UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UTILIZAÇÃO DE GERADORES DE APLICAÇÃO EM PROCESSOS ÁGEIS DE REENGENHARIA

RAQUEL GONÇALVES DE FREITAS

ORIENTADORA: PROFA. DRA. ROSÂNGELA AP. D. PENTEADO

São Carlos/SP

2006

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

F862ug

Freitas, Raquel Gonçalves de.
Utilização de geradores de aplicação em processos ágeis de reengenharia / Raquel Gonçalves de Freitas. -- São Carlos : UFSCar, 2007.
92 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2006.

1. Reengenharia de Software. 2. Gerador (Programas de computador). 3. Geradores de aplicação. 4. *Frameworks*. 5. Reuso. 6. XML (Linguagem de marcação de documento). I. Título.

CDD: 005.1 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Utilização de Geradores de Aplicação em Processos Ágeis de Reengenharia "

RAQUEL GONÇALVES DE FREITAS

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Membros da Banca:

Profa. Dra. Rosângela Ap. Dellosso Penteado
(Orientadora – DC/UFSCar)

Prof. Dr. Antonio Francisco do Prado
(DC/UFSCar)

Profa. Dra. Maria Istela Cagnin Machado
(UNIVEM/Marília)

São Carlos
Dezembro/2006

AGRADECIMENTOS

Á Deus e à Nossa Senhora por essa oportunidade em minha vida e por terem sido meu asilo nos momentos difíceis.

À professora Rosângela A. Delloso Penteado pela orientação, incentivo e paciência, apesar da dedicação não exclusiva ao mestrado pela orientanda. E também, pela amizade e ensinamentos além desta dissertação.

Ao professor Fernão, pelo auxílio no momento crucial da elaboração desta dissertação, demonstrando confiança ao trabalho realizado e transmitindo sua experiência.

Aos meus queridos pais Maria Alice e Orlando, grandes incentivadores desta jornada e essenciais para a realização deste trabalho. É por vocês que sempre sigo em frente, pois me dão todo o apoio necessário com muito amor.

Aos meus irmãos Márcio e Rafael que mesmo de longe, sempre estiveram torcendo por mim.

Aos funcionários e professores do Departamento de Computação que colaboraram pela realização deste trabalho.

À Maria Istela e Anderson Pazin, pela confiança de permitirem a utilização de seus trabalhos e principalmente pelo grande auxílio dado, desde dúvidas pontuais a discussões importantes para a dissertação.

À Márcia Bratfisch pela grande amizade e apoio que se mostraram sempre presente em qualquer situação.

À Rosana Goldbeck pela convivência na execução deste trabalho, não deixando o cansaço e a tristeza abater.

Ao Dirceu Lopes pelo incentivo e amizade, tornando possível a continuidade da vida profissional em conjunto com o mestrado.

Aos amigos Leopoldo, Luis e Henrique pelo apoio e confiança demonstrados em vários momentos que necessitei.

Aos amigos adquiridos ao longo desse período, em especial aos moradores do meu prédio em Campinas.

À todos aqueles que colaboraram de forma direta ou indiretamente à este trabalho.

RESUMO

Métodos ágeis, através de seus princípios de valores, propõem tornar o desenvolvimento de software mais flexível, satisfazendo assim o cliente de modo rápido e personalizado. Qualidade tem papel fundamental tanto em desenvolvimento quanto em manutenção de software. Sabe-se que, para um produto ter qualidade, diretrizes devem ser seguidas ao longo de todo o processo de desenvolvimento bem como de manutenção de software. Reengenharia de software é solução para aqueles sistemas nos quais a manutenção tornou-se difícil ao longo do tempo. Alguns processos de reengenharia usam padrões de software – que fornecem soluções de sucesso para problemas recorrentes – e, também buscam agilidade evitando o modelo de processo em cascata. Geradores de aplicação possibilitam agilidade no desenvolvimento de software, pois ajudam a automação do processo para um certo domínio. Esta dissertação visa usar geradores de aplicação, mais especificamente GAwCRe (Gerador de aplicações para *web* de Clínicas de Reabilitação) junto com o ARA (Arcabouço de Reengenharia Ágil) em vez do *framework* GREN (Gestão de REcursos de Negócio). Por meio de um estudo de caso prospectivo, conduzido com quatro sistemas legados pertencentes ao domínio de clínicas médicas, foi observado que o ARA apóia o uso de geradores de aplicação, apesar de ser necessário fazer uma adaptação para permiti-lo. Isso é verdade apesar do domínio dos sistemas legados não ser o mesmo que o do gerador de aplicações. A adaptação do gerador foi facilitada porque ele é especificado em XML. Como o código por ele gerado é em Java, foi possível usar o ambiente Eclipse para obter os diagramas de classes necessários para futura manutenção. Isso obrigou a adaptação do processo de reengenharia de modo que a produção dos diagramas de classes preconizada para ser feita antes da geração de código passasse a ser realizada após essa geração.

ABSTRACT

Agile methods, through their principles and values, propose to make software development more flexible, thus satisfying the customer in a rapid and customized way. Quality has a fundamental role both in software development and in software maintenance. It is known that, for a product to have quality, guidelines have to be followed along the whole development process as well as software maintenance. Software reengineering is a solution for those systems in which maintenance grew difficult with time. Some reengineering processes use software patterns – which provide success solutions for recurrent problems – and, also seek agility, avoiding the waterfall process model. Application generators provide agility in software development, as they help the process automation for a certain domain. This dissertation aims to use application generators, more specifically GawCRe (Gerador de aplicações para web de Clínicas de Reabilitação – web application generator for rehabilitation clinics) together with ARA (Arcabouço de Reengenharia Ágil – Agile Reengineering Approach), instead of the GREN (Gestão de REcursos de Negócio – Business Resource Management) framework. Through a prospective case study, conducted with four legacy systems belonging to the medical clinics domain, it has been observed that ARA supports that applications generators use, although it is necessary to make an adaptation in order to allow it. This is true even though the legacy systems domain is not the same as that of the application generator. The generator adaptation has been easier because it is specified in XML. As the code that it generates is in Java, it has been possible to use the Eclipse environment though to produce the classes diagrams necessary to future maintenance. This enforced the reengineering process adaptation so that the classes diagrams production commended to be prepared before code generation passed to be produced after that generation.

SUMÁRIO

LISTA DE FIGURAS ..... **V**

LISTA DE TABELAS ..... **VI**

LISTA DE ABREVIATURAS ..... **VIII**

CAPÍTULO 1 - INTRODUÇÃO **1**

1.1 Contexto 1
1.2 Motivação e Objetivo 2
1.3 Organização da dissertação 3

CAPÍTULO 2 – ASSUNTOS RELACIONADOS **4**

2.1 Considerações Iniciais 4
2.2 Padrões, Linguagens e Famílias de Padrões para o Processo de Reengenharia 6
2.3 Frameworks, Geradores de Aplicação e Transformadores 9
2.4 Gerador de Aplicações baseadas na Web para o Domínio de Clínicas de Reabilitação (GAwCRe) 11
2.5 Arcabouço de Reengenharia Ágil (ARA) 14
2.6 Processo Ágil em Framework no domínio de sistemas de Informação com técnicas de VV&T (PARFAIT) 15
2.7 Considerações Finais 18

CAPÍTULO 3 – ESTUDO DE CASO PROSPECTIVO **19**

3.1 Considerações Iniciais 19
3.2 Detalhamento do Estudo de Caso 20
3.3 Considerações Finais 42

CAPÍTULO 4 – PROCESSO ÁGIL DE REENGENHARIA COM GERADORES DE APLICAÇÃO **43**

4.1 Considerações Iniciais 43
4.2 Fases e Atividades 43
4.3 Comparação PARFAIT x Processo com Gerador 47
4.4 Arcabouço de Reengenharia Ágil (ARA) 48
4.5 Considerações Finais 49

CAPÍTULO 5 – CONSIDERAÇÕES FINAIS **50**

5.1 Considerações Iniciais 50
5.2 Contribuições deste Trabalho 51
5.3 Limitações do Trabalho 52
5.4 Trabalhos Futuros 53

REFERÊNCIAS BIBLIOGRÁFICAS **54**

APÊNDICE I – LINGUAGEM DE PADRÕES SIGCLI **59**

APÊNDICE 2 – ESTUDO DE CASO DOS OUTROS SISTEMAS **73**

LISTA DE FIGURAS

Figura 1 – GRN (Braga; Masiero, 2002) 7
Figura 2 – SiGCli (Pazin et. al 2004) 8
Figura 3 - Arquitetura do *Framework* GREN (Braga, 2003) 10
Figura 4 – Regras da Gramática definida para a LMA baseada na SiGCli (Pazin, 2004) 12
Figura 5 – Ordem das tags XML (Pazin, 2004) 13
Figura 6 – Arcabouço ARA (Cagnin, 2005) 14
Figura 7 – Visão geral do PARFAIT (Adaptado de Cagnin, 2003b) 17
Figura 8 – Diagrama de Caso de Uso Cadastrar Pacientes 30
Figura 9 – Caso de Uso Cadastrar Pacientes 30
Figura 10 – Diagrama de Classes do sistema alvo 32
Figura 11 – Cadastro de Pacientes no sistema alvo – versão 0 32
Figura 12 – Cadastro de Pacientes no sistema legado e alvo 33
Figura 13 – Versão 1 com modificação da LMA 39
Figura 15 – Dados da Tabela Pacientes do sistema legado 41
Figura 16 – XML com as tags <atributo> ordenadas 41
Figura 17 – ARA com Gerador 48
Figura 18 – SiGCli (Pazin, 2004) 59
Figura 19 – Diagrama de classes para o padrão Identificar Pacientes (Pazin, 2004) 60
Figura 20 – Diagrama de classes para o padrão Definir Serviços (Pazin, 2004) 62
Figura 21 – Diagrama de classes para o padrão Realizar Vendas (Pazin, 2004) 63
Figura 22 – Diagrama de classes para o padrão Processar Guias (Pazin, 2004) 65
Figura 23 – Diagrama de classes para o padrão Agendar Atendimentos (Pazin, 2004) 66
Figura 24 – Diagrama de classes para o padrão Identificar Atendentes (Pazin, 2004) 67
Figura 25 – Diagrama de classes para o padrão Realizar Acompanhamento (Pazin, 2004) 69
Figura 26 – Diagrama de classes para o padrão Realizar Compras (Pazin, 2004) 70
Figura 27 – Diagrama de classes para o padrão Controlar Faturamento (Pazin, 2004) 71
Figura 28 – Diagrama de casos de uso (ucs atendidos pela SiGCli) 75
Figura 29 – Diagrama de casos de uso (ucs não atendidos pela SiGCli) 75
Figura 30 – Caso de Uso Cadastrar Pacientes 76
Figura 31 – Versão 0 do Sistema Alvo 78
Figura 32 – Versão 1do Sistema Alvo 78
Figura 33 – Diagrama de Classes 79
Figura 34 – Diagrama de Casos de Uso 81
Figura 35 – Casos de Uso 81
Figura 36 – Versão 0 do Sistema Alvo 84
Figura 37 – Versão 1 do Sistema Alvo 85
Figura 38 – Diagrama de Classes do Sistema Alvo 85
Figura 39 – Diagrama de Classes do Sistema Alvo 87
Figura 40 – Caso de Uso 87
Figura 41 – Versão 0 do Sistema Alvo 90
Figura 42 – Versão 1 do Sistema Alvo 91
Figura 43 – Diagrama de Classes do Sistema Alvo 92

