

**UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA
COMPUTAÇÃO**

Rodrigo Bianco

**SICOGEF: Sistema Computacional do Genograma
Estrutural da Família**

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Prof. João Bosco Manguiera Sobral, Dr.

Florianópolis, Agosto de 2002

SICOGEF: Sistema Computacional do Genograma Estrutural da Família

Rodrigo Bianco

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Prof. Fernando Álvaro O. Gauthier, Dr. (UFSC)
Coordenador do Curso

Banca Examinadora

Prof. João Bosco Mangueira Sobral, Dr. (UFSC)
Orientador

Prof. Hugo Fuks, Ph.D. (PUC-Rio)

Prof. Paulo Sergio da Silva Borges, Dr. (UFSC)

Prof. Rosvelter João Coelho da Costa, Dr. (UFSC)

“Visão sem ação não passa de um sonho.
Ação sem visão é só um passatempo. Visão
com ação pode mudar o mundo.”

Gilclér Regina

A minha esposa Heloisa e ao meu filho Leonardo que me apóiam para lutar e atingir meus objetivos.

A meu pai Domingos Bianco e mãe Dalvina Jarros Bianco, que sempre estiveram ao meu lado nessa caminhada.

AGRADECIMENTOS

A Márcio Alberto Pires, Camila Cezarino Martins e Iracema pelo apoio desde o meu ingresso no curso;

A meu primo Jaime Bianco Júnior pela boa vontade em me apoiar principalmente no início do curso;

A meu irmão Ricardo Bianco, que mesmo estando longe e com pouco contato, sempre deu apoio para continuar meus estudos;

A Everton Schönardie Pasqual pelo companheirismo e atenção nos momentos difíceis;

A meu orientador, Professor João Bosco Mangureira Sobral, pelo incentivo e encorajamento para levar a bom termo este trabalho;

Ao Professor Juan Carlos Sotuyo e ao Professor Jorge Habib por terem me indicado como sendo apto a ingressar num curso de pós-graduação;

A Sirlei T. S. Fernandes e Adriana Mendes Pires de Campos, pois sem elas não existiria a idéia de se construir o SICOGEF;

A Carlos Augusto Santana Braga pela revisão do resumo em inglês.

A toda minha família que, direta ou indiretamente, ajudaram-me a concluir o curso.

SUMÁRIO

LISTA DE FIGURAS	X
LISTA DE ANEXOS.....	XIV
ABREVIATURAS E SIGLAS	XV
RESUMO	XIV
ABSTRACT	XV
1 INTRODUÇÃO.....	1
1.1 DEFINIÇÃO DO PROBLEMA.....	4
1.2 JUSTIFICATIVAS	4
1.3 PRINCIPAIS OBJETIVOS	5
1.4 FUNDAMENTAÇÃO TEÓRICA.....	5
1.4.1 <i>Genograma Estrutural da Família</i>	5
1.4.2 <i>Desenvolvimento Utilizando Componentes</i>	6
1.4.3 <i>Web Services e XML</i>	6
2 GENOGRAMA ESTRUTURAL DA FAMÍLIA	8
2.1 ESTADO DA ARTE	9
2.1.1 <i>Padrão GEDCOM</i>	9
2.1.2 <i>Software GenoPro</i>	9
2.2 PRINCIPAIS SÍMBOLOS DO GENOGRAMA ESTRUTURAL DA FAMÍLIA.....	10
2.3 MAPA ESTRUTURAL DA FAMÍLIA	15

3 DESENVOLVIMENTO COM COMPONENTES	18
3.1 COMPONENTES NAS INDÚSTRIAS	19
3.2 O CASO DO SOFTWARE.....	20
3.3 A UTILIZAÇÃO DE COMPONENTES NO SICOGEF.....	21
3.3.1 Componentes de Negócio.....	21
3.3.2 Componentes Visuais.....	22
3.3.3 Componentes para Manipulação de XML.....	22
3.3.4 Componentes para Acesso ao Banco de Dados.....	22
4 WEB SERVICES	23
4.1 VISÃO GERAL.....	23
4.1.1 Arquitetura de Comunicação para Web Service.....	24
4.1.2 Publicação das Interfaces de um Web Service na Internet	25
4.1.3 Empacotando as Mensagens para serem Transferidas pela Internet	26
5 REPRESENTANDO O GENOGRAMA ESTRUTURAL DA FAMÍLIA COM UML E XML.....	27
5.1 A UML PARA MODELAR O GENOGRAMA E O MAPA ESTRUTURAL DA FAMÍLIA.....	27
5.1.1 Representação de acordo com o Gênero.....	28
5.1.2 Representação de Paciente Identificado de acordo com o Gênero	28
5.1.3 Representação de Morte de acordo com o Gênero.....	28
5.1.4 Representação de um Matrimônio	29
5.1.5 Representação de uma Separação	30
5.1.6 Representação de um Divórcio	31
5.1.7 Representação de Convivência	31
5.1.8 Representação de uma Gravidez.....	32
5.1.9 Representação de Crianças que Nasceram Mortas	32
5.1.10 Representação de um Aborto Espontâneo.....	33
5.1.11 Representação de um Aborto Induzido.....	33
5.1.12 Representação de Filhos Naturais de acordo com o Gênero	33
5.1.13 Representação de Filhos Adotivos de acordo com o Gênero	34
5.1.14 Representação de Gêmeos Idênticos	34

5.1.15	<i>Representação de Gêmeos Fraternos</i>	35
5.1.16	<i>Representação para o caso de Aliança</i>	35
5.1.17	<i>Representação para o caso de Emaranhado</i>	35
5.1.18	<i>Representação para o caso de Conflito</i>	35
5.1.19	<i>Representação para o caso de Quebra no Relacionamento</i>	36
5.1.20	<i>Representação para o caso de Distância</i>	36
5.2	MAPEAMENTO DA UML PARA XML	36
5.2.1	<i>Classes UML para Elementos XML</i>	36
5.2.2	<i>Herança</i>	36
5.2.3	<i>Atributos UML para XML</i>	36
5.2.4	<i>Valores de Atributos Enumerados</i>	37
5.2.5	<i>Mapeando Composições UML</i>	37
5.2.6	<i>Mapeando Associações UML</i>	37
5.3	MAPEAMENTO DOS DIAGRAMAS DE CLASSES DO GENOGRAMA PARA XML	38
5.3.1	<i>Mapeamento de acordo com o Gênero</i>	38
5.3.2	<i>Mapeamento para Paciente Identificado de acordo com o Gênero</i>	39
5.3.3	<i>Mapeamento de Morte de acordo com o Gênero</i>	39
5.3.4	<i>Mapeamento de um Matrimônio</i>	40
5.3.5	<i>Mapeamento para o caso de uma Separação</i>	40
5.3.6	<i>Mapeamento para o caso de um Divórcio</i>	41
5.3.7	<i>Mapeamento para o caso de Convivência</i>	42
5.3.8	<i>Mapeamento para o caso de uma Gravidez</i>	43
5.3.9	<i>Mapeamento de Crianças que Nasceram Mortas</i>	43
5.3.10	<i>Mapeamento para o caso de um Aborto Espontâneo</i>	44
5.3.11	<i>Mapeamento para o caso de um Aborto Induzido</i>	45
5.3.12	<i>Mapeamento de Filhos Naturais de acordo com o Gênero</i>	45
5.3.13	<i>Mapeamento de Filhos Adotivos de acordo com o Gênero</i>	46
5.3.14	<i>Mapeamento de Gêmeos Idênticos</i>	46
5.3.15	<i>Mapeamento de Gêmeos Fraternos</i>	47
5.3.16	<i>Mapeamento para o caso de Aliança</i>	47
5.3.17	<i>Mapeamento para o caso de Emaranhado</i>	47

5.3.18 Mapeamento para o caso de Conflito	48
5.3.19 Mapeamento para o caso de Quebra no Relacionamento.....	48
5.3.20 Mapeamento para o caso de Distância	48
6 SICOGEF – IMPLEMENTAÇÃO DO SISTEMA.....	49
6.1 MONTAGEM INICIAL DO PROJETO PARA GERAR DLL DE NEGÓCIO	51
6.2 CONFIGURAÇÕES DO AMBIENTE C++ BUILDER 6 ENTERPRISE.....	57
6.3 DETALHES SOBRE PADRÕES DE CODIFICAÇÃO	63
6.3.1 Tratamento de Exceções	63
6.3.2 Tratamento do XML	64
6.3.3 Observações sobre Classes de Persistência.....	66
6.4 MONTAGEM INICIAL DO PROJETO PARA GERAR DLL WEB SERVICE SERVIDOR.....	67
6.5 MONTAGEM DO CONSUMIDOR DE WEB SERVICE	71
6.6 MONTAGEM DA APLICAÇÃO CLIENTE.....	74
7 TRABALHOS FUTUROS.....	76
8 CONCLUSÃO.....	79
9 ANEXOS	81
10 GLOSSÁRIO.....	90
11 REFERÊNCIAS BIBLIOGRÁFICAS	91

LISTA DE FIGURAS

Figura 2.1: Símbolos de acordo com o gênero	10
Figura 2.2: Símbolos para paciente identificado de acordo com o gênero	11
Figura 2.3: Símbolos de acordo com o gênero e representando a morte do indivíduo ...	11
Figura 2.4: Representação de um matrimônio	12
Figura 2.5: Representação de separação (A) e divórcio (B).....	12
Figura 2.6: Representação de convivência.....	12
Figura 2.7: Representação de gravidez.....	13
Figura 2.8: Crianças que nasceram mortas. A letra A representa uma menina e a B um menino	13
Figura 2.9: Representações para o aborto. A letra A representa um aborto espontâneo e a B um induzido.....	14
Figura 2.10: Representação de filhos naturais. A letra A representa uma menina e a B um menino	14
Figura 2.11: Símbolos para crianças adotivas. A letra A representa uma menina adotada e a B um menino	14
Figura 2.12: Símbolo para gêmeos. A letra A representa os gêmeos fraternos e a B os gêmeos idênticos	15
Figura 2.13: Símbolo para representar a aliança	15
Figura 2.14: Representação para emaranhamento	15
Figura 2.15: Símbolo para representar conflito.....	16
Figura 2.16: Representação para quebra do relacionamento	16
Figura 2.17: Símbolo para representar distância.....	16
Figura 2.18: Exemplo de genograma com mapa estrutural da família	16

Figura 5.1: Representação do elemento Pessoa.....	28
Figura 5.2: Representação da morte de uma pessoa	29
Figura 5.3: Representação da associação matrimônio.....	29
Figura 5.4: Representação de uma separação	30
Figura 5.5: Representação de um divórcio	31
Figura 5.6: Representação de convivência.....	31
Figura 5.7: Representação de uma gravidez	32
Figura 5.8: Representação quando uma criança nasce morta	32
Figura 5.9: Representação de aborto.....	33
Figura 5.10: Representação de filhos naturais	33
Figura 5.11: Representação de filhos adotivos	34
Figura 5.12: Representação de gêmeos.....	34
Figura 5.13: Representação do mapa estrutural.....	35
Figura 5.14: XML para representação de acordo com o gênero	39
Figura 5.15: XML para representação de um paciente identificado.....	39
Figura 5.16: XML para representar a morte de uma pessoa	40
Figura 5.17: XML para representar o matrimônio entre duas pessoas.....	40
Figura 5.18: XML para representar a separação entre duas pessoas	41
Figura 5.19: XML para representar o divórcio entre duas pessoas	42
Figura 5.20: XML para representar a convivência entre duas pessoas	43
Figura 5.21: XML para representar uma gravidez.....	43
Figura 5.22: XML para representar quando uma criança nasce morta	44
Figura 5.23: XML para representar um aborto.....	45
Figura 5.24: XML para representar o filhos	45
Figura 5.25: XML para representar gêmeos.....	47
Figura 5.26: XML para representar o mapa estrutural.....	47
Figura 6.1: Opção para criar uma ActiveX Library.....	51
Figura 6.2: Arquivos existentes durante a criação de uma Active Library	52
Figura 6.3: Projeto já com os nomes atualizados após o salvamento.....	53
Figura 6.4: Escolhendo a opção COM Object para a criação do componente	54
Figura 6.5: Tela inicial para configuração do componente COM+	54

Figura 6.6: Tela inicial do Type Library.....	55
Figura 6.7: Configuração dos parâmetros de entrada e saída.....	56
Figura 6.8: Salvando arquivo do Type Library que conterá a implementação COM+....	57
Figura 6.9: Configuração de cache para código pré-compilado	58
Figura 6.10: Configuração do diretório intermediário.....	59
Figura 6.11: Tela com configurações necessárias para debug DLL.....	60
Figura 6.12: Tela da ferramenta Serviços de Componente.....	61
Figura 6.13: Tela de propriedade do aplicativo contendo ID necessário para “debug” ..	62
Figura 6.14: Exemplo de tratamento de exceções nas classes limítrofes.....	63
Figura 6.15: Exemplo de tratamento de exceções nas classes de negócio	64
Figura 6.16: Exemplo de FOR alinhado para percorrer XML entrante e acionar métodos de negócio apropriados	65
Figura 6.17: Exemplo mostrando como converter os dados do banco em XML	66
Figura 6.18: SOAP Server Application para implementação do Web Service	67
Figura 6.19: Configuração do tipo de Web Service.....	68
Figura 6.20: Confirmação se cria ou não a Interface para o Web Service	68
Figura 6.21: Configuração do nome para o Web Service.....	69
Figura 6.23: Arquivo com os métodos que serão disponibilizados pelo Web Service....	70
Figura 6.24: Exemplo de código fonte que cria o componente de negócio.....	70
Figura 6.25: Página de retorno WSDL.....	72
Figura 6.26: Tela principal do Web Services Importer	73
Figura 6.27: Acesso aos métodos através de uma Interface Remota de Invocação.....	74
Figura 7.1: Forma atual de representação do genograma.....	77
Figura 7.2: Nova proposta de representação do genograma	78
Figura 9.1: Diagrama de implantação para o SICOGEF	82
Figura 9.2: Exemplo genérico de mapeamento da UML para XML [CAR01]	83
Figura 9.3: Diagrama de classes.....	84
Figura 9.4: MER – Primeira parte.....	85
Figura 9.5: MER – Segunda parte	86
Figura 9.6: Composição geral do XML genograma.....	87
Figura 9.7: Pais de uma pessoa.....	87

Figura 9.8: Referência de uma pessoa a uma fonte	88
Figura 9.9: Dados binários.....	88
Figura 9.10: Representação completa	89

LISTA DE ANEXOS

Anexo 1 – Diagrama de implantação do SICOGEF.....	82
Anexo 2 – Exemplo genérico de mapeamento UML para XML	83
Anexo 3 – Diagrama de classes.....	84
Anexo 4 – MER – Modelo entidade relacionamento	85
Anexo 5 – Outras representações em UML do XML	87

ABREVIATURAS E SIGLAS

Sigla	Descrição
ADO	Activex Data Objects
DLL	Dinamic Link Library
DOM	Document Object Model
DTD	Document Type Definition
GEDCOM	Genealogical Data Communications
GUI	Graphics User Interface
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
PI	Paciente Identificado
RUP	Rational Unified Process
SICOGEF	Sistema Computacional do Genograma Estrutural da Família
SQL	Structured Query Language
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
UML	Unified Modeling Language
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
Xlink	XML Linking Language
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language

RESUMO

O objetivo deste trabalho é mostrar uma abordagem em Sistemas Distribuídos para o desenvolvimento de um sistema chamado SICOGEF, o qual implementa o Genograma – uma arquitetura para o entendimento de padrões familiares.

Como solução ao problema de distribuição das aplicações, utilizou-se a tecnologia de Web Services. A modelagem da arquitetura foi desenvolvida em cima do paradigma de modelagem de aplicações XML (Extensible Markup Language) com UML (Unified Modeling Language). A utilização de componentes de Software permitiu um desenvolvimento rápido com qualidade.

ABSTRACT

The purpose of this work is to show an approach in Distributed Computing for the development of a system called SICOGEF, which implements the Genogram – a framework for understanding family patterns.

As a solution to the problem of distribution of the applications, the Web Services technology was used. The architecture modeling was developed on the paradigm of modeling XML (Extensible Markup Language) applications with UML (Unified Modeling Language). The use of software components allowed a rapid development with quality.

1 INTRODUÇÃO

Muitos profissionais de diversas áreas podem tirar proveito das vantagens que um software possa trazer. Porém, devemos ser cuidadosos ao adquirir ou construir um software, como *Capers Jones* reconhece em seu livro *Software Assessments, Benchmarks, and Best Practices*:

“Software tem sido uma boa tecnologia por mais de 50 anos. Computadores e software são as maiores ferramentas de negócios, governos e organizações militares, porém eles podem também ser problemáticos, caros e propensos a erros” [JON00].

Com essas idéias, podemos perceber que software é algo interessante e amplamente utilizado, porém, se não for bem avaliada a forma de sua construção, problemas poderão aparecer no futuro. Mas existem meios de se minimizar esses problemas referentes ao software: “O caminho mais efetivo para reduzir os problemas de software é estudar todas as possíveis formas de se construir e manter aplicativos” – para usar a frase de *Jones*.

Essa dissertação tem como objetivo central, propor um sistema distribuído de computador que atenda as necessidades de vários profissionais em utilizar a técnica do genograma através desse sistema, possibilitando diversas vantagens em relação à forma manual de elaboração atual.

Segundo a sugestão colocada por *Jones*, várias formas de se construir e manter aplicativos foram avaliadas, sendo o desenvolvimento com componentes o escolhido para a aplicação em si e o recurso de Web Services para a interoperabilidade e troca de informações entre os aplicativos que estarão distribuídos.

Esse trabalho procura demonstrar como as novas tecnologias e teorias da área da computação podem ser aplicadas para apoiar e procurar amenizar um problema encontrado pelos profissionais da área humana (psicólogos, sociólogos, terapeutas e outros), que é fazer o genograma com papel e caneta.

Embora ainda não sejam práticas amplamente utilizadas (no caso a utilização de componentes e processos afins), como se expressa *Capers Jones*:

“Algumas práticas de reutilização interessantes ainda não tem sido utilizadas por nossos clientes para determinar se elas irão se tornar as melhores práticas. A prática mais significativa é o desenvolvimento baseado em componentes. Quando esse livro foi escrito, poucos dos nossos clientes usavam desenvolvimento baseado em componentes, poucos componentes estavam disponíveis e os resultados do desenvolvimento de componentes ainda são ambíguos” [JON00].

Outros escritores reconhecidos crêem num futuro brilhante para esse tipo de desenvolvimento, onde, acredita-se que seja a solução para a crise de software que ainda existe, como *Peter M. Heinckiens* diz em suas observações:

“Uma tendência fundamental em computação é a evolução das aplicações de códigos monolíticos para sistemas baseados em componentes. Evidências de aumento na modularização do software e distribuição são visíveis” [HEI98].

Ainda segundo *Carpes Jones* com seus estudos, a indústria de software têm dois problemas crônicos que devem ser resolvidos antes que o desenvolvimento baseado em componentes tenha sucesso total:

“Primeiro, convenções de interface padrão devem ser aceitas para poder unir os componentes de forma organizada. Segundo, controle de qualidade de software deve ser capaz de construir componentes que aproximam ou alcançam níveis de defeitos zero” [JON00].

Com relação a interface, Web Services e o protocolo SOAP (Simple Object Access Protocol) está tornando possível que os componentes possam cooperarem entre si através da Internet, independente de linguagem ou plataforma onde estejam esses componentes [ZAE01].

Já, com relação a qualidade dos componentes desenvolvidos, a medida que os engenheiros de software estão tendo acesso a novos processos de desenvolvimento, ferramentas e reutilização de códigos, deve-se esperar que a qualidade melhore, principalmente nas empresas que tem algum componente no mercado e que já passou por várias fases de amadurecimento.

Essa preocupação com a qualidade dos componentes de software é justificável por dois dados estatísticos muito interessantes retirados do livro de *Jones*:

“O fator que mais influencia positivamente a produtividade do desenvolvimento de software, é o uso de materiais reutilizáveis de alta qualidade. Da mesma forma que esses materiais de alta qualidade ajudam, por outro lado, a utilização de materiais de baixa qualidade são os que mais atrapalham na produtividade do desenvolvimento de software” [JON00].

Convém lembrar que esses materiais reutilizáveis não se restringem apenas aos componentes, podendo ser: requerimentos, projetos, código fonte, material para teste, documentação do usuário e outros.

Uma forma de se ter componentes de qualidade é adquiri-los de empresas comerciais especializadas no desenvolvimento de componentes. Normalmente essas empresas são altamente centradas em soluções específicas, conseguindo com isso obter grande conhecimento na área e, como reflexo, a evolução com qualidade dos seus componentes.

Nos capítulos seguintes, iremos esclarecer o que é o genograma com seus símbolos atuais; revisar alguns conceitos teóricos do desenvolvimento com componentes e meios práticos de se chegar ao sistema proposto através dessa filosofia de desenvolvimento; modelar as informações XML (Extensible Markup Language) com UML (Unified Modeling Language); e por fim, como fazer para que o SICOGEF possa funcionar de forma

distribuída utilizando os recursos e conceitos de Web Services.

1.1 Definição do Problema

Profissionais que trabalham com famílias, constantemente precisam lidar com diversas informações sobre cada um dos membros ou dados referentes à família como um todo. A situação se complica quando esses profissionais precisam repassar essas informações entre eles para troca de opiniões ou tomada de decisão coletiva [LEO97].

O ciclo de vida familiar é um fenômeno complexo. Ele é uma espiral da evolução familiar, na medida em que as gerações avançam no tempo em seu desenvolvimento do nascimento à morte [CAR95]. Se fosse possível mapear essa evolução familiar através de um sistema que permitisse a modelagem do genograma (técnica utilizada para mapear a situação atual de uma família), esses profissionais teriam melhores condições de intervir junto ao indivíduo ou na família, para que possam achar a solução para os problemas.

1.2 Justificativas

Hoje, esses profissionais realizam essas tarefas utilizando meios tradicionais como papel e lápis para rascunhar o genograma. Além da péssima qualidade, tanto pela falta de prática de alguns profissionais em desenhar, como pelo próprio artefato utilizado para a construção do genograma, esses profissionais enfrentam grandes dificuldades para trocar informações uns com os outros devido as limitações físicas do genograma desenhado. Esse problema se agrava, ainda mais, quando o outro profissional se encontra em uma localidade geográfica diferente, sendo o processo de transferência do genograma feito via FAX, degradando ainda mais a qualidade, além de trazer grandes custos com ligações telefônicas para explicar a situação da família representada no genograma em questão [BRO98]. Outra limitação é a pobre quantidade de informações que são adicionadas ao genograma, pois existe um limite espacial que impede a inclusão de muitas informações.

Esperamos com o SICOGEF atender essas demandas e melhorar substancialmente a forma de trabalho desses profissionais, refletindo diretamente na

qualidade do serviço realizado.

1.3 Principais Objetivos

- Esclarecer quais são os símbolos usados no genograma;
- Mostrar que os princípios básicos de construção através de componentes e comunicação com SOAP é algo factível;
- Demonstrar que Web Services podem ser construídos e consumidos com as tecnologias correntes;
- Mostrar como representar os principais símbolos do genograma familiar através da UML;
- Como realizar o mapeamento da UML para XML usando como base diagramas de classes extraídos de cada estrutura básica do genograma familiar;
- Montar uma arquitetura de software que atenda às necessidades específicas de um problema da área humana (modelagem do genograma).

1.4 Fundamentação Teórica

Três fundamentos teóricos deverão estar explícitos para o sucesso do desenvolvimento do SICOGEF. O primeiro deles é entender realmente como funciona e quais são os principais símbolos utilizados no genograma. O outro fundamento é o domínio dos conceitos e técnicas de programação de Web Services e XML [MCI01], visto que os genogramas terão que ser convertidos para esse padrão, de modo que, possibilite a troca de informações entre usuários SICOGEF e outros sistemas que porventura queiram visualizar esses genogramas. Por fim, o último fundamento será o de desenvolvimento utilizando componentes, o qual tem um grande embasamento teórico e na prática existem vários componentes para serem reutilizados, diminuindo o tempo de programação e erros encontrados no produto final.

1.4.1 Genograma Estrutural da Família

A abordagem sobre o genograma basicamente consistirá em demonstrar seus símbolos

básicos e em capítulos posteriores demonstrar como esses símbolos e suas informações podem ser modelados através da UML e convertidos em XML para o caso de trocas através de solicitações Web Services.

1.4.2 Desenvolvimento Utilizando Componentes

O paradigma de desenvolvimento utilizando componentes será utilizado para a implementação do SICOGEF. Existem vários motivos para essa escolha, entre os principais: reutilização de código existente de qualidade; componentes já amplamente testados; diminuição do tempo de entrega do produto final [GAE99]. Existe todo um contexto histórico e de evolução até chegar ao nível que estamos hoje de reutilização de componentes, sendo esse contexto histórico e de evolução detalhados num próximo capítulo.

1.4.3 Web Services e XML

Um Web Service é uma classe escrita em uma linguagem que tem suporte para implementá-lo e que pode ser acessada via protocolo HTTP (Hypertext Transfer Protocol). Isso significa que se pode acessar qualquer Web Service disponível na Web e utilizar todas as funcionalidades do mesmo [AOY02].

O acesso sempre será via HTTP, mas internamente existe um documento XML que está empacotado em um protocolo SOAP (Simple Object Access Protocol). O SOAP é um padrão aberto criado pela Microsoft, Ariba e IBM para padronizar a transferência de dados em diversas aplicações, por isso, se dá em XML.

Extensible Markup Language (XML) é uma linguagem de marcação de dados (meta-markup language) que provê um formato para descrever dados estruturados. O XML permite a definição de um número infinito de tags. Enquanto no HTML, se as tags podem ser usadas para definir a formatação de caracteres e parágrafos, o XML fornece um sistema para criar tags para dados estruturados. Como as tags XML são adotadas por Intranets de organizações, e também via Internet, haverá uma correspondente habilidade em manipular e procurar por dados independentemente das aplicações [ROY01].

No caso do SICOGEF é importante a utilização do XML para representar os genogramas, pois facilita na hora de trocar informações entre os usuários ou outros

dispositivos. Uma grande vantagem do XML é que ele é um padrão aberto proposto pela W3C (World Wide Web Consortium), consórcio que controla os padrões para Internet. Junto com o XML surgiram várias outras tecnologias para apoiar a sua utilização, entre as principais: DOM (Document Type Definition), uma nova forma de acessar os elementos do documento XML usando qualquer linguagem de programação; DTD (Document Type Definition), define quais as tags possíveis para um documento XML; XSL (Extensible Stylesheet Language), responsável pela formatação do documento XML em HTML (HyperText Markup Language) ou outro formato; e muito mais, porém foge do escopo desse trabalho detalhar cada uma.

