMember( in Feature::s57Feature feature );
        void setName( in string name );
    };. . .
```

Descrição dos métodos:

- insertFeature, removeFeature - insere e remove uma feição ou um novo grupo no grupo corrente;

- groupUnion, groupIntersect, groupDifference - operações sobre conjuntos. O grupo sobre o qual é invocado o método recebe o resultado final da operação;

- IsMember - verifica se a feição fornecida é membro do grupo.

O arquivo IDL completo da carta fica então:

```
// chart.idl
#include "feature.idl"

module Chart
{
    interface featureIterator
    {
        Feature::s57Feature getFirst( );
        Feature::s57Feature getNext( );
        void destroy( );
    };
```

```
        interface s57Group;

        interface s57GeoSpace
        {
                s57Group getFeatureNeighborhood( in Feature::s57Feature feature );
                s57Group getFeatureNeighborhoodByName( in string name );
                s57Group getFeatureNeighborhoodById( in string id );
                Feature::s57Feature getFeatureByName( in string name );
                Feature::s57Feature getFeatureById( in string id );
                s57Group getFeatureByType( in string type);
                s57Group getPointFeature( );
                s57Group getLineFeature( );
                s57Group getAreaFeature( );
                featureIterator createFeatureIterator ( );
        };

        interface s57Group : s57GeoSpace, Feature::s57Feature
        {
                void insertFeature( in Feature::s57Feature feature );
                void removeFeature( in Feature::s57Feature feature );
                void groupUnion( in s57Group group );
                void groupIntersect( in s57Group group );
                void groupDifference( in s57Group group );
                boolean isMember( in Feature::s57Feature feature );
                void setName( in string name );
        };
};
```

5.3.2.3 SERVIDOR DE CARTAS

O servidor de cartas está presente em todas as localidades onde a base de cartas reside e sua principal função é atender às requisições de dados dos usuários. Sempre que um mapa é requisitado, o servidor de cartas verifica se o mesmo já se encontra instanciado em memória,

caso contrário, ele carrega a carta náutica em memória local e retorna a referência CORBA da carta para o requisitante. Sempre que um novo servidor desejar se incluir no sistema, ele deverá se cadastrar no gerente e fornecer os nomes de todos as cartas presentes na sua base de cartas. O servidor de cartas provê serviços para abrir e fechar mapas (openChart e closeChart) e mantém contadores de clientes que estão acessando determinado mapa. A cada operação de openChart o contador é incrementado; no closeChart, ele é decrementado e, se chegar a zero, o objeto carta é destruído junto com todos os seus geo-objetos. É permitido a um cliente abrir mais de uma carta ao mesmo tempo.

A servidor de cartas fornece métodos para criar e destruir grupos e sempre cria grupos vazios, isto é, sem elementos, que devem ser inseridos através dos métodos do grupo.

A interface IDL do servidor de cartas é apresentada a seguir. No arquivo chart.idl é definida a interface da Carta.

```
// chartServer.idl

#include "chart.idl"

module ChartServer
{
        typedef sequence<string> chartNameSeq;

        interface s57Server
        {
                Chart::s57Chart openChart ( in string name );
                void closeChart ( in string name );
                chartNameSeq getCharts( );
                Chart::s57Group createGroup( in string name );
                void destroyGroup ( in Chart::s57Group group );
                Chart::s57Group getGroup( in string name );
                void isAlive();
        };
};
```

O método openChart retorna a referência CORBA da carta recém aberta. O getCharts retorna uma lista de cartas oferecidas pelo servidor.

O método isAlive é invocado ciclicamente pelo gerente para implementar um nível de tolerância a falhas. Caso o gerente, após um determinado tempo, não chamar esse método, infere-se que houve uma falha no gerente. O servidor de cartas procura então um novo gerente para se cadastrar.

5.3.2.4 BASE DE CARTAS

São os arquivos contendo os mapas vetorizados no formato S-57. Não é permitido a uma base de cartas possuir replicação de dados, isto é, dois ou mais mapas idênticos; admitida, porém, a replicação entre base de dados diferentes, sendo essa uma abordagem não conveniente devido a problemas de atualizações freqüentes do dado geográfico.

5.3.2.5 GERENTE

O gerente tem que ser único em todo sistema e é onde são cadastrados todos os servidores de cartas do sistema. O gerente é um objeto global que utiliza o serviço de nomes do CORBA para publicar seus serviços permitindo que todos os objetos possam ter fácil acesso a ele. Provê um nível mínimo de tolerância a falhas verificando continuamente, através do método isAlive do servidor de cartas, o acesso aos servidores nele cadastrados. Caso o ORB retorne uma exceção indicando a indisponibilidade do objeto remoto, por exemplo “OBJECT_NOT_EXIST”, o servidor é descadastrado da lista.

Um segundo nível de tolerância a falhas é necessário para garantir que uma falha no gerente não prejudique todo o sistema. É utilizada a técnica de redundância de objetos associada ao serviço de transações atômicas do CORBA. Neste caso, teríamos o gerente (gerente mestre) e várias cópias do gerente (gerentes escravos) distribuído pela rede, de maneira que se ocorrer uma falha no mestre, um dos escravos assume o seu lugar. Aqui, da mesma maneira que o gerente verifica se o servidor está ativo, os escravos verificam se o mestre está ativo através do método “isAlive” do gerente. Se um dos escravos demorar muito a invocar o método, é descadastrado do sistema.

O mestre possui uma lista de escravos onde armazena a referência CORBA de todos os seus escravos e dois métodos (“insertSlave” e “removeSlave”) para cadastrar e descadastrar os escravos. Todos os escravos devem possuir uma cópia dessa lista. Assim que um escravo é cadastrado, o mestre invoca o “insertSlave” do escravo várias vezes, passando as referência CORBA dos escravos, uma de cada vez, que estão contidos na sua lista. Dessa forma a lista do escravo fica igual à lista do mestre. Além disso, o mestre invoca o “insertSlave” dos demais escravos para inserir a referência CORBA do novo escravo nos demais, mantendo as listas de escravos iguais em todo o sistema.

Caso um dos escravos perceba alguma falha no mestre, através da ocorrência de exceções geradas pelo ORB, inicia-se o processo de eleição de um novo mestre utilizando o algoritmo Bully de Garcia-Molina citado por TANENBAUM [24]. Tão logo seja eleito, o mestre se cadastra no serviço de nomes do CORBA tornando sua interface pública para o sistema.

O algoritmo Bully, aqui adaptado para a visão de objetos, baseia-se no fato que se o escravo, *P*, perceber que o mestre não está mais respondendo ao método “isAlive”, *P* inicia a eleição de um novo mestre como se segue:

1. *P* invoca o método de eleição para todos os escravos que estão numa posição anterior da sua na lista de escravos;
2. Se ninguém responder, *P* ganha a eleição e torna-se o novo mestre;
3. Se o escravo, *Q*, responder à requisição de *P*, a tarefa de *P* está feita e *Q* inicia um novo processo de eleição.

Aquele que vencer a eleição invoca o método “setMaster” de todos os demais escravos anunciando a sua vitória e encerrando o processo de eleição.

O serviço de transações atômicas do CORBA é utilizado no cadastramento e descadastramento de escravos, garantindo, dessa forma, que as listas de escravos sejam iguais em todo o sistema.

O gerente fornece serviços para:

- Cadastramento e descadastramento de servidores de cartas. Um servidor, para poder ser acessado, deve antes se cadastrar no gerente fornecendo a sua referência CORBA e uma lista de mapas que possui. O cadastramento de um servidor já cadastrado sobrescreve o cadastramento anterior;

- Busca de cartas através do nome da carta. Permite a um cliente (usuário) acessar uma carta apenas fornecendo o seu nome. Este método retorna a referência CORBA do servidor que possui a carta solicitada;

- Acesso à lista de servidores e suas respectivas cartas. Permite que os clientes acessem a lista de servidores interna através de objetos de iteração;

- Cadastramento e descadastramento de gerentes escravos. O cadastramento de um gerente já cadastrado sobrescreve o cadastramento anterior;

- Verificação de falhas no gerente;

- Iniciar a eleição de um novo gerente para assumir o posto de mestre.

A interface IDL do gerente é apresentada a seguir. No arquivo chartServer.idl é definida a interface do servidor de cartas chartServer::s57Server que foi apresentada no item 5.3.2.3 .