LISTA DE TABELAS

Tabela 1 – Análise das Atividades em Relação ao PARFAIT 20
Tabela 2 – XML e Aplicação antes de depois de alterações 21
Tabela 3 – Aplicação antes e depois de alterações 22
Tabela 4 – XML antes e depois de alterações 23
Tabela 5 – `Gerador.java` antes e depois de alterações 24
Tabela 6 – Aplicação `e GeradorCodigo.java` antes e depois de alterações 25
Tabela 7 – Comparação das Funções dos Sistemas Legados 27
Tabela 8 – Comparação do Sistema Psychologist com o Gerador 28
Tabela 9 – Casos de Teste 31
Tabela 10 – Comparação dos atributos Cadastro de Pacientes 33
Tabela 11 – Inclusão de Atributo no Gerador 34
Tabela 12 – Inclusão de Atributo no Sistema Alvo 34
Tabela 13 – Inclusão de relatório no Gerador 36
Tabela 14 – Inclusão de relatório no Sistema Alvo 37
Tabela 15 – Inclusão de Cadastro com dependência no Gerador 38
Tabela 16 – Inclusão de Cadastro com dependência no Atributo no Sistema Alvo 39
Tabela 17 – Informações da Tabela de Pacientes do Sistema Legado e Alvo 40
Tabela 18 – Fase de Concepção 44
Tabela 19 – Fase de Elaboração 45
Tabela 20 – Fase de Construção 45
Tabela 21 – Fase de Transição 46
Tabela 22 – Adaptações do processo PARFAIT pela utilização de geradores 47
Tabela 23 – Padrões da GRN usados para definir o padrão *Identificar Pacientes (Pazin, 2004)* 60
Tabela 24 – Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão *Identificar Pacientes* (Pazin, 2004) 61
Tabela 25 – Padrões da GRN usados para definir o padrão Definir Serviços (Pazin, 2004) 61
Tabela 26 – Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão *Definir Serviços (Pazin, 2004)* 62
Tabela 27 – Padrões da GRN usados para definir o padrão *Realizar Vendas (Pazin, 2004)* 63
Tabela 28 – Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão Realizar Vendas 64
Tabela 29 – Padrões da GRN usados para definir o padrão Processar Guias (Pazin, 2004) 64
Tabela 30 – Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão Processar Guias (Pazin, 2004) 65
Tabela 31 – Padrões da GRN usados para definir o padrão Agendar Atendimentos (Pazin, 2004) 66
Tabela 32 – Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão Agendar Atendimento (Pazin, 2004) 67
Tabela 33 – Padrões da GRN usados para definir o padrão Identificar Atendentes (Pazin, 2004) 67
Tabela 34 – Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão Identificar Atendentes (Pazin, 2004) 68
Tabela 35 – Padrões da GRN usados para definir o padrão Realizar Compras (Pazin, 2004) 70
Tabela 36 – Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão Realizar Compras (Pazin, 2004) 70

Tabela 37 – Padrões da GRN usados para definir o padrão Controlar Faturamento (Pazin, 2004) 71
Tabela 38 – Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão Controlar Faturamento 72
Tabela 39 – Funções Legado x Padrões SiGCli 74
Tabela 40 – Casos de Teste 76
Tabela 41 – Funções Legado x Padrões SiGCli 80
Tabela 42 – Casos de Teste 82
Tabela 43 – Funções Legado x Padrões SiGCli 86
Tabela 44 – Casos de Teste 88

LISTA DE ABREVIATURAS

AM	*Agile Modeling*
CMM	*Capability Maturity Model*
CMMI	*Capability Maturity Model Integrated*
FaPRE/OO	Família de Padrões de Reengenharia
FDD	*Feature Driven Development*
GawCRe	Gerador de Aplicações baseadas na Web para sistemas de gestão de Clínicas de Reabilitação
GREN	*Framework* para Gestão de Recursos de Negócios
GRN	Linguagem de Padrões para Gestão de Recursos de Negócios
IEEE	*Institute of Electrical and Electronics Engineering*
IPPD	*Integrated Product and Process Development*
ISO	*International Organization of Standardization*
KPA	*Key Process Areas*
LMA	Linguagem de Modelagem de Aplicação
MA	Modelagem Ágil
PA	Process Area
PRE/OO	Processo de Reengenharia Orientada a Objetos
RUP	*Rational Unified Process*
SA-CMM	*Software Acquisition*
SE-CMM	*System Engineering*
SEI	*Software Engineering Institute*
SiGCli	Linguagem de Padrões para Sistemas de Gerenciamento de Clínicas de Reabilitação

SPICE	*Software Process Improvement and Capability dEtermination*
SW-CMM	Software Engineering
UML	Unified Modeling Language
XP	eXtreme Programming
XML	EXtensible Markup Language

CAPÍTULO 1 - INTRODUÇÃO

1.1 Contexto

Mesmo com a evolução dos computadores, das técnicas, tecnologias e ferramentas que apóiam o desenvolvimento de software, nos últimos anos, a produção de software confiável, correto e entregue dentro dos prazos e custos estipulados ainda é muito difícil. Sendo assim, surgiram algumas importantes iniciativas para melhor abordar os problemas relacionados ao desenvolvimento de software. Dentre essas, podem-se citar os métodos ágeis, com a proposta de aumentar o enfoque nas pessoas e não nos processos de desenvolvimento, e, além disso, produzir apenas documentação essencial e dar mais atenção à resolução de problemas de forma iterativa. A meta do desenvolvimento ágil, de acordo com Abrahamsson et. al. (2003), é aumentar a habilidade de reagir e responder às necessidades mutáveis da tecnologia, dos clientes e dos negócios em todos os níveis organizacionais.

A manutenção é um dos principais problemas das organizações de desenvolvimento de software. A dificuldade está na manutenção dos chamados "softwares legados" que já estão ultrapassados quanto à tecnologia atual e possuem pouca ou nenhuma documentação, porém têm regras de negócio úteis que não foram documentadas em todas as alterações realizadas. Uma das formas de solucionar esse problema é aplicar processos de reengenharia para recuperar as informações intrínsecas ao código, para que um novo sistema seja desenvolvido com tecnologia atual e qualidade, porém preservando a funcionalidade já existente de acordo com as necessidades do cliente. Na aplicação da reengenharia há a possibilidade de inserção de algumas novas funções dependendo da necessidade do cliente, bem como a retirada daquelas funções que não mais interessam ao cliente.

Os processos clássicos de reengenharia adotam o desenvolvimento em cascata, tornando o processo longo, de alto custo, com pouca participação do cliente e o produto liberado somente ao final do processo. Com o surgimento da abordagem ágil, novos processos de reengenharia estão sendo criados, um deles é o PARFAIT (Cagnin, 2005), adotando técnicas ágeis para que a implementação do sistema advindo de um sistema legado seja desenvolvida mais rapidamente e com maior adaptabilidade de acordo com as necessidades do cliente durante o processo. O PARFAIT apóia a migração de sistemas procedimentais para sistemas orientados a objetos com base em linguagem de padrões e *framework.* Produz uma versão inicial do sistema alvo o mais rápido possível que evolui até atingir a versão final.

Nessa mesma perspectiva de agilidade, geradores de aplicação surgiram para facilitar a criação de aplicações a partir da sua especificação em uma linguagem de alto nível, com a limitação de sistema para um determinado domínio (Smaragdakis; Batory, 1998). Com eles, sistemas alvo podem ser gerados a partir de sistemas legados desde que pertençam ao mesmo domínio, realizando a reengenharia de forma automatizada. Um gerador sob a ótica de *frameworks, é como se* as partes fixas do produto gerado correspondessem aos *frozen-spots e* as partes variáveis, que devem ser adaptadas correspondessem aos *hot-spots*.

O GAwCRe – Gerador de Aplicações baseadas na *web* para Clínicas de Reabilitação, é um exemplo de gerador de aplicação, criado por Pazin (2004), que abrange o domínio de clínicas de reabilitação física. Esse domínio é representado pela linguagem de padrões SiGCli (Pazin, 2004) e, por meio de uma Linguagem de Modelagem de Aplicação (LMA) (Weiss; Lai, 1999) suas variabilidades são documentadas em XML, possibilitando ao gerador ler suas definições e gerar uma aplicação com base nessas informações.

1.2 Motivação e Objetivo

A abordagem ágil tem crescente interesse tanto no meio acadêmico quanto no empresarial. Ela não é uma "bala de prata" para resolver todos os problemas da engenharia de software, mas tenta amenizá-los, principalmente quanto a custos, prazos de entrega e satisfação do cliente (Abrahamsson et. al.,2003).

As empresas precisam manter as regras de negócios que estão embutidas em sistemas legados, que passaram por diversas manutenções, mas devem adaptá-los às novas tecnologias. O processo de reengenharia com técnicas ágeis de desenvolvimento tem se mostrado como uma boa solução para esses casos. Assim, um sistema legado pode utilizar apenas nova tecnologia e/ou incorporar novos requisitos aos existentes.

O arcabouço de reengenharia ágil ARA (Cagnin, 2005), possibilita a migração de sistemas legados procedimentais para sistemas alvo orientados a objetos, utilizando a abordagem iterativa e incremental, além de técnicas de reuso, práticas ágeis e ferramentas, com o intuito de obter um sistema alvo de qualidade o mais rápido possível. Para apoiar o processo de reengenharia utiliza-se *framewoks,* que no estudo realizado por Cagnin teve-se como exemplo o *framework* GREN (Braga, 2003), desenvolvido com base na linguagem de padrões GRN (Braga; Masiero, 2002) e implementado na linguagem orientada a objetos Smalltalk (Shafer, 1991). Algumas características do *framework* são passadas para os sistemas obtidos com a sua instanciação. Uma delas é a linguagem de implementação. Assim, os

sistemas alvo são produzidos em Smalltalk, que não é uma linguagem muito difundida no meio empresarial. Ao contrário do gerador GAwCRe (Pazin, 2004), que também apóia o processo de reengenharia, mas gera sistemas para *web* na linguagem Java.

Este trabalho tem como objetivo utilizar o GAwCRe (Pazin, 2004) do domínio de clínicas de reabilitação física no arcabouço de reengenharia ágil ARA (Cagnin 2005), definindo um processo ágil de reengenharia com geradores de aplicação, adaptado do Processo Ágil de Reengenharia Apoiado por *Framework* (PARFAIT). Para tal, um estudo de caso prospectivo foi realizado utilizando quatro sistemas encontrados na *Internet*, do domínio de clínicas médicas, sendo esse um pouco mais abrangente que o domínio do GAwCRe.

O estudo foi baseado nas atividades do PARFAIT, sendo que, nem todas as atividades foram utilizadas, com atenção voltada para verificar o comportamento do gerador durante a aplicação do arcabouço ARA. A reengenharia é realizada de forma incremental e iterativa.

Algumas funções não puderam ser atendidas pelo gerador, por serem pertencentes a um domínio mais abrangente, de clínicas médicas. Verifica-se também que, algumas funções presentes nesses legados são inerentes a outros subdomínios, como por exemplo, Segurança de Dados, e que não são tratados pela SiGCLi e, nem pelo gerador. As funções atendidas por essa linguagem tiveram alterações para se adaptarem aos requisitos do sistema legado. Essas alterações puderam ser realizadas modificando somente a especificação do gerador em XML.