2 GENOGRAMA ESTRUTURAL DA FAMÍLIA

Genograma familiar é uma representação gráfica de uma constelação familiar de várias gerações [FER99]. Para que o conceito fique claro, vamos explicar de uma forma menos formal. O genograma familiar procura através de símbolos gráficos, que são atribuídos para cada membro de uma família que esteja sendo analisada, demonstrar as principais características dos indivíduos e da família como um todo. Alguns autores também utilizam o nome genetograma.

“Os genetogramas são retratos gráficos da história e do padrão familiar, mostrando a estrutura básica, a demografia, o funcionamento e os relacionamentos da família. Eles são uma taquigrafia utilizada para descrever os padrões familiares à primeira vista” [CAR95].

Normalmente no genograma aparecem nomes, idades, datas de matrimônio, divórcio, mortes, enfermidades, paciente identificado, escolaridade, status familiar, altura, posição estrutural de cada indivíduo e as etapas do ciclo vital familiar [FER99].

Com o genograma familiar, vários profissionais da área terapêutica, jurídica, social e outros que trabalham diretamente ou não com famílias, podem fazer a leitura da realidade social dessas famílias no presente e juntamente com elas, tentar projetar um futuro diferente.

Os principais símbolos serão demonstrados em seções posteriores juntamente com a representação em UML e o mapeamento para XML.

2.1 Estado da Arte

A idéia de representar a estrutura familiar não é nova, existindo padrões e vários software no mercado para realizar as tarefas de modelar um genograma. Entre os padrões o mais significativo é o GEDCOM ("GEnealogical Data COMmunications" - Comunicação de Dados Genealógicos), utilizado para representar informações genealógicas. No caso de software um que tem aceitação é o GenoPro.

Mesmo com as opções existentes no mercado, a maioria não tem recursos específicos que terapeutas familiares precisam para realmente mapear a semântica de uma família. Entre esses recursos, podemos citar a representação de limites familiares, mapa estrutural (dizendo como é a relação dos indivíduos entre si) e elaboração de histórico de genogramas de uma mesma família.

2.1.1 Padrão GEDCOM

O formato GEDCOM é um protocolo de intercâmbio de dados padronizado para simplificar o trabalho de troca de informações deste tipo.

Embora, o GEDCOM seja o padrão oficialmente adotado, ele possui limitações, entre elas, o difícil entendimento por seres humanos (não possui uma estrutura clara como ocorre no XML). Propostas de substituição do GEDCOM por padrões em XML já estão sendo estudadas. Não é objetivo do SICOGEF elaborar uma proposta de substituição para o GEDCOM, porém um formato em XML será elaborado para uso interno do SICOGEF.

2.1.2 Software GenoPro

O SICOGEF não será o primeiro software para modelagem do genograma. Existem vários outros no mercado (Personal Ancestral File – PAF, Fzip Family Tree 1.7, GenoPro, etc.) sendo um dos mais populares o GenoPro. Porém, no contexto brasileiro e com as informações que os nossos profissionais necessitam, o SICOGEF é um projeto inédito que buscará atender às necessidades desses profissionais.

Para maiores informações sobre GenoPro, www.genopro.com.

2.2 Principais Símbolos do Genograma Estrutural da Família

Um genograma é um mecanismo para representar uma árvore genealógica que registra informações sobre os membros de uma família e seus relacionamentos de pelo menos três gerações [MCG99].

“Quando avaliamos o lugar de uma família no ciclo de vida, os genogramas e as cronologias familiares constituem instrumentos úteis. Eles proporcionam uma visão de um quadro trigeracional de uma família e de seu movimento através do ciclo de vida” [CAR95].

Os genogramas mostram informações familiares de uma forma que provê uma visão rápida dos padrões familiares complexos e uma fonte rica de hipótese sobre como um problema clínico pode estar conectado com o contexto familiar e a evolução do problema e do contexto com o tempo [MAR89].

O principal recurso de um genograma é a sua capacidade de descrever graficamente como diferentes membros de uma família estão biologicamente e legalmente relacionados uns com os outros entre as gerações. Cada membro da família é representado com um quadrado ou um círculo dependente do seu gênero (masculino ou feminino).

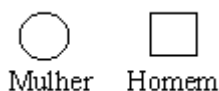


Figura 2.1: Símbolos de acordo com o gênero

O paciente identificado (PI), o qual é o foco central para a construção do genograma, se identifica com linhas duplas. Segundo *Sirlei T. S. Fernandes* o paciente identificado é caracterizado como:

“É a pessoa que apresenta o sintoma da disfuncionalidade familiar. O PI sempre é o motivo da busca de ajuda. Nem sempre é o motivo de ajuda. Nem sempre é ele que precisa de auxílio. Muitas vezes, a sintomatologia está a serviço de uma desviação de conflito, ou seja, a separação dos pais, doença na família, desemprego, drogadição, alcoolismo e outros” [FER99].

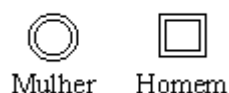


Figura 2.2: Símbolos para paciente identificado de acordo com o gênero

Para o caso de pessoas que já faleceram, um “X” é colocado dentro do símbolo (quadrado ou círculo, de acordo com o gênero em questão).

Em genogramas estendidos que alcançam mais de três gerações, os símbolos que estão no passado distante usualmente não levam um “X”, dado que estão presumidamente mortos.

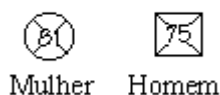


Figura 2.3: Símbolos de acordo com o gênero e representando a morte do indivíduo

Os símbolos que representam os membros de uma família estão conectados por linhas que demonstram seus relacionamentos biológicos e legais. As pessoas que estão casadas, estão conectadas por uma linha simples, com o marido à esquerda e a esposa à direita. Como podemos ver na Fig. 2.4, uma letra (para o matrimônio utilizamos a letra “m”) seguida por um ano (dois dígitos) indica quando ocorreu o matrimônio.

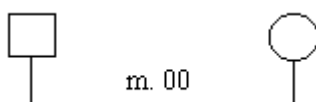


Figura 2.4: Representação de um matrimônio

A mesma linha que une um casal também é o lugar onde as separações e divórcios serão indicados. Linhas oblíquas significam uma interrupção no matrimônio sendo: uma diagonal para separação e duas para representar um divórcio, como é ilustrado na Fig. 2.5A e 2.5B respectivamente.



Figura 2.5: Representação de separação (A) e divórcio (B)

No caso da Fig. 2.5A o “s. 01” significa que houve uma separação no ano de 2001. Já na Fig. 2.5B ocorreu um divórcio no ano de 2002.

Para representar um relacionamento de convivência seguimos o mesmo princípio que o de matrimônio, porém a linha é tracejada. A data representada no relacionamento é quando o casal se conheceu e passaram a viver juntos.

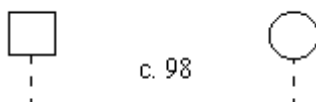


Figura 2.6: Representação de convivência

Por enquanto vimos somente os símbolos de acordo com o gênero, como representar o paciente identificado, mortes e as formas de relacionamento entre um casal (matrimônio, separação, divórcio e convivência). Vamos agora verificar como representar gravidez e seus problemas, bem como os tipos de filhos.

Para representar uma gravidez, basta utilizar o símbolo representado na Fig. 2.7 conectado ao relacionamento que tem a gravidez. No caso, a gravidez é representada por um triângulo.



Figura 2.7: Representação de gravidez

Quando uma criança nasce morta, seguimos o mesmo princípio do caso de adultos, ou seja, basta colocar um “X” no símbolo da criança que estaria nascendo, podendo ser menino ou menina.

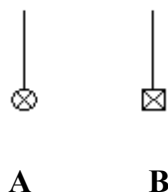


Figura 2.8: Crianças que nasceram mortas. A letra A representa uma menina e a B um menino

Para o caso do Aborto temos duas condições para serem representadas. A primeira é o aborto espontâneo, que é representado com uma bola negra. A outra é o aborto induzido, que é representado somente com um “X”.

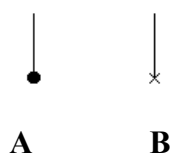


Figura 2.9: Representações para o aborto. A letra A representa um aborto espontâneo e a B um induzido

No caso de filhos naturais do casal a representação é simples, bastando colocar o símbolo do gênero da criança abaixo do relacionamento do casal.

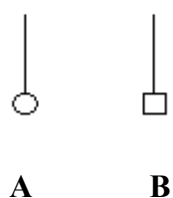


Figura 2.10: Representação de filhos naturais. A letra A representa uma menina e a B um menino

Quando uma criança é adotada por um casal, o símbolo ao invés de ter uma haste normal, terá uma que seja tracejada. A Fig. 2.11 demonstra os símbolos para crianças adotivas.

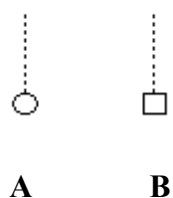


Figura 2.11: Símbolos para crianças adotivas. A letra A representa uma menina adotada e a B um menino

O conflito poderá aparecer em qualquer movimento de inter-relação dos subsistema.



Figura 2.15: Símbolo para representar conflito

A quebra de um relacionamento é algo que, normalmente, ocorre devido a fatos que leve ao rompimento total do relacionamento.

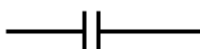


Figura 2.16: Representação para quebra do relacionamento

O último símbolo que veremos é o de distância, representando que as partes envolvidas estão distantes. Sua representação é através de uma linha pontilhada.



Figura 2.17: Símbolo para representar distância

A descrição dos símbolos para o mapa estrutural da família não faz sentido sem uma ilustração prática de sua utilização. Por exemplo, um casal pode estar casado legalmente, porém na vida real pode ter um relacionamento distante. É isso que tenta representar o pequeno genograma (Fig. 2.18) já com o mapa estrutural sendo demonstrado com a linha tracejada entre os indivíduos do casamento.

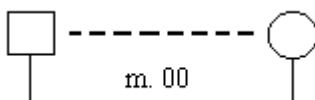


Figura 2.18: Exemplo de genograma com mapa estrutural da família

Os símbolos que acabamos de descrever são usados na atualidade para montar um genograma. Pode-se perceber que não se trata de um conjunto poderoso de símbolos para representar toda a complexidade de informações que existe em uma família. Porém, consegue ilustrar alguns aspectos interessantes e, mesmo que seja um conjunto simples, não deixa de ser um instrumento útil para os profissionais.

O escopo dessa dissertação fica em torno desses símbolos, deixando para trabalhos futuros, propor novos símbolos e melhorias no modo de se construir um genograma.

3 DESENVOLVIMENTO COM COMPONENTES

Os conceitos de reutilização e de componentes são relativamente antigos na ciência da computação. A utilização prática desses conceitos revestia-se de aspectos muito simples e não preenchia, até o momento, as expectativas em torno de indicadores como a produtividade, a qualidade e a extensibilidade.

Porém, esse quadro vem mudando e hoje já é possível reutilizar componentes, principalmente os comerciais, para desenvolver aplicações com qualidade e com uma produtividade boa [GRU01].

Apesar de a tentativa de reutilização de código ser uma aspiração antiga, a sua adoção era limitada, fundamentalmente porque era difícil projetar software reutilizável [MEY00]. A construção de código reutilizável requer um elevado nível de abstração, paralelamente a uma boa técnica de programação, porque as variáveis, assim como as combinações das diferentes situações de utilização, poderão ser numerosas.

Como podemos perceber nas palavras de *Szyperski* essa limitação está sendo superada e outras vantagens surgem da utilização de componentes:

“Criando novas soluções combinando componentes comprados e feitos, aumentam a qualidade e suporta desenvolvimento rápido, levando a um curto tempo de entrega para o mercado. Ao mesmo tempo, uma adaptação mais rápida para mudança de requerimentos pode ser alcançada investindo somente em mudanças-chaves de uma solução baseada em componentes, ao invés de se responsabilizar em entregar uma mudança maior” [SZY98].

A indústria de software apresenta problemas evidentes desde os anos 50, por não dar vazão ao crescimento exponencial da procura e por continuar a revelar lacunas em termos de qualidade e confiabilidade. Constata-se, igualmente, que nos últimos anos a dimensão dos sistemas aumentou, a sua complexidade cresceu, e a necessidade de convivência com outros sistemas tornou-se um requisito comum.

“A evolução de manipulação de dados para manipulação de informações tem implicações significantes para engenheiros de software. Essa evolução vem tornando os sistemas de software modernos muito mais complexos. Adicione ainda a evolução explosiva de recursos de hardware disponível, requerimentos de usuários sempre crescendo e a demanda por interfaces gráficas para o usuário (GUIs). Além disso tudo, acrescente a necessidade de ficar compatível com velhos terminais baseados em caracteres e a demanda por flexibilidade, portabilidade e extensibilidade” [HEI98].

Neste cenário, a questão do desenvolvimento de sistemas baseado em componentes retoma relevância e, com isso, a necessidade de se verem reforçadas as boas práticas de projeto, centradas na elevada coesão e no baixo acoplamento entre as partes de um sistema.

3.1 Componentes nas Indústrias

Em indústrias mais tradicionais que a do software, a questão dos componentes e das respectivas técnicas de construção amadureceram antes. Compare, por exemplo, o desenvolvimento de software com a construção de uma ponte. A indústria de software é relativamente recente e imatura, mas sabemos que se constroem pontes desde três mil anos atrás. Depois, em geral, as pontes são construídas no prazo, dentro do orçamento previsto e habitualmente não caem. Isso porque esse tipo de construção se rege por planos, desenhos e cálculos de engenharia bem detalhados, que variam pouco ao longo do ciclo de projeto.

No caso do desenvolvimento de software, não ocorre dessa forma. É habitual

que as especificações se alterem, às vezes de forma significativa, ao longo do projeto. Por outro lado, quando uma ponte cai, isso se deve, regra geral, a eventos catastróficos (por exemplo terremotos). Na indústria de construção de software a situação é diferente. As falhas e os insucessos tendem freqüentemente a serem minimizados e escondidos, geralmente para não desprestigiar a imagem pública da organização.

“Quais são os benefícios de software componente? A resposta mais simples é: componentes é o caminho para seguir, pois todas as outras disciplinas de engenharia introduziram componentes que se tornaram maduros - e ainda são utilizados” [SZY98].

3.2 O Caso do Software

Podemos dizer que o software ainda é construído por uma determinada entidade, por exemplo, por uma empresa de desenvolvimento (Software House) ou pelos departamentos de sistemas de informação das empresas. Se é certo que nesse desenvolvimento são usados instrumentos de terceiros, tais como compiladores e verificadores de erros, também se sabe que raramente se incorporam componentes externos à aplicação. Quando muito, pode ser reutilizado algum código de outro projeto, desde que esteja devidamente documentado ou quando da equipe de desenvolvimento faz parte, pelo menos, um programador que se lembra das funcionalidades incorporadas no referido código.

Mas, esse cenário vem mudando. Cada vez mais, as empresas preocupam-se em criar componentes ou adquiri-los de empresas terceiras, já prontos ou sob encomenda. Todo um movimento da indústria de software está propiciando e viabilizando a utilização dos conceitos de desenvolvimento utilizando componentes.

Livros teóricos de qualidade sobre o assunto estão disponíveis no mercado, congressos são realizados e pesquisas de empresas e Universidades dão força para o crescimento. Tecnologias como COM+, JavaBeans, CORBA e vários outros servidores de aplicações (que podemos imaginar como servidores de componentes) dão o respaldo prático

para a construção de sistemas distribuídos complexos.¹

3.3 A Utilização de Componentes no SICOGEF

Quando dizemos que o SICOGEF estará utilizando componentes para a montagem de sua arquitetura, podemos entendê-los como sendo de negócio, componentes visuais, componentes para a manipulação de XML e para acesso ao banco de dados. A seguir vamos detalhar melhor cada um deles.

3.3.1 Componentes de Negócio

Os componentes de negócio serão DLL (Dynamic Link Library) criadas com a linguagem C++ através dos recursos disponíveis no ambiente de programação C++ Builder. Essas DLL serão desenvolvidas de modo a suportar sua instalação no COM+. Basicamente, um componente de negócio será composto por várias classes que podem ser: classes limítrofes, classes de controle e classes entidade.

As classes limítrofes são as que normalmente estão na fronteira do componente. No nosso caso, serão as classes limítrofes que farão interface com os Web Services, que por sua vez, podem ser considerados limítrofes da parte servidora como um todo.

Já, as classes de controle gerenciam processos e regras de negócio que possam envolver vários objetos. Regra individual que se aplica ao próprio objeto, como a validação de um atributo é responsabilidade do objeto de negócio que está modelado numa classe de entidade. Basicamente cada caso de uso² ganhou uma classe de controle própria.

¹ É claro que essa tendência para a utilização de componentes também tem um lado financeiro, pois sai mais barato usando esses conceitos. “O investimento adicional requerido para produzir componentes, ao invés de soluções especializadas completas, somente pode ser justificado se o retorno do investimento vier.” [SZY98]

² Caso de uso é a descrição de um conjunto de seqüência de ações realizadas pelo sistema que proporciona resultados observáveis de valor para um determinado ator. Um caso de uso é utilizado para estruturar o comportamento de itens em um modelo. [GAR99]

3.3.2 Componentes Visuais

Dentro do que chamamos de componentes visuais, podemos dizer que, qualquer componente padrão do Delphi (o ambiente escolhido para fazer a parte cliente do SICOGEF) para Interface se encaixa nesse escopo.

Dentre os componentes visuais, temos que destacar a utilização de um componente comercial específico para representar fluxogramas. O componente em questão é o FlowChart da empresa DevExpress (www.devexpress.com), o qual permitiu, de uma forma rápida, achar algumas soluções básicas quando se trata de manipulação de elementos gráficos. Se não fosse possível a utilização de um componente com essas características, a complexidade de se desenvolver um seria, de tamanha complexidade que, provavelmente, inviabilizaria um desenvolvimento rápido e de qualidade.

3.3.3 Componentes para Manipulação de XML

Tanto na parte servidora como na cliente, temos que ter alguns recursos para facilitar na hora de manipular documentos XML. Esses recursos estão disponíveis no ambiente Windows (as vezes é necessário fazer algumas atualizações no sistema operacional para constar as últimas versões) como componentes que podem ser utilizados. Tanto no Delphi como no C++ Builder serão componentes que podem ser importados para sua aplicação através do menu *Project | Import Type Library* e escolher o Microsoft XML v3.0.

3.3.4 Componentes para Acesso ao Banco de Dados

Para o caso de acesso ao banco de dados utilizaremos o MDAC 2.7 (Microsoft Data Access Components), que segue a mesma filosofia de utilização como os componentes para manipulação de XML. Uma vez importados esses componentes, podemos criar instâncias e utilizar seus recursos. Como diria *Szyperski*:

“Uma coisa pode ser declarada com certeza: componentes são para composição. Composição habilita reutilizar coisas pré-fabricadas rearranjando-as sempre em novas composições” [SZY98].

4 WEB SERVICES

Os conceitos básicos de Web Services não são difíceis de entender e muito do que eles utilizam já são tecnologias consolidadas à algum tempo. A complicação maior fica, em como montar a arquitetura do software, considerando essas novas possibilidades de deixar a comunicação entre clientes e servidores o mais resiliente possível [CHE00].

4.1 Visão Geral

“Web Services são importantes porque eles definem um caminho padrão para programadores de diferentes plataformas não somente trocar dados, mas também, usar cada um as funcionalidades do outro” [JOH01].

Web Services utilizam um protocolo chamado SOAP (Simple Object Access Protocol) para transmitir suas mensagens. Este protocolo tem uma estrutura genérica e uma parte reservada para o XML, que é definido de acordo com a necessidade das informações a serem transportadas. O processo de definição desse XML será baseado nas idéias contidas no livro *Modeling XML Applications With UML: Practical E-Business Applications* de David Carlson que se justifica de acordo com as palavras de Jeffrey Hammond no próprio livro de Carlson:

“Um manual de tradução entre o mundo da UML e o XML é muito útil porque ele irá acelerar o esforço dos desenvolvedores de software que querem criar aplicações empresariais na Web” [CAR01].

Quando queremos nos comunicar com qualquer tipo de empresa, temos a necessidade de pensar em uma forma de trabalhar com um grande número de plataformas. Para cada plataforma tem-se, tradicionalmente, criado seus próprios protocolos, geralmente de natureza binária, para integração máquina-para-máquina (não aplicativo-para-aplicativo). Como resultado, aplicações que trabalham entre plataformas, geralmente apenas compartilham os dados. Com o reconhecimento dessas limitações, houve um grande esforço para a elaboração de um formato padrão para troca de dados. A idéia seria dar a sistemas diferentes a capacidade de comunicarem-se e compartilharem dados de modo que eles não precisem estar ligados entre si [TRE02].

Este é o objetivo dos Web Services. Um Web Service é uma aplicação lógica, programável, acessível, que usa os protocolos padrões da Internet, para que se torne possível à comunicação transparente de máquina-para-máquina e aplicação-para-aplicação [NAR02].

Para desenvolvermos um Web Service, utilizamos tecnologias pensadas sob esta visão, como o HTTP (HyperText Transfer Protocol), WSDL (Web Service Description Language) e SOAP (Simple Object Access Protocol), usadas para passar mensagens através das máquinas. Estas mensagens podem variar muito na complexidade, desde a chamada de um método, às solicitações de um genograma completo.

4.1.1 Arquitetura de Comunicação para Web Service

Lembrando que uma das definições para Web Service é: algum processo que pode ser integrado em um sistema externo através de documentos XML válidos através dos protocolos da Internet [ZAE01]. Esses protocolos, hoje, são representados pelo TCP/IP (Transmission Control Protocol / Internet Protocol) e o UDP (User Datagram Protocol), sendo o primeiro amplamente utilizado pela Internet por garantir a entrega de seus pacotes, enquanto o outro não tem essa garantia. Como o objetivo não é ficar discutindo os detalhes de um ou de outro protocolo, vamos nos preocupar no que é importante: confirmar que Web Services podem ser construídos e consumidos com as tecnologias correntes.

Pensando na camada de aplicação ao nível de rede para os Web Services, temos várias opções: HTTP (HyperText Transfer Protocol), HTTPS (Secure HTTP), FTP (File Transfer Protocol) e SMTP (Simple Mail Transfer Protocol). O HTTP é sem dúvida um dos

protocolos mais utilizados na Internet e permite que máquinas de arquiteturas diferentes e sistemas operacionais distintos possam comunicar de forma transparente. Como esses protocolos são tecnologias consolidadas, os Web Services ainda tiram vantagens com relação aos recursos de segurança, que já existem para esses protocolos [LAR00].

4.1.2 Publicação das Interfaces de um Web Service na Internet

Para que um Web Service seja utilizado, temos que ter uma forma de saber quais interfaces ele possui e como acessá-las. Esse tipo de informação é possível de ser obtida através de um documento WSDL (Web Service Definition Language) que também é descrito como um XML [TRE02].

Quando construímos uma aplicação servidora, podemos publicar um WSDL que forneça as principais informações das interfaces do Web Service. Muitas das ferramentas de programação que tem suporte a Web Services, geram o WSDL a partir dos métodos definidos como sendo do tipo interface.

Por outro lado, quando construímos uma aplicação cliente para consumir um Web Service, basta ter acesso ao documento WSDL e utilizar os recursos do ambiente de programação para gerar uma interface de chamada para os recursos disponíveis.

Uma vez criado o WSDL de um Web Service, contendo detalhes dos serviços fornecidos, onde achá-lo na Web e como interagir com ele, essas informações podem ser registradas num lugar único e conhecido. Esse lugar trata-se do UDDI (Universal Description, Discovery, and Integration). O UDDI é algo parecido com o DNS (Domain Name System) e como, a princípio, não faremos uso de seus recursos nesse trabalho, não entraremos em maiores detalhes [ROY01].

Um arquivo WSDL contém cinco grupos de informações. Esses dados são empilhados uns sobre os outros para fornecer uma descrição completa do Web Service. Na primeira camada temos o elemento `<service>` que descreve o provedor e as URLs do Web Service. Na seqüência temos os elementos `<binding>`, `<portType>` e `<message>` que são importantes para futuras configurações para quem deseja consumir o Web Service. Por fim, o elemento `<types>` permite definir novos tipos de dados com XML Schema, se for necessário.

4.1.3 Empacotando as Mensagens para serem Transferidas pela Internet

Quando falamos em implementação de Web Services e troca de informações pela Internet, temos que ter um meio de organizar essas mensagens para serem transferidas. O SOAP (Simple Object Access Protocol) faz justamente isso. Ele é uma especificação para mensagens que descreve como organizar os dados e regras para comunicação baseada em XML [GAB02].

O que faz um Web Service ser transparente e ter a capacidade de interagir com outros sistemas é adoção do protocolo SOAP sobre o HTTP. O SOAP define uma notação baseada em XML para fazer requisições de execução de métodos de um objeto no servidor, passando parâmetros para ele e, define também, um formato para respostas a essas chamadas.

Da mesma forma que o WSDL, a definição e criação dos pacotes SOAP, muitas vezes é transparente para o desenvolvedor, pois a maioria dos ambientes de programação atuais encapsulam isso.

5 REPRESENTANDO O GENOGRAMA ESTRUTURAL DA FAMÍLIA COM UML E XML

O objetivo desse capítulo é demonstrar como representar os principais símbolos do genograma familiar através da UML e também como realizar o mapeamento das informações representadas com UML para XML.

Convém lembrar que esse capítulo leva em consideração que os leitores já tenham conhecimento prévio da UML e XML, pois não é nosso objetivo ficar entrando em detalhes dessas tecnologias, e sim, como realizar o mapeamento da UML para XML usando como base diagramas de classes extraídos de cada estrutura básica do genograma familiar.

Como um exemplo genérico desse tipo de mapeamento podemos observar a Fig. 9.2 [CAR01] que se encontra no ANEXO 2. Nela vemos como os elementos de um diagrama de classes UML são convertidos para uma representação em XML.

5.1 A UML para Modelar o Genograma e o Mapa Estrutural da Família

Para podermos definir um vocabulário de forma consciente e preciso para o genograma, vamos utilizar os recursos de modelagem encontrados na UML [SCH00]. Na verdade, iremos usar somente os diagramas estáticos, pois cada vocabulário sempre tem um modelo implícito ou explícito que pode ser representado através desse tipo de diagrama [CAR01].

Essa limitação de utilizar somente os diagramas estáticos e, no nosso caso mais especificamente o diagrama de classes, se justifica pelo fato de um documento XML poder ser definido a partir de um conjunto de definições simples de classes que especificam todos os termos básicos requeridos por um vocabulário de aplicação [CAR01].

5.1.1 Representação de acordo com o Gênero

Como um primeiro exemplo de representação utilizando a notação UML (Fig. 5.1), temos uma classe *Pessoa* com vários atributos (*ID*, *PI*, *genero*, *nota* e *nome*). Essa estrutura está representando os símbolos do genograma para as pessoas tanto com gênero masculino como feminino. No genograma, essa diferença era percebida através da forma das figuras que os representavam (quadrado para o homem e círculo para mulher). Quando se pensa na representação dessas informações no mundo computacional, escolhemos a opção de criar um atributo *genero* na classe *Pessoa*.