```
// manager.idl

#include "chartServer.idl"

module manager
{
    typedef sequence<string> chartNameSeq;

    interface serverIterator
```

```
        {
                chartServer::s57Server getFirst( );
                chartServer::s57Server getNext( );
                unsigned short getCount( );
                void destroy( );
        };

        interface s57Manager
        {
                void insertS57Server ( in chartServer::s57Server server, in
                chartNameSeq chartNames );
                void removeS57Server ( in chartServer::s57Server server );
                chartServer::s57Server getS57Server  ( in string name );
                serverIterator createServerIterator ( );
                void  insertManager( in s57Manager slave );
                void removeManager( in s57Manager slave );
                void isAlive();
                void election();
                void setMaster ( in s57Manager master );
        };
};
```

A interface serverIterator define quatro métodos de acesso. Os métodos getFirst e getNext são usados para percorrer a lista de servidores e retornam referências nulas quando não existir servidores na lista ou quando já atingiu o fim da lista, respectivamente. O método getCount retorna a quantidade de servidores cadastrados no gerente e o método destroy destrói o iterador.

O método insertS57Server da interface s57Manager é utilizado no cadastramento de servidores no gerente e recebe como parâmetros a referência CORBA do servidor e uma lista de nomes contendo os nomes das cartas que o servidor possui. Para descadastrar um servidor é utilizado o removeS57Server. A interface possui ainda métodos para obter um servidor que forneça determinada carta (getS57Server) e para criar iteradores (createS57Iterator).

Os métodos insertManager e removeManager são utilizados no cadastramento e descadastramento de gerentes escravos. O método isAlive é invocado pelos escravos para verificar ocorrência de falhas no gerente.

No apêndice C são apresentados alguns exemplos de código C++ mostrando várias possibilidades de um cliente acessar a base de dados distribuída de acordo com a arquitetura aqui apresentada.

CAPÍTULO 6

CONCLUSÃO

Apesar do significativo aprimoramento dos últimos anos, a difusão da tecnologia SIG ainda encontra inúmeras dificuldades. Carecem ainda arquiteturas de software que promovam a distribuição efetiva e racional da informação geográfica. Os SIGs são sistemas bastante complexos e diferentes dos demais sistemas de informações, pois manipulam grande quantidade de dados e requerem conceitos avançados para descrever a geometria de objetos e o relacionamento topológico entre eles. A Geomática vem buscando soluções nas modernas técnicas e nos conceitos avançados da ciência da computação. Neste contexto, o paradigma da orientação a objetos se aproxima da maneira natural de representar o mundo real. A visão de objetos é mais próxima do usuário final e do programador, que não precisam realizar esforços mentais para recriar a realidade geográfica a partir de conceitos não-OO.

Diversos organismos de padronização, antevendo a tendência do mercado, realizam esforços na inclusão da visão OO na difusão de seus padrões de dados, ressaltando-se aqui os padrões S-57, o STDS, o SAIF e o DIGEST.

A Geomática procura uma exploração do potencial de comunicação de dados de grandes redes de computadores a fim de racionalizar o acesso aos dados geográficos fornecidos pelos inúmeros organismos de levantamento cartográfico. Tirando proveito da natural distribuição dos dados geográficos acoplado a uma visão OO do espaço geográfico, CORBA surge como uma poderosa ferramenta na implementação de sistemas com um alto grau de transparência como apontado na seção 4.2. . CORBA aparece como a solução ideal para a interoperabilidade entre sistemas em ambientes distribuídos altamente heterogêneos.

6.1. CONTRIBUIÇÕES

Este trabalho buscou apresentar uma arquitetura para distribuição da informação geográfica vetorizada baseada no formato S-57, abrangendo a pesquisa de vários artigos, livros e periódicos principalmente no concernente aos seguintes itens:

- Foi apresentada uma arquitetura de distribuição da informação geográfica, baseada no modelo S-57, junto com todo o modelo de interface CORBA necessária a uma distribuição transparente do dado geográfico;

- Foi mostrado como utilizar diretamente o modelo S-57, sem a necessidade de conversão para outros formatos, a fim de preservar a integridade da informação geográfica original;