Nessa etapa verificou-se a flexibilidade do gerador por estar implementado em XML (XML, 2006), que permitiu agilidade no tratamento das mudanças. Além disso, o XML facilita a legibilidade da LMA, não sendo necessário muito conhecimento prévio do desenvolvedor. Dessa forma, optou-se pela reengenharia do gerador ao longo do processo, para que o sistema alvo fosse mais facilmente adaptado ao sistema legado.

1.3 Organização da dissertação

Esta dissertação está organizada em cinco capítulos, sendo que neste primeiro, foi apresentado o contexto em que este trabalho se enquadra, com uma breve explicação dos motivos que levaram à sua elaboração. O Capítulo 2 trata dos assuntos relacionados ao projeto, principalmente o ARA (Cagnin, 2005) e o GAwCRe (Pazin, 2004). O Capítulo 3 mostra o estudo de caso realizado, para exemplificar, analisar e avaliar o processo ARA utilizando geradores de aplicação. O Capítulo 4 mostra o processo ágil de reengenharia com geradores de aplicação. O Capítulo 5 aborda as considerações finais comentando os resultados obtidos e os trabalhos futuros propostos.

CAPÍTULO 2 – Assuntos Relacionados

2.1 Considerações Iniciais

Um software, com o passar do tempo, apesar de bom funcionamento, pode não atender mais totalmente aos requisitos da empresa, estar ultrapassado à tecnologia atual, ou estar com o código "bagunçado" devido às manutenções. Assim, a reengenharia torna-se um dos meios para que, esse sistema que ainda possui a base do negócio, chamado de sistema legado, seja recuperado. Uma boa prática na reengenharia é a utilização de padrões de software.

Padrões auxiliam o desenvolvimento de software com soluções de sucesso, que quando agrupados em uma seqüência temporal de decisões, transformam-se em linguagem de padrões, formando um guia para o processo de desenvolvimento. Assim, a linguagem de padrões auxilia na criação de novas aplicações em um domínio específico, que pode ser utilizada como base para a construção de *frameworks* (Johnson; Foote, 1998) e geradores (Smaragdakis; Batory, 1998). *Frameworks* e Geradores de aplicação conseguem automatizar grande parte da atividade de desenvolvimento de software, ficam em um nível menos abstrato e mais completo que a linguagem de padrões, permitindo a reutilização de códigos executáveis existentes.

O reuso de software aproveita artefatos no desenvolvimento de software, incluindo componentes de software, ambiente de teste, projeto e documentação que tem como principais objetivos: redução de custo e risco do projeto, e também aumento da qualidade. O desenvolvimento baseado em linhas de produtos de software é considerado uma estratégia para reutilização de software em grande escala. Uma linha de produtos consiste numa família de produtos pertencentes a determinado domínio, que tem como base uma arquitetura em comum. Sendo assim, a parte em comum de uma família de aplicações é reutilizada cada vez que uma nova aplicação é necessária, e, no caso de uma aplicação específica, esta é especializada. O GAwCRe (Pazin, 2004) abrange a linguagem de padrões SiGCli (Pazin et. al, 2004), definida a partir de uma família de produtos e que reutiliza padrões da linguagem de padrões GRN. Para controle da parte em comum e variabilidades da SiGCli, uma Linguagem de Modelagem de Aplicação (LMA) foi definida, que são documentadas em XML, utilizado para gerar artefatos da aplicação.

Apesar de existirem muitos pontos positivos na aplicação da reengenharia, ela ainda é discutida com certa cautela, pois muitos questionam seu custo, sua duração e a utilização da

reengenharia tradicional. Nesse contexto, Cagnin (2005) propõe o arcabouço de reengenharia ágil ARA, que utiliza conceitos de reuso e agilidade, de maneira incremental e iterativa, migrando um sistema legado para um sistema alvo com qualidade e em um curto espaço de tempo.

O PARFAIT (Cagnin, 2005) é um processo que foi criado para viabilizar o arcabouço ARA, é baseado em *frameworks*, cuja construção está baseada em linguagem de padrões, com atividades de VV&T associadas, proporcionando versões do sistema ao longo de suas iterações, para que os usuários possam acompanhar e avaliar o andamento do projeto, possibilitando alterações e inclusão de novos requisitos, em qualquer momento.

O desenvolvimento em cascata tem diversos problemas provenientes de sua seqüência de fases no processo, em contrapartida, o paradigma incremental e iterativo é bem aceito e vem sendo utilizado por várias metodologias de desenvolvimento de software. Juntamente com essa tendência, estão presentes também, as técnicas de reuso em várias etapas do desenvolvimento e as práticas ágeis que, procuram principalmente promover uma maior interação com o usuário para garantir o sucesso final do projeto.

Para apoiar a tendência de evolução de sistemas legados diversos métodos de reengenharia têm sido propostos, porém, poucos possuem apoio computacional efetivo, e nenhum utiliza *framework* baseado em linguagem de padrões. Sendo assim, Cagnin (2005) propõe a elaboração de um arcabouço de reengenharia ágil baseado em *framework*, que realiza a engenharia reversa do sistema legado com o apoio de linguagem de padrões de análise, fornecendo entendimento e documentação necessários para instanciar o *framework*. Para permitir a reengenharia com o apoio do arcabouço definido, um processo ágil de reengenharia chamado PARFAIT (Cagnin, 2005) também foi criado.

Este capítulo está organizado da seguinte maneira: na Seção 2.2 padrões, linguagens e famílias de padrões de Reengenharia são apresentados e exemplificados; *Frameworks,* Geradores de Aplicação e Transformadores que podem ser utilizados para auxiliar a reengenharia são apresentados na Seção 2.3. O GAwCRe (Pazin, 2004) é mostrado na Seção 2.4; a Seção 2.5 apresenta o arcabouço ARA e, em seguida, a Seção 2.6 apresenta o processo de reengenharia ágil denominado PARFAIT e as considerações finais estão na Seção 2.7.

2.2 Padrões, Linguagens e Famílias de Padrões para o Processo de Reengenharia

O reuso de software existente tem sido considerado como uma solução possível para os problemas de melhoria de qualidade e de produtividade de desenvolvimento de software (Canfora et. al, 1995). Chikofsky e Cross (1990), definem reengenharia como sendo a reconstrução de sistemas, tendo como propósito a busca por melhorias que permitam produzir algo de qualidade melhor ou comparável ao produto inicial. O entendimento do sistema é a maior dificuldade da reengenharia, pois normalmente a documentação necessária sobre ele não está disponível ou está completamente desatualizada. Além disso, seus desenvolvedores geralmente não estão mais disponíveis para mantê-lo ou comentar as modificações que já realizaram.

O reuso pode ser conseguido por meio de padrões de software, que descrevem soluções de sucesso para os problemas que ocorrem freqüentemente no desenvolvimento de software, objetivando recuperar e documentar a experiência adquirida para que essa seja passada aos menos experientes (Fayad et. al, 1999). Gamma et. al (1995) foram os introdutores dos padrões de projeto para auxiliarem os projetistas por meio de reuso; outros autores também apresentam padrões para atender às mais diferentes etapas de desenvolvimento.

Nenhum padrão é uma entidade isolada, cada padrão é apoiado por outros padrões (Alexander, 1979). Uma coleção estruturada e organizada de padrões que se apóiam entre si para transformar requisitos e restrições numa arquitetura forma uma linguagem de padrões (Coplien, 1998). Uma linguagem de padrões representa a seqüência de decisões que levam ao projeto completo de uma aplicação, tornando-se um método para guiar o processo de desenvolvimento (Brugali et al., 2000). As linguagens de padrões podem ser encontradas em diferentes etapas do desenvolvimento, porém neste trabalho somente as de reengenharia serão comentadas.

Demeyer et al. (2000) apresentam uma linguagem de padrões para auxiliar no processo de engenharia reversa de sistemas orientados a objetos, porém suas idéias podem ser adaptadas para permitir a engenharia reversa de sistemas legados procedimentais, conforme proposto na Família de Padrões de Reengenharia - FaPRE/OO (Família de Padrões para a Reengenharia Orientada a Objetos) (Recchia, 2002). A FaPRE/OO é utilizada especialmente para migrar sistemas legados implementados em Clipper para sistemas orientados a objetos implementados em Delphi (Recchia et. al, 2002 e Recchia, 2002).

Uma linguagem de padrões para Gestão de Recursos de Negócios, chamada **GRN** (Braga; Masiero, 2002) é composta por quinze padrões, agrupados de acordo com o seu propósito. Na Figura 1 pode-se observar os relacionamentos e dependências entre os padrões existentes na GRN. As linhas mais espessas indicam os principais padrões da linguagem: LOCAR O RECURSO, COMERCIALIZAR O RECURSO E MANTER RECURSO. Uma linguagem de padrões permite o uso conjunto ou separado de cada um dos padrões que a compõem.

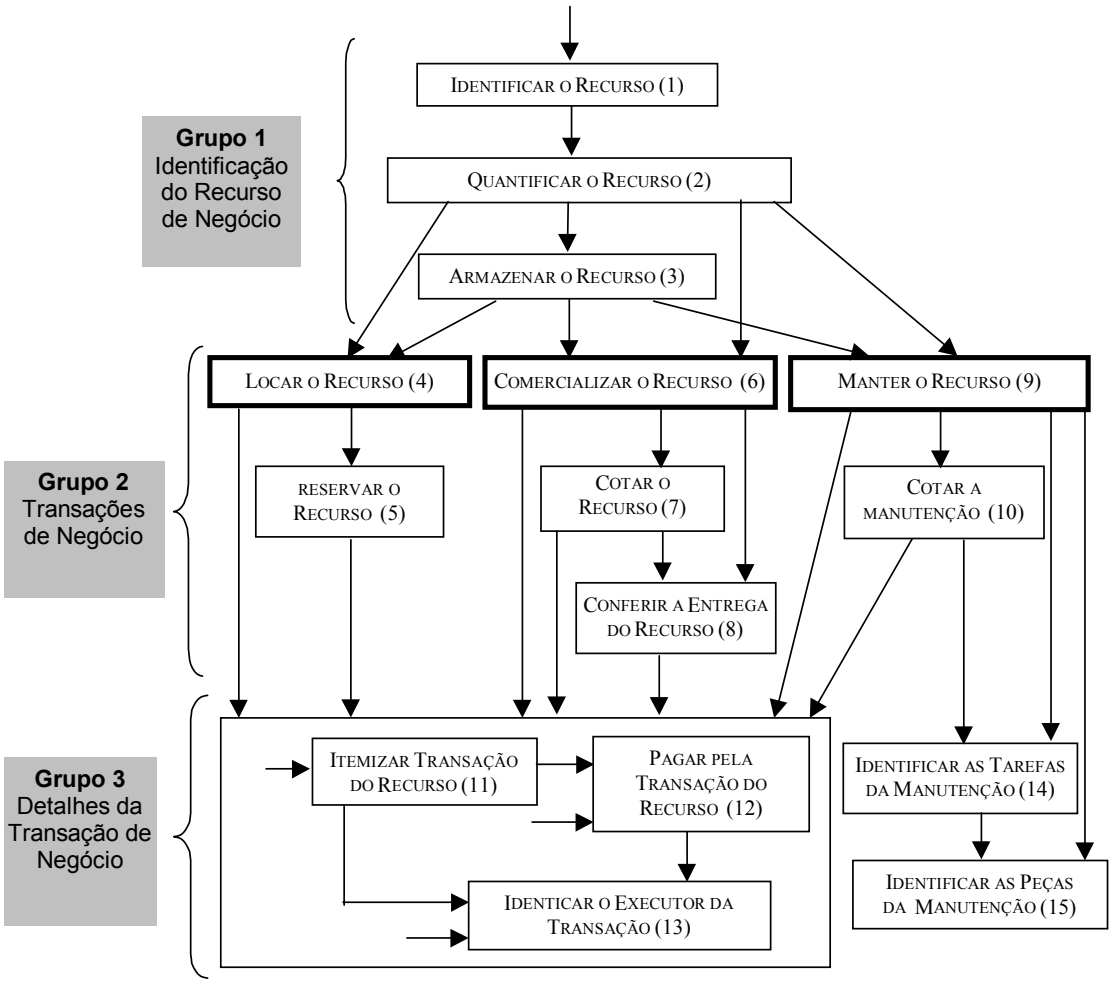

Figura 1 – GRN (Braga; Masiero, 2002)