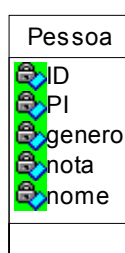


Figura 5.1: Representação do elemento Pessoa

5.1.2 Representação de Paciente Identificado de acordo com o Gênero

Embora no genograma temos um símbolo especial para representar um Paciente Identificado, na UML nós simplesmente colocamos um atributo dizendo se é ou não um PI a pessoa em questão. A representação é a mesma que a da Fig. 5.1.

5.1.3 Representação de Morte de acordo com o Gênero

Nos símbolos básicos do genograma, a morte é representada com um “X” no meio do símbolo do indivíduo que morreu. Na UML, representamos a morte como uma agregação, como podemos ver na Fig. 5.2. Para o nosso trabalho definimos, por enquanto, somente dois atributos (*local* e *data*) para a classe *Morte*. Isso porque toda a modelagem e estruturação dos símbolos do genograma em UML e posteriormente em XML é totalmente maleável, permitindo evolução dos atributos necessários para capturar informações mais detalhadas associada à devida classe. No caso da classe *Morte* poderíamos ter, além de *local* e *data*, outros atributos como: tipo da morte, motivo, descrição de como ocorreu, etc.

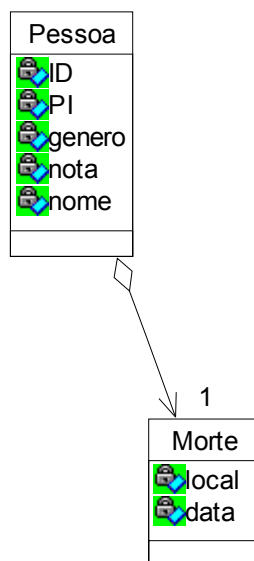


Figura 5.2: Representação da morte de uma pessoa

5.1.4 Representação de um Matrimônio

Como exemplo simples de diagrama de classes para ilustrar o matrimônio entre dois indivíduos podemos ter:

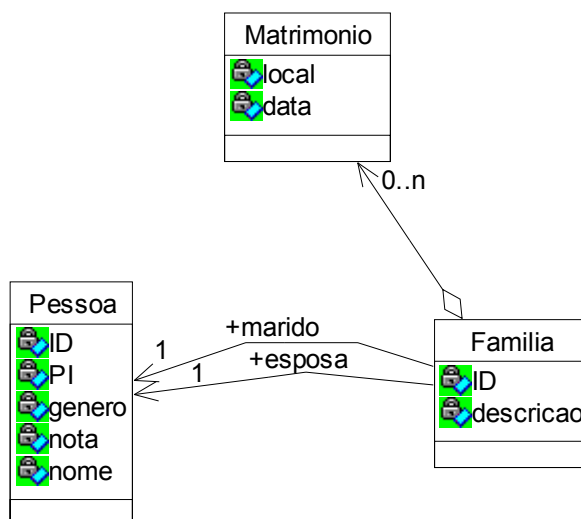


Figura 5.3: Representação da associação matrimônio

Convém lembrar que não teremos muita preocupação em identificar todos os atributos de cada elemento, visto que um dos objetivos principais é demonstrar de uma

forma geral, como realizar a modelagem do genograma em UML e o seu mapeamento para XML.

Para representar um matrimônio entre dois indivíduos, tudo acaba sendo configurado a partir de uma classe principal chamada *Familia*. Como podemos ver na Fig. 5.3, é a classe *Familia* que possui duas referências a classe *Pessoa*, sendo uma das referências o *Marido* e a outra representando a *Esposa* daquela família. Uma vez definido quem é o marido e a mulher, podemos, através da classe *Familia*, dizer que houve um casamento entre os dois em determinada *data* e *local*, que é o papel da classe *Matrimonio* agregada à classe *Familia*.

5.1.5 Representação de uma Separação

De forma muito similar à representação de um matrimônio, a separação adota o mesmo princípio de ser uma agregação da classe *Familia* dizendo que houve uma separação do casal.

Quando temos o mapeamento da cardinalidade de “0..n” da classe *Familia* para a classe *Separação* (Fig. 5.4), isso quer dizer que em uma família podemos não ter nenhuma separação ou até muitas, dependendo da dinâmica dessa família durante o ciclo de vida.

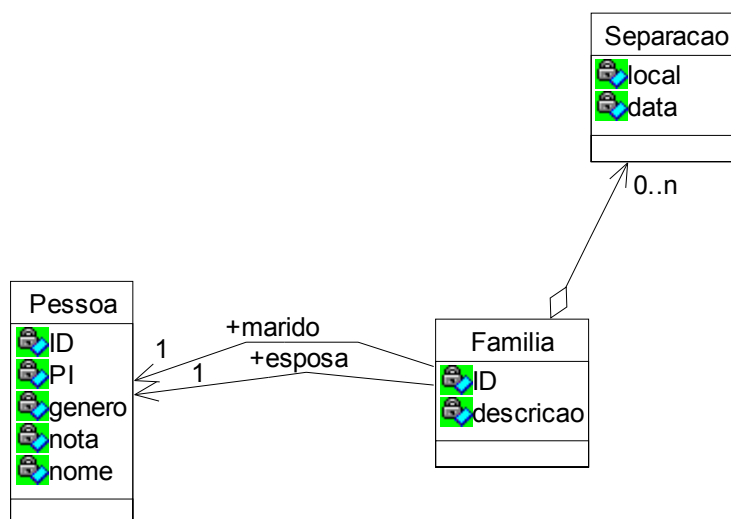


Figura 5.4: Representação de uma separação

5.1.6 Representação de um Divórcio

Seguindo o mesmo princípio da representação de matrimônio e separação, o divórcio também ganha uma classe chamada *Divorcio*³ agregada a família, tendo o mesmo tipo de cardinalidade. Podemos ver a representação de um divórcio na Fig. 5.5.

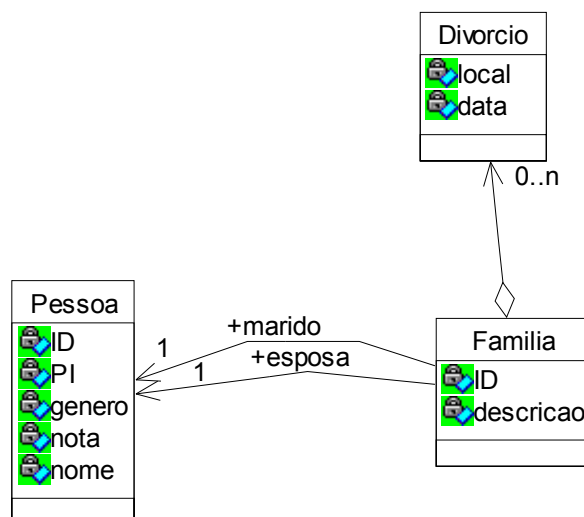


Figura 5.5: Representação de um divórcio

5.1.7 Representação de Convivência

A representação de convivência não foge do padrão e possui uma estrutura muito parecida com as anteriores já comentadas. A Fig. 5.6 mostra a estrutura em UML para o caso de convivência.

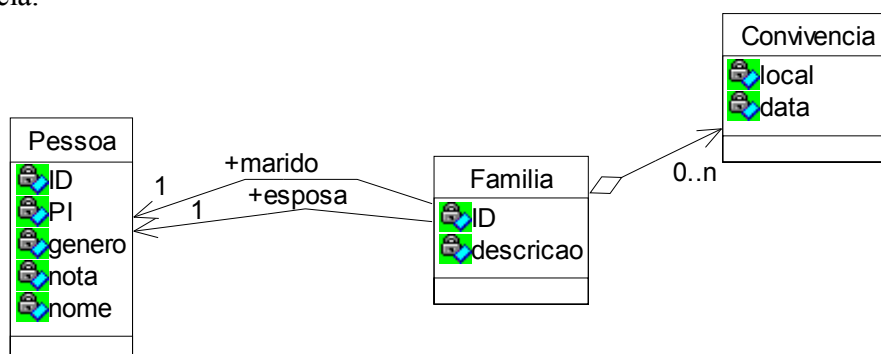


Figura 5.6: Representação de convivência

3 Os nomes das classes estão sem acentuação gráfica de propósito para facilitar na hora de gerar os trechos de XML, que também estarão sem acentuação de forma consciente.

5.1.8 Representação de uma Gravidez

A gravidez também tem sua representação feita como uma agregação da classe *Familia* (Fig. 5.7).

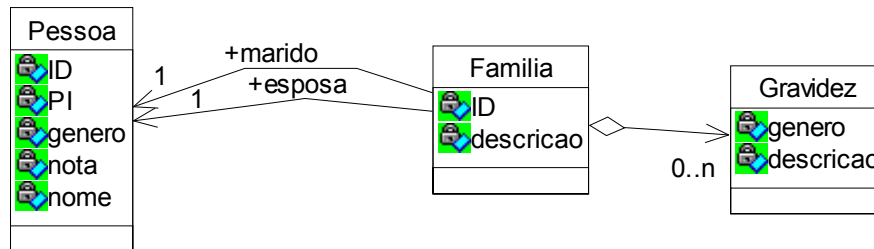


Figura 5.7: Representação de uma gravidez

5.1.9 Representação de Crianças que Nasceram Mortas

O nascimento de uma criança, porém morta ou que morre durante o parto, traz conseqüências para a família e deve ser documentado. A Fig. 5.8 mostra o relacionamento da classe agregada chamada *CriancaNascMorta*.

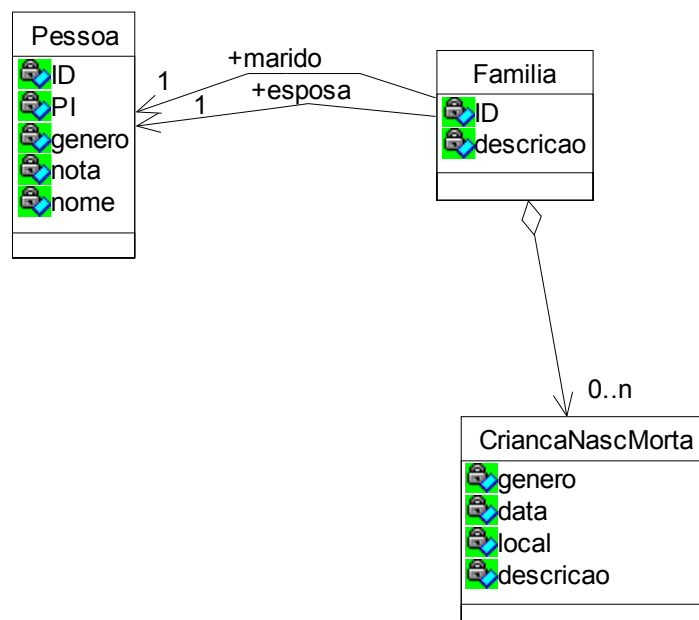


Figura 5.8: Representação quando uma criança nasce morta

5.1.10 Representação de um Aborto Espontâneo

A estrutura geral para representar aborto pode ser vista na Fig. 5.9. Usamos o termo “geral” devido ao fato de o aborto poder ser tanto espontâneo como induzido. Para o caso de ser espontâneo o atributo *tipo* da classe *Aborto* recebe o valor “Espontâneo”. Os outros atributos (*local*, *data* e *descricao*), assim como em outras classes, são alto explicativos.

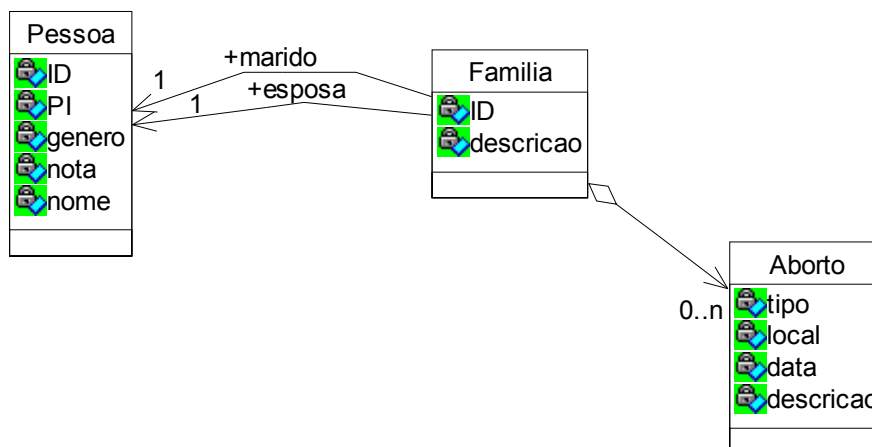


Figura 5.9: Representação de aborto

5.1.11 Representação de um Aborto Induzido

O aborto induzido segue a mesma estrutura da Fig. 5.9, porém o atributo *tipo* recebe o valor “Induzido”.

5.1.12 Representação de Filhos Naturais de acordo com o Gênero

O núcleo principal de uma família é a composição de pais (*marido* e *esposa* na Fig. 5.10) e filhos. Nesse caso estamos falando de filhos naturais, ou seja, que são biologicamente descendentes de seus pais. A representação em UML é semelhante a de marido e mulher de uma família, porém o relacionamento *filhoNatural* possui a cardinalidade “0..n”.

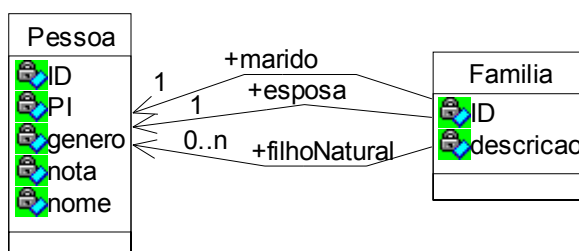


Figura 5.10: Representação de filhos naturais

5.1.13 Representação de Filhos Adotivos de acordo com o Gênero

Também é comum um casal não poder ter filhos biológicos devido a vários fatores. Para esses casos, casais optam pela adoção de crianças para serem seus filhos. A representação ilustrada na Fig. 5.11 mostra um relacionamento chamado *filhoAdotivo* para essas crianças que são adotadas.

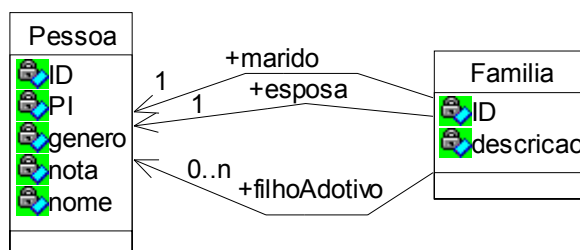


Figura 5.11: Representação de filhos adotivos

5.1.14 Representação de Gêmeos Idênticos

Para representar o caso de gêmeos em uma família, a estrutura em UML fica mais elaborada. Analisando a Fig. 5.12, podemos ver que a classe *Familia* possui uma agregação de “0..n” com a classe *Gemeos*. Isso quer dizer que uma família pode ou não ter gêmeos. Se acontecer de ter gêmeos, a classe *Gemeos* então tem que ter no mínimo dois ou mais irmãos (relacionamento *irmaos*), que no fim são pessoas representadas pela classe *Pessoa*. Para o caso de gêmeos idênticos, teremos o atributo *tipo* com o valor “idênticos”.

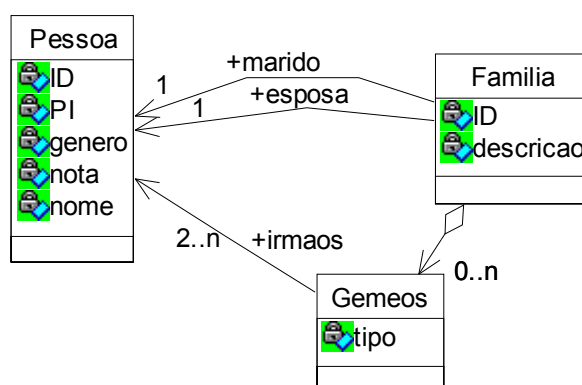


Figura 5.12: Representação de gêmeos

5.1.15 Representação de Gêmeos Fraternos

O caso de gêmeos fraternos segue a mesma estrutura da Fig. 5.12, porém o atributo *tipo* passa a ter o valor “Fraternos”.

5.1.16 Representação para o caso de Aliança

Quando entramos nos casos de representar o mapa estrutural de uma família, chegamos a conclusão que os casos são sempre entre duas pessoas. Com isso chegamos a uma representação (Fig. 5.13) onde uma pessoa possui uma instância da classe *MapaEstrutural* a qual está vinculada a uma outra pessoa através de um relacionamento. Basta agora dizer que tipo de interação está acontecendo. No caso de aliança configuramos o atributo *tipo* da classe *MapaEstrutural* para “Aliança”.

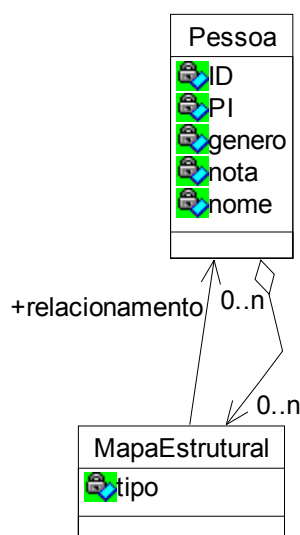


Figura 5.13: Representação do mapa estrutural

5.1.17 Representação para o caso de Emaranhado

Segue a estrutura da Fig. 5.13, porém configurando o atributo *tipo* para “Emaranhado”.

5.1.18 Representação para o caso de Conflito

Segue a estrutura da Fig. 5.13, porém configurando o atributo *tipo* para “Conflito”.

5.1.19 Representação para o caso de Quebra no Relacionamento

Segue a estrutura da Fig. 5.13, porém configurando o atributo *tipo* para “Relacionamento”.

5.1.20 Representação para o caso de Distância

Segue a estrutura da Fig. 5.13, porém configurando o atributo *tipo* para “Distância”.

5.2 Mapeamento da UML para XML

Para podermos fazer o mapeamento da UML para XML utilizando como exemplo os diagramas feitos na seção anterior, teremos que antes, revisar alguns conceitos teóricos.

5.2.1 Classes UML para Elementos XML

Em uma classe podemos encontrar características estruturais através de seus atributos, associações, composições, referências e comportamentos através dos métodos. Porém, para a definição de um vocabulário, somente a parte arquitetural é relevante.

Em XML, um elemento serve como um container para atributos e elementos filhos. Dessa forma, como ocorre em UML, o elemento XML conterá todos os atributos e regras de associação de uma determinada classe correspondente em UML. Com isso, o mapeamento das classes para elementos em XML é quase que direto.

5.2.2 Herança

O uso de herança é um recurso existente na modelagem orientada a objetos, porém, no padrão corrente da XML não temos ainda um mecanismo para representar herança. A solução que se tem é a de copiar tudo que se tem na superclasse para suas classes filhas na hora do mapeamento.

5.2.3 Atributos UML para XML

Podemos ter duas formas de realizar esse tipo de mapeamento:

- Dada uma instância de uma classe UML, podemos começar o mapeamento pegando cada um dos atributos e transformando-os normalmente como um elemento filho

separado.

- **Atributos UML para Atributos XML:** quando os tipos de dados dos atributos UML são primitivos (inclusive o tipo de dado) ou enumerações, o seu mapeamento deve ser para um atributo XML.

No caso de atributos multivalorados não existe representação em XML, então esses atributos devem ser transformados em elementos XML.

5.2.4 Valores de Atributos Enumerados

Um atributo do tipo enumerado requer que ele seja configurado a partir de uma lista de possíveis valores predefinidos.

Para realizar o mapeamento para XML, a filosofia é a mesma adotada para atributos simples, porém é adicionada uma nova capacidade do XML DTD checar se é um valor válido para atributo enumerado.

5.2.5 Mapeando Composições UML

Composições são sempre mostradas em diagramas UML usando um diamante sólido no final da associação indicando que o objeto relacionado é por valor e não por referência.

Da mesma forma em um elemento XML, seus elementos filhos são por valor, significando que se o elemento pai for apagado os filhos também serão.

Para realizar o mapeamento, o nome da regra da associação existente na composição é mapeada para um elemento XML, de forma que, sirva como um container para os objetos pertencentes à composição.

5.2.6 Mapeando Associações UML

Para que possamos fazer o mapeamento das associações UML para XML, cada classe em UML deve ter um atributo do tipo *XMI.ID*. Esse recurso do XMI (XML Metadata Interchange) permite a criação de associações entre elementos no mesmo documento. A única restrição, é que *ID* deve ser único dentro do escopo do documento. Se existisse o caso de ter que referenciar elementos em outros documentos, o XML possui um recurso próprio para esses casos, que é o *Xlink*, mas como foge do escopo desse trabalho, não entraremos em detalhes.

XMI define duas formas de fazer o mapeamento de associações para XML: uma utilizando elemento e outra atributo. No caso da utilização de elemento, a regra de associação é usada para criar o nome do elemento, semelhante à criação de elementos a partir de atributos UML. Entretanto, ao invés de incluir a definição completa do elemento como um elemento filho (como foi feito para o caso de composição), um elemento *Proxy* é incluído para referenciar uma definição completa para o elemento associado.

O atributo XML *XMI.IDREF* representa uma referência a um *XMI.ID* contido no mesmo documento. A especificação XMI também define outro atributo, *XMI.LABEL*, que deve ser incluído no elemento *Proxy*, contendo um título ou uma breve descrição do elemento referenciado [CAR01]. Usando esse *LABEL*, uma aplicação pode apresentar para o usuário sobre o que se trata o link sem ter que buscar o elemento referenciado.

A segunda forma de realizar o mapeamento de associações UML para XML é através do uso de atributos. Nesse caso um atributo com o mesmo nome da associação é criado contendo a referência para o outro elemento.

Todos esses conceitos ficarão mais claros quando apresentarmos o mapeamento XML propriamente dito para cada um dos diagramas UML identificados.

5.3 Mapeamento dos Diagramas de Classes do Genograma para XML

Nesta seção, para cada diagrama de classe criado a partir dos símbolos padrões do genograma e do mapa estrutural na seção 5.1, vamos fazer o mapeamento para XML. A quebra em partes é para facilitar o entendimento e documentação, porém nada impede que uma representação mais elaborada seja montada através dessas partes, pois um documento XML pode ser montado com vários elementos filhos.

Em nossos exemplos, todos os documentos XML tem como elemento raiz *Genograma* (no ANEXO 5 Fig. 9.6 podemos visualizar um diagrama de classe representando essa raiz). A idéia de se montar a estrutura do XML dessa forma está embasada nos exemplos contidos no livro *XML Bible* de *Elliotte Rusty Harold* [HAR99].

5.3.1 Mapeamento de acordo com o Gênero

Na representação em UML da classe *Pessoa* tínhamos vários atributos, os quais podem ser

mapeados para XML como sendo atributos ou elementos. No caso de *Pessoa* - *ID*, *PI* e *Genero* tornaram-se atributos e *Nome* e *Nota* foram mapeados como sendo elementos do elemento *Pessoa* no XML com raiz *Genograma*, como podemos observar na Fig. 5.14.

```
<GENOGRAMA>
  <PESSOA ID="p1" PI="N" GENERO="M">
    <NOME>Rodrigo Bianco</NOME>
    <NOTA></NOTA>
  </PESSOA>
</GENOGRAMA>
```

Figura 5.14: XML para representação de acordo com o gênero

5.3.2 Mapeamento para Paciente Identificado de acordo com o Gênero

O atributo *ID* que aparece no XML para uma pessoa é um identificador único no escopo do sistema, que serve como referência à essa pessoa em outros XML. O *PI* é o atributo para dizer se essa pessoa é um *Paciente Identificado* ou não (“S” para sim e “N” para não). O *Genero* assume “F” para feminino e “M” para masculino. Na Fig. 5.15 temos a representação de uma pessoa do gênero feminino que é um *Paciente Identificado*.

```
<GENOGRAMA>
  <PESSOA ID="p2" PI="S" GENERO="F">
    <NOME>Heloisa Cezarino Martins Bianco</NOME>
    <NOTA>Paciente Identificado</NOTA>
  </PESSOA>
</GENOGRAMA>
```

Figura 5.15: XML para representação de um paciente identificado

5.3.3 Mapeamento de Morte de acordo com o Gênero

A representação de morte de uma pessoa na notação UML é uma agregação da classe *Morte* com a classe *Pessoa*. Essa agregação, quando fazemos o mapeamento para XML, transformamos em um novo elemento do elemento que tem a agregação. Por sua vez, os atributos da classe *Morte* também se tornam elementos com os valores apropriados. Isso fica ilustrado na Fig. 5.16.


```

<GENOGRAMA>
  <PESSOA ID="p1" PI="N" GENERO="M">
    <NOME>Mariano Martins</NOME>
    <NOTA>Falecimento por motivos de parada cardíaca.</NOTA>
    <MORTE>
      <LOCAL>Porto Rico</LOCAL>
      <DATA>11/12/2000</DATA>
    </MORTE>
  </PESSOA>
</GENOGRAMA>

```

Figura 5.16: XML para representar a morte de uma pessoa

5.3.4 Mapeamento de um Matrimônio

Para que possamos representar em XML um matrimônio, este deve estar vinculado a uma família. A família, por sua vez, tem que ter uma indicação de quem é a pessoa que representa o marido e quem é a esposa. Por fim, tanto o elemento *Familia* como os elementos *Pessoa* ficam abaixo da raiz principal *Genograma*, fazendo referências entre eles.

```

<GENOGRAMA>
  <PESSOA ID="p1" PI="N" GENERO="M">
    <NOME>Rodrigo Bianco</NOME>
    <NOTA></NOTA>
  </PESSOA>
  <PESSOA ID="p2" PI="S" GENERO="F">
    <NOME>Heloisa Cezarino Martins Bianco</NOME>
    <NOTA>Paciente Identificado</NOTA>
  </PESSOA>
  <FAMILIA ID="f1">
    <MARIDO PESSOA="p1" />
    <ESPOSA PESSOA="p2" />
    <DESCRICAO>Família de Exemplo</DESCRICAO>
    <MATRIMONIO ID="m1">
      <LOCAL>Loanda</LOCAL>
      <DATA>29/12/2001</DATA>
    </MATRIMONIO>
  </FAMILIA>
</GENOGRAMA>

```

Figura 5.17: XML para representar o matrimônio entre duas pessoas

5.3.5 Mapeamento para o caso de uma Separação

Semelhante ao caso de matrimônio, uma separação é representada como uma agregação na