- Foi criado o conceito de hierarquia e grupo de objetos geográficos possibilitando a realização de consultas espaciais sobre uma arquitetura OO bem como a possibilidade de realização de operações de conjuntos sobre os objetos geográficos;

- Foram apresentados alguns exemplos de codificação C++ utilizando a arquitetura proposta percorrendo as principais características do sistema tais como: acesso aos dados geográficos, utilização de iteradores, criação e manipulação de grupos, consultas espaciais e operações sobre conjuntos.

6.2. TRABALHOS FUTUROS

A finalidade deste trabalho de pesquisa, de um modo geral, foi de construir uma infra-estrutura básica de distribuição da informação geográfica a ser utilizado na Marinha do Brasil, possibilitando, dessa maneira, a criação de futuros SIGs distribuídos nos mais diversos ramos de atividades navais, tais como: navegação, controle de tráfego, exploração de recursos naturais e avaliação tática e estratégica utilizando dados geográficos.

A partir deste trabalho, pode-se evoluir para outros pontos importantes de pesquisa aprofundando ainda mais o nível de abrangência e de detalhamento da dissertação. A arquitetura aqui apresentada procurou focalizar o problema da distribuição da informação geográfica, no âmbito marítimo, sobre o padrão de dados S-57. Novas pesquisas podem ser desenvolvidas sobre a ótica da apresentação, para o usuário, da informação geográfica bem como a adaptação para outras arquiteturas e outros formatos neutros de dados geográficos. Podemos citar:

- Acompanhamento e adaptação à especificação CORBA do padrão OpenGis;

- Aprofundamento com outros padrões de dados geográficos: STDS, SAIF, DIGEST, etc;

- Adaptação para Web permitindo a visualização e consultas espaciais através de navegadores;

REFERÊNCIAS BIBLIOGRÁFICAS

[1] ALMEIDA, M. A. *SDIG-PB: Proposta de um Sistema Distribuído de Informação Geográfica para Auxílio à Gestão de Recursos Hídricos da Paraíba.* Tese de M.Sc., UFPB, Campina Grande, PB, Brasil, 1999.

[2] BOOCH, G. *Object-Oriented Analysis and Design with Applications*. 2 ed. Benjamin/ Cummings Publishing Company, Redwood City, CA, USA, 1994.

[3] CÂMARA G. *Modelos, Linguagens e Arquiteturas para Bancos de Dados Geográficos*. Tese de D.Sc. INPE, São José dos Campos, SP, Brasil, 1995.

[4] CÂMARA G., CASANOVA, M. A., HERMELY, A. S., et al. *Anatomia de Sistemas de Informação Geográfica*. 1 ed. SBC, Escola de Computação, Campinas, SP, Brasil, 1996.

[5] COX, B. *Object-Oriented Programming: An Evolutionary Approach.* 2 ed. Addison-Wesley, Reading, MA, USA, 1991.

[6] EGENHOFER, M. J., FRANK, A. *Object Oriented Modeling for GIS*. Journal of the Urban and Regional Information Systems URISA, v. 4, n. 2, pp 3-19, 1992.

[7] FIRESMITH, D. G. *Object-Oriented Requirements Analisys and Logical Design: A Software Engineering Approach*. 1 ed. John Wiley & Sons. Reading, MA, USA, 1992.

[8] FONSECA, F. T., EGENHOFER, M. J. *Sistemas de Informação Geográficos Baseados em Ontologias*. Revista IP – Informática Pública, ano 1 - n° 2, PRODABEL, Belo Horizonte, MG, Dez, 1999. Disponível em: <www.ip.pbh.gov.br/revista0102/ip0102fonseca.pdf>. Acesso em: 26 ago. 2001.

[9] IHO. *International Hydrographic Organization: Object Catalogue.* Revision 3.1. International Hydrographic Bureau, Mônaco, 2000.

[10] IHO. *International Hydrographic Organization: Special Publication 57.* Revision 3.1. International Hydrographic Bureau, Mônaco, 2000.

[11] INP. *Visibroker for C++ - Programmer's Guide*, 4 ed. Inprise Corporation, USA, 2000.

[12] JACOBSON, I., MAGNUS, C., PATRIK, J., et al. *Object-Oriented Software Engineering: A Use Case Driven Approach.* 1 ed. Addison-Wesley, Reading, MA, USA, 1992.

[13] KUMAR, K., SINHA, P., BHATT, P. *DO-GIS - A distributed and object oriented GIS*. In: Advances in GIS Research II: Proceedings 7th International Symposium on Spatial Data Handling, pp 37-49, Delft University of Technology, Nederland, Aug. 1996.

[14] LACOSTE, Y. *A Geografia - Isso Serve, em Primeiro Lugar, para Fazer a Guerra*. 5 ed. Papirus Editora, São Paulo, SP, Brasil, 1989.

[15] LORCH, C. *Marinha do Brasil – Poder Naval*. 1 ed. Serviço de Relações Públicas da Marinha. Action editora, Rio de Janeiro, RJ, Brasil, 1997.

[16] MITROVIC, A. *OO paradigm meets GIS: a new era in spatial data management*. Invited paper presented at *YUGIS'96*, pp 141-148, Belgrade, Yugoslavia, Mar 1996.

[17] OGC. *Simple Features Specification For CORBA*, Open GIS Consortium, Inc. Revision 1.0, USA, 1998. Disponível em: <http://www.opengis.org/techno/specs.htm>. Acesso em: 29 mar. 2002.