O Grupo 1 é composto por três padrões, que tratam da identificação e possível qualificação, quantificação e armazenagem dos recursos gerenciados pelo negócio, no Grupo 2 estão os padrões relacionados à manipulação dos recursos de negócio pelo sistema e o Grupo 3 possui cinco padrões que cuidam de detalhes das transações efetuadas com o recurso. No terceiro grupo, os padrões ITEMIZAR TRANSAÇÃO DO RECURSO, PAGAR PELA TRANSAÇÃO DO RECURSO e IDENTIFICAR O EXECUTOR DA TRANSAÇÃO são

aplicáveis a quaisquer transações do grupo 2 e os padrões IDENTIFICAR AS TAREFAS DA MANUTENÇÃO e IDENTIFICAR AS PEÇAS DA MANUTENÇÃO são aplicáveis às transações contidas nos padrões MANTER O RECURSO e COTAR A MANUTENÇÃO (Braga; Masiero, 2002).

Pazin et. al (2004), procuraram, sempre que possível, reusar ou adaptar os padrões da GRN (Braga; Masiero, 2002) na elaboração dos padrões da **SiGCli**, apresentada a seguir.

A linguagem de padrões SiGCli (Pazin et. al 2004) foi elaborada para o domínio de sistemas de gestão de clínicas de reabilitação, que incluem clínica de fisioterapia, terapia ocupacional e educação física. A SiGCli possui um conjunto de padrões que engloba as principais funções que devem ser realizadas por uma clínica, considerando vários pontos, como o controle de guias de convênios médicos, a comercialização de serviços e de produtos, o controle de faturamento e o acompanhamento de tratamento de um paciente. Os padrões da SiGCli tiveram por base os padrões da GRN, sendo que em alguns foi mantida a estrutura original. Na Figura 2 são mostrados os nove padrões que constituem a SiGCli, organizados em três grupos de acordo com suas funções.

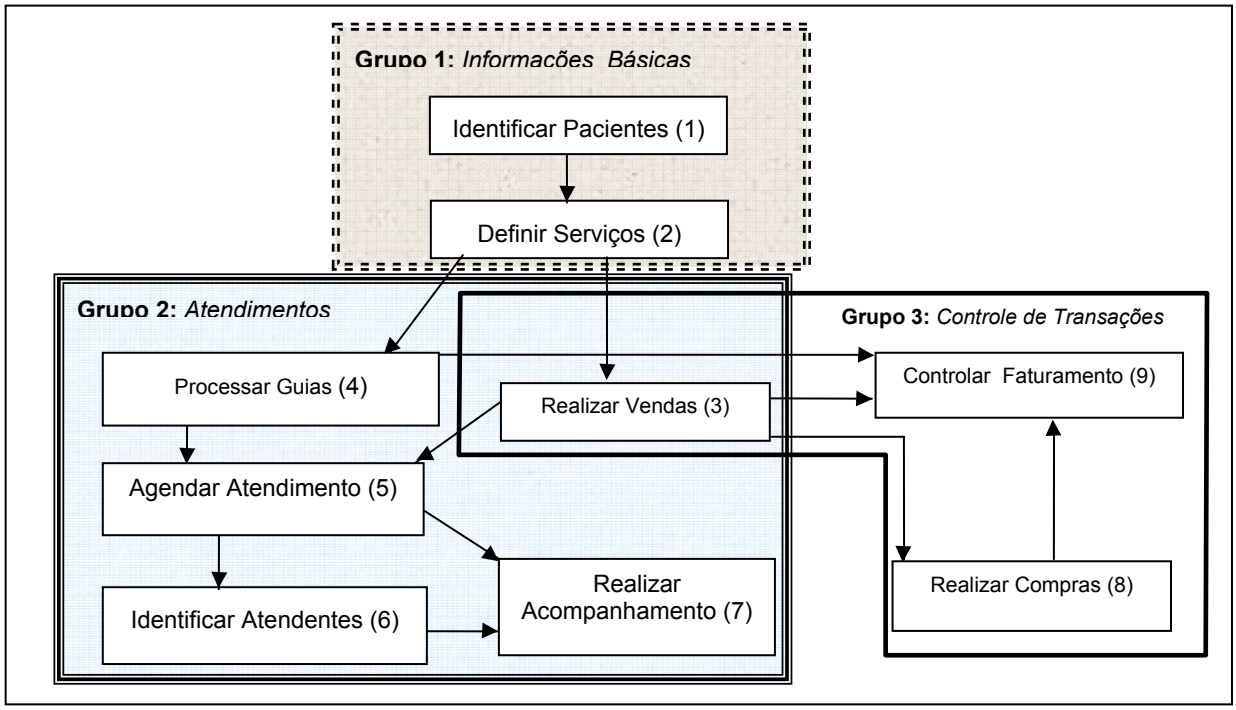

Figura 2 – SiGCli (Pazin et. al 2004)

Os padrões do Grupo 1 devem ser aplicados obrigatoriamente em todos os sistemas desse domínio e os demais padrões são opcionais. Este grupo trata das informações básicas relacionadas às clínicas, a identificação dos pacientes e dos serviços prestados. O Grupo 2

cuida do gerenciamento dos atendimentos e o Grupo 3 trata do controle financeiro das clínicas.

Na Figura 2, os padrões Identificar Atendentes (1) e Definir Serviços (2) são aplicados obrigatoriamente em todos os sistema desse domínio. Para que o padrão Agendar Atendimentos (5), o padrão Realizar Vendas(3) ou Processar Guias(4) deve ser realizado. O padrão (5) leva aos Padrões Identificar Atendentes (6) e/ou Realizar Acompanhamento (7). Os padrões (4) ou (3) pode acarretar o padrão Controlar Faturamento (9). E o padrão Realizar Compras (8) é aplicado a partir do padrão Realizar Vendas (3).

Embora o grafo da Figura 2 defina uma ordem que parece ser a mais adequada para a aplicação dos padrões, eles podem ser usados independentemente ou em uma ordem diferente da definida pelo grafo (Pazin et. al 2004). Os padrões que compõem a SiGCli estão no Apêndice 1.

2.3 Frameworks, Geradores de Aplicação e Transformadores

De acordo com Johnson, R. E.; Foote, B. (1998) um *framework* é uma aplicação semi-completa reutilizável que, quando especializada, produz aplicações personalizadas. Pode ser ainda definido como sendo um conjunto de blocos de software pré-fabricados que podem ser utilizados, ou, estentidos e adaptados para soluções computacionais específicas (Taligent, 1993). Os geradores de aplicação são ferramentas de software que conseguem automatizar o processo de implementação no desenvolvimento de software, para um mesmo domínio, transformando especificações de alto nível em aplicações.

Existem muitas semelhanças entre um gerador de aplicações e um *framework*. Analisando um gerador sob a ótica de *frameworks*, as partes fixas do produto gerado correspondem aos *frozen-spots*, já as partes variáveis, que devem ser adaptadas correspondem aos *hot-spots*. A instanciação de um *framework*, no contexto do gerador corresponde aos mecanismos de geração (Franca; Staa, 2001). Já os sistemas transformadores reestruturam um primeiro sistema em um segundo sistema aplicando um conjunto de transformações, preservando as semânticas.

Com base na Linguagem de Padrões GRN foi construído o *framework* **GREN** (Braga, 2003) que permite criar aplicações no domínio de sistemas de gestão de recursos de negócios e foi desenvolvido em Smalltalk. A arquitetura do *framework* GREN é mostrada na Figura 3.

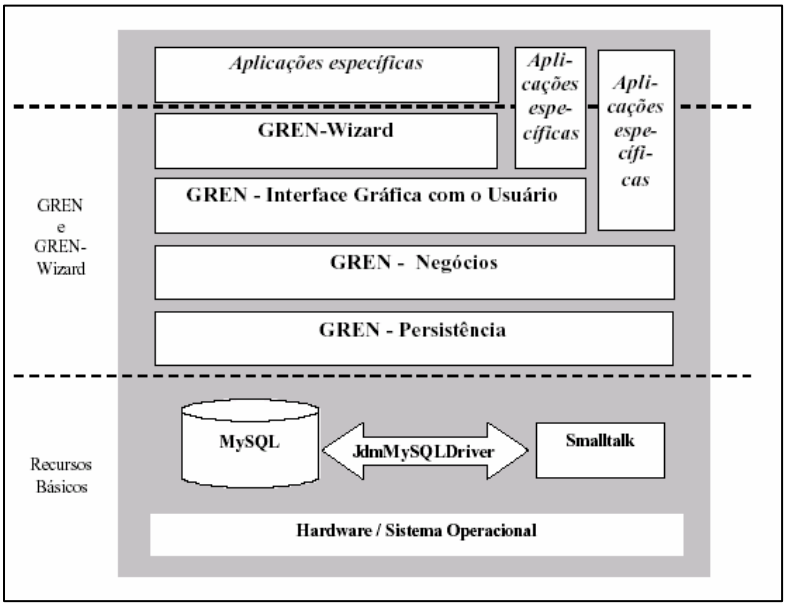

Figura 3 - Arquitetura do *Framework* GREN (Braga, 2003)

As instanciações das aplicações no GREN são guiadas pela aplicação da Linguagem de Padrões GRN, que gera diagramas de classes da aplicação desejada. Com base nesses diagramas utilizam-se classes pré-programadas do *Framework* GREN para criar o código da nova aplicação.

Como exemplo de sistemas transformacionais, tem-se o **Draco-PUC** (Prado, 1992). É um sistema transformador genérico baseado nas idéias do paradigma Draco (Neighbors, 1984) que vem sendo utilizado na Reengenharia de Software. Possui ainda, um código portável e que pode operar em diferentes versões de sistemas operacionais, como Unix, Windows 9x, NT, 2000 e XP.

Sistemas transformacionais podem trabalhar com apenas um conjunto restrito de linguagens de descrição de artefatos e de operadores, identificados como transformadores específicos. Nesta categoria enquadram-se os compiladores, já que não podem ser configurados para trabalhar com diversas linguagens e também possuem um conjunto pré-configurado e restrito de operadores de transformação. Ou então, podem permitir ampla configuração para diversas situações de uso, chamados de transformadores genéricos.