UML e torna-se um elemento de *Familia* em XML. Podemos dizer que em uma família, onde existe um casamento, pode não ocorrer separação dos cônjuge (ficando sem o elemento *Separacao* no XML), ou ainda, ocorrer uma ou várias separações (sendo colocado um elemento *Separacao* para cada uma que ocorrer). Isso acontece devido ao fato de o casal se separar e voltar a ficar juntos no futuro, podendo novamente se separar e assim por diante. Uma separação seria algo que não exige o envolvimento de cartórios ou algo do tipo e sim, somente o afastamento dos indivíduos.

```

<GENOGRAMA>
  <PESSOA ID="p1" PI="N" GENERO="M">
    <NOME>Rodrigo Bianco</NOME>
    <NOTA></NOTA>
  </PESSOA>
  <PESSOA ID="p2" PI="S" GENERO="F">
    <NOME>Heloisa Cezarino Martins Bianco</NOME>
    <NOTA>Paciente Identificado</NOTA>
  </PESSOA>
  <FAMILIA ID="f1">
    <MARIDO PESSOA="p1" />
    <ESPOSA PESSOA="p2" />
    <DESCRICAO>Família de Exemplo</DESCRICAO>
    <SEPARACAO ID="s1">
      <LOCAL>Curitiba</LOCAL>
      <DATA>22/05/2002</DATA>
    </SEPARACAO>
  </FAMILIA>
</GENOGRAMA>

```

Figura 5.18: XML para representar a separação entre duas pessoas

5.3.6 Mapeamento para o caso de um Divórcio

O mapeamento para o caso de divórcio é análogo ao caso de matrimônio. Porém, quando dizemos divórcio, está se referindo aos casos legais onde antes tinha que existir um casamento oficial pelo menos em cartório. Devido a essa premissa, não tem problema os dois elementos existirem em conjunto, pois, primeiro existe um casamento e, se for o caso, depois vem um divórcio, mas as duas informações existem. A Fig. 5.19 esclarece como fica o mapeamento para XML.

```

<GENOGRAMA>
  <PESSOA ID="p1" PI="N" GENERO="M">
    <NOME>Rodrigo Bianco</NOME>
    <NOTA></NOTA>
  </PESSOA>
  <PESSOA ID="p2" PI="S" GENERO="F">
    <NOME>Heloisa Cezarino Martins Bianco</NOME>
    <NOTA>Paciente Identificado</NOTA>
  </PESSOA>
  <FAMILIA ID="f1">
    <MARIDO PESSOA="p1" />
    <ESPOSA PESSOA="p2" />
    <DESCRICAO>Família de Exemplo</DESCRICAO>
    <MATRIMONIO ID="m1">
      <LOCAL>Loanda</LOCAL>
      <DATA>29/12/2001</DATA>
    </MATRIMONIO>
    <DIVORCIO ID="d1">
      <LOCAL>Foz do Iguaçu</LOCAL>
      <DATA>20/06/2002</DATA>
    </DIVORCIO>
  </FAMILIA>
</GENOGRAMA>

```

Figura 5.19: XML para representar o divórcio entre duas pessoas

5.3.7 Mapeamento para o caso de Convivência

Seguindo o padrão do casamento, divórcio e separação, o mapeamento de convivência não é diferente na forma de estruturar o XML. Um detalhe importante, é que os elementos criados para representar essas situações também recebem um *ID* único para futuras referências.

```

<GENOGRAMA>
  <PESSOA ID="p1" PI="N" GENERO="M">
    <NOME>Rodrigo Bianco</NOME>
    <NOTA></NOTA>
  </PESSOA>
  <PESSOA ID="p2" PI="S" GENERO="F">
    <NOME>Heloisa Cezarino Martins Bianco</NOME>
    <NOTA>Paciente Identificado</NOTA>
  </PESSOA>
  <FAMILIA ID="f1">
    <MARIDO PESSOA="p1" />
    <ESPOSA PESSOA="p2" />
    <DESCRICAO>Família de Exemplo</DESCRICAO>
    <CONVIVENCIA ID="c1">
      <LOCAL>Foz do Iguaçu</LOCAL>
    </CONVIVENCIA>
  </FAMILIA>
</GENOGRAMA>

```

```

<DATA>25/07/2002</DATA>
  </CONVIVENCIA>
</FAMILIA>
</GENOGRAMA>

```

Figura 5.20: XML para representar a convivência entre duas pessoas

5.3.8 Mapeamento para o caso de uma Gravidez

A gravidez também está vinculada a uma família, por isso, é mapeada como um elemento *Gravidez* que possui um atributo *ID* para sua identificação.

```

<GENOGRAMA>
  <PESSOA ID="p1" PI="N" GENERO="M">
    <NOME>Rodrigo Bianco</NOME>
    <NOTA></NOTA>
  </PESSOA>
  <PESSOA ID="p2" PI="S" GENERO="F">
    <NOME>Heloisa Cezarino Martins Bianco</NOME>
    <NOTA>Paciente Identificado</NOTA>
  </PESSOA>
  <FAMILIA ID="f1">
    <MARIDO PESSOA="p1" />
    <ESPOSA PESSOA="p2" />
    <DESCRICAO>Família de Exemplo</DESCRICAO>
    <GRAVIDEZ ID="g1">
      <GENERO>Masculino</GENERO>
      <DESCRICAO>Gravidez não esperada.</DESCRICAO>
    </GRAVIDEZ>
  </FAMILIA>
</GENOGRAMA>

```

Figura 5.21: XML para representar uma gravidez

5.3.9 Mapeamento de Crianças que Nasceram Mortas

Quando uma criança nasce morta, esse acontecimento deve ser representado e também pertence à família onde ocorreu. Podemos ver um exemplo de XML para esse caso na Fig. 5.22.

```

<GENOGRAMA>
  <PESSOA ID="p1" PI="N" GENERO="M">
    <NOME>Rodrigo Bianco</NOME>

```

```

<NOTA></NOTA>
</PESSOA>
<PESSOA ID="p2" PI="S" GENERO="F">
  <NOME>Heloisa Cezarino Martins Bianco</NOME>
  <NOTA>Paciente Identificado</NOTA>
</PESSOA>
<FAMILIA ID="f1">
  <MARIDO PESSOA="p1" />
  <ESPOSA PESSOA="p2" />
  <DESCRICAO>Família de Exemplo</DESCRICAO>
  <CRIANCANASCORTA ID="cm1">
    <GENERO>Masculino</GENERO>
    <LOCAL>Porto Rico</LOCAL>
    <DATA>15/07/1998</DATA>
    <DESCRICAO>Complicação no Parto.</DESCRICAO>
  </CRIANCANASCORTA>
</FAMILIA>
</GENOGRAMA>

```

Figura 5.22: XML para representar quando uma criança nasce morta

5.3.10 Mapeamento para o caso de um Aborto Espontâneo

O aborto tem uma representação idêntica tanto para o espontâneo como para o induzido. Também é um fato associado a uma família, portanto torna-se um elemento filho de *Familia*. Para o caso de ser espontâneo, o elemento *Tipo* de *Aborto* recebe o valor “Espontâneo”. A Fig. 5.23 ilustra esse caso.

```

<GENOGRAMA>
  <PESSOA ID="p1" PI="N" GENERO="M">
    <NOME>Rodrigo Bianco</NOME>
    <NOTA></NOTA>
  </PESSOA>
  <PESSOA ID="p2" PI="S" GENERO="F">
    <NOME>Heloisa Cezarino Martins Bianco</NOME>
    <NOTA>Paciente Identificado</NOTA>
  </PESSOA>
  <FAMILIA ID="f1">
    <MARIDO PESSOA="p1" />
    <ESPOSA PESSOA="p2" />
    <DESCRICAO>Família de Exemplo</DESCRICAO>
    <ABORTO ID="a1">
      <TIPO>Espontâneo</TIPO>
      <LOCAL>Loanda</LOCAL>
      <DATA>11/08/1999</DATA>
      <DESCRICAO>Aborto ocorrido com três meses de gestação.</DESCRICAO>
    </ABORTO>
  </FAMILIA>
</GENOGRAMA>

```

```

</ABORTO>
</FAMILIA>
</GENOGRAMA>

```

Figura 5.23: XML para representar um aborto

5.3.11 Mapeamento para o caso de um Aborto Induzido

Nos mesmos moldes do espontâneo, só mudando o elemento *Tipo* de “Espontâneo” para “Induzido”.

5.3.12 Mapeamento de Filhos Naturais de acordo com o Gênero

O mapeamento para filhos segue o mesmo princípio adotado para marido e esposa. Cria-se um elemento *pessoa*, que conterà as informações referentes ao filho e no elemento *Familia*, criamos o elemento *Filho* ligando com o *ID* da pessoa que o representa. Ainda no elemento *Filho*, temos que dizer se ele é natural ou adotivo. No exemplo da Fig. 5.24 temos a demonstração do XML para o caso de um filho natural do casal.

```

<GENOGRAMA>
  <PESSOA ID="p1" PI="N" GENERO="M">
    <NOME>Rodrigo Bianco</NOME>
    <NOTA></NOTA>
  </PESSOA>
  <PESSOA ID="p2" PI="S" GENERO="F">
    <NOME>Heloisa Cezarino Martins Bianco</NOME>
    <NOTA>Paciente Identificado</NOTA>
  </PESSOA>
  <PESSOA ID="p3" PI="N" GENERO="M">
    <NOME>Leonardo Martins Bianco</NOME>
    <NOTA>Filho único</NOTA>
  </PESSOA>
  <FAMILIA ID="f1">
    <MARIDO PESSOA="p1" />
    <ESPOSA PESSOA="p2" />
    <FILHO PESSOA="p3">
      <TIPO>Natural</TIPO>
    </FILHO>
  </FAMILIA>
</GENOGRAMA>

```

Figura 5.24: XML para representar o filhos

5.3.13 Mapeamento de Filhos Adotivos de acordo com o Gênero

Similar ao caso de filho natural, porém o atributo *Tipo* de *Filho* ao invés de ser “Natural” será “Adotivo”.

5.3.14 Mapeamento de Gêmeos Idênticos

Quando vamos representar os casos de gêmeos em uma família, fazemos da seguinte forma. No elemento *Familia* colocamos o elemento *Gemeos* com uma identificação única; um atributo para dizer que tipo de gêmeos se trata (idênticos ou fraternos); e quais as pessoas que são irmãos entre si. É um mapeamento interessante, pois temos um relacionamento de muitos para muitos entre *Familia* e *Gemeos*, que por sua vez, também tem um relacionamento de muitos para muitos com *Pessoa*. Na figura abaixo temos o XML para o caso dos gêmeos serem idênticos.

```
<GENOGRAMA>
  <PESSOA ID="p1" PI="N" GENERO="M">
    <NOME>Rodrigo Bianco</NOME>
    <NOTA></NOTA>
  </PESSOA>
  <PESSOA ID="p2" PI="S" GENERO="F">
    <NOME>Heloisa Cezarino Martins Bianco</NOME>
    <NOTA>Paciente Identificado</NOTA>
  </PESSOA>
  <PESSOA ID="p3" PI="N" GENERO="M">
    <NOME>Leonardo Martins Bianco</NOME>
    <NOTA>Nasceu primeiro do que seu irmão gêmeo Eduardo.</NOTA>
  </PESSOA>
  <PESSOA ID="p4" PI="N" GENERO="M">
    <NOME>Eduardo Martins Bianco</NOME>
    <NOTA></NOTA>
  </PESSOA>
  <FAMILIA ID="f1">
    <MARIDO PESSOA="p1" />
    <ESPOSA PESSOA="p2" />
    <FILHO PESSOA="p3">
      <TIPO>Natural</TIPO>
    </FILHO>
    <FILHO PESSOA="p4">
      <TIPO>Natural</TIPO>
    </FILHO>
    <GEMEOS ID="g1">
      <TIPO>Idêntico</TIPO>
      <IRMAO PESSOA="p3"></IRMAO>
      <IRMAO PESSOA="p4"></IRMAO>
    </GEMEOS>
  </FAMILIA>
</GENOGRAMA>
```

```

</GEMEOS>
</FAMILIA>
</GENOGRAMA>

```

Figura 5.25: XML para representar gêmeos

5.3.15 Mapeamento de Gêmeos Fraternos

Idem à representação para gêmeos idênticos, porém, no caso dos fraternos, os irmãos podem ser de gênero diferente e o elemento *Tipo* muda para “Fraterno”.

5.3.16 Mapeamento para o caso de Aliança

No exemplo abaixo temos um XML que procura demonstrar a forma básica para o mapa estrutural e suas configurações entre pessoas. O elemento *Pessoa* passa a ter um novo elemento que é o *MapaEstrutural*, no qual temos o *Tipo* que ele pode ser (Aliança, Emaranhado, Conflito, Quebra de Relacionamento e Distância) e a pessoa com quem é feito o *Relacionamento*.

```

<GENOGRAMA>
  <PESSOA ID="p1" PI="N" GENERO="M">
    <NOME>Rodrigo Bianco</NOME>
    <NOTA></NOTA>
  </PESSOA>
  <PESSOA ID="p2" PI="S" GENERO="F">
    <NOME>Heloisa Cezarino Martins Bianco</NOME>
    <NOTA>Paciente Identificado</NOTA>
    <MAPAESTRUTURAL>
      <TIPO>Aliança</TIPO>
      <RELACIONAMENTO PESSOA="p1"></RELACIONAMENTO>
    </MAPAESTRUTURAL>
  </PESSOA>
</GENOGRAMA>

```

Figura 5.26: XML para representar o mapa estrutural

5.3.17 Mapeamento para o caso de Emaranhado

A mesma coisa que 5.3.16, mudando somente o *Tipo* para “Emaranhado”.

5.3.18 Mapeamento para o caso de Conflito

Idem 5.3.16, com as devidas configurações do *Tipo* para “Conflito”. Convém lembrar que o *MapaEstrutural* pode ser unidirecional, bidirecional e reflexivo, ou seja, para o caso unidirecional uma pessoa *P1* pode estar em conflito com uma outra *P2* e esta não estar com problemas com ninguém; para bidirecional uma pessoa *P1* pode estar em conflito com outra *P2* que por sua vez também está em conflito com *P1*; e por fim, o reflexivo seria quando uma pessoa *P1* está em conflito com *P2* que por sua vez está em conflito com *P3*. Podemos achar outras propriedades interessantes, mas, por enquanto, não vem ao caso. Os exemplos acima foram ilustrados usando o tipo “Conflito”, porém poderia ser qualquer uma das outras opções para o mapa estrutural.

5.3.19 Mapeamento para o caso de Quebra no Relacionamento

Da mesma forma que 5.3.16, somente configurar o *Tipo* para “Relacionamento”. Voltamos a lembrar que a quantidade de elementos e atributos apresentados, são o mínimo necessário para representar a estrutura básica para a representação de um genograma e seu mapa estrutural. Sendo necessário, podemos aumentar a quantidade de atributos e elementos para poder representar novas informações. Por exemplo, para o caso de quebra no relacionamento, talvez fosse interessante um elemento *Observacao* no *MapaEstrutural* para que comentários livres pudessem ser feitos com relação ao fato ocorrido.

5.3.20 Mapeamento para o caso de Distância

Idem 5.3.16, configurando o *Tipo* para “Distância”.

6 SICOGEF – IMPLEMENTAÇÃO DO SISTEMA

Para demonstrar que o desenvolvimento de aplicações baseado em componentes (podendo estes serem comerciais ou não) é viável e que o SOAP através de suas mensagens XML consegue fazer a comunicação entre os aplicativos, vamos utilizar os seguintes recursos:

- C++ Builder Enterprise 6.0: ferramenta de desenvolvimento da Inprise (www.inprise.com) que oferece suporte para desenvolvimento de Web Services na linguagem C++. Sua utilização será para criar a infra-estrutura servidora de modo que ficará esperando solicitações através de mensagens SOAP;
- Delphi Enterprise 6.0: também outra ferramenta de desenvolvimento da empresa Inprise, porém com a linguagem Object Pascal como sendo a padrão. Será utilizada para desenvolver a aplicação cliente (através de componentes para a criação de interface) que farão as solicitações ao servidor através do SOAP;
- Access 2000: Ferramenta com recurso de banco de dados da Microsoft que será utilizado num primeiro momento, podendo ser substituído por um Sistema Gerenciador de Banco de Dados (SGBD);
- Componente FlowChart: Componente comercial fornecido pela empresa Devexpress (www.devexpress.com) que implementa as principais funcionalidades de diagramas;
- Componente Chat: Componente gratuito que implementa os recursos necessários para a construção de um Chat;
- Componentes do Delphi e C++ Builder: Componentes padrões desses ambientes, principalmente os componentes básicos visuais;
- COM+: Os componentes de negócio serão implementados de modo que possam ser instalados no ambiente COM+.

“Para aumentar a produtividade no desenvolvimento de Software, Rapid Application Development (RAD) tornou-se muito popular e tem dado um crescimento explosivo para sistemas com linguagens de quarta geração (4GL). Entretanto, ser capaz de desenvolver uma aplicação rapidamente é uma coisa. Agora, como nós fazemos para seguir com um ciclo de vida de software inteiro, particularmente manutenção e extensibilidade? Estudos recentes tem mostrado que em várias companhias, mais de 93% de todo esforço em software vão para a manutenção e esse quadro está aumentando” [HEI98].

A proposta do SICOGEF é tentar utilizar o poder das ferramentas RAD e componentes comerciais, mas sem esquecer de questões básicas de arquitetura e de engenharia de software [KIR01], pois como podemos ver nos dados fornecidos por *Heinckeins* em suas observações, mais de 93% do esforço acaba sendo gasto na manutenção do sistema. Então, já prevendo esse tipo de situação, procuramos seguir alguns critérios para a elaboração da arquitetura do software.

A seguir iremos descrever os passos para a montagem de uma arquitetura com todas essas tecnologias, de modo que, o sistema para a construção do genograma seja algo realmente possível de ser implementado. Temos que deixar bem claro que, o objetivo não é fazer o sistema ficar pronto para o usuário final, e sim, mostrar que os princípios básicos de construção através de componentes e comunicação com SOAP é algo realmente factível.

Para a parte servidora da implementação, existem alguns detalhes a mais que será interessante comentar. Os componentes de negócios serão construídos utilizando a tecnologia COM+, que por sua vez, terão que se comunicar com a camada Web Services para poderem ser visíveis no mundo exterior. Esses componentes COM+ serão construídos utilizando objetos em C++ que seguirão uma arquitetura em camadas muito parecida com a Windows DNA (Distributed interNet Application). Podemos resumir essas camadas da parte servidora conforme ilustra a Fig. 9.1 no ANEXO 1.

Os objetos de negócio poderão ser de dois tipos: os transientes e os não transientes. Os não transientes são aqueles que realizarão alterações no banco de dados. Já os transientes irão somente fazer consultas através de comandos SQL (igualmente para os não transientes) implementados nos objetos “Shadom” (encarregados de saber fazer

operações no banco de dados) de cada objeto de negócio.

Poderíamos ter feito o controle de transação da camada de negócio através de recursos disponíveis no COM+. Porém, devido às experiências anteriores mal sucedidas no que diz respeito à utilização de controle de transação via COM+ fora do ambiente Microsoft, ficou claro que esse controle deveria ficar sobre a responsabilidade da camada de negócio utilizando os recursos do ADO (Activex Data Objects). Quando é dito “fora do ambiente da Microsoft” quer dizer não utilizar as ferramentas de desenvolvimento dessa empresa. Talvez, a tecnologia fique consolidada e ferramentas de diversas empresas possam utilizar os recursos do COM+ sem muitos problemas.

6.1 Montagem Inicial do Projeto para Gerar DLL de Negócio

Considerando a ferramenta C++ Builder 6 Enterprise, o passo inicial para a criação da DLL é através do *Menu File | New | Other* e escolher a opção de criar uma ActiveX Library como é demonstrado na Fig. 6.1. Seguindo esses passos, já podemos salvar o projeto com o nome desejado, que será o nome da DLL a ser gerada.

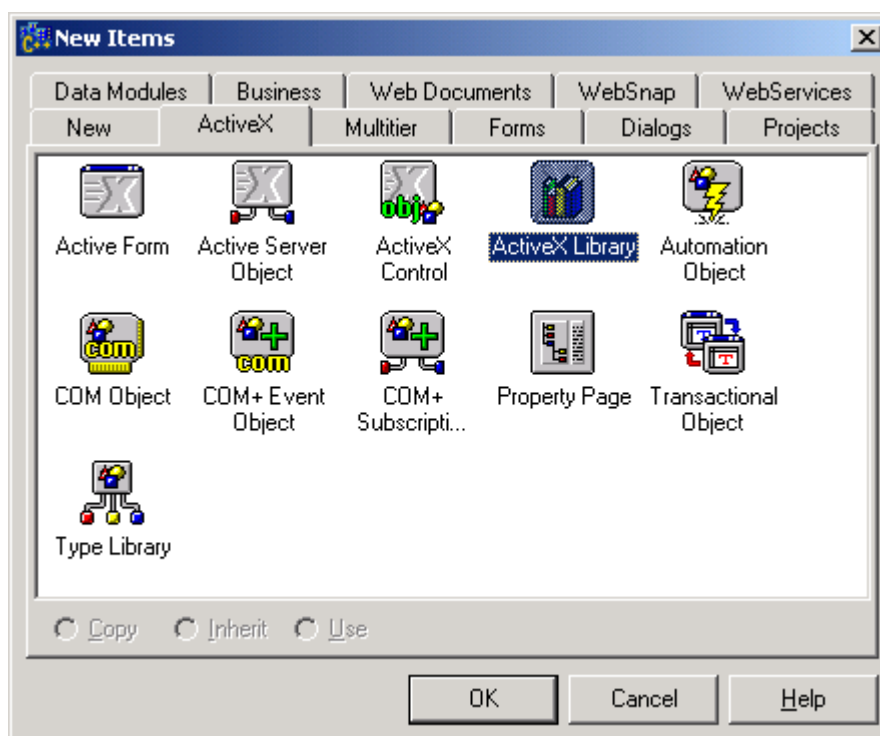
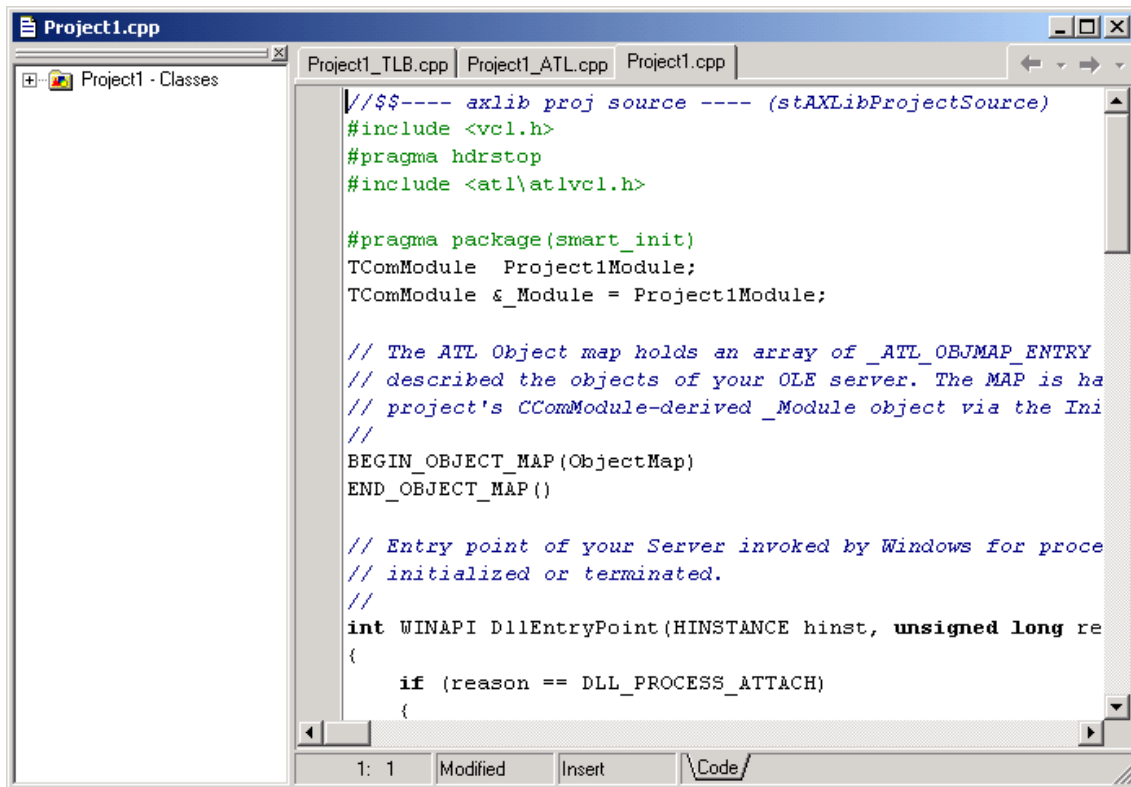


Figura 6.1: Opção para criar uma ActiveX Library

Quando mandamos salvar o projeto pela primeira vez, basicamente três arquivos nesse momento terão que ser salvos. Esses arquivos estão ilustrados na Fig. 6.2 e são totalmente gerados e utilizados pelo ambiente para implementar as funcionalidades desejadas.



```

//$$---- axlib proj source ---- (stAXLibProjectSource)
#include <vcl.h>
#pragma hdrstop
#include <atl\atlvc1.h>

#pragma package(smart_init)
TComModule Project1Module;
TComModule &_Module = Project1Module;

// The ATL Object map holds an array of _ATL_OBJMAP_ENTRY
// described the objects of your OLE server. The MAP is ha
// project's CComModule-derived _Module object via the Ini
//
BEGIN_OBJECT_MAP(ObjectMap)
END_OBJECT_MAP()

// Entry point of your Server invoked by Windows for proce
// initialized or terminated.
//
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long re
{
    if (reason == DLL_PROCESS_ATTACH)
    {

```

Figura 6.2: Arquivos existentes durante a criação de uma Active Library

A geração de código automatizada é muito comum nas ferramentas de desenvolvimento atuais, pois muitos dos trechos de código são padrão para qualquer tipo de aplicação que venha a ter que utilizar um determinado recurso.

Para o nosso projeto, foi adotado o nome *genograma* como sendo o escolhido para a DLL que conterà os componentes de negócios. Uma vez salvo o projeto com *genograma* sendo a palavra chave (substituindo a sugerida pelo ambiente que é *Project1*) teremos algo do tipo ilustrado na Fig. 6.3.