[18] OMG. *The Common Object Request Broker: Architecture and Specification*. Object Management Group. Revision 2.6, USA, 2001.

[19] OMG. *OMG Unified Modeling Language Specification*. Object Management Group, v. 1.3, USA, Jun. 1999.

[20] ORFALI, R., HARKEY D., Edwards J. *The Essential Distributed Objects Survival Guide*. 1 ed. John Wiley & Sons. USA, 1996.

[21] PEREZ, C. R., FERRAZ , C. A. G. *Suporte ao Processamento de Dados Geográficos Distribuídos*, Universidade Federal de Pernambuco, RT-DI 006/97, PE, Brasil, 1997.

[22] PIRES, P. F., MATTOSO, M. L. Q. *Arquiteturas de Sistemas de Banco de Dados Distribuídos – Novas Tecnologias*. Universidade Federal do Rio de Janeiro. Relatório Técnico PESC/COPPE/UFRJ ES-382/96, RJ, Brasil, 1996.

[23] RUMBAUGH, J. *Object-Oriented Modelling and Design*. 1 ed. Addison-Wesley, Englewood Cliffs, N.J., USA, 1991.

[24] TANENBAUM, A. S. *Distributed Operating Systems*. 1 ed. Prentice-Hall International. Englewood Cliffs, N.J., USA, 1995.

[25] TEIXEIRA, A. L. A., MORETI, E., CHRISTOFOLETTI A. *Introdução aos Sistemas de Informação Geográfica*. Edição do autor. Rio Claro, MG, Brasil, 1992.

Apêndice A - Feições Definidas no S-57

A.1 Geo Object Classes

Descrição	Tipo	Código
Administration Area (Named)	ADMARE	1
Airport/airfield	AIRARE	2
Anchor berth	ACHBRT	3
Anchorage area	ACHARE	4
Beacon, cardinal	BCNCAR	5
Beacon, isolated danger	BCNISD	6
Beacon, lateral	BCNLAT	7
Beacon, safe water	BCNSAW	8
Beacon, special purpose/general	BCNSPP	9
Berth	BERTHS	10
Bridge	BRIDGE	11
Building, single	BUISGL	12
Built-up area	BUAARE	13
Buoy, cardinal	BOYCAR	14
Buoy, installation	BOYINB	15
Buoy, isolated danger	BOYISD	16
Buoy, lateral	BOYLAT	17
Buoy, safe water	BOYSAW	18
Buoy, special purpose/general	BOYSPP	19
Cable area	CBLARE	20
Cable, overhead	CBLOHD	21
Cable, submarine	CBLSUB	22
Canal	CANALS	23
Canal bank	CANBNK	24
Cargo transhipment area	CTSARE	25
Causeway	CAUSWY	26
Caution area	CTNARE	27
Checkpoint	CHKPNT	28
Coastguard station	CGUSTA	29
Coastline	COALNE	30
Contiguous zone	CONZNE	31
Continental shelf area	COSARE	32
Control point	CTRPNT	33
Conveyor	CONVYR	34
Crane	CRANES	35
Current - non-gravitational	CURENT	36
Custom zone	CUSZNE	37
Dam	DAMCON	38
Daymark	DAYMAR	39
Deep water route centerline	DWRTCL	40
Deep water route part	DWRTPT	41
Depth area	DEPARE	42
Depth contour	DEPCNT	43

Distance mark	DISMAR	44
Dock area	DOCARE	45
Dredged area	DRGARE	46
Dry dock	DRYDOC	47
Dumping ground	DMPGRD	48
Dyke	DYKCON	49
Exclusive economic zone	EXEZNE	50
Fairway	FAIRWY	51
Fence/wall	FNCLNE	52
Ferry route	FERYRT	53
Fishery zone	FSHZNE	54
Fishing facility	FSHFAC	55
Fishing ground	FSHGRD	56
Floating dock	FLODOC	57
Fog signal	FOGSIG	58
Fortified structure	FORSTC	59
Free port area	FRPARE	60
Gate	GATCON	61
Gridiron	GRIDRN	62
Harbour area (administrative)	HRBARE	63
Harbour facility	HRBFAC	64
Hulk	HULKES	65
Ice area	ICEARE	66
Incineration area	ICNARE	67
Inshore traffic zone	ISTZNE	68
Lake	LAKARE	69
Lake shore	LAKSHR	70
Land area	LNDARE	71
Land elevation	LNDELV	72
Land region	LNDRGN	73
Landmark	LNDMRK	74
Light	LIGHTS	75
Light float	LITFLT	76
Light vessel	LITVES	77
Local magnetic anomaly	LOCMAG	78
Lock basin	LOKBSN	79
Log pond	LOGPON	80
Magnetic variation	MAGVAR	81
Marine farm/culture	MARCUL	82
Military practice area	MIPARE	83
Mooring/Warping facility	MORFAC	84
Navigation line	NAVLNE	85
Obstruction	OBSTRN	86
Offshore platform	OFSPLF	87
Offshore production area	OSPARE	88
Oil barrier	OILBAR	89
Pile	PILPNT	90
Pilot boarding place	PILBOP	91
Pipeline area	PIPARE	92
Pipeline, overhead	PIPOHD	93
Pipeline, submarine/on land	PIPSOL	94

Pontoon	PONTON	95
Precautionary area	PRCARE	96
Production/storage area	PRDARE	97
Pylon/bridge support	PYLONS	98
Radar line	RADLNE	99
Radar range	RADRNG	100
Radar reflector	RADRFL	101
Radar station	RADSTA	102
Radar transponder beacon	RTPBCN	103
Radio calling-in point	RDOCAL	104
Radio station	RDOSTA	105
Railway	RAILWY	106
Rapids	RAPIDS	107
Recommended route centerline	RCRTCL	108
Recommended track	RECTRC	109
Recommended traffic lane part	RCTLPT	110
Rescue station	RSCSTA	111
Restricted area	RESARE	112
Retro-reflector	RETRFL	113
River	RIVERS	114
River bank	RIVBNK	115
Road	ROADWY	116
Runway	RUNWAY	117
Sand waves	SNDWAV	118
Sea area/named water area	SEAARE	119
Sea-plane landing area	SPLARE	120
Seabed area	SBDARE	121
Shoreline construction	SLCONS	122
Signal station, traffic	SISTAT	123
Signal station, warning	SISTAW	124
Silo/tank	SILTNK	125
Slope topline	SLOTOP	126