As transformações realizadas pelo Draco-PUC são orientadas a domínio, assim, parte-se de descrições no domínio do código legado para se obter descrições no domínio alvo, permitindo recuperar o projeto do sistema legado e gerar sua implementação em uma nova linguagem. Um domínio é definido por uma Linguagem e esta por sua vez é formada por:

uma gramática, que especifica as regras de produção dessa linguagem, um *parser*, ou interpretador, que associa ações semânticas às regras de produção da linguagem, e *prettyprinter*, ou *unparser*, responsável pela geração do código.

Com a estratégia proposta por Prado (Prado, 1992), é possível a reconstrução de um software pela "transformação" direta do código fonte em uma linguagem para linguagens de outros domínios. Para efetuar essa "transformação" é necessário construir os domínios das linguagens de origem e alvo do sistema, e a automatização ou semi-automatização, realizada por meio das regras de transformação.

2.4 Gerador de Aplicações baseadas na Web para o Domínio de Clínicas de Reabilitação (GAwCRe)

Geradores de aplicação traduzem especificações em programas. Tais especificações descrevem o problema a ser resolvido pelo programa. Nesse sentido, um gerador de aplicação opera de forma similar a um compilador de linguagem, que traduz informações de alto nível em implementações de baixo nível (Cleaveland, 1988).

O Gerador de Aplicações baseadas na *Web* para sistemas de gestão de Clínicas de Reabilitação, denominado GAwCRe (Pazin, 2004), facilita instanciações de aplicações no domínio coberto pela linguagem de padrões SiGCli (Pazin et. al, 2004), definida a partir de uma família de produtos de software de Sistemas de Gestão de Clínicas de Reabilitação, como já apresentado na Seção anterior.

A Linguagem de Modelagem de Aplicação (LMA, originalmente AML – *Application Modeling Language*) contém especificações para garantir certas propriedades importantes ao domínio (Weiss; Lai, 1999). Uma LMA foi elaborada com base na SiGCli, para possibilitar a definição de aplicações a partir de especificações em alto nível no GAwCRe (Pazin, 2004). Na LMA são representadas as possíveis variações de clínicas de reabilitação, que são documentadas em XML, utilizado para gerar artefatos da aplicação.

O gerador GAwCRe (Pazin, 2005) foi construído baseado em uma LMA, onde estão representadas as variabilidades presentes nos padrões, sendo possível gerar um conjunto distinto de aplicações, considerando as diferentes combinações permitidas. Um fator importante é a possibilidade de reusar a estrutura do gerador para outros domínios, bastando definir uma LMA e mapeando essa linguagem para o documento XML (Pazin et. al, 2004).

Na Figura 4 é exibida a gramática definida para a LMA, sendo que para instanciar uma aplicação basta optar pelas regras definidas na gramática.

Para o armazenamento das informações referentes à linguagem de padrões e à LMA um meta-modelo em XML (XML,2004) foi elaborado por Pazin (2004). Abaixo é descrito o papel de cada *tag*:

```
LMA ::=  <lista_padrões>
<lista_padrões> ::=  <padrão> ; <lista_padrões>| λ
<padrão> ::= Identificar_Paciente | <Definir_Serviço> | <Realizar_Vendas> |
            Processar_Guias |Agendar_Atendimentos | <Identificar_Atendente>
            |<Realizar_Acompanhamento>| Realizar_Compra |Controlar_Faturamento
<Definir_Serviço> ::= Definir_Serviço ; <Com_Tipo_Serviço>
<Com_Tipo_Servico> ::= Com_Tipo_Serviço | λ
<Realizar_Venda> ::= Realizar_Venda ; <Com_Produto>
<Com_Produto> ::= Com_Produto | λ
<Identificar_Atendente>::=Identificar_Atendente ; <Com_Atributo_Atendentes>
<Com_Atributos_Atendentes>::= especialidade ; salário | especialidade |
salário|λ
<Realizar_Acompanhamento> ::= Realizar_Acompanhamento ; <lista_avaliações>
<lista_avaliações> ::= <avaliações> ; <lista_avaliações> | λ
<avaliações> ::= <avaliações_fisio> | <avaliações_to> | <avaliações_ef>
<avaliações_fisio> ::= <tipo_avaliação_fisio> ; <avaliações_fisio>
<tipo_avaliação_fisio> ::= Neurológica_Infantil | Neurológica_Adulto
|Ortopedica | Pneumologica | Cardiológica | Diária |  λ
<avaliações_to> ::= <tipo_avaliação_to> ; <avaliações_to>
<tipo_avaliação_to> ::= Anamnese_Infantil | Anamnese_Adulto|
Neurológica_Infantil|Psicomotora | TDE |Física_Infantil | Física_Adulto |
Portage | Diária  | λ
<avaliações_ef> ::= <tipo_avaliação_ef> ; <avaliações_ef>
<tipo_avaliação_ef> ::= Anamnese | Antropométrica | Física
|Controle_Diário_Atividades | Ficha_Musculação | λ
```

Figura 4 – Regras da Gramática definida para a LMA baseada na SiGCli (Pazin, 2004)

- Tag <linguagem padrão> ... </linguagem padrão>: usada para definir qual o nome da linguagem de padrões usado pelo gerador
- Tag <padrão>...</padrão>: usada para definir os padrões da linguagem
- Tag <classe>...</classe>: usada para definir as classes de um padrão, nela devem ser especificados os atributos, métodos e associações entre classes.
- Tag <atributo>...</atributo>: usada para definir os atributos de uma classe.
- Tag <valor>...</valor>: usada quando a tag <atributo> possui o atributo de tag tipoCampo definido como select ou radio. Nesse caso cria-se uma lista com valores pré-definidos.
- Tag <associacao>...</associacao>: usada para definir as associações entre as classes dos padrões
- Tag <metodo>...<metodo>: usada para definir os métodos de cada classe.
- Tag <parametro>...</parametro>: usada para definir os parâmetros usados pelos métodos.

- Tag <valor>...</valor>: usada quando a tag <parâmetro> possui o atributo de tag tipoCampo definido como select ou radio. Nesse caso cria-se uma lista de valores pré-definidos.
- Tag <corpoMetodo>...<corpoMetodo>: usada para definir o corpo do método.
- Tag <variante>...<variante>: usada para definir a variante de um padrão.

Na Figura 5 são apresentadas as regras de criação de um documento XML, seguindo a ordem de uso das *tags* para que o gerador funcione. Cada padrão da linguagem de padrões possui uma ou mais classes, podendo possuir nenhuma ou muitas variantes. As variantes também podem possuir nenhuma ou muitas classes. As classes são constituídas de atributos, associações e métodos. Cada método deve possuir um corpo e nenhum ou muitos parâmetros.

```
<linguagemPadrao>
        <padrao>
         <classe>
            <atributo> </atributo> ...
             <associacao> </associacao> ...
                       <metodo>
                         <parametro>
                         <corpoMetodo> <corpoMetodo>
                   </metodo>
       </classe>
            <variante>
                  <classe>... </classe>
            </variante>
      </padrao>
<linguagemPadrao>
```

Figura 5 – Ordem das tags XML (Pazin, 2004)

Tudo o que é gerado por um gerador, desde os *scripts* de criação do banco de dados até a aplicação final são considerados artefatos. O GAwCRe utiliza gabaritos (*templates*) na geração desses artefatos, os gabaritos possuem partes fixas, que estarão presentes em todos os artefatos gerados e partes variáveis que possibilitam a geração de diferentes artefatos. Para a geração dos artefatos, as partes fixas são previamente definidas no código fonte do gerador e as partes variáveis são substituídas pelas informações existentes no documento XML.

Na Seção seguinte, o arcabouço ARA utilizado neste trabalho, juntamente com o processo ágil de reengenharia PARFAIT é apresentado.

2.5 Arcabouço de Reengenharia Ágil (ARA)

O arcabouço de reengenharia ágil, denominado ARA (Cagnin, 2005), auxilia na transformação de um sistema procedimental para um sistema orientado a objetos. A reengenharia é realizada conforme ilustrado na Figura 6; utiliza abordagem iterativa e incremental, juntamente com mecanismos como: práticas ágeis, processos, ferramentas, reengenharia guiada por testes; e, também, ao longo da reengenharia os usuários do sistema legado auxiliam no processo.

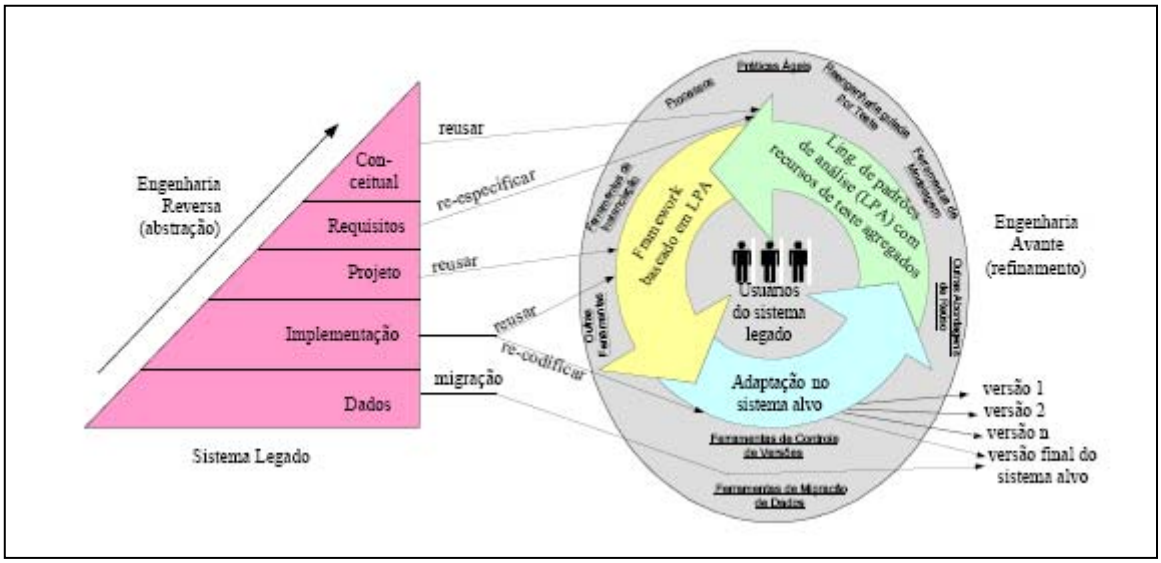

Figura 6 – Arcabouço ARA (Cagnin, 2005)

De acordo com Cagnin (2005), a utilização de linguagem de padrões de análise, juntamente com recursos de teste funcional auxiliam o entendimento do sistema legado. A execução desse, é baseada na reutilização de casos de teste construídos a partir da linguagem de padrões de análise de mesmo domínio, contribuindo de forma efetiva para o entendimento e análise do legado.

O projeto e a implementação do sistema são reutilizados, em grande parte, do *framework*, que é baseado na mesma linguagem de padrão de análise utilizada. As funções que o *framework* não engloba, sendo, específicas do legado ou identificadas posteriormente pelos usuários, são implementadas diretamente no sistema alvo. A aplicação sistema legado se mantém separada durante todo o processo do sistema alvo, que é uma instanciação do *framework* (Cagnin, 2005).