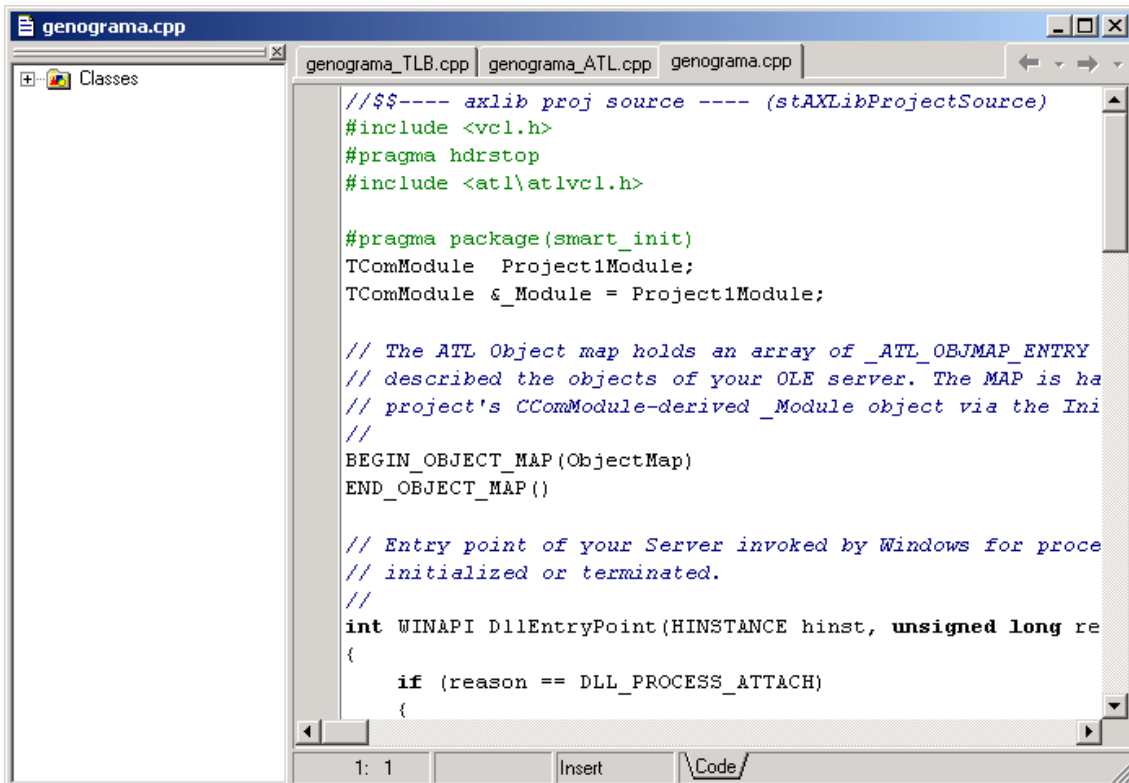


Figura 6.3: Projeto já com os nomes atualizados após o salvamento

Para a criação dos componentes COM+, foi adotado a utilização do *COM Object*. Para criar um novo componente, basta seguir os mesmos passos para a criação de uma *Active Library*, porém na aba *ActiveX* escolher a opção *COM Object*, como podemos ver na Fig. 6.4. Uma vez escolhida essa opção será mostrada a tela ilustrada na Fig. 6.5.

Essa fase inicial de criação do projeto é muito tranquila devido ao fato de os novos ambientes de programação deixarem cada vez mais declarativa a forma de se criar certos recursos, ou seja, não exige nenhuma programação da parte do desenvolvedor para criar qualquer um dos componentes ilustrados na Fig. 6.4. O máximo que poderá exigir do desenvolvedor é ir respondendo algumas telas básicas de configuração do componente que se pretende criar.

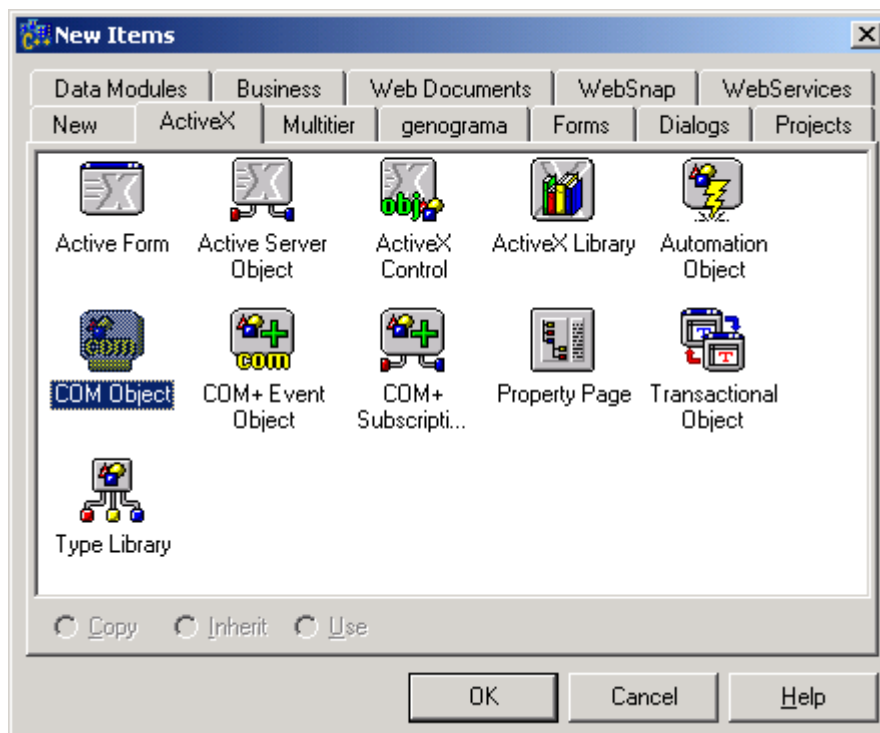


Figura 6.4: Escolhendo a opção COM Object para a criação do componente

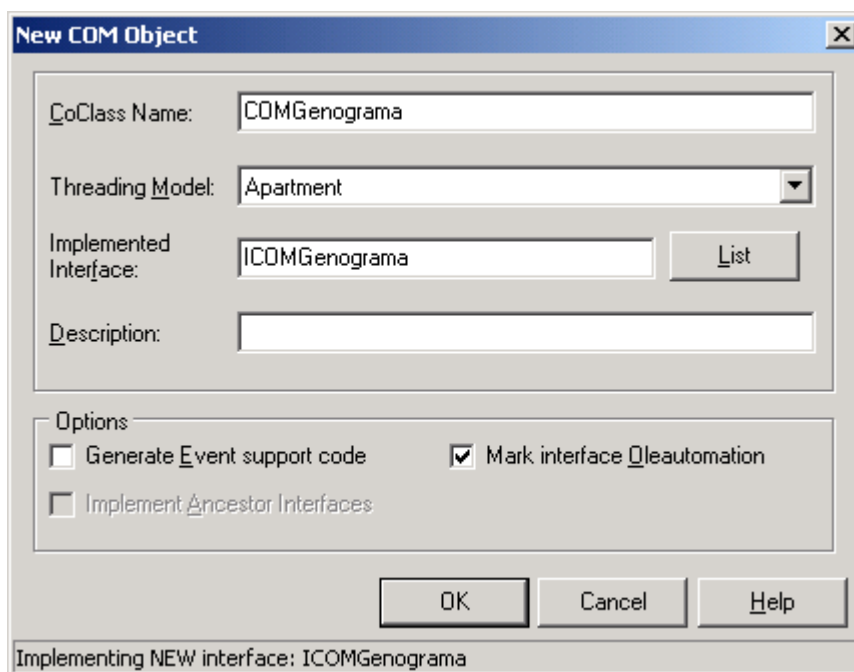


Figura 6.5: Tela inicial para configuração do componente COM+

No primeiro campo da Fig. 6.5, é fornecido o nome para o componente, o qual será referenciado posteriormente pelos clientes que fizerem uso desse componente (na nossa arquitetura o cliente, no caso, será o Web Service que, por sua vez, será acessado pelos clientes do sistema). Como opção de configuração, foi definido que o *Threading Model* seria do tipo *Apartment*, o que é padrão para todos os novos componentes que são criados. É importante que essa opção continue configurada assim, pois é ela que permitirá que o componente crie novas threads se a demanda for muito grande.

No momento em que é pressionado o botão *Ok* da tela da Fig. 6.5, o *Type Library* é aberto conforme ilustra a Fig. 6.6.

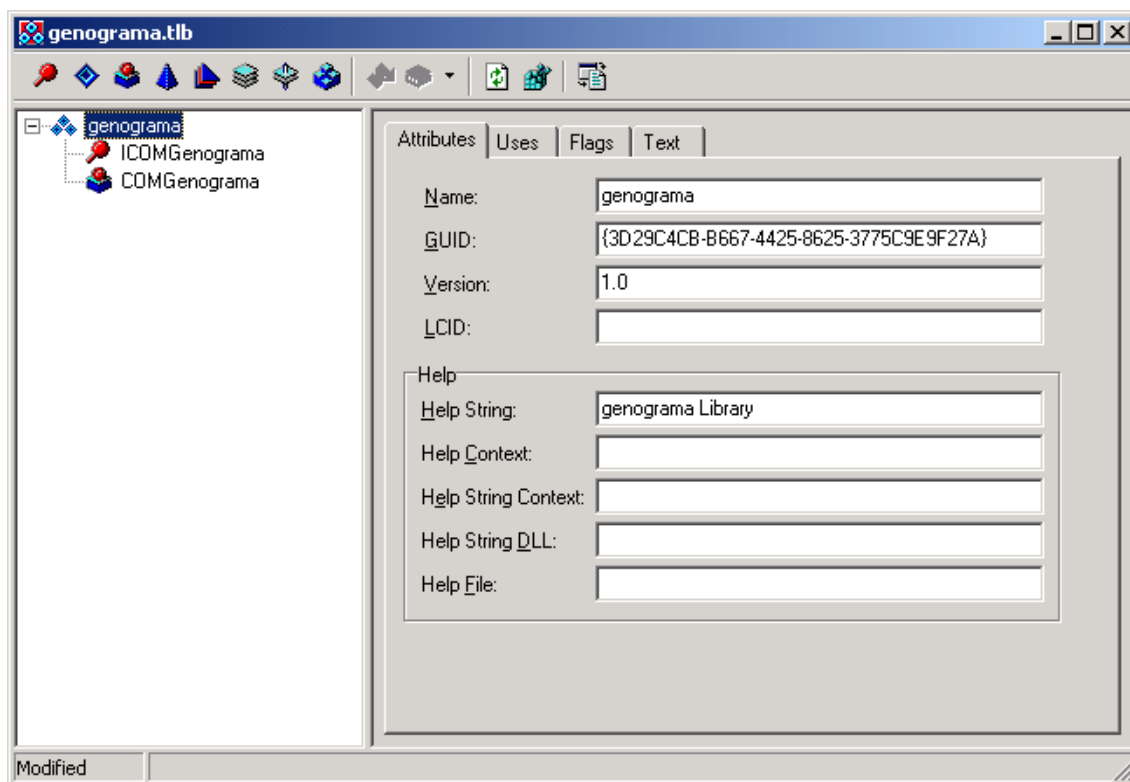


Figura 6.6: Tela inicial do Type Library

O *Type Library* é um editor para visualmente criarmos os métodos que serão disponibilizados pelos componentes. Para se criar um novo método basta posicionar o mouse sobre o *ICOMGenograma* (a letra “I” na frente do nome é adicionada automaticamente pelo ambiente e vem de *Interface*) e clicar com o botão direito, onde

aparecerá a opção *New* e, dentro dessa opção, escolha *Method*. Basta agora fornecer o nome para o novo método criado e configurar os parâmetros de entrada e saída como ilustra a Fig. 6.7.

Como podemos observar na Fig. 6.7, o parâmetro *ovarRG* é de entrada devido a configuração do campo *Modifier* estar em [in]. Já o *ovarGenograma* é um parâmetro de saída devido ao [out], ou seja, esses parâmetros de saída são variáveis que conterão possíveis valores para que o cliente que chamou o método possa ter um retorno, visto que por padrão o retorno real do método é sempre *HRESULT*. Uma vez feita toda a configuração dos parâmetros, basta agora pressionar o botão *Refresh Implementation* e verificar na unit criada para o componente o novo trecho de código gerado para a assinatura do método.

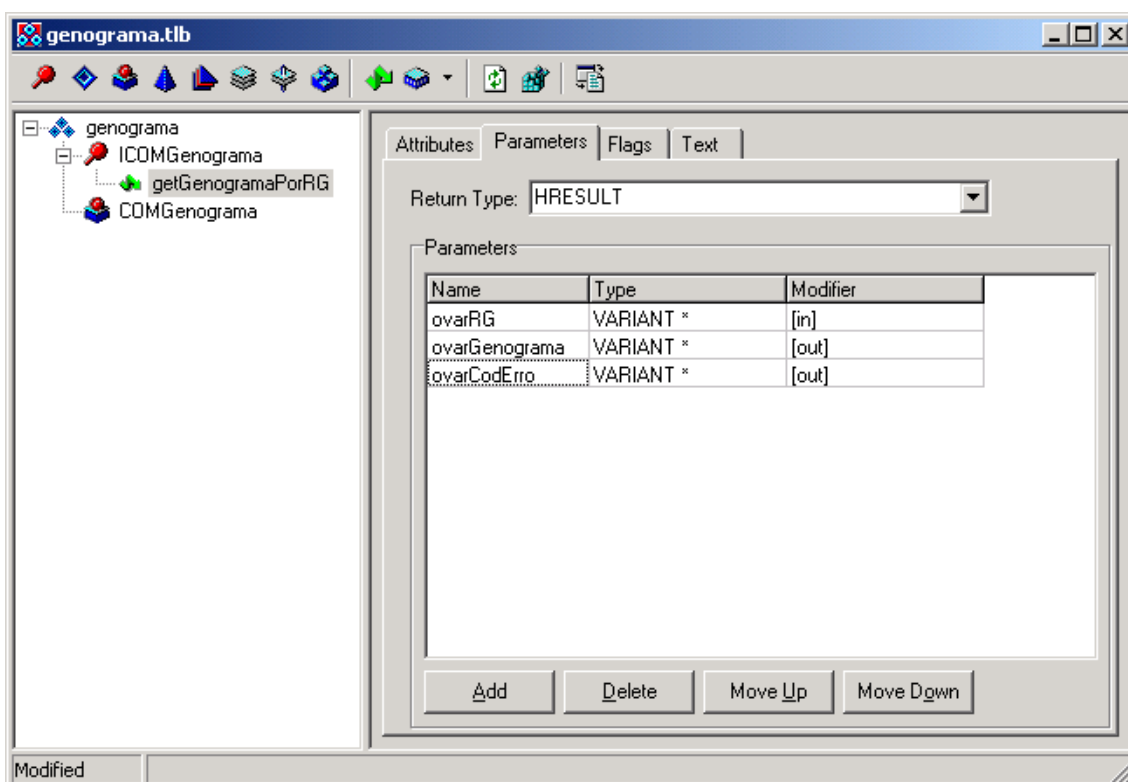


Figura 6.7: Configuração dos parâmetros de entrada e saída

Uma vez declarados e gerado o código fonte para os métodos necessários, basta agora salvar o arquivo da *Type Library* que conterá a implementação dos métodos criados.

Podemos ver um exemplo na Fig. 6.8.

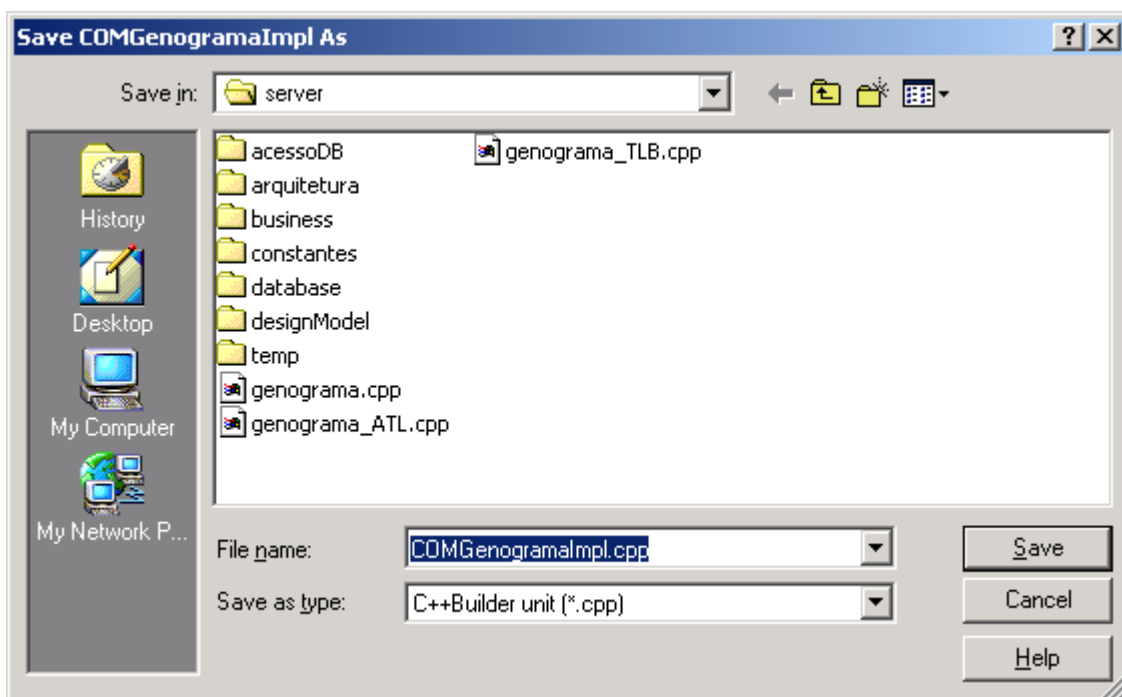


Figura 6.8: Salvando arquivo do Type Library que conterá a implementação COM+

6.2 Configurações do Ambiente C++ Builder 6 Enterprise

Por questões de otimização para a compilação e controle dos arquivos intermediários gerados pelo C++ Builder 6 Enterprise, é necessário fazer algumas configurações no ambiente. A primeira delas é criar um arquivo que conterá um repositório dos trechos de código fonte pré-compilado. A configuração é ilustrada na Fig. 6.9.

A compilação de projetos em C++ nesse ambiente tende a ser muito demorada se as devidas configurações não forem feitas. Como a tendência dos sistemas é crescer e ter cada vez mais classes para serem compiladas, pode chegar a um ponto que, se para compilar uma classe, essa tiver que compilar novamente todas as outras, isso poderia atrapalhar o rendimento do desenvolvedor.

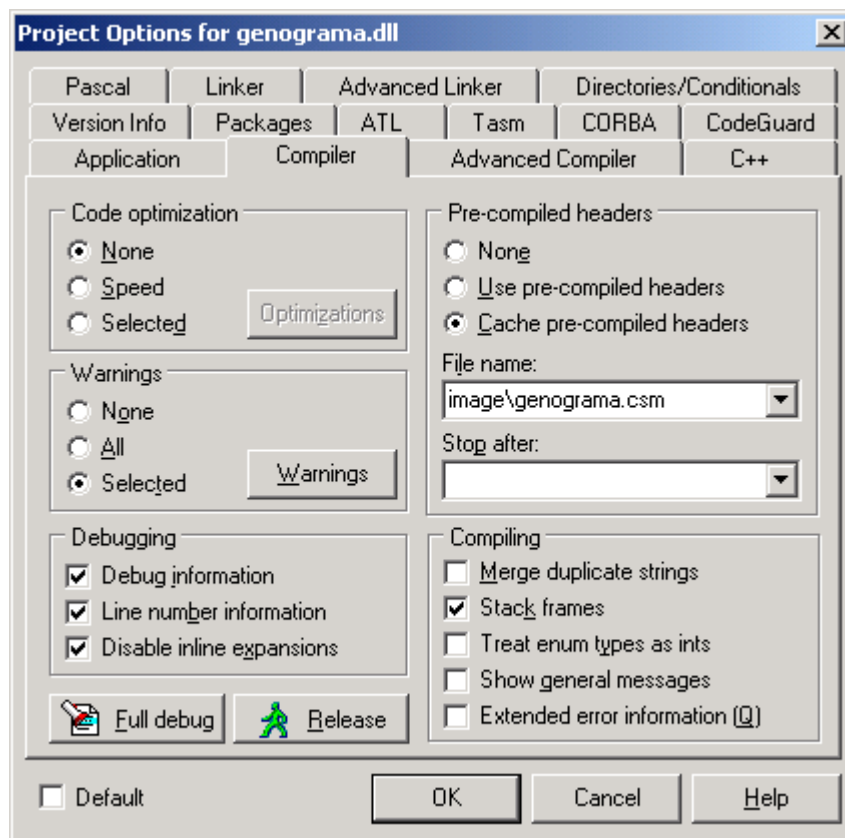


Figura 6.9: Configuração de cache para código pré-compilado

Outra configuração interessante é a do diretório intermediário de saída, ou seja, é um diretório que conterá todos os arquivos com extensão “obj”, que posteriormente serão unificados num processo de “linking” para gerar o arquivo final, que pode ser uma DLL, EXE ou algo do tipo. A configuração do diretório é ilustrada na Fig. 6.10.

Pode ocorrer, durante o desenvolvimento de algum sistema nesse ambiente de programação (C++ Builder), de o desenvolvedor realizar uma alteração no código fonte, compilar novamente a classe em questão mandando gerar uma nova aplicação, e esta não realizar o que deveria ser realizado de acordo com as mudanças feitas no código fonte. Isso ocorre devido ao fato de o ambiente tentar controlar a compilação, por questões de desempenho, e durante esse processo, se perde não gerando a aplicação como deveria ser. Se isso estiver ocorrendo durante o desenvolvimento, o melhor é forçar a compilação ou simplesmente apagar todos os arquivos com a extensão “obj” que estarão centralizados no diretório que configurarmos para isso (no nosso caso o diretório “temp”).

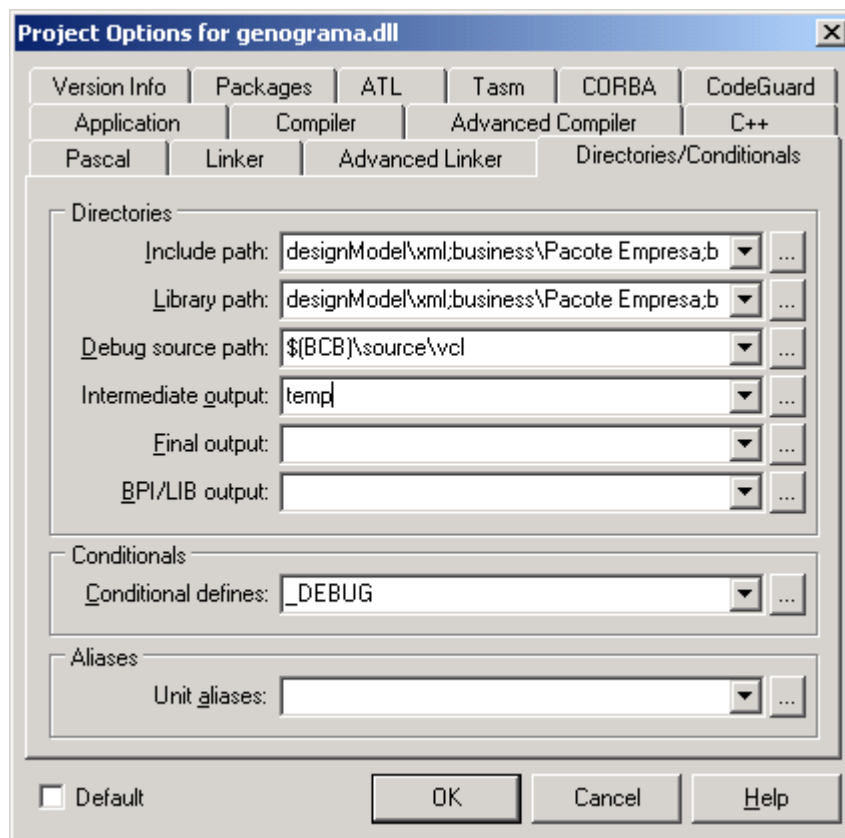


Figura 6.10: Configuração do diretório intermediário

Um dos grandes recursos encontrados nos ambientes de programação é o de acompanhar passo a passo a execução de um programa (ir acompanhando linha por linha de código conforme é executado). Nesses ambientes podemos verificar os valores das variáveis, funcionamento das condições, enfim, averiguar se tudo está correto naquele determinado trecho de código fonte. Essa é uma prática amplamente estimulada pela engenharia de software, pois acompanhando pelo menos uma vez seu código fonte em tempo de execução, diminui as chances de estar ficando algum erro para trás.

Para que o C++ Builder possa acompanhar (debug) a execução das DLL geradas, algumas configurações precisam ser feitas. A principal delas é no menu *Run | Parameters*, onde aparece a tela ilustrada na Fig. 6.11. É importante fazer essa configuração, pois o recurso de acompanhar o código fonte executando a aplicação (debug) é extremamente necessária para verificar possíveis causas de erros, garantir que o fluxo de execução está correto e entender melhor o que está acontecendo em tempo de execução.

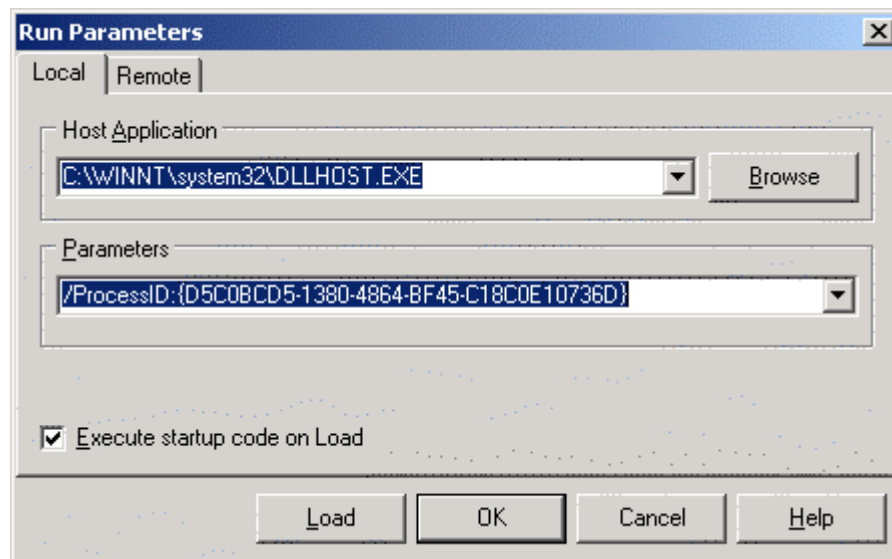


Figura 6.11: Tela com configurações necessárias para debug DLL

O primeiro campo deve conter o caminho para o *DLLHOST.EXE* como é demonstrado na Fig. 6.11. Normalmente esse arquivo vem no diretório “system32” dentro do diretório padrão do Windows (no caso, “c:\WINNT”). Se ele não se encontra nesse caminho, uma busca no disco deverá ser feita para encontrá-lo. Já o segundo campo é configurado com o *ID* específico que é gerado para a aplicação que conterá o componente. O *ID* em questão é um número único (segundo a Microsoft é praticamente impossível existir dois *ID* iguais em todo o mundo, isso devido ao algoritmo utilizado para sua geração). Para pegar esse *ID* basta ir à ferramenta *Serviços de Componente* (Fig. 6.12) clicar com o botão direito sobre o *aplicativo* criado para as DLLs e escolher *propriedades*. Feito isso basta copiar o número *Ident. de Aplicativo* que aparecerá na tela, conforme ilustra a Fig. 6.13.