Sloping ground	SLOGRD	127
Small craft facility	SMCFAC	128
Sounding	SOUNDG	129
Spring	SPRING	130
Square	SQUARE	131
Straight territorial sea baseline	STSLNE	132
Submarine transit lane	SUBTLN	133
Swept Area	SWPARE	134
Territorial sea area	TESARE	135
Tidal stream - flood/ebb	TS_FEB	160
Tidal stream - harmonic prediction	TS_PRH	136
Tidal stream - non-harmonic prediction	TS_PNH	137
Tidal stream panel data	TS_PAD	138
Tidal stream - time series	TS_TIS	139
Tide - harmonic prediction	T_HMON	140
Tide - non-harmonic prediction	T_NHMN	141
Tide - time series	T_TIMS	142
Tideway	TIDEWY	143
Topmark	TOPMAR	144

Traffic separation line	TSELNE	145
Traffic separation scheme boundary	TSSBND	146
Traffic separation scheme crossing	TSSCRS	147
Traffic separation scheme lane part	TSSLPT	148
Traffic separation scheme roundabout	TSSRON	149
Traffic separation zone	TSEZNE	150
Tunnel	TUNNEL	151
Two-way route part	TWRTPT	152
Underwater/awash rock	UWTROC	153
Unsurveyed area	UNSARE	154
Vegetation	VEGATN	155
Water turbulence	WATTUR	156
Waterfall	WATFAL	157
Weed/Kelp	WEDKLP	158
Wreck	WRECKS	159

A.2 Meta Object Classes

Descrição	Tipo	Código
Accuracy of data	M_ACCY	300
Compilation scale of data	M_CSCL	301
Coverage	M_COVR	302
Horizontal datum of data	M_HDAT	303
Horizontal datum shift parameters	M_HOPA	304
Nautical publication information	M_NPUB	305
Navigational system of marks	M_NSYS	306
Production information	M_PROD	307
Quality of data	M_QUAL	308
Sounding datum	M_SDAT	309
Survey reliability	M_SREL	310
Units of measurement of data	M_UNIT	311
Vertical datum of data	M_VDAT	312

A.3 Collection Object Classes

Descrição	Tipo	Código
Aggregation	C_AGGR	400
Association	C_ASSO	401
Stacked on/stacked under	C_STAC	402

P = ponto, L = Linha, A = Área, N = Nenhum.

ACHARE	P		A
BCNCAR	P		
BCNSPP	P		
BOYISD	P		
BRIDGE	P	L	A
CAUSWY		L	A
CGUSTA	P		
CONZNE			A
CTRPNT	P		
DAMCON	P	L	A
DISMAR	P		
DMPGRD	P		A
EXEZNE			A
FNCLNE		L	
FSHFAC	P	L	A
GRIDRN	P		A
ICEARE			A
LNDARE	P	L	A
LIGHTS	P		
LOGPON	P		A

ACHBRT	P		A
BCNISD	P		
BERTHS	P	L	A
BOYLAT	P		
BUAARE	P		A
CBLARE			A
CHKPNT	P		A
COSARE			A
CTSARE	P		A
DAYMAR	P		
DOCARE			A
DYKCON		L	A
FAIRWY			A
FOGSIG	P		
FSHGRD			A
HRBARE			A
ICNARE	P		A
LNDELV	P	L	
LITFLT	P		
LOKBSN			A

ADMARE			A
BCNLAT	P		
BOYCAR	P		
BOYSAW	P		
BUISGL	P		A
CBLOHD		L	
COALNE		L	
CRANES	P		A
CURENT	P		
DEPARE		L	A
DRGARE			A
DWRTCL		L	
FERYRT		L	A
FORSTC	P	L	A
FSHZNE			A
HRBFAC	P		A
ISTZNE			A
LNDMRK	P	L	A
LITVES	P		
MAGVAR	P	L	A

AIRARE	P		A
BCNSAW	P		
BOYINB	P		
BOYSPP	P		
CANALS		L	A
CBLSUB		L	
CONVYR		L	A
CTNARE	P		A
CUSZNE			A
DEPCNT		L	
DRYDOC			A
DWRTPT			A
FLODOC		L	A
FRPARE			A
GATCON	P	L	A
HULKES	P		A
LAKARE			A
LNDRGN	P		A
LOCMAG	P	L	A
MARCUL	P	L	A

MIPARE	P		A
OFSPLF	P		A
PILPNT	P		
PONTON		L	A
RADLNE		L	
RAILWY		L	
RDOCAL	P	L	
RETRFL	P		
RTPBCN	P		
SILTNK	P		A
SLOTOP		L	
SNDWAV	P	L	A
SUBTLN			A
TOPMAR	P		
TSSCRS			A
TWRTPT			A
WATFAL	P	L	

MORFAC	P	L	A
OSPARE			A
PIPARE	P		A
PRCARE	P		A
RADRNG			A
RAPIDS	P	L	A
RDOSTA	P		
RIVERS		L	A
RUNWAY	P	L	A
SISTAT	P		
SLOGRD	P		A
SPLARE	P		A
SWPARE			A
TSELNE		L	
TSSLPT			A
UNSARE			A
WATTUR	P	L	A

NAVLNE		L	
OILBAR		L	
PIPOHD		L	
PRDARE	P		A
RADRFL	P		
RCRTCL		L	
RECTRC		L	A
ROADWY	P	L	A
SBDARE	P	L	A
SISTAW	P		
SMCFAC	P		A
SPRING	P		
TESARE			A
TSEZNE			A
TSSRON			A
UWTROC	P		
WEDKLP	P		A

OBSTRN	P	L	A
PILBOP	P		A
PIPSOL	P	L	
PYLONS	P		A
RADSTA	P		
RCTLPT	P		A
RESARE			A
RSCSTA	P		
SEAARE	P		A
SLCONS	P	L	A
SOUNDG	P		
STSLNE		L	
TIDEWY		L	A
TSSBND		L	
TUNNEL	P	L	A
VEGATN	P	L	A
WRECKS	P		A

C_AGGR	N	N	N
M_CSCL			A
M_QUAL			A
T_HMON	P		A
TS_PAD	P		A

C_ASSO	N	N	N
M_HOPA			A
M_SDAT			A
T_NHMN	P		A
TS_PNH	P		A

M_ACCY			A
M_NPUB	P		A
M_SREL		L	A
T_TIMS	P		A
TS_PRH	P		A

M_COVR			A
M_NSYS			A
M_VDAT			A
TS_FEB	P		A
TS-TIS	P		A

Apêndice C – Exemplos de Codificação em C++

A seguir são apresentados alguns trechos de código C++ para o cliente. Esses exemplos foram compilados usando o ambiente C++ Builder 5.0 e foi utilizado o Visibroker 4.0 como ORB. O sistema operacional utilizado foi o Windows XP. Maiores detalhes de programação encontram-se em INP [11].

C.1 EXEMPLO 1

Abaixo segue um exemplo básico de um processo cliente que mostra:

- Como obter a referência CORBA do Gerente utilizando o serviço de nomes;
- Como localizar um Servidor de Cartas que forneça a carta desejada;
- Como abrir uma carta e acessar uma feição da mesma;
- Como acessar os dados espaciais da feição.

Nota-se a total transparência tanto na complexidade da rede quanto na localização dos objetos distribuídos. Toda a comunicação em rede é realizada através dos métodos dos objetos.