A abordagem incremental e iterativa no ARA (Cagnin, 2005), permite que os usuários estabeleçam a prioridade dos requisitos a serem submetidos à reengenharia, assim como, suas validações sejam realizadas em versões parciais do sistema. Essa validação tem como objetivo verificar se as funções implementadas estão equivalentes à do sistema legado e para isso, são reutilizados os mesmos casos de teste utilizados anteriormente, durante a engenharia reversa.

O arcabouço ARA (Cagnin, 2005) possui também controle de versão de todos os artefatos produzidos, facilitando assim a mudança de requisitos ao longo do processo. A migração dos dados do sistema alvo é realizada com o apoio de ferramentas, e todos os casos de teste são novamente executados no sistema alvo para garantir a sua confiabilidade e maturidade.

Algumas das vantagens do arcabouço ARA, são: a) agrega diversos benefícios de métodos ágeis, por exemplo, é adaptativo; tem participação dos usuários e clientes; é incremental, produzindo uma primeira versão do sistema alvo o mais rápido possível; utiliza testes desde o inicio da reengenharia; incentiva programação em pares; garante propriedade coletiva do código, por meio de controle de versão de todos os artefatos produzidos durante a reengenharia; incentiva a jornada de trabalho de no máximo quarenta horas semanais e garante a prática metáfora, por meio do uso da linguagem de padrões de análise;b) utiliza ferramentas e técnicas que colaboram para o reuso, diminuindo o tempo e o custo da reengenharia e c) O uso de classes herdadas do *framework* aumenta a confiabilidade do sistema alvo uma vez que essas classes já foram testadas anteriormente.

A partir do arcabouço ARA, o processo ágil de reengenharia PARFAIT foi construído, sendo apresentado a seguir.

2.6 Processo Ágil em Framework no domínio de sistemas de Informação com técnicas de VV&T (PARFAIT)

O PARFAIT (Cagnin, 2005) tem como objetivo principal migrar sistemas procedimentais para o paradigma orientado a objetos e garantir que a qualidade do software resultante da reengenharia seja confiável e aceitável pelos usuários. O PARFAIT utiliza como apoio: a) a linguagem de padrões GRN, que por meio de reutilização dos casos de teste da linguagem de padrões de análise, consegue um melhor e mais rápido entendimento do sistema legado, além de auxiliar na documentação OO, sendo que o legado deve ser do mesmo domínio da GRN; b) o *framework* GREN que auxilia na migração dos sistemas legados para a linguagem Smalltalk e o banco de dados MySQL, reutilizando seu projeto e código fonte. Porém, quando há regras de negócios específicas, esta implementação é realizada diretamente

no sistema alvo, que está separado do legado; c) devido à abordagem incremental e iterativa, os requisitos são priorizados a cada iteração, e são validados em cada versão parcial do sistema, por meio da reutilização de casos de teste; d) controle de versão de todos os artefatos.

Desta forma, o PARFAIT atende às seguintes práticas do XP (Beck, 2000): versões pequenas, cliente presente, testes constantes, jogo do planejamento, programação em pares, propriedade coletiva do código e integração contínua, metáfora, semana de 40 horas. As demais práticas ágeis do XP, como, refatoração constante, padrão de codificação e projeto simples, assim como outras práticas ágeis, podem ser utilizadas caso o responsável da reengenharia verifique a necessidade, desde que crie viabilidade para aplicá-las. O PARFAIT também atende a Modelagem Ágil (Ambler,2002), que busca a modelagem e a documentação do sistema legado de maneira efetiva e ágil, facilitando o entendimento e comunicação entre os analistas.

Esse processo é também baseado na estrutura estática do RUP (*Rational Unified Process*) (Rational, 2004). O RUP é um processo de software que determina tarefas e responsabilidades, que ainda, pode ser adaptado e estendido. Portanto ele possui fases, atividades, passos, executores, inspeções, artefatos de entrada, artefatos de saída produzidos por cada atividade e gabaritos que são exemplos dos artefatos que devem ser produzidos. No final de cada fase, existem marcos de referência (*milestones*) que verificam a condução do projeto de reengenharia e fornecem subsídios para que o engenheiro de software decida por continuar ou não a reengenharia, de acordo com os riscos considerados. E ao término de cada iteração, os usuários indicam os requisitos que serão utilizados na próxima iteração.

Como o PARFAIT foi construído para reengenharia de sistemas, ele não utiliza a estrutura dinâmica do RUP, ou seja, o conteúdo de suas atividades, fases, planos de iteração e marcos de referência, pois esses elementos foram definidos no RUP especificamente para o contexto de desenvolvimento de software.

Na Figura 7 é apresentado o processo PARFAIT com suas atividades, que são realizadas durante as fases de Concepção, Elaboração, Construção e Transição. As abordagens incremental e iterativa, utilizadas por esse processo, são identificadas pelas setas e pelos retângulos com borda dupla. Ressalta-se que o engenheiro de software pode retornar a qualquer atividade do processo, estando em qualquer fase, a fim de refinar os artefatos produzidos. Antes de passar de uma fase para a outra, durante as iterações das atividades do processo, é necessário realizar verificações por meio de marcos de referência, representados na Figura 7 por triângulos.

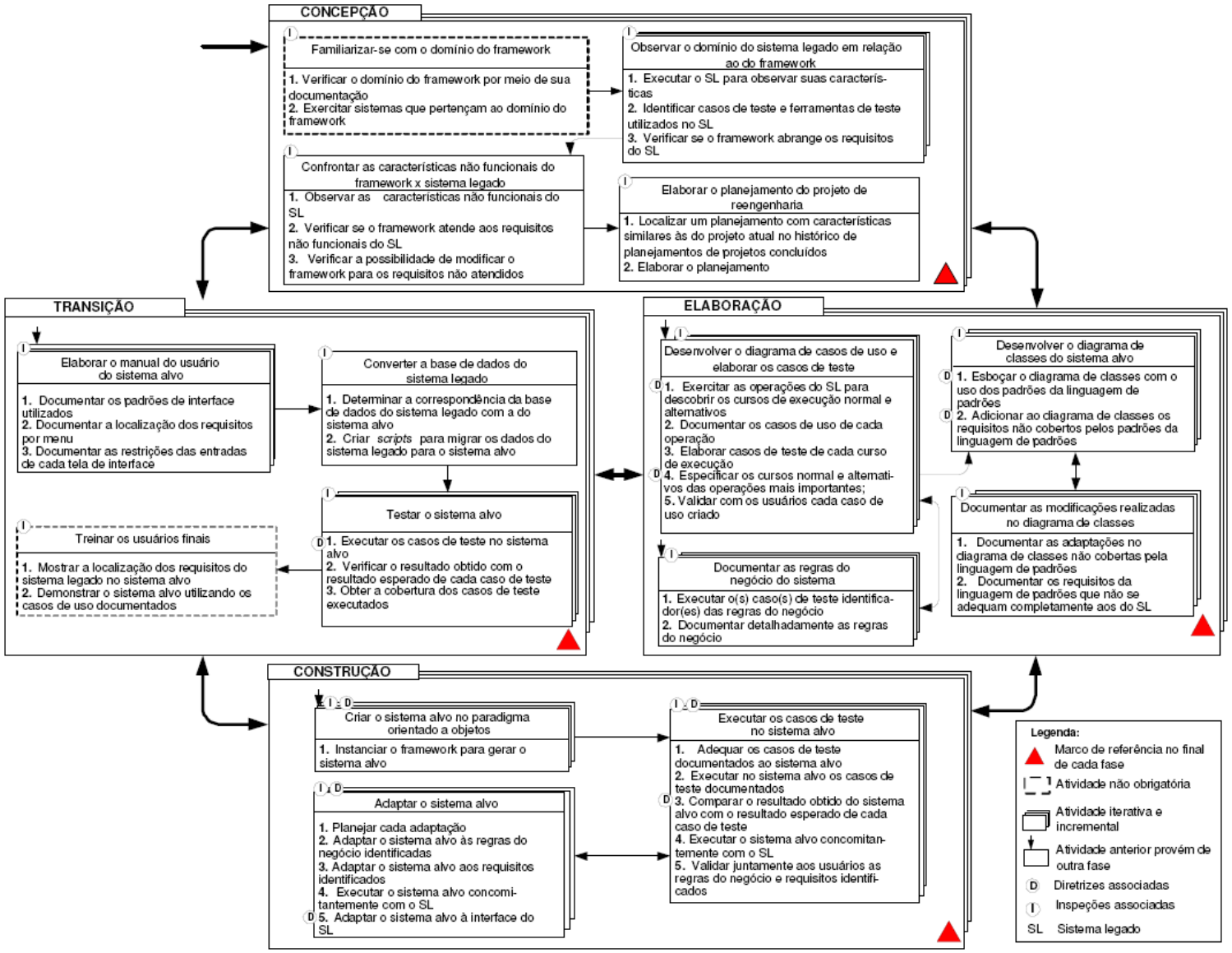

Figura 7 – Visão geral do PARFAIT (Adaptado de Cagnin, 2003b)

2.7 Considerações Finais

O ARA busca agilidade, aumento de qualidade e produtividade ao utilizar um *framework* baseado em uma linguagem de padrões, que oferece aos analistas o reuso na análise e no desenvolvimento. A utilização de padrões em um desenvolvimento de software significa reutilizar soluções bem sucedidas de projetos de reengenharia já realizados, garantindo assim a qualidade do processo de reengenharia a ser adotado e, consequentemente, a qualidade do produto. Assim como os *frameworks*, os geradores de aplicações também asseguram uma melhor qualidade e agilidade no desenvolvimento, pois também são reutilizáveis para um mesmo domínio e desta forma diminuem o tempo de produzir uma aplicação.

Analogamente ao Draco-PUC (Prado, 1992), o PARFAIT contribui para a automatização do processo de reengenharia, ainda que de maneira diferente. Este trabalho tem a finalidade de adaptar o processo de reengenharia ágil PARFAIT para que, ao invés de utilizar *frameworks*, utilize geradores de aplicações. Essa adaptação utiliza o gerador de aplicação GAwCRe (Pazin, 2004) apresentado neste capítulo, que é baseado na linguagem de padrões SiGCLi, cujas variações são representadas em LMA e especificadas em XML.

CAPÍTULO 3 – ESTUDO DE CASO PROSPECTIVO

3.1 Considerações Iniciais

Um estudo de caso prospectivo foi realizado, utilizando o gerador GAwCre (Pazin, 2004) ao invés do *framework* GREN (Braga, 2003) no arcabouço ARA (Cagnin, 2005). O GAwCre abrange o domínio de clínicas de reabilitação física, porém, optou-se pela utilização de um domínio mais abrangente e, consequentemente, com maior disponibilidade de sistemas legados no mercado. Quatro sistemas foram examinados: Psychologist (BioManager, 2006), Medical Office (Medical Office, 2006), Clínicas Integradas (D&D Consultoria, 2006) e CIC (Tesche & Vasconcelos, 2006).

O arcabouço ARA possui abordagem incremental e iterativa, além disso, é constituído de técnicas de reuso, ferramentas e práticas ágeis. Essas características têm como objetivo construir um sistema alvo orientado a objetos, a partir de um sistema legado procedimental, com qualidade em um curto espaço de tempo. O processo PARFAIT (Cagnin, 2005), que refere-se à reengenharia de sistemas procedimentais para sistemas orientados a objetos, também foi utilizado.