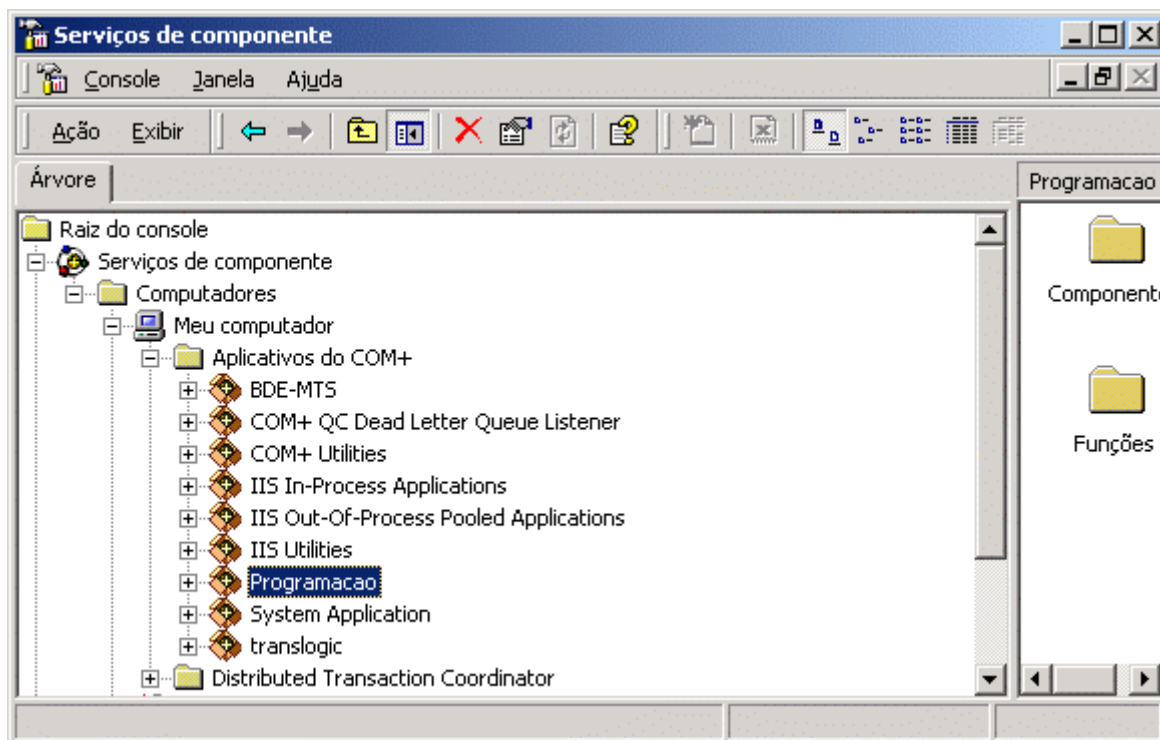


Figura 6.12: Tela da ferramenta Serviços de Componente

A ferramenta *Serviços de Componente* ilustrada na Fig. 6.12 mostra onde são gerenciados os componentes COM+ instalados na máquina, ou referências para outros componentes em outros servidores.

Nosso escopo, dentro dessa ferramenta, fica limitado a pasta chamada *Aplicativos do COM+*. É nela que vamos criando novas aplicações (que são uma espécie de container para instalar uma ou várias DLLs que contém componentes COM+) conforme a necessidade. Normalmente, dentro dessa pasta, já vem algumas aplicações criadas pelo próprio Windows para utilização interna. Aplicativos que se destacam e que são criados pelo Windows são: *IIS In-Process Applications*, *IIS Out-Of-Process Pooled Applications* e *IIS Utilities*, sendo todos utilizados pelo IIS (Internet Information Services) e que é requisito básico para a disponibilização do Web Service na Internet.

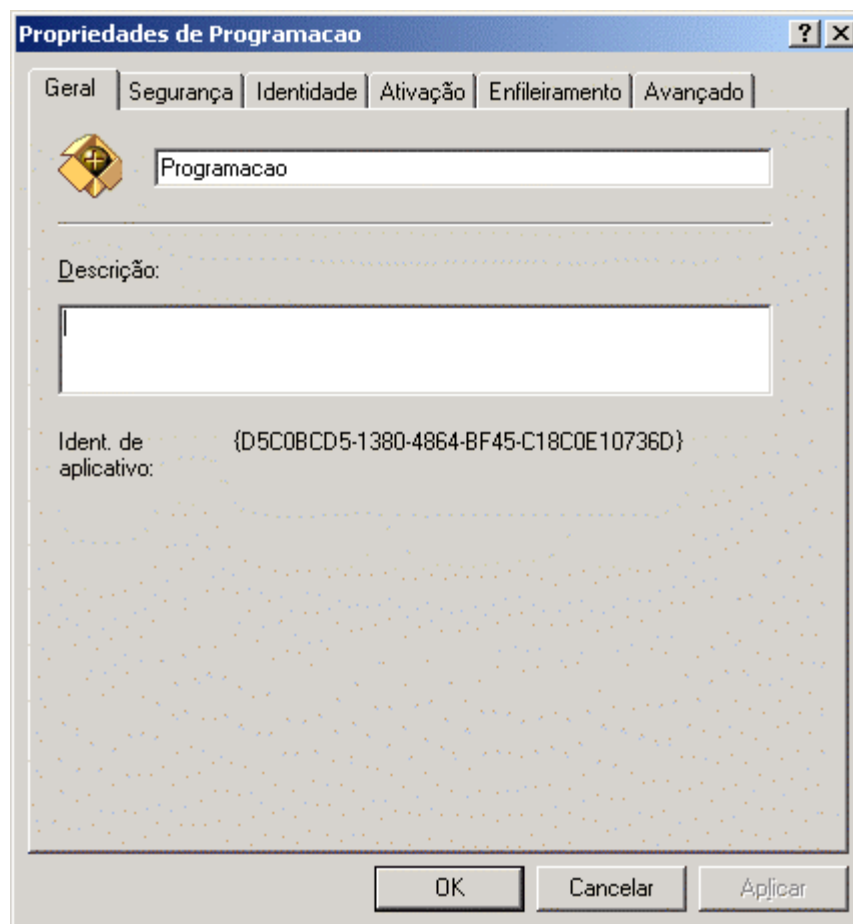


Figura 6.13: Tela de propriedade do aplicativo contendo ID necessário para “debug”

Na tela de propriedade do aplicativo (Fig. 6.13), além do *ID* necessário para configurar o recurso no C++ Builder de acompanhar o código fonte em tempo de execução, temos outras abas, com recursos específicos, fornecidos pelo COM+. Dentre esses recursos, o que se destaca, é o de segurança, que permitiria, por exemplo, não deixar certos usuários fazer uso de determinados componentes ou até mesmo métodos de um componente.

O COM+ fornece vários recursos interessantes (Pool de Objetos, Segurança, Controle de Transação, etc.) e necessários nas aplicações distribuídas que precisamos desenvolver hoje, porém, para esse trabalho não utilizaremos nenhum desses recursos em especial, somente o ambiente para registrar nossa DLL no COM+.

6.3 Detalhes sobre Padrões de Codificação

6.3.1 Tratamento de Exceções

Para as classes limítrofes, representadas pelas geradas para o ambiente COM+, nos métodos que são chamados pelos clientes ou camadas intermediárias para os clientes (Web Service), a seguinte estrutura de tratamento de exceções estará sendo utilizada (Fig. 6.14):

```
try
{
    ... IMPLEMENTAÇÃO ...
}
catch ( Geno_Exception& eGENO)
{
    *ovarCodErro = eGENO.Message;
}
catch (Exception &eErro)
{
    *ovarCodErro = eErro.Message;
}
catch (...)
{
    *ovarCodErro = _ERRO_DESCONHECIDO;
}
```

Figura 6.14: Exemplo de tratamento de exceções nas classes limítrofes

Caso alguma exceção é gerada no decorrer da execução do código protegido pelo “*try {}*” os “*catch*” apropriados capturarão essa exceção e atribuirão a mensagem vinda na exceção a variável de saída *ovarCodErro*, que poderá ser testada pelo cliente fazendo o tratamento apropriado dependendo do que estiver na variável que pode ser: vazia, caso nenhuma exceção tenha sido gerada; conter uma mensagem, caso tenha acontecido alguma exceção; e, conter uma mensagem iniciada com : (dois pontos) significando que ocorreu um “Warning” (no caso de regras de negócios específicas não serem atendidas que não gerem exceção e sim um “Warning”, uma espécie de aviso de que algo não está legal).

Podemos ter três tipos de exceções: a *Geno_Exception* que é um tipo de exceção especificamente projetada para os casos em que objetos de negócios da aplicação geram alguma exceção; a *Exception* que é a exceção padrão em C++; e o terceiro *catch (...)*

captura qualquer outro tipo de exceção que não tenha sido capturada nos *catch* anteriores.

Para as classes de negócio temos uma estrutura de tratamento de exceções bem parecida com as das classes limítrofes (Fig. 6.15), a única diferença é que nesse caso o tratamento de exceções simplesmente encaminha (através do *throw*) as mensagens recebidas em exceções vindas das classes de persistência, que são as responsáveis em grande parte da geração das mensagens de erro.

```
try
{
    ... IMPLEMENTAÇÃO ...
}
catch (Geno_Exception &eGENO)
{
    throw Geno_Exception(eGENO.Message);
}
catch (Exception &eErro)
{
    throw Exception(eErro.Message);
}
catch (...)
{
    throw Exception(_ERRO_DESCONHECIDO);
}
```

Figura 6.15: Exemplo de tratamento de exceções nas classes de negócio

6.3.2 Tratamento do XML

Todo o tratamento do XML é feito nas classes limítrofes (classes geradas para o ambiente COM+). Existem basicamente duas situações de manipulação do XML: uma quando o XML é entrante com dados para serem persistidos; e a outra, quando o XML é de saída, tendo que ser gerado a partir de dados trazidos da base de dados.

Para o caso de XML entrante, um conjunto de *FOR* alinhados percorrem o XML para pegar os dados necessários e chama o método adequado da classe de negócio para tratar os dados retirados do XML. A Fig. 6.16 possui um trecho de código que ilustra essa situação. É claro que, para cada conjunto de dados XML que precise de uma lógica de negócio própria para sua persistência ou qualquer outro processamento, terá que ter um trecho de código parecido com o da Fig. 6.16 para decompor a informação.

```

xmlDom->loadXML(WideString(VarToStr(ovarGenograma)));
raiz = xmlDoc->documentElement;
for (tabela = raiz->firstChild; false != tabela; tabela = tabela->nextSibling){
    for (linha = tabela->firstChild; false != linha; linha = linha->nextSibling){
        dadosColuna->Clear();
        for (coluna = linha->firstChild; false != coluna; coluna = coluna->nextSibling){
            //montar array com colunas para serem persistidas
            dadosColuna->Add(WideString(coluna->text));
        }
        if (WideString(tabela->nodeName) == WideString("PESSOA")) {
            atributoID = linha->attributes->get_item(0);
            atributoPI = linha->attributes->get_item(1);
            atributoGenero = linha->attributes->get_item(2);
            genograma->inserirPessoa( atributoID->Text,
                                     atributoPI->Text,
                                     atributoGenero->Text,
                                     dadosColuna->Strings[0], //Nome
                                     dadosColuna->Strings[1] ); //Nota
        }
    }
}
}

```

Figura 6.16: Exemplo de FOR alinhado para percorrer XML entrante e acionar métodos de negócio apropriados

Já para o caso de XML de saída, inicialmente é chamado um método de negócio passando como parâmetro um *TADOQuery* que conterà ou não os dados da base de dados que deverão ser convertidos para XML. Uma vez, tendo esses dados, podemos percorrê-los e ir montando o XML de acordo com os elementos que definimos através da modelagem em UML e exemplos em XML. Um exemplo para esse caso, pode ser avaliado na Fig. 6.17.

```

genograma->getPessoas(oadoqListaPessoas);
oadoqListaPessoas->First();
ostrXML = "<GENOGRAMA>";
while (! oadoqListaPessoas->Eof)
{
    ostrXML = ostrXML + "<PESSOA ";
    ostrID = BDTString(oadoqListaPessoas->FieldByName("ID")->Value);
    ostrXML = ostrXML + BDTString("ID=" + StrQuoted(ostrID) + " ");
    ostrPI = BDTString(oadoqListaPessoas->FieldByName("PI")->Value);
    ostrXML = ostrXML + BDTString("PI=" + StrQuoted(ostrPI) + " ");
    ostrGenero = BDTString(oadoqListaPessoas->FieldByName("GENERO")->Value);
    ostrXML = ostrXML + BDTString("GENERO=" + StrQuoted(ostrGenero) + ">");
    ostrNome = BDTString(oadoqListaPessoas->FieldByName("NOME")->Value);
    ostrXML = ostrXML + BDTString("<NOME>" + ostrNome + "</NOME>");
}

```

```

oastrNota = BDTString(oadoqListaPessoas->FieldByName("NOTA")->Value);
oastrXML = oastrXML + BDTString("<NOTA>" + oastrNota + "</NOTA>");
oastrXML = oastrXML + "</PESSOA>";
oadoqListaPessoas->Next();
}
oastrXML = oastrXML + "</GENOGRAMA>";
return oastrXML;

```

Figura 6.17: Exemplo mostrando como converter os dados do banco em XML

6.3.3 Observações sobre Classes de Persistência

Toda classe de negócio tem uma classe “Shadow” de persistência. Essas classes de persistência tem sempre um método store() e restore(). Os store() montam os SQL com os parâmetros necessários para realizar a persistência dos dados. Para os restore(), comandos em SQL são montados junto com os parâmetros de filtro passados pelo cliente e executa esse SQL podendo ou não trazer dados dependendo dos parâmetros de filtragem.

No livro *Developing Application With Visual Basic and UML* o autor *Paul R. Reed* sugere a utilização de classes “Shadow” da mesma forma que utilizamos nesse projeto. Segundo palavras do REED:

“Outra é ter uma classe de tradução de dados, por exemplo CustomerDT (DT vem de Data Translation), para cada classe de negócio, por exemplo Customer” [REE00].

Ele ainda justifica essa solução dizendo que “dessa forma temos o benefício de ter uma solução mais orientada a objetos. Entretanto, isso deixa o sistema com mais classes.”. A questão de deixar o sistema com mais classe não é tão relevante do ponto de vista de desempenho, pois com o poder de processamento que temos hoje, vale mais deixar o sistema bem projetado, ao invés de querer economizar instanciamento de classes.

6.4 Montagem Inicial do Projeto para Gerar DLL Web Service Servidor

Nosso servidor Web Service também será um projeto feito com a ferramenta C++ Builder 6 Enterprise. Essa versão da ferramenta já possui grandes facilidades para a construção tanto de consumidores como servidores Web Services.

Para a construção de um novo servidor Web Service, devemos entrar na ferramenta e fechar o projeto padrão. Feito isso, devemos ir a *File | New | Other* e escolher a pasta *Web Services* no repositório de objetos. Dentro dessa pasta (Fig. 6.18) encontramos o ícone *SOAP Server Application*, o qual acionamos para ter acesso a tela de configuração de um novo servidor.

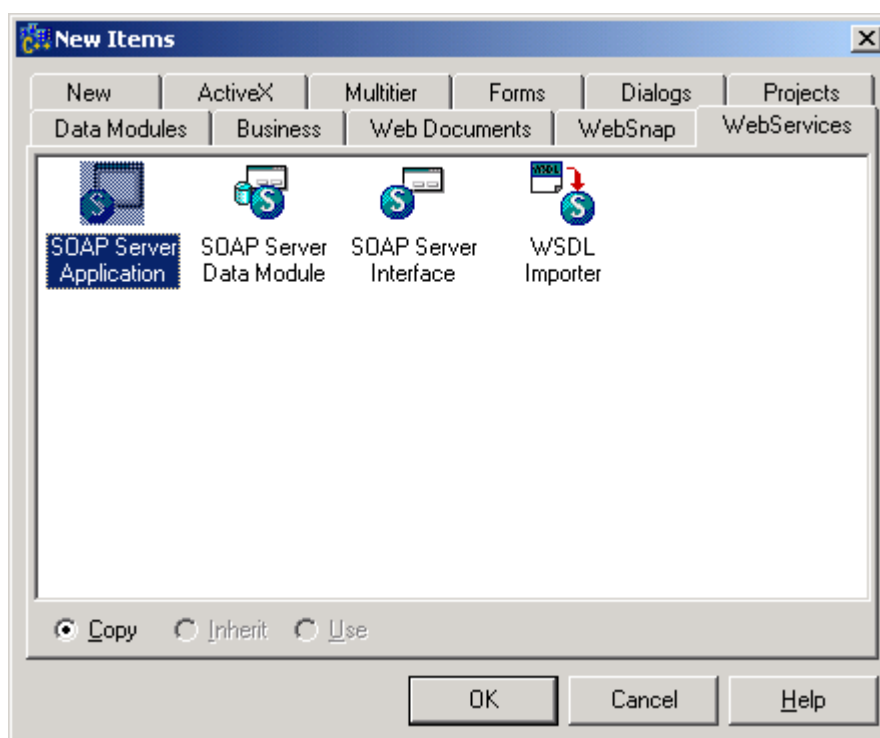


Figura 6.18: SOAP Server Application para implementação do Web Service

A tela de configuração está ilustrada na Fig. 6.19. Nela escolhemos qual será a estrutura básica do nosso Web Service. Os nomes, por si só, já dão uma noção do que se trata cada um deles. Nós vamos escolher a primeira opção (*ISAPI/NSAPI Dynamic Link Library*) por ser uma das mais indicadas para ambiente Windows com IIS (Internet Information Services).

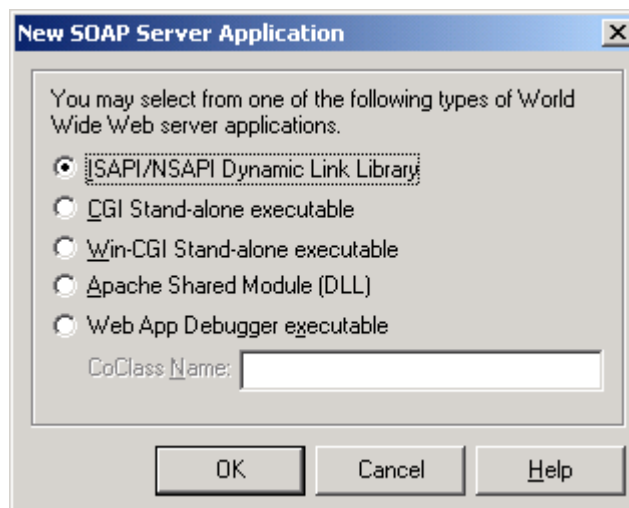


Figura 6.19: Configuração do tipo de Web Service

Feita a escolha da estrutura básica, recebemos uma pergunta (Fig. 6.20) se queremos criar a *Interface* para o módulo SOAP. Essa *Interface* será a responsável em gerar o WSDL para quem solicitar informações sobre o Web Service. O WSDL informa quais métodos estão disponíveis e como devemos chamá-los.

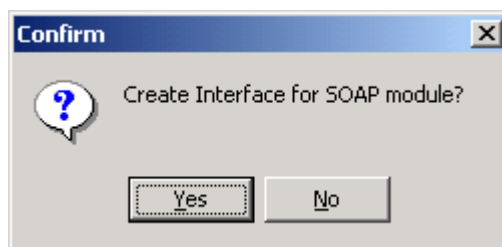


Figura 6.20: Confirmação se cria ou não a Interface para o Web Service

A próxima tela de configuração é para fornecer o nome do Web Service que está sendo criado⁴ e algumas opções de geração de comentários e métodos de exemplo. A opção *Service Activation Model* será configurada com *Per Request*, o que significa que o serviço será ativado de acordo com cada requisição. Essas opções estão ilustradas na Fig. 6.21.

⁴ A DLL gerada terá esse nome e quando for acessar o Web Service via Internet também será utilizado esse nome como complemento da URL de acesso ao serviço.

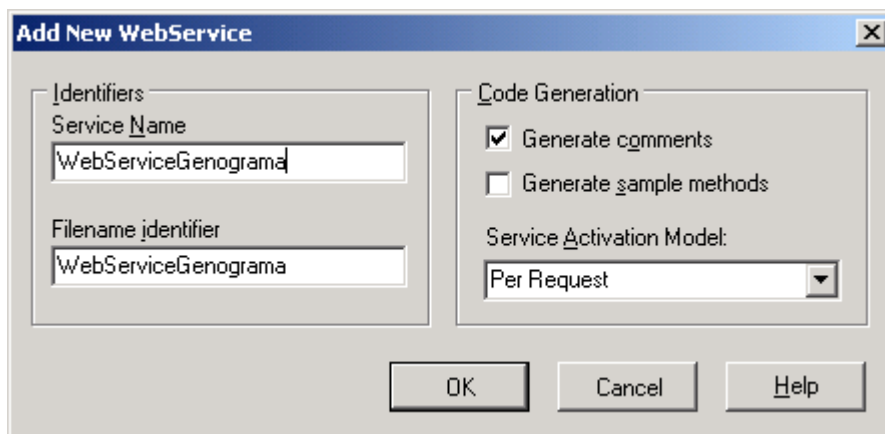


Figura 6.21: Configuração do nome para o Web Service

Feitas as devidas configurações, o gerador cria o projeto já contendo alguns componentes padrão para um servidor Web Service dentro de um *WebModule*. Podemos ver esses componentes na Fig. 6.22 que são:

- Um componente invocador (*THTTPSOPCInvoker*): O invocador converte entre as mensagens SOAP e os métodos de alguma interface registrada que pode ser invocada no servidor Web Service.
- Um componente despachante (*THTTPSopDispatcher*): Esse componente responde automaticamente a entradas de mensagens SOAP e as encaminha para o invocador.
- Um componente publicador de WSDL (*TWSDLHTMLPublish*): Responsável em publicar um documento WSDL que descreve as interfaces e como chamá-las.

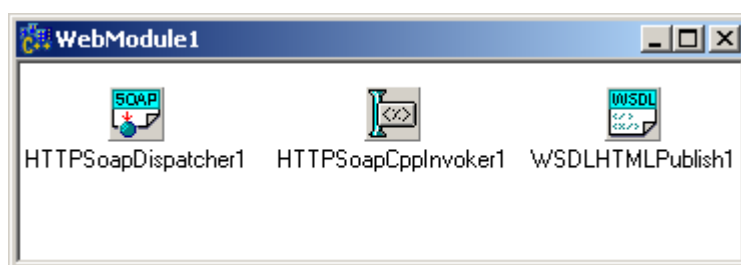
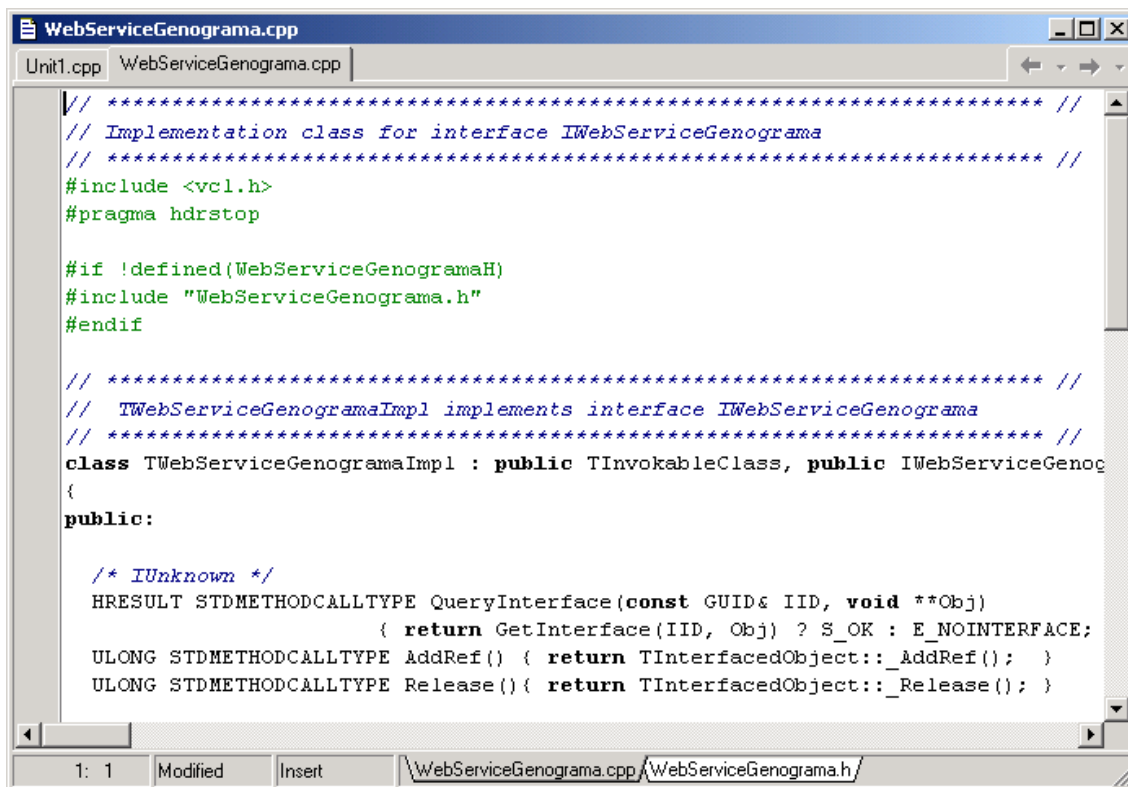


Figura 6.22: Três componentes gerados automaticamente para o Web Service

Nesse ponto, nosso Web Service já tem toda a infra-estrutura necessária para implementar seus métodos de negócio (Fig. 6.23), podendo conter lógicas de negócio ou acionar outros componentes que contenham as regras de negócio. O nosso caso se encaixa

na segunda opção, pois o Web Service funciona como uma fachada para que todos os nossos componentes de negócio (que estarão instalados no COM+) com seus métodos estejam disponíveis via Internet.



```

// ***** //
// Implementation class for interface IWebServiceGenograma
// ***** //
#include <vcl.h>
#pragma hdrstop

#if !defined(WebServiceGenogramaH)
#include "WebServiceGenograma.h"
#endif

// ***** //
// TWebServiceGenogramaImpl implements interface IWebServiceGenograma
// ***** //
class TWebServiceGenogramaImpl : public TInvokableClass, public IWebServiceGenog
{
public:

    /* IUnknown */
    HRESULT STDMETHODCALLTYPE QueryInterface(const GUID& IID, void **Obj)
    { return GetInterface(IID, Obj) ? S_OK : E_NOINTERFACE; }
    ULONG STDMETHODCALLTYPE AddRef() { return TInterfacedObject::_AddRef(); }
    ULONG STDMETHODCALLTYPE Release() { return TInterfacedObject::_Release(); }

```

Figura 6.23: Arquivo com os métodos que serão disponibilizados pelo Web Service

A Fig. 6.24 está ilustrando como criar um componente que se encontra no COM+ e a partir disso invocar seus métodos para realizar alguma ação.

```

TCOMICOMGenograma* genograma = new TCOMICOMGenograma;
genograma->getGenogramaPorRG(ovarRG, ovarGenograma, ovarCodErro);
delete genograma;

```

Figura 6.24: Exemplo de código fonte que cria o componente de negócio

Tendo o projeto do Web Service todo montado fazendo chamadas para métodos de componentes instalados no COM+, temos agora que compilar esse projeto e gerar uma DLL que deve ser colocado no diretório “/inetpub/scripts”, para o caso do IIS.