```
int main(int argc, char* const* argv)
{
  try
  {
    // Inicia o ORB
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

    // Obtêm o Id do gerente
    PortableServer::ObjectId_var managerId =
                    PortableServer::string_to_ObjectId("S57Manager");

    // Localiza o gerente. Fornece o nome completo do POA e o Id do gerente.
```

```
Manager::s57Manager_var manager =
        Manager::s57Manager::_bind("/S57_poa", managerId);

// Invoca os métodos do gerente
// Procura por um servidor que forneça a carta 1501
ChartServer::s57Server_var chart_server = manager->getS57Server("1501");

if (CORBA::is_nil(chart_server))
{
  cout << "Não foi encontrado nenhum servidor" << endl;
}
else
 {
  // Servidor encontrado. Abre a carta 1501
  Chart::s57Chart_var chart = chart_server->openChart("1501");

  // Seleciona uma feição da carta
  Feature::s57Feature_var farol = chart->getFeatureByName("Farol de Cabo Frio");

  if (CORBA::is_nil(farol))
  {
    cout << "Feição inexistente nesta carta" << endl;
  }
  else
  {
    // Obtém os dados espaciais da feição
    Feature::spatial_data *spatial = farol->getSpatialData();

    // O S57 define que a representação de farol só pode ser
    // do tipo PONTO
    if (spatial->representation != Feature::REP_POINT)
    {
      cout << "Erro na representação espacial" << endl;
        }
```

```
          else
          {
            // Exibe as coordenadas do farol
            cout << "Latitude: " << spatial->node[0].latitude << endl;
            cout << "Longitude: " << spatial->node[0].longitude << endl;
          }

          delete spatial;
        }
      }
    }
    catch(const CORBA::Exception& e)
    {
      cerr << e << endl;
      return 1;
    }

    return 0;
}
```

C.2 EXEMPLO 2

Abaixo segue um exemplo de um processo cliente que mostra a utilização de iteradores de feições. Neste exemplo é aberta uma carta e é criado um iterador para percorrer todas as feições da carta imprimindo o nome delas. O conceito de iteradores é importante em um sistema multi-usuário onde os usuários podem percorrer a lista de feições simultaneamente cada qual com o seu próprio iterador, sem afetar as consultas dos demais.

```
int main(int argc, char* const* argv)
{
 try
 {
  // Inicia o ORB
  CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

  // Obtêm o Id do gerente
  PortableServer::ObjectId_var managerId =
                PortableServer::string_to_ObjectId("S57Manager");

  // Localiza o gerente. Fornece o nome completo do POA e o Id do gerente.
  Manager::s57Manager_var manager =
            Manager::s57Manager::_bind("/S57_poa", managerId);

  // Invoca os métodos do gerente
  // Procura por um servidor que forneça a carta 1501
  ChartServer::s57Server_var chart_server = manager->getS57Server("1501");

  if (CORBA::is_nil(chart_server))
  {
    cout << "Não foi encontrado nenhum servidor" << endl;
  }
  else
  {
    // Servidor encontrado. Abre a carta 1501
```

```
      Chart::s57Chart_var chart = chart_server->openChart("1501");

      // Cria um iterador para percorrer a lista de feições
      Chart::featureIterator_var iterador = chart->createFeatureIterator();

      // Obtém cada feição da lista
      Feature::s57Feature_var feicao = iterador->getFirst();

      while (!CORBA::is_nil(feicao))
      {
        cout << "Nome: " << feicao->getName() << endl;
        feicao = iterador->getNext();
      }

      iterador ->destroy();
    }
  }
  catch(const CORBA::Exception& e)
  {
    cerr << e << endl;
    return 1;
  }

  return 0;
}
```

C.3 EXEMPLO 3

Abaixo segue um exemplo de um processo cliente que realiza uma consulta sobre o objeto Carta para obter um grupo contento todas as feições da carta que possuem representação de ponto. Em seguida obtém uma feição representada por uma área e verifica quais feições pontuais estão contidas nesta área.

A função SpatialRelationship::contains() verifica se um ponto está contido numa área e deverá ser implementada pelo cliente.