Este capítulo está organizado da seguinte maneira: na Seção 3.2 é apresentado o estudo de caso realizado, tendo como exemplo o sistema Psychologist e as considerações finais encontram-se na Seção 3.3.

3.2 Detalhamento do Estudo de Caso

Esta Seção descreve todas as atividades realizadas durante o estudo de caso para incorporar o gerador GAwCre (Pazin, 2004) ao ARA. Esse gerador inicialmente utilizava banco de dados Oracle, e, posteriormente, em um trabalho de iniciação científica (Rizzo, 2005) o banco de dados MySql passou a ser utilizado, por ser de domínio público e assim facilitar o uso do GAwCre.

Na Tabela 1 são mostradas algumas das atividades propostas pelo PARFAIT que foram utilizadas no estudo de caso (a) e, outras tiveram que ser criadas (b). A segunda coluna informa a fase em que cada uma das atividades foi executada.

Tabela 1 – Análise das Atividades em Relação ao PARFAIT

	Fase	**Atividades**
(a) Atividades Executadas do PARFAIT	Concepção	1) Familiarizar-se com o domínio do gerador
		2) Observar o domínio do sistema legado em relação ao gerador
	Elaboração	3) Desenvolver o diagrama de casos de uso e elaborar os casos de testes
		4) Desenvolver o diagrama de classes do sistema alvo
	Construção	5) Criar o sistema alvo no paradigma orientado a objetos
		6) Executar os casos de teste no sistema alvo
		7) Adaptar o sistema alvo
	Transição	8) Converter banco de dados
		9) Testar o sistema alvo
(b) Atividades Criadas	Concepção	1) Buscar sistemas legados
		2) Verificar domínio dos sistemas legados

1) O estudo de caso teve início com a atividade Familiarizar-se com o domínio do gerador. Como o gerador GAwCRe (Pazin, 2004) é baseado na linguagem de Padrões SiGCli(Pazin, 2004) do domínio de clínicas de reabilitação, essa linguagem foi estudada para o entendimento do domínio.

Um segundo passo nessa atividade foi executar o gerador para visualizar as funções já estudadas na linguagem. Nesse ponto, alguns erros foram encontrados que impediam ou dificultavam a continuidade dessa atividade. Assim, algumas modificações fizeram-se necessárias e são mostradas a seguir.

a) nas classes `anamnese_coronapatia.java` e `anamnese_saude_clinica.java` havia duplicidade da declaração da variável `fk_atendente`. A solução adotada foi retirar a duplicidade dessa declaração no XML em ambas as classes, assim a primeira coluna mostra o antes da correção e a segunda o depois da correção tanto no XML quanto nas classes descritas acima.

Tabela 2 – XML e Aplicação antes de depois de alterações

	Antes	**Depois**
XML	`<associacao ocorrenciaMax="1" classe="Atendente" opcional="S" padraoId="6" varianteId="default"> responsavel</associacao>` ***<associacao ocorrenciaMax="1" classe="Atendente" opcional="S" padraoId="6" varianteId="1"> responsavel</associacao>***	`<associacao ocorrenciaMax="1" classe="Atendente" opcional="S" padraoId="6" varianteId="1"> responsavel</associacao>`
Aplicação	`// Atributos da classe` `private int codigoId;` `private String data;` `.` `private String rigidez_noturna;` `private String fk_atendimento;` ***`private int fk_atendente;`*** ***`private int fk_atendente;`***	`// Atributos da classe` `private int codigoId;` `Private String data;` `.` `private String rigidez_noturna;` `private String fk_atendimento;` ***`private int fk_atendente;`***

b) em várias classes da aplicação gerada, como por exemplo, `atendimento.java`, `cidade.java`, `convenio.java`, `pagamento.java`, e outras; métodos estavam vazios. O `Gerador.java` não capturava o que estava dentro da tag CDATA no XML, que tem como função não deixar que o texto que está dentro de sua Seção, seja interpretado (XML, 2006). A solução adotada foi a seguinte:

Algumas *tags* foram inseridas no documento XML e o `Gerador.java` foi modificado de modo que entendesse o significado dessas *tags*. Como por exemplo, o atributo `pev` (sigla que significa ponto e vírgula), serve para indicar ao gerador que na escrita de uma linha de código de saída, deve-se colocar um ponto-e-vírgula no final da linha se o `pev` for igual a 1. Se o `pev` não existir, significa que o gerador não escreve o ponto-e-vírgula no final da linha. Quanto ao &, ele é um código especial do HTML e do XML para permitir que alguns símbolos especiais sejam escritos na tela sem confundir com alguns símbolos que também são usados pra definir *tags* e comandos HTML. Por exemplo, o símbolo < indica o início de uma *tag*, então se escrito dentro de um texto, para que o XML não confunda o texto com uma nova *tag* XML, escreve-se "<" utilizando o código especial &.

A partir da solução adotada, as Tabelas 3, 4 e 5 mostram respectivamente a aplicação, o código XML do gerador e o `Gerador.java`. A primeira coluna ("antes") contém o código original e a segunda ("depois") o código após as alterações realizadas devido aos erros de compilação encontrados.

Tabela 3 – Aplicação antes e depois de alterações

	Antes	**Depois**
Aplicação	`/**Métodos específicos da classe**/` `public String GerarPK(String codPaciente) {` `}// fim do método` `public String getDataAtual() {` `}// fim do metodo` `public String obterGuiasemAberto () {` `}// fim do metodo` `public String imprimirComprovante(int codPaciente, int codguia) {` `}// fim do metodo` `public String imprimirFichaAssinatura (int codPaciente, int codguia) {` `}// fim do metodo` `public String imprimirFatura() {` `}// fim do metodo` `public void gerarFaturas() {` `}// fim do metodo`	`/**Métodos específicos da classe**/` `public String GerarPK(String codPaciente) {` `Calendar data = null;` `data=Calendar.getInstance();` `String pk = ""+data.get(data.YEAR)+(data.get(data.MONTH)+1)+data.get(data.DAY_OF_MONTH)+codPaciente;` `return pk;` `}// fim do metodo` `public String getDataAtual() {` `Calendar data = null;` `data=Calendar.getInstance();` `return data.get(data.DAY_OF_MONTH)+"/"+(data.get(data.MONTH)+1)+"/"+data.get(data.YEAR);` `}// fim do metodo` `public String obterGuiasemAberto() {` `return null;` `}// fim do metodo` `public String imprimirComprovante(int codPaciente, int codguia) {` `return null;` `}// fim do metodo` `public String imprimirFichaAssinatura(int codPaciente, int codguia) {` `return null;` `}// fim do metodo` `public String imprimirFatura() {` `// METODO para listar faturas geradas pelas guias` `return null;` `}// fim do metodo` `public void gerarFaturas() {` `// METODO para listar as despesas` `}// fim do método`

Tabela 4 – XML antes e depois de alterações

	Antes	**Depois**
XML	<metodo tipo="public" tipoRetorno="String">GerarPK <parametro tipo="String" tamanho="4">codPaciente</parametro> <corpoMetodo> ***<![CDATA[*** ***Calendar data = null;*** ***data=Calendar.getInstance();*** ***String pk =*** ***""+data.get(data.YEAR)+(data.get(data.MONTH)+1)+data.get(data.DAY_OF_MONTH)+codPaciente;*** ***return pk;*** ***]]>*** </corpoMetodo> </metodo> <metodo tipo="public" tipoRetorno="String">getDataAtual <corpoMetodo> ***<![CDATA[*** ***Calendar data = null;*** ***data=Calendar.getInstance();*** ***Return*** ***data.get(data.DAY_OF_MONTH)+*** ***"/"+(data.get(data.MONTH)+1)+"/"*** ***+data.get(data.YEAR);*** ***]]>*** </corpoMetodo> </metodo> <metodo tipo="public" tipoRetorno="String" menu="relatorio" nome="Listar guias em aberto">obterGuiasemAberto <corpoMetodo> ***<![CDATA*** ***[return " ";*** ***]]>*** </corpoMetodo> </metodo> . . .	<metodo tipo="public" tipoRetorno="String">GerarPK <parametro tipo="String" tamanho="4">codPaciente</parametro> ***<corpo inicio="1" />*** ***<corpo pev="1"> Calendar data = null</corpo>*** ***<corpo pev="1"> data=Calendar.getInstance()</corpo>*** ***<corpo pev="1">String pk = ""+data.get(data.YEAR)+(data.get(data.MONTH)+1)+data.get(data.DAY_OF_MONTH)+codPaciente</corpo>*** ***<corpo pev="1">return pk</corpo>*** ***<corpo fim="1" />*** </metodo> <metodo tipo="public" tipoRetorno="String">getDataAtual ***<corpo inicio="1" />*** ***<corpo pev="1">Calendar data = null </corpo>*** ***<corpo pev="1"> data=Calendar.getInstance()</corpo>*** ***<corpo pev="1"> return data.get(data.DAY_OF_MONTH)+"/"+(data.get(data.MONTH)+1)+"/"+data.get(data.YEAR)</corpo>*** ***<corpo fim="1" />*** </metodo> <metodo tipo="public" tipoRetorno="String" menu="relatorio" nome="Listar guias em aberto">obterGuiasemAberto ***<corpo inicio="1" />*** ***<corpo pev="1">return null</corpo>*** ***<corpo fim="1" />*** </metodo> . . .

Tabela 5 – `Gerador.java` antes e depois de alterações

	Antes	**Depois**
Gerador Código .java	`Element tipo = (Element)parametros.item(p);` `// se houver parametro` `System.out.println("Tag name:"+tipo.getTagName().toString());` `if(tipo.getTagName().equals("parametro")){` `met +=virgula + tipo.getAttribute("tipo").toString()+ " ";` `Node parametro = tipo.getFirstChild();` `met += preparaString(parametro.getNodeValue().toString().trim());` `virgula=", ";` ***} else if(tipo.getTagName().equals("corpoMetodo")){*** ***met += ") {\n\t\t";*** ***Node corpoMetodo = tipo.getFirstChild();*** ***met += corpoMetodo.getNodeValue();*** ***met += "\n\t}// fim do metodo\n\n";*** ***}***	`Element tipo = (Element)parametros.item(p);` `System.out.println("Tipo: " + tipo);` `// se houver parametro` `System.out.println("Tag name:"+tipo.getTagName().toString());` `if(tipo.getTagName().equals("parametro")){` `met +=virgula + tipo.getAttribute("tipo").toString()+ " ";` `Node parametro = tipo.getFirstChild();` `met += preparaString(parametro.getNodeValue().toString().trim());` `virgula=", ";` `} else if(tipo.getTagName().toString().equals("corpo")){` `if ( tipo.getAttribute("inicio").toString().equals("1") ) {` `met += ") {";` `}` `}` `String pev;` `if ( tipo.getAttribute("pev").toString().equals("1") )` `pev = ";";` `else` `pev = "";` `if ( !tipo.getTextContent().equals("") ) {` `met += "\n\t\t" + tipo.getTextContent();` `met += pev;` `}` `if (tipo.getAttribute("fim").toString().equals("1") )` `met += "\n\t}// fim do metodo\n\n";`

c) nas classes `guia.java` e `patologia.java` algumas palavras apareciam com acento como por exemplo Convênio está na Tabela 11. A solução adotada foi utilizar o método `preparaString`, que faz a retirada de acentos em palavras. Na Tabela 6 é mostrada a aplicação e o `GeradorCodigo.java`. antes e depois das alterações.