6.5 Montagem do Consumidor de Web Service

Considerando que o nosso Web Service já se encontra parcialmente implementado em C++ fazendo uso dos componentes de negócio instalados no COM+, podemos agora construir o módulo do lado do cliente (usando os recursos do Delphi 6 Enterprise Edition) que irá consumir os dados do nosso servidor Web Service.

Antes de começarmos a demonstrar como se implementa um consumir de Web Service em Delphi 6, nós podemos verificar que nosso Web Service já se encontra disponível⁵. Para fazermos um teste, basta acessarmos o endereço: “http://localhost/Scripts/WebServiceSicogef.dll/wsdl/IWebServiceGenograma”. Podemos verificar na Fig. 6.25 a página de retorno que esse endereço gerou. O “/wsdl/IWebServiceGenograma” no final do endereço diz para meu Web Service retornar as definições do mesmo através de WSDL (Web Service Description Language). Vale lembrar que esse endereço é acionado e utiliza o protocolo HTTP, podendo o Web Service estar localizado em qualquer tipo de máquina e sistema operacional, desde que tenham suporte aos recursos de Web Service.

A utilização de dois ambientes de desenvolvimento diferentes (C++ Builder e Delphi) com suas respectivas linguagens (C++ e Object Pascal) foi proposital justamente para mostrar que, além da independência de plataforma e sistema operacional, os Web Services trocam informações entre sistemas desenvolvidos com linguagens diferentes.

⁵ No caso, como os testes são em uma única máquina os links são sempre para o localhost por questões de limitação de outras máquinas na Internet para teste, porém o princípio é o mesmo e funciona na Web.

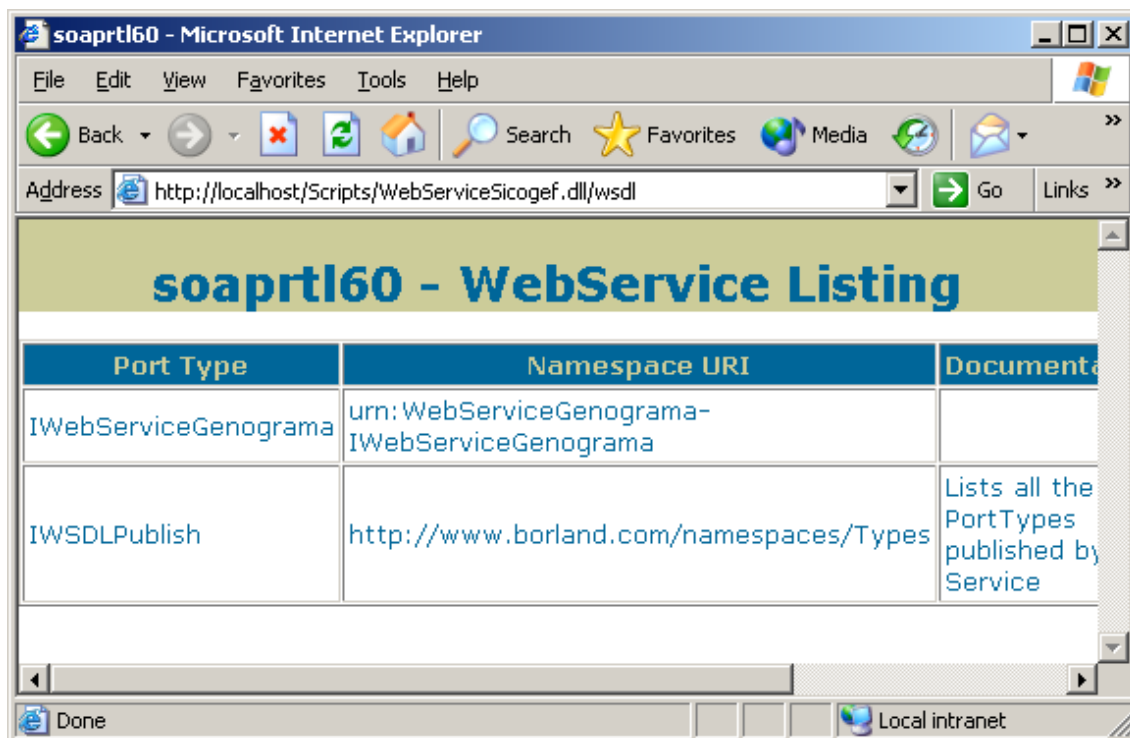


Figura 6.25: Página de retorno WSDL

Uma vez tendo esse endereço funcionando, nós precisamos gerar uma *Unit* importadora no Delphi que represente a Interface do Web Service. Essa *Unit* irá descrever o que podemos fazer com ele. Para gerar a *Unit* importadora no Delphi 6 é muito fácil, pois temos um auxiliar chamado *Web Services Importer* que faz todo o trabalho para nós. Basta ir ao menu *File | New | Other* e escolher a pasta *Web Services* do Repositório de Objetos. Chegando nesse ponto, basta escolher o ícone *Web Services Importer* e apertar o botão *Ok*. Com isso veremos o importador como ilustra a Fig. 6.26.

Como nós conhecemos o endereço do WSDL que queremos utilizar para gerar a *Unit* importadora fica fácil o processo. Agora, se tivéssemos que achar esse endereço na Internet, teríamos que usar os recursos do UDDI para localizá-lo, é claro que levando em consideração que ele tenha sido publicado no UDDI, senão não seria possível fazer essa localização.

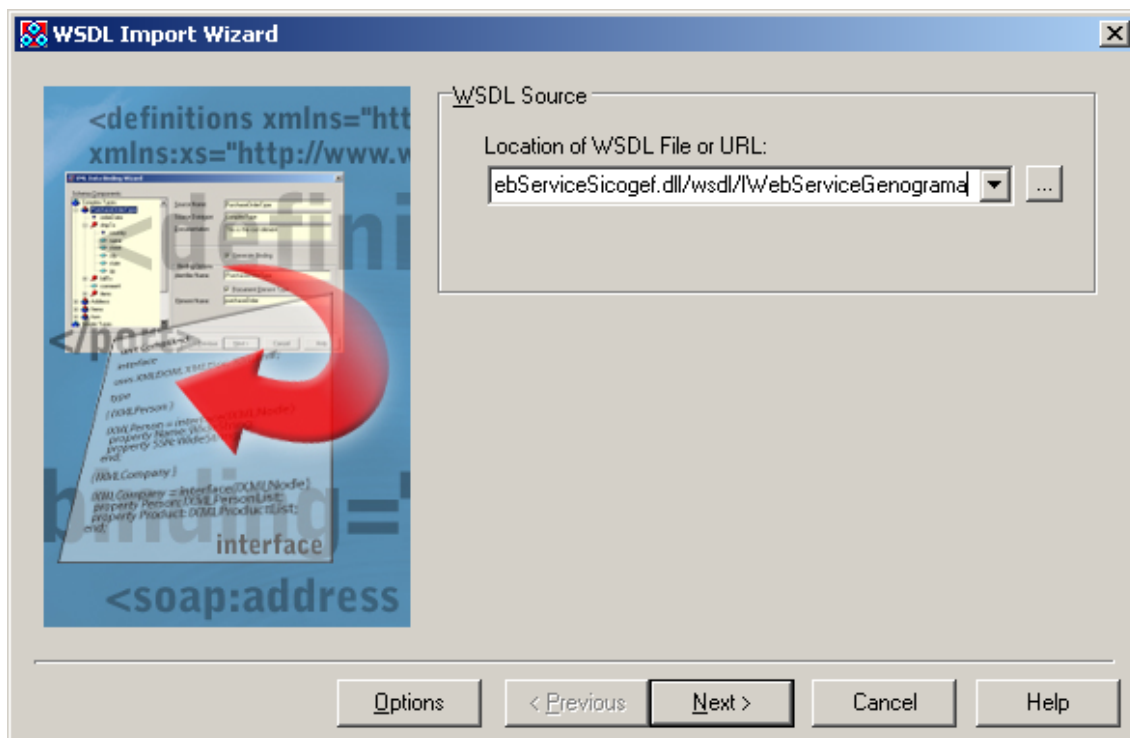


Figura 6.26: Tela principal do Web Services Importer

Na tela principal basta digitar o endereço com o WSDL do Web Service desejado (poderia ser um arquivo local com o WSDL) e apertar o botão *Next* e em seguida *Finish* para gerar a *Unit* importadora.

Com a *Unit* gerada, podemos adicioná-la em diferentes projetos Delphi que tenham a necessidade de utilizar os métodos expostos por ela. No nosso caso o projeto que contém a interface visual para o SICOGEF é quem fará a adição dessa *Unit* para fazer chamadas aos métodos que persistem e retornam XML com as informações necessárias para a criação e manutenção dos genogramas no lado do cliente.

Nesse ponto, temos que ressaltar que embora o cliente do SICOGEF seja um EXE, sua filosofia de funcionamento é fortemente baseada na Web e, devido a isso, o usuário, na hora de criar um novo genograma, terá que antes pedir essa criação no servidor, e depois sim, manipular as informações, acrescentando, alterando, excluindo, porém, sempre enviando os comandos para o servidor. Da mesma forma que não podemos recuperar uma página HTML sem uma conexão com a Internet ou um servidor ativo, também não poderemos ter acesso a certas funcionalidades no SICOGEF caso não tenhamos as

condições mínimas.

Finalmente, para que o projeto que fará uso da *Unit* importadora tenha condições de acessar o nosso Web Service e recuperar as informações desejadas através de solicitações SOAP (que são transparentes para o programador, pois a *Unit* encapsula isso), basta acrescentar o componente *HTTPRIO* (HTTP Remote Invokable Object) encontrado na pasta *Web Services* do Delphi.

O *HTTPRIO* também precisa de algumas configurações para que ele possa funcionar corretamente e fazer com que o projeto que estiver fazendo uso da *Unit* importadora, consiga alcançar o Web Service.

A primeira configuração importante é, da mesma forma que no *Web Services Importer*, fornecer o endereço com WSDL do Web Service desejado. Esse endereço deve ser fornecido na propriedade *WSDLLocation* do componente. Além dessa propriedade, temos também que configurar as propriedades *Service* e *Port* com *IwebServiceGenogramaservice* e *IwebServiceGenogramaPort* respectivamente. Com essas propriedades configuradas corretamente podemos ter acesso aos métodos como ilustra a Fig. 6.27 abaixo.

```
...
(HTTPRIO1 AS IwebServiceGenograma).getGenogramaPorRG(ovarRG, ovarGenograma, ovarCodErro);
...
```

Figura 6.27: Acesso aos métodos através de uma Interface Remota de Invocação

6.6 Montagem da Aplicação Cliente

Embora toda parte de persistência e busca de informações atualizadas fique por conta da camada servidora, recebendo e enviando XML através do Web Service, o cliente também possui uma pequena base de dados para armazenar as informações que são recebidas.

A lógica de funcionamento dessa base de dados no cliente é muito simples: toda vez que o usuário precisar criar um novo genograma, terá que fazer isso solicitando para a camada servidora, que por sua vez envia a estrutura básica (em XML) para essa criação. Essa estrutura básica será replicada na base de dados do cliente, o qual pode realizar todo o

trabalho de construção do genograma. Quando esse trabalho estiver concluído, ele solicita a atualização junto ao banco de dados na camada servidora, que faz uma validação de versão do que está sendo enviado para atualização, isso para evitar que, conteúdo atualizado por outro usuário no mesmo genograma não seja perdido.

Para que o cliente possa ter acesso ao Web Service no servidor, foi criado uma *Unit* importadora (discutida em 6.5) para permitir que esse cliente seja um consumidor de Web Service. Junto a essa *Unit* importadora que permite a comunicação via HTTP com o Web Service, teremos um módulo que pega as informações em XML que a classe importadora recebe e se responsabiliza em persistir na base de dados local para que possam ser trabalhados.

Uma vez os dados estando na base de dados do cliente, o componente de interface principal da aplicação (FlowChart, com as devidas adaptações no código fonte para atender ao nosso negócio) consegue ler essas informações e mostrar de forma gráfica o genograma que está armazenado na tupla em questão. Convém lembrar que a parte gráfica tanto na camada servidora como na cliente é armazenada de forma binária e transferida assim entre as partes.

O importante é destacar que, os testes básicos de funcionamento dessa estruturação, foram feitos e se mostraram propícios para a construção completa de um sistema usando essa arquitetura.

7 TRABALHOS FUTUROS

A preocupação com a segurança da aplicação é uma parte importante e que merece atenção em trabalhos futuros. Essa preocupação se deve, principalmente, pelo fato de o genograma de uma família ser algo confidencial e que merece sigilo.

A abordagem com Web Services é nova, mas como sua base é o HTTP, este já possui alguns mecanismos de segurança que poderão ser utilizados tal como: SSL, Criptografia dos Dados e outros.

Outro tópico, que ficará para trabalhos futuros, é a redefinição dos símbolos e como eles são conectados para representar os dados de uma família. Essa redefinição deve ser aprimorada, pois estará mexendo com padrões internacionais para a elaboração do genograma. Porém, essa mudança se faz necessária por vários motivos:

- Ter a possibilidade de representar novas configurações familiares que estão surgindo na sociedade moderna, tal como: casamento homossexual, bebê de proveta e outras. Além disso, as formas de relacionamento entre os indivíduos são diversas e não devem ficar atreladas somente ao emaranhamento, aliança saudável, distante, conflitual, e outros que existem no mapa estrutural da família como conhecemos hoje.

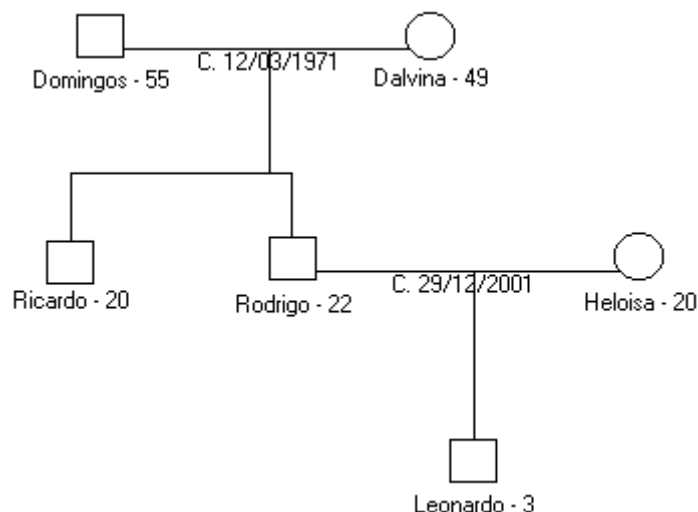


Figura 7.1: Forma atual de representação do genograma

- Um outro motivo é a falta de embasamento matemático na abordagem atual para a montagem de um genograma. A idéia seria, redefinir como é montado o genograma, tentando chegar a um modelo próximo do conceito de grafo. Então, ao invés de termos uma representação do genograma da forma atual como é ilustrado na Fig. 7.1, teríamos uma como é ilustrado na Fig. 7.2. A diferença principal seria, tirar as ligações de indivíduos com relacionamentos (o que ocorre no genograma atual para o caso de filhos de um casal) e, de certa forma, unificar a representação estrutural com a simbologia para representar a realidade social da família, de modo que, com uma única aresta com símbolos diferenciados nas extremidades (pesos) possa ter um poder de representação maior.
- Uma vez tendo essa nova forma de representação pronta, podemos aplicar técnicas de Inteligência Artificial para extrair informações importantes ou identificar padrões nos genogramas representados nesse novo modelo, sendo a aplicação de técnicas de Inteligência Artificial mais uma proposta de trabalho futuro.

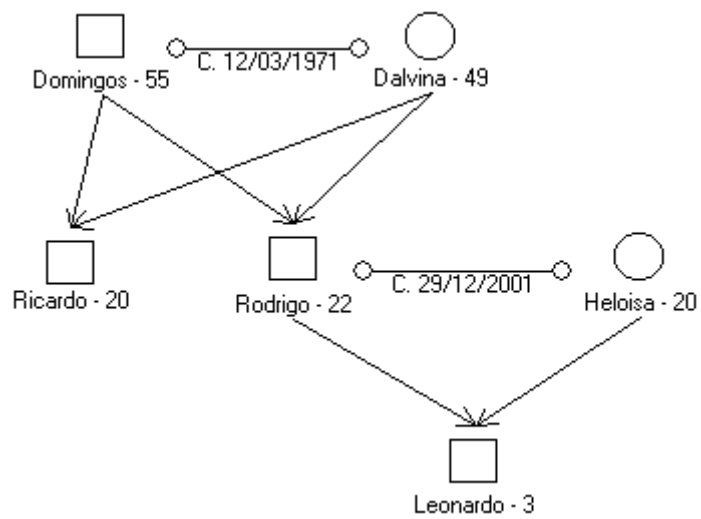


Figura 7.2: Nova proposta de representação do genograma

8 CONCLUSÃO

O desenvolvimento do SICOGEF trará grandes contribuições tanto para o lado da terapia familiar como para a ciência da computação. Para a terapia familiar, podemos destacar a facilidade de armazenamento e transmissão dos dados e a possibilidade de interação através da Internet. Além disso, quem for utilizar os recursos fornecidos pelo sistema, um Juiz, por exemplo, terá grande facilidade de extrair informações dessa representação gráfica, pois seriam necessárias várias páginas para descrever todas as informações que podemos obter apenas olhando para um genograma. Imaginando uma situação, onde esse Juiz tenha que tomar uma decisão sobre a guarda dos filhos de um casal, de acordo com as informações contidas num genograma, a chance de uma tomada de decisão coerente pode ser maior. Já para a ciência da computação, tentaremos demonstrar como realizar um trabalho utilizando um desenvolvimento baseado em componentes; Web Services e XML para representar e trocar informações dando um embasamento teórico e prático da utilização de componentes para o desenvolvimento de software.

As facilidades que os recursos de Web Services trouxeram no sentido de termos as aplicações cliente em qualquer parte, onde tenha Internet, são de grande valor, principalmente com a demanda que existe, hoje, de aplicações distribuídas.

A modelagem das informações XML em UML também contribui muito para a manutenção, elaboração e implementação em XML dessas informações. Uma vez que a UML pode se tornar padrão de comunicação entre desenvolvedores, fica mais fácil discutir sobre as informações necessárias com esse tipo de linguagem.

A possibilidade de guardar as informações dos genogramas de várias famílias em meio digital, de forma organizada, é uma fonte para a realização de pesquisas elaboradas sobre o comportamento humano, podendo vir a identificar novos padrões e configurações

familiares nos tempos modernos.

O desenvolvimento de sistemas distribuídos nunca foi uma tarefa fácil. Com as pesquisas já desenvolvidas na área, hoje temos algo mais consolidado em termos de teorias e implementações práticas, onde conseguimos bons resultados de forma viável. A teoria de desenvolvimento baseado em componentes, COM+ da Microsoft e os milhares de componentes gratuitos e comerciais disponíveis, são provas de que uma grande indústria de software veio para ficar.

Em termos arquiteturais, uma das grandes vantagens de utilizar um Web Service como meio de acesso para a camada de negócio é a possibilidade de, se for preciso, trocar a camada de apresentação sem grandes problemas. Isso se deve ao fato de o protocolo SOAP deixar o sistema resiliente, ou seja, as camadas de apresentação, negócio e de dados ficarem independentes entre si. Podemos conseguir essa independência, utilizando conceitos de arquitetura e engenharia de software juntamente com as novas tecnologias disponíveis no mercado (XML, ADO, Web Services).

A filosofia de programação declarativa através de telas de configuração, embora tenha gerado uma quantidade grande de figuras nesse trabalho, na hora de fazer a implementação de um sistema ajuda, pois evita de ficar codificando tudo, linha por linha, sendo bem mais agradável e rápido através da declaração do que se espera.

Mesmo com uma quantidade enorme de novas tecnologias e conceitos para a construção de sistemas distribuídos, podemos concluir que os objetivos desse trabalho foram atingidos, conseguindo montar uma arquitetura de software que atendesse um problema específico da área humana (modelagem do genograma através de um sistema computacional).

9 ANEXOS

ANEXO 1 – DIAGRAMA DE IMPLANTAÇÃO DO SICOGEF

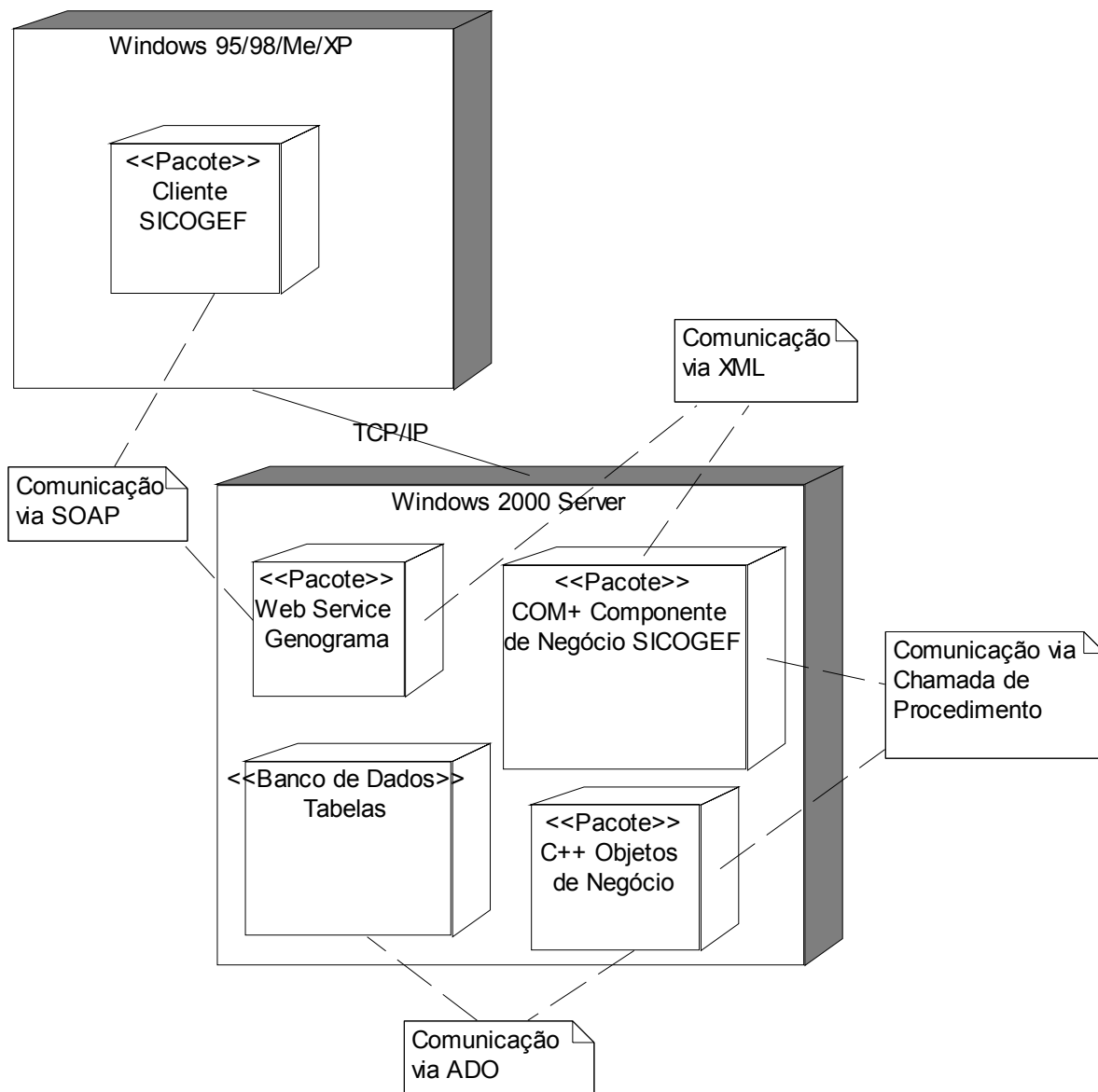


Figura 9.1: Diagrama de implantação para o SICOGEF

ANEXO 2 – EXEMPLO GENÉRICO DE MAPEAMENTO UML PARA XML

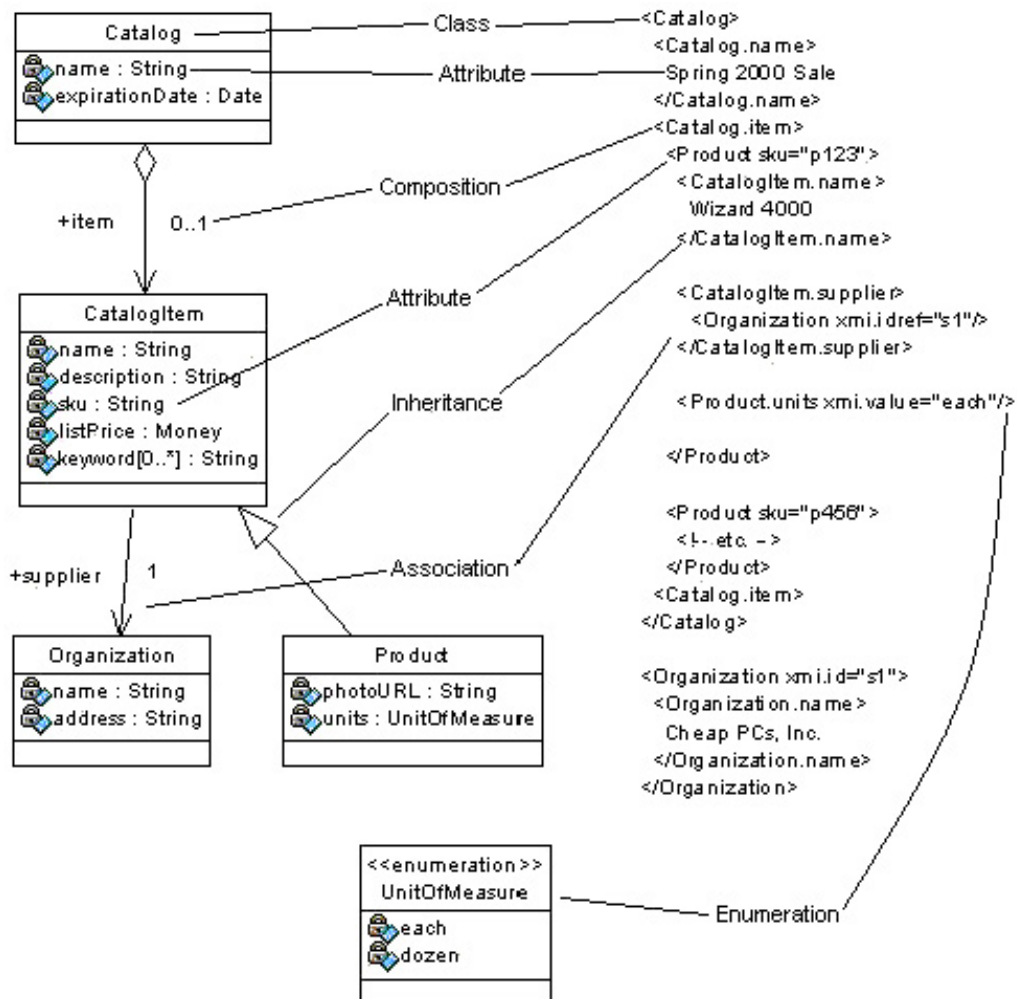


Figura 9.2: Exemplo genérico de mapeamento da UML para XML [CAR01]