```
class SpatialRelationship
{
public:
  static bool contains(Feature::spatial_data *area, Feature::spatial_data *ponto)
      {
        // Função implementada pelo cliente
        . . .
        . . .
      };
};

int main(int argc, char* const* argv)
{
 try
 {
   // Inicia o ORB
   CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

   // Obtêm o Id do gerente
   PortableServer::ObjectId_var managerId =
                PortableServer::string_to_ObjectId("S57Manager");

   // Localiza o gerente. Fornece o nome completo do POA e o Id do gerente.
   Manager::s57Manager_var manager =
            Manager::s57Manager::_bind("/S57_poa", managerId);
```

```
// Invoca os métodos do gerente
// Procura por um servidor que forneça a carta 1501
ChartServer::s57Server_var chart_server = manager->getS57Server("1501");

if (CORBA::is_nil(chart_server))
{
  cout << "Não foi encontrado nenhum servidor" << endl;
}
else
{
  // Servidor encontrado. Abre a carta 1501
  Chart::s57Chart_var chart = chart_server->openChart("1501");

  Chart::s57Group_var grupo_ponto = chart->getPointFeature( );

  // Cria um iterador para percorrer a lista de feições do grupo
  Chart::featureIterator_var iterador = grupo_ponto->createFeatureIterator();

  // Obtém cada feição do grupo
  Feature::s57Feature_var feicao = iterador->getFirst();

  // Exibe os nomes das feições pertecentes ao grupo
  while (!CORBA::is_nil(feicao))
  {
    cout << "Nome: " << feicao->getName() << endl;
    feicao = iterador->getNext();
  }

  // Verifica quais feições estão contidas na "Ilha de Mocanguê"
  Feature::s57Feature_var mocangue = chart->getFeatureByName("Ilha de
  Mocanguê");

  if (CORBA::is_nil(mocangue))
```

```
    {
      cout << "Feição inexistente nesta carta" << endl;
    }
    else
    {
      // Obtém os dados espaciais da ilha
      Feature::spatial_data *area_mocangue = mocangue->getSpatialData();

      // Verifica quais feições do grupo estão contidas na área
      // delimitada pela "Ilha de Mocanguê"
      Feature::spatial_data *ponto;

      feicao = iterador->getFirst();

      while (!CORBA::is_nil(feicao))
      {
        ponto = feicao->getSpatialData();

        bool contem = SpatialRelationship::contains(area_mocangue, ponto);

        if (contem)
        {
          cout << "Ilha de Mocanguê contém " << feicao->getName() << endl;
        }

        feicao = iterador->getNext();
      }
    }

    iterador ->destroy ();
  }
}
catch(const CORBA::Exception& e)
{
```

```
    cerr << e << endl;
    return 1;
  }

  return 0;
}
```

C.4 EXEMPLO 4

Abaixo segue um exemplo de um processo que acessa duas bases de dados pertencentes a dois servidores de cartas distintos e constrói uma hierarquia de grupo utilizando os elementos contidos nas duas bases de dados.

O programa abre duas cartas distintas (carta 1511 e carta 1512) em servidores diferentes e monta a hierarquia a partir das bóias de águas seguras (BOYSAW) e das bóias de perigo à navegação (BOYISD), contidas em ambas as cartas, segundo a figura abaixo.

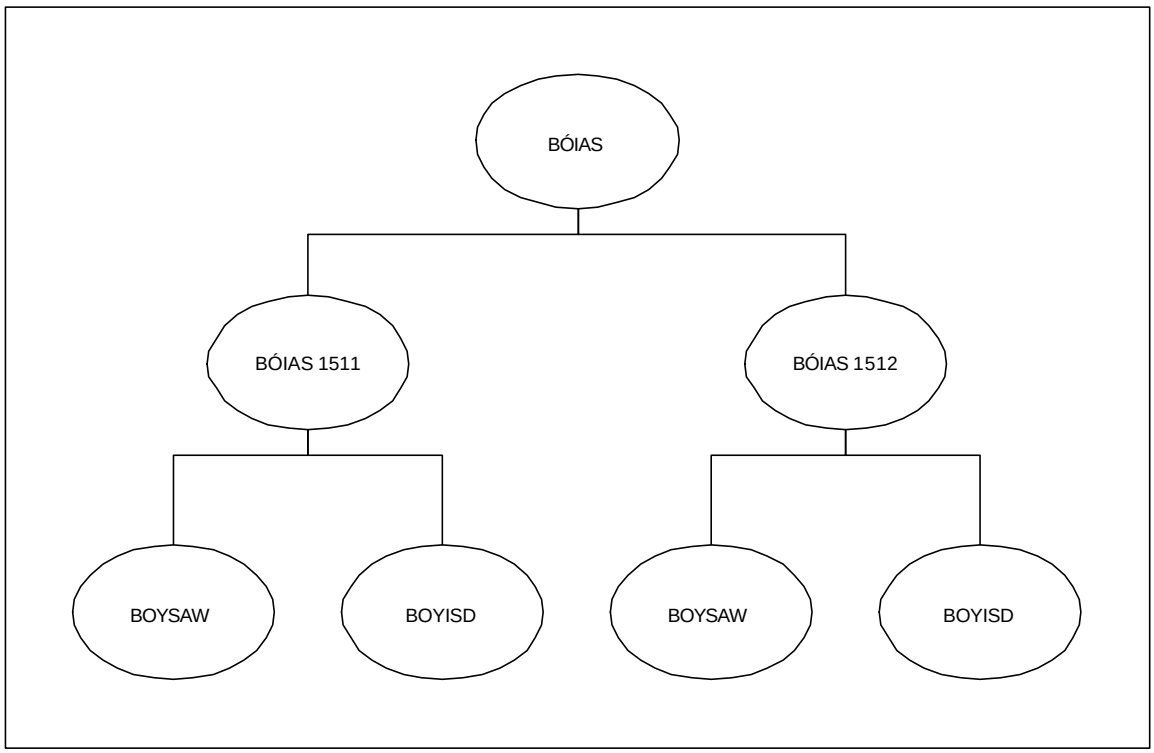


Fig. 35 - Hierarquia de grupo de bóias.

Este exemplo mostra a capacidade de manipulação de várias bases de dados permitindo ao cliente cruzar informações de fontes distintas transparentemente.

```
int main(int argc, char* const* argv)
{
 try
 {
  // Inicia o ORB
  CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
```

```
// Obtêm o Id do gerente
PortableServer::ObjectId_var managerId =
                PortableServer::string_to_ObjectId("S57Manager");

// Localiza o gerente. Fornece o nome completo do POA e o Id do gerente.
Manager::s57Manager_var manager =
          Manager::s57Manager::_bind("/S57_poa", managerId);

// Procura por um servidor que forneça a carta 1511
ChartServer::s57Server_var chart_server1511 = manager->getS57Server("1511");

if (CORBA::is_nil(chart_server1511))
{
  cout << "Não foi encontrado nenhum servidor que forneça a carta 1511" << endl;
  return 1;
}

// Procura por um servidor que forneça a carta 1512
ChartServer::s57Server_var chart_server1512 = manager->getS57Server("1512");

if (CORBA::is_nil(chart_server1512))
{
  cout << "Não foi encontrado nenhum servidor que forneça a carta 1512" << endl;
  return 1;
}

Chart::s57Chart_var chart1511 = chart_server1511->openChart("1511");

Chart::s57Chart_var chart1512 = chart_server1512->openChart("1512");

// Cria, no servidor 1511, um grupo de Bóias de Águas Seguras e
// um grupo de Bóias de Perigo à Navegação
```

```
  Chart::s57Group_var boysaw1511group = chart1511-> getFeatureByType
  ("BOYSAW");

  Chart::s57Group_var boyisd1511group = chart1511->getFeatureByType
  ("BOYISD");

 // Idem para o servidor 1512
  Chart::s57Group_var boysaw1512group = chart1512->getFeatureByType
  ("BOYSAW");

  Chart::s57Group_var boyisd1512group = chart1512->getFeatureByType
  ("BOYISD");

 // Constrói, a partir do servidor 1511, uma hierarquia de grupos tendo
 // como grupo mestre um grupo denominado BÓIAS
 Chart::s57Group_var boias = chart_server1511->createGroup("BÓIAS");

 Chart::s57Group_var boias1511 = chart_server1511->createGroup("BÓIAS1511");

 Chart::s57Group_var boias1512 = chart_server1511->createGroup("BÓIAS1512");

 boias1511->insertFeature( boysaw1511group );

 boias1511->insertFeature( boyisd1511group );

 boias1512->insertFeature( boysaw1512group );

 boias1512->insertFeature( boyisd1512group );

 boias->insertFeature( boias1511 );

 boias->insertFeature( boias1512 );
}
catch(const CORBA::Exception& e)
```

```
  {
    cerr << e << endl;
    return 1;
  }

  return 0;
}
```

C.5 EXEMPLO 5

Abaixo segue um exemplo de um processo cliente que realiza uma consulta sobre o grupo de bóias criado no exemplo anterior. O programa obtém a referência CORBA do grupo e realiza a operação de interseção entre o grupo da carta 1511 e o grupo da carta 1512 para obter as bóias comuns às cartas. Note que os grupos “BÓIAS 1511” e “BÓIAS 1512” são considerados feições inseridas no grupo pai “BÓIAS”.

```
int main(int argc, char* const* argv)
{
 try
 {
  // Inicia o ORB
  CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

  // Obtêm o Id do gerente
  PortableServer::ObjectId_var managerId =
                PortableServer::string_to_ObjectId("S57Manager");

  // Localiza o gerente. Fornece o nome completo do POA e o Id do gerente.
  Manager::s57Manager_var manager =
            Manager::s57Manager::_bind("/S57_poa", managerId);

  // Invoca os métodos do gerente
  // Procura por um servidor que forneça o grupo de bóias
  ChartServer::s57Server_var chart_server = manager->getS57Server("BÓIAS");

  if (CORBA::is_nil(chart_server))
  {
   cout << "Não foi encontrado nenhum servidor" << endl;
   return 1;
  }
  else
  {
```

```
    Chart::s57Group_var boias = chart_server->getGroup("BÓIAS");

    // Servidor encontrado. Obtêm a referencia CORBA do primeiro subgrupo
    Feature::s57Feature_var feicao1 = boias->getFeatureByName( "BÓIAS1511" );

    if (CORBA::is_nil(feicao1)) return 1;

    // Faz a mudança de tipo CORBA (Narrowing)
    Chart::s57Group_var boias1511 = Chart::s57Group::_narrow(feicao1);

    // Idem para o segundo subgrupo
    Feature::s57Feature_var feicao2 = boias->getFeatureByName( "BÓIAS1512" );

    if (CORBA::is_nil(feicao2)) return 1;

    Chart::s57Group_var boias1512 = Chart::s57Group::_narrow(feicao2);

    // Cria um grupo vazio
    Chart::s57Group_var intersecao = chart_server->createGroup("BÓIAS COMUNS
    ÀS DUAS CARTAS");

    // Insere as bóias da carta 1511 no novo grupo
    intersecao->groupUnion( boias1511 );

    // Faz a inteseção das bóias das duas cartas
    intersecao->groupIntersect( boias1512 );
  }
}
catch(const CORBA::Exception& e)
{
  cerr << e << endl;
  return 1;
}
```

```
  return 0;
}
```