ANEXO 3 – DIAGRAMA DE CLASSES

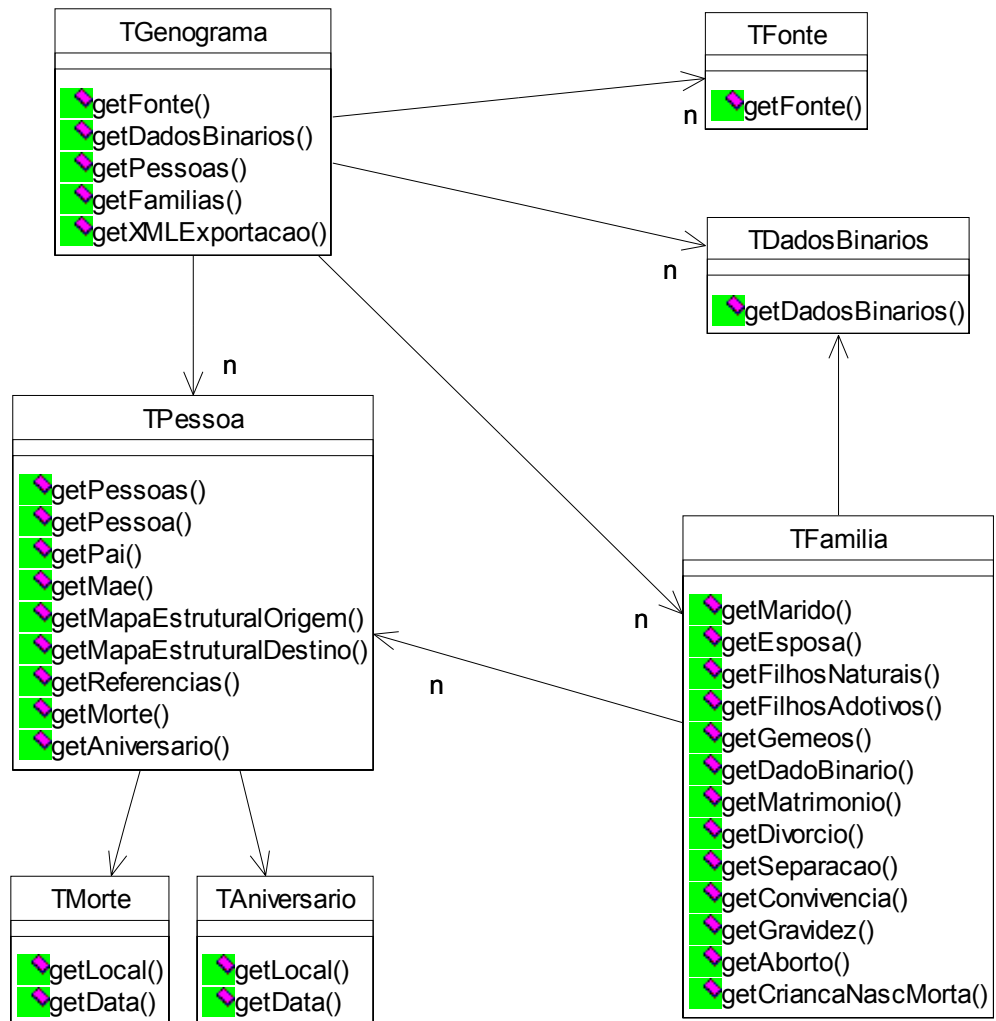


Figura 9.3: Diagrama de classes

ANEXO 4 – MER – MODELO ENTIDADE RELACIONAMENTO

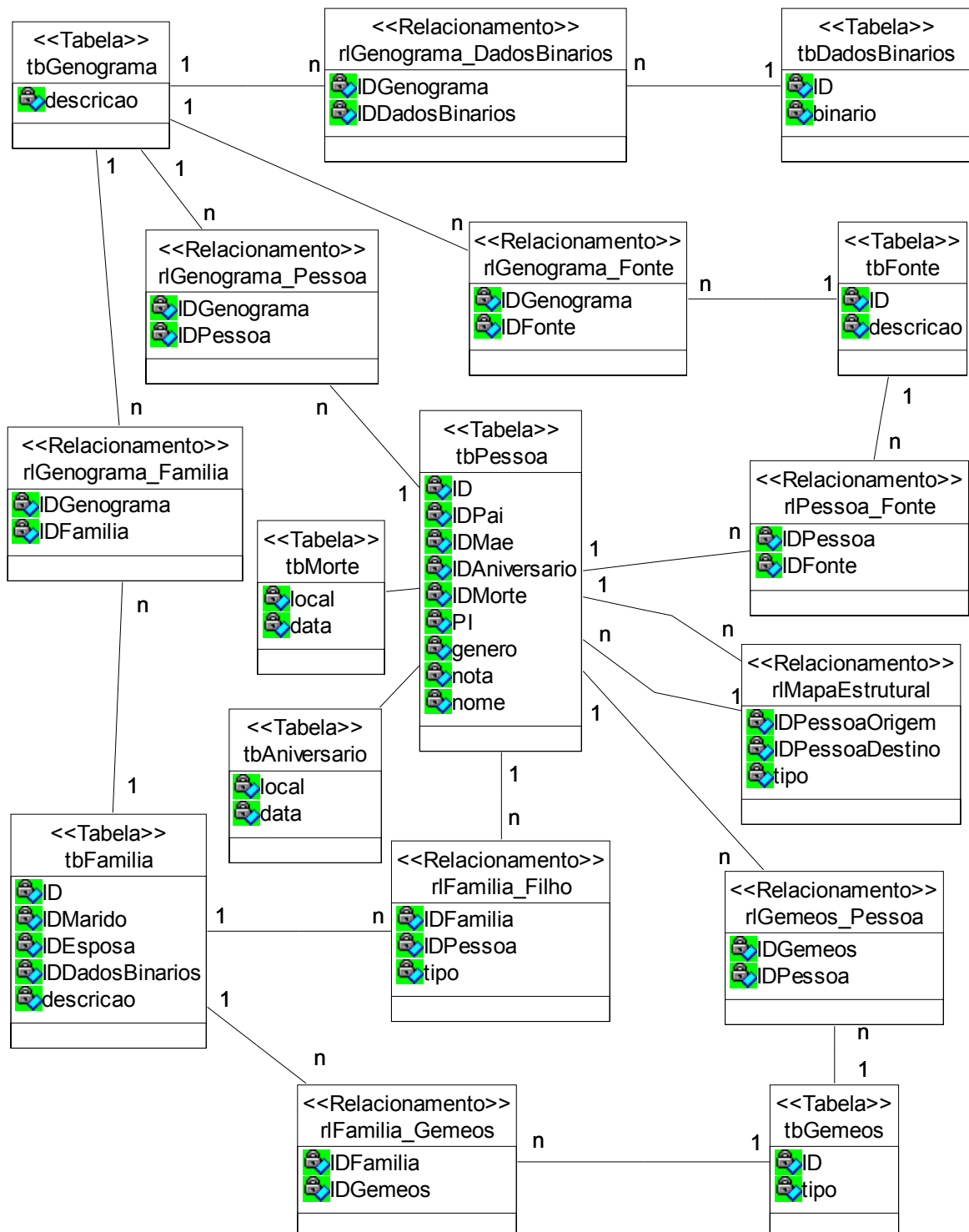


Figura 9.4: MER – Primeira parte

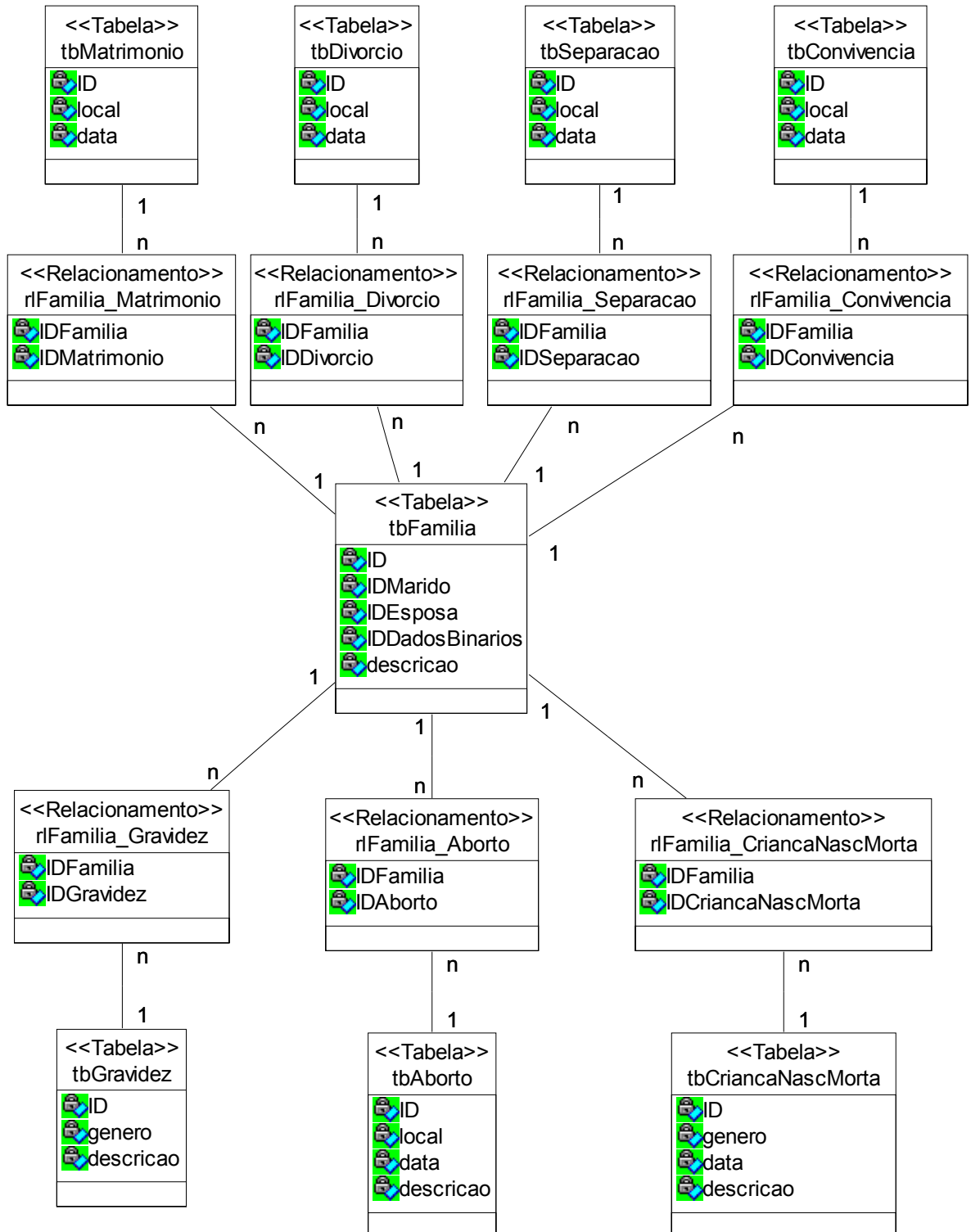


Figura 9.5: MER – Segunda parte

ANEXO 5 – OUTRAS REPRESENTAÇÕES EM UML DO XML

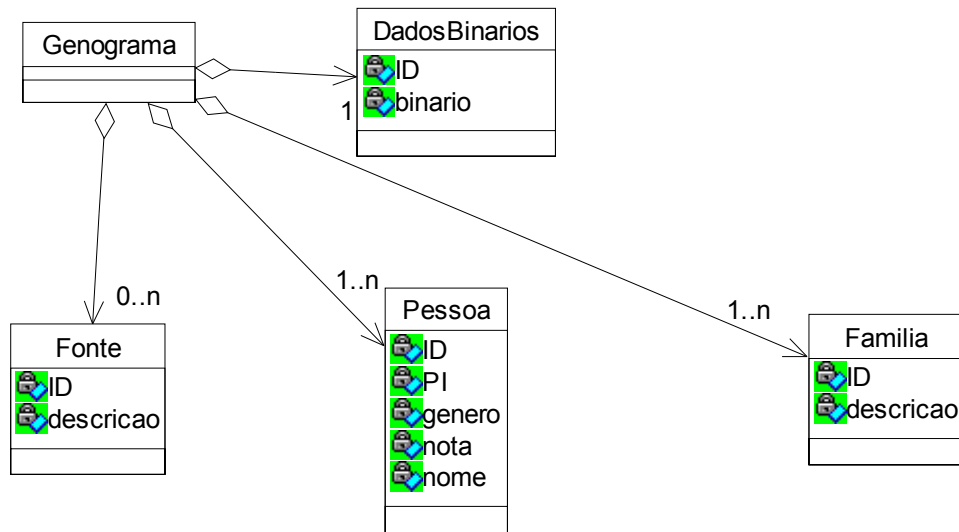


Figura 9.6: Composição geral do XML genograma

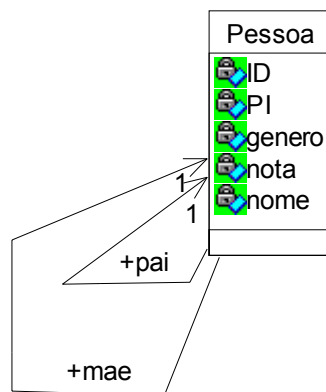


Figura 9.7: Pais de uma pessoa

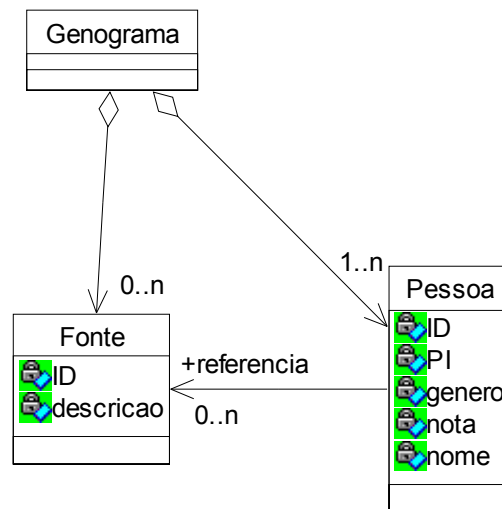


Figura 9.8: Referência de uma pessoa a uma fonte

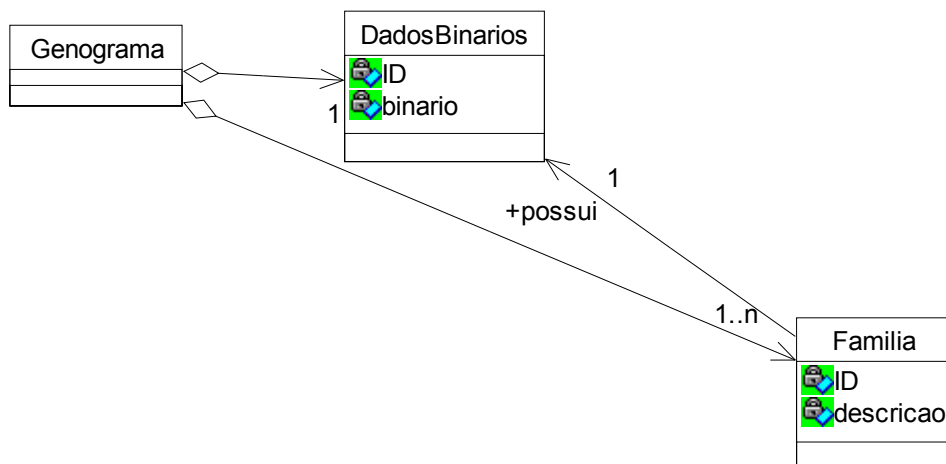


Figura 9.9: Dados binários

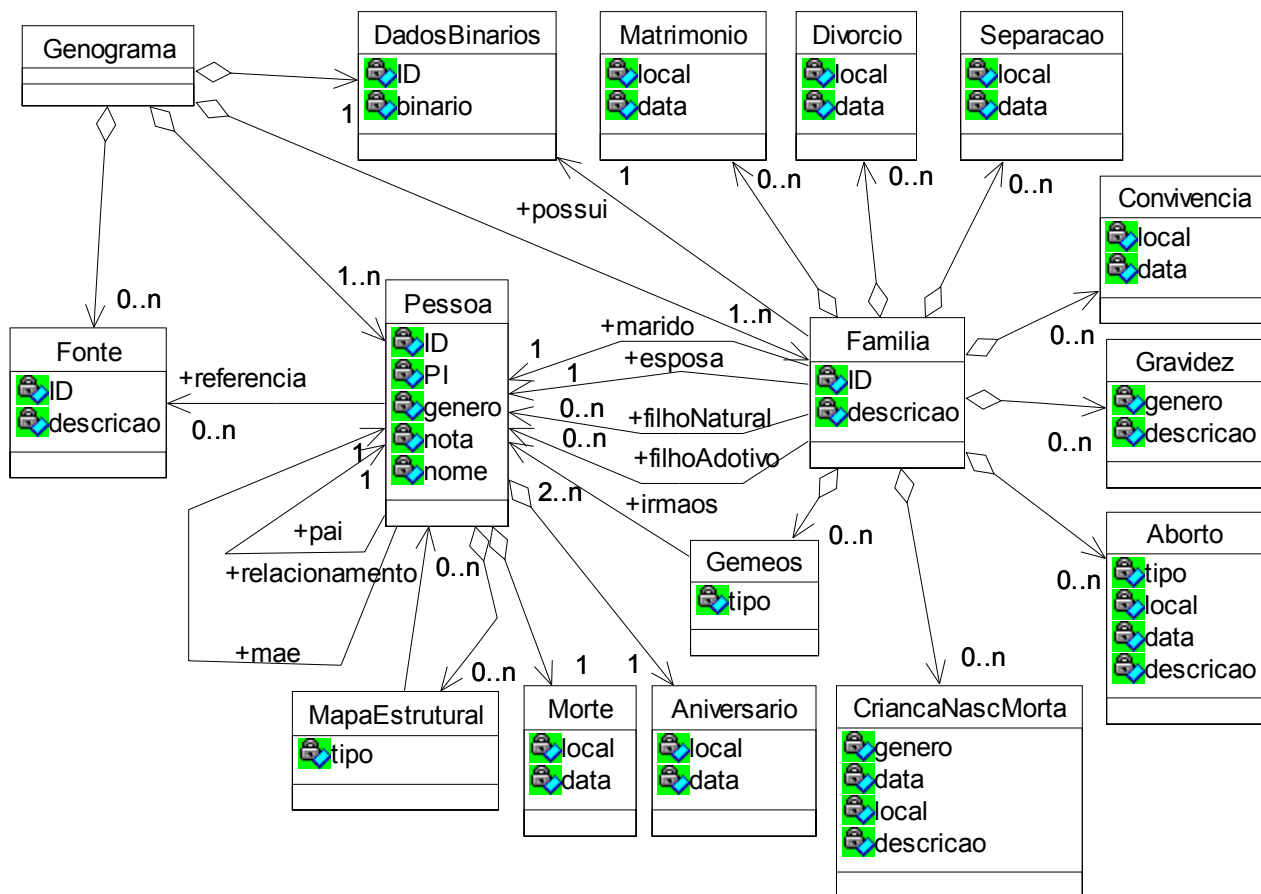


Figura 9.10: Representação completa

10 GLOSSÁRIO

COM+: Recurso tecnológico disponibilizado pela Microsoft em algumas versões do sistema operacional Windows. Oferece vários recursos para os desenvolvedores, tais como: Pool de Objetos, Controle de Transações, Escalabilidade, etc.

Genograma: Nome da técnica utilizada por vários profissionais para fazer a leitura de informações de uma família de forma gráfica.

Objetos “Shadom”: Conceito arquitetural para centralizar responsabilidades com relação a acessos ao banco de dados. Toda classe de negócio que é persistente terá uma classe específica para tratar sobre os recursos necessários para sua manipulação no banco. Essa classe específica é quem dá origem aos Objetos “Shadom”.

SICOGEF: Sigla encontrada para servir como nome do sistema. Ela vem de “Sistema Computacional do Genograma Estrutural da Família”.

Web Service: Recurso que possibilita deixar serviços disponíveis na Internet de forma transparente.

Windows DNA: Proposta de arquitetura de desenvolvimento da Microsoft mostrando quais as melhores práticas para se construir uma aplicação utilizando sua tecnologia.

11 REFERÊNCIAS BIBLIOGRÁFICAS

- [AOY02] AOYAMA, Mikio; **A Business-Driven Web Service Creation Methodology**; IEEE; 2002.
- [BRO98] BROWN, Debbie; **Local Guru Explores the World of Virtual Work**; Virtual Teams – Real Results; Harvard College; 1998.
- [BOO00] Booch, Grady; Rumbaugh, James; Jacobson, Ivar; **UML – Guia do Usuário**; Editora Campus; 2000.
- [BOO01] BOOCH, Grady. **Web Services: The Economic Argument**. Disponível na Internet. <http://www.sdmagazine.com/print/documentID=16851> 26. fev. 2002.
- [CAR95] Carter, Betty; McGoldrick, Mônica; **As mudanças no ciclo de vida familiar – uma estrutura para a terapia familiar**; 2 ed.; Editora Artes Médicas Sul Ltda; 1995.
- [CHE00] CHEUNG, David; et all; **Distributed and Scalable XML Document Processing Architecture for E-Commerce Systems**; IEEE; 2000.
- [CAR01] Carlson, David; **Modeling XML Applications With UML – Practical e-Business Applications**; Addison-Wesley; 2001.
- [CER02] CERAMI, Ethan. **Top Ten FAQs for Web Services**. Disponível na Internet. <http://web.oreillynet.com/lpt/a//webservices/2002/02/12/webservicefaqs.html> 27. fev. 2002.
- [FER99] Fernandes, Sirlei T. S.; **Unidade familiar – A representação estrutural e configurativa das inter-relações familiares**; Gráfica Planeta; 1999.
- [FOW00] Fowler, Martin; Scott, Kendall; **UML Distilled – A Brief Guide to the Standard Object Modeling Language**; 2 ed. John Wiley & Sons. Inc.; 2000.
- [GAE99] GAEDKE, Martin; et all; **Object-oriented Web Engineering for Large-scale Web Service Management**; IEEE; 1999.

- [GAR99] GARCEZ, Roberto Cartaxo. Trabalhando com Use Cases em Sistemas Corporativos Distribuídos. **Palestras – OD.99 – Objetos Distribuídos**, São Paulo, p.4-17, dez. 1999.
- [GRU01] GRUNDY, John; Wang, Xing; Hosking, John; **Building Multi-device, Component-based, Thin-client Groupware: Issues and Experiences**; Australian Computer Society; 2001.
- [GAB02] GABRICK, Kurt A.; Weiss, David B.; **J2EE and XML Development**; Manning; 2002.
- [HEI98] Heinckiens, Peter M.; **Building Scalable Database Applications – Object Oriented Design, Architectures, and Implementations**; Addison Wesley; 1998.
- [HAR99] HAROLD, Elliotte Rusty; **XML Bible**; IDG Books WorldWide, Inc.; 1999.
- [JON00] Jones, Capers; **Software Assessments, Benchmarks, and Best Practices**; Addison Wesley; 2000.
- [JOH01] JOHNSTON, Joe; **Binary Data to Go: Using XML-RPC to Serve Up Charts on the Fly**; Disponível na Internet; <http://web.oreilly.com>; 27. fev. 2002.
- [KIR01] KIRDA, Engin; **Engineering of Web services with XML and XSL**; ACM; 2001.
- [LEO97] LEONARD, Dorothy; Straus, Susaan; **Putting Your Company's Whole Brain to Work**; Harvard College; 1997.
- [LAR00] LARSON, Eric; Stephens, Brian; **Administrating Web Servers, Security, & Maintenance**; Prentice Hall PTR Prentice-Hall; 2000.
- [MAR89] MARLIN, Emily; **Genograms: The New Tool for Exploring the Personality, Carrer, and Love Patterns You Inherit**; McGraw Hill; 1989.
- [MCG99] MCGOLDRICK, Monica; Gerson, Randv; Shellenberger, Svlvia; **Genograms – Assessment and Intervention**; 2 ed.; W.W. Norton & Company; 1999.
- [MEY00] MEYER, Bertrand. Contracts for Components. **Software Development**, n. 7, p. 51-53, jul. 2000.
- [MCI01] MCILRAITH, Sheila A.; Son, Tran Cao; Zeng, Honglei; **Semantic Web Services**; IEEE; 2001.
- [NAR02] NARAYANAN, Srini; McIlraith, Sheila A.; **Simulation, Verification and Automated Composition of Web Services**; ACM; 2002.

- [ORC02] ORCHARD, David. **Web Services Pitfalls**. Disponível na Internet.
<http://www.xml.com/lpt/a/2002/02/06/webservices.html> 27. fev. 2002.
- [PRE02] PRESCOD, Paul. **Second Generation Web Services**. Disponível na Internet.
<http://www.xml.com/lpt/a/2002/02/06/rest.html> 27. fev. 2002.
- [REE00] Reed, Paul R. Jr.; **Developing Applications with Visual Basic and UML**;
Addison Wesley; 2000.
- [ROY01] ROY, Jaideep; Ramanujan, Anupama; **XML Schema Language: Taking XML
to the Next Level**; IEEE; 2001.
- [SZY98] SZYPERSKI, Clemens. **Component Software: Beyond Object-Oriented
Programming**; Addison-Wesley; 1998.
- [SCH00] SCHRANZ, Markus W.; et all; **Engineering Complex World Wide Web
Services with JESSICA and UML**; IEEE; 2000.
- [SCH01] SCHNEIDER, Jeff. **Convergence of Peer and Web Services**. Disponível na
Internet. <http://www.openp2p.com/lpt/a//2001/07/20/convergence.html> 27. fev. 2002.
- [SNE01] SNELL, James; Tidwell, Doug; Kulchenko, Pavel. **Programming Web
Services with SOAP**. Disponível na Internet.
<http://www.oreilly.com/catalog/progwebsoap/chapter/ch03.html> 27. fev. 2002.
- [SZY01] SZYPERSKI, Clemens. **Components and Web Services**. Disponível na
Internet. <http://www.sdmagazine.com/print/documentID=11651> 26. fev. 2002.
- [SZY02] SZYPERSKI, Clemens. **Services Rendered**. Disponível na Internet.
<http://www.sdmagazine.com/print/documentID=20209> 26. fev. 2002.
- [TRE02] TREESE, Win; **XML, Web services, and UML**; ACM; 2002.
- [ZAE01] ZAEV, Zoran; et all; **Professional XML Web Services**; Wrox; 2001.