Tabela 6 – Aplicação e `GeradorCodigo.java` antes e depois de alterações

	Antes	**Depois**
Aplicação	`//Associacoes entre as classe` `private Paciente paciente[];` ***`private Convênio Convênio;`*** `private Patologia patologia[];` `private Atendente Atendente;` `private Medico medico[];`	`//Associacoes entre as classe` `private Paciente paciente[];` `private Convenio Convenio;` `private Patologia patologia[];` `private Atendente Atendente;` `private Medico medico[];`
GeradorCodigo.Java	`/**método para Escrever os atributos das classes**/` `private String escreveAtributos()throws Exception{` `String atributo="// Atributos da classe \n";` `for (int k=0;k<this.atrib.size();k++){` `atributo+="\tprivate "+this.tipos.elementAt(k)+" "+this.atrib.elementAt(k)+";\n";` `n_atributos++;` `}// for k` `atributo+="\n//Associacoes entre as classe\n";` `for (int k=0; k<this.associacao.size();k++){` `atributo+="\tprivate "+this.tipoAssociacao.elementAt(k)+" "+this.associacao.elementAt(k)+";\n";` `}` `return atributo;` `}` `.........`	`/**método para Escrever os atributos das classes**/` `private String escreveAtributos()throws Exception{` `String atributo="// Atributos da classe \n";` `for (int k=0;k<this.atrib.size();k++){` ***`atributo+= "\tprivate " + preparaString( this.tipos.elementAt(k).toString() ) + " " + preparaString( this.atrib.elementAt(k).toString() ) + ";\n";`*** `n_atributos++;` `}// for k` `atributo+="\n//Associacoes entre as classe\n";` `for (int k=0; k<this.associacao.size();k++){` ***`atributo += "\tprivate " + preparaString( this.tipoAssociacao.elementAt(k).toString() ) + " " + preparaString( this.associacao.elementAt(k).toString() ) + ";\n";`*** `}` `return atributo;` `}` `.........`

Com o gerador funcionando corretamente, ele foi executado diversificando a utilização dos padrões para a geração de aplicações que atendam o domínio de clínicas de reabilitação. Dessa forma, podem-se verificar quais funções fazem parte de cada padrão e como elas se relacionam, e, conseqüentemente, como os padrões se relacionam.

2) Um segundo passo para a realização da reengenharia foi "Buscar sistemas legados" que atendessem ao mesmo domínio que o do gerador. Para essa identificação, buscaram-se sistemas na *Internet* por palavras chaves que referenciam o domínio e, posteriormente, cada sistema foi analisado com uma execução rápida.

Nesta atividade, verificou-se a dificuldade de encontrar sistemas para clínicas somente de reabilitação, ou seja, fisioterapia, terapia ocupacional e educação física, portanto um domínio mais amplo foi adotado, clínicas de qualquer especialidade médica. Optou-se pela escolha de mais de um sistema para que o resultado do estudo de caso fosse mais expressivo. Assim, os sistemas abaixo foram escolhidos para este trabalho:

- Psychologist (Bio Manager, 2006): é um sistema de clínica de psicologia que contém o cadastro de pacientes, anamnese, ficha clínica de acompanhamento por sessão, módulo financeiro, contas a pagar, contas a receber, *backup* de dados, avisos, etc. Foi desenvolvido em JD *Builder* e com *applets* em C++, sendo seu SGBD desenvolvido em Access.
- Medical Office (Medical Office, 2006): é um aplicativo destinado a consultórios e clínicas médicas de todas as especialidades, e tem como objetivo o gerenciamento das principais informações clínicas e financeiras de clínica ou consultório. Desenvolvido em *Visual Basic Microsoft*, pode operar em ambiente multi-usuário (rede local). O sistema permite que todos os usuários acessem o banco de dados, simultaneamente, por meio da rede.
- Clínicas Integradas (D&D Consultorias): faz o gerenciamento de consultórios médicos, odontológicos e fisioterápicos. Controla pacientes ativos e inativos, agenda de atendimento, fluxo de caixa simples, emissão de receitas e atestados, controle e impressão de contratos, controle e impressão de orçamentos, agenda de contatos. Desenvolvido na linguagem *Object Pascal*, com ferramenta Delphi, sem orientação a objetos.
- CIC (Tesche & Vasconcelos, 2006): permite que uma clínica faça o atendimento ao paciente, controla o histórico de todos os atendimentos, agenda com calendário para vários médicos, contas a pagar e receber, recebimentos por convênios, relatórios, emissão de receita, atestado padrão, controle de exames e aniversariantes do mês.

3) A atividade "Verificar Domínio dos Sistemas Legados" fez-se necessária para que, os sistemas identificados como sendo do domínio de clínicas médicas em um primeiro momento, fossem validados como pertencentes realmente a esse domínio. Assim, cada sistema foi executado separadamente, suas funções obtidas e, para melhor visualização organizada uma tabela, reproduzida na Tabela 7, com todas as macro-funções, coluna 1,

juntamente com uma subdivisão em funções, coluna 2, para que todos os legados estudados pudessem ser comparados. As colunas 3, 4, 5 e 6 referem-se aos sistemas legados escolhidos e marcados "X" a função que está presente em cada sistema. Na última coluna "X" indica as funções que podem ser atendidas pelos padrões que compõem a SiGCli (Pazin, 2004) e com "Y" indica os padrões que podem ser utilizados, desde que adaptados. Exemplo desse caso é o da função Anamnese, que no gerador, as perguntas são fixas para fisioterapia, terapia ocupacional ou educação física e no sistema legado qualquer pergunta pode ser cadastrada.

Tabela 7 – Comparação das Funções dos Sistemas Legados

Macro Funções	**Funções**	**Clínicas Integradas**	**Medical Office**	**Psychologist**	**CIC**	**SiGCli**
Cadastros	**Pacientes**	**X**	**X**	**X**	**X**	**X**
	Médicos	**X**	**X**		**X**	**X**
	Convênios		X		X	X
	Indicações		X			
	Especialidades		X			
	Diagnósticos		X		X	
	Trat Clínico		X			X
	Trat Cirúrgico		X			
	Exames		X		X	
	Remédios		X		X	X
	Dietas		X			
	Orientações		X			
	Laudos		X			X
	Anamnese		X	X		Y
	Anotações		X			
	Procedimentos				X	
Agenda	**Consultas**	**X**	**X**	**X**	**X**	**X**
	Espera		X			
	Telefones	X	X			
	Compromissos		X			
	Recados		X	X		
	Aniversariantes			X		
Compra	Pedido	X				X
Financeiro	Contas Pagas e Recebidas	X				X
	Contas a Pagar			X		X
	Contas a Receber			X	X	X
	Conta do Paciente			X	X	Y
Acompanha/o	**Prontuário**		**X**	**X**	**X**	**X**
	Receita	X			X	
	Atestado	X			X	
	Orçamento	X				
		X	**X**	**X**	**X**	**X**
	Backup	**X**	**X**	**X**	**X**	
Relatórios / Impressões	Histórico	X				
Manutenção	Recupera BD		X	X		
	Compactar / Reparar		X			
	Troca Senha	**X**	**X**	**X**	**X**	
	Manutenção		X			
Usuários	Troca Usuário		X			

Como pode-se observar, algumas funções estão presentes em todos (ou quase todos) os sistemas legados. Essa característica deve ser levada em consideração ao longo da reengenharia com o gerador, pois são importantes ao domínio, e, conseqüentemente ao gerador. Com exceção das funções "Backup" e "Troca de Senha", que não se referem diretamente ao domínio de clínicas médicas e sim, ao subdomínio de Segurança que deve estar presente em todos os sistemas. Nota-se também pela Tabela 7, que outras funções não estão presentes na SiGCli (Pazin, 2004), como por exemplo o cadastro de Dietas, já que o domínio do gerador é restrito para clínicas de reabilitação.

4) A partir da atividade "Observar o domínio do sistema legado em relação ao gerador" os sistemas legados foram tratados separadamente. Na Tabela 8 são apresentadas quais funções existentes no sistema Psychologist podem ser implementadas pelo GAwCRe (Pazin, 2004), por meio dos padrões da SiGCli.

Tabela 8 – Comparação do Sistema Psychologist com o Gerador

Psychologist		**SiGCli**
Macro Funcionalidade	Funções	Padrões
Arquivo	Bloco de Notas	
	Calculadora	
	Word	
	Paint	
	Explorer	
Avisos	Avisos de Aniversariantes	
	Avisos Gerais	
Backup	Backup	
	Recuperação BD	
Agenda	**Agenda de Consultas**	**5- Agendar Atendimento** **3- Realizar Vendas**
	Calendário	
Cadastros	Cadastrar Perguntas	
	Anamnese do Paciente	
	Paciente	**1- Identificar Pacientes**
	Ficha Clinica	**6- Identificar Atendentes** **7- Realizar Acompanhamento** **2- Definir Serviços**
Financeiro	**Contas a Pagar**	**9- Controlar Faturamento**
	Contas a Receber	**9- Controlar Faturamento**
	Conta do Paciente	**9- Controlar Faturamento**
Acesso	Modificar Senha	

Na terceira coluna da Tabela 8 os padrões da SiGCli utilizados obedecem ao número do padrão como apresentado na Seção 2.2:

1- Identificar Pacientes;

2- Definir Serviços: o único serviço disponibilizado pela clínica é "Consulta";

3- Realizar Vendas: o serviço a ser comercializado pela clínica é "Consulta";

5- Agendar Atendimentos;

6- Identificar Atendentes;

7- Realizar Acompanhamento;

9- Controlar Faturamento.

A Tabela 8 possibilitou notar que, funções relacionadas à "Arquivo", "Avisos", "Backup" e "Acesso" não são atendidos pela SiGCli, porém pode-se concluir que são subdomínios conexos, não relacionados diretamente a clínicas de reabilitação física. Também observou-se que uma importante função, a anamnese não é utilizada porque as perguntas existentes no GAwCre não se aplicam ao sistema Psychologist, já que são perguntas voltadas para o subdomínio de reabilitação física. Assim, o padrão da SiGCli poderia ser reutilizado somente com a alteração das perguntas, o que implica na modificação do código XML do gerador. Isso não foi realizado por não ter sido possível obter quais questões devem ser abordadas em um sistema de psicologia.

5) A atividade "Desenvolver o diagrama de casos de uso e elaborar os casos de testes" permitiu ao desenvolvedor aumentar o entendimento do sistema, sendo necessário realizar um estudo mais preciso das funções do sistema. Como o processo é incremental e iterativo, os requisitos foram priorizados para cada iteração. Neste estudo são consideradas as funções de Cadastro de Pacientes, Consultas, Prontuário e alguns relatórios. Para efeito de exemplificação do processo de reengenharia, a função de Cadastro de Pacientes é considerada nesta atividade e nas demais.

Assumiu-se que os sistemas legados não tinham nenhuma documentação, já que nos *sites* em que estavam disponíveis, havia somente a opção de *download* do executável. Assim, os seguintes artefatos foram desenvolvidos para ajudar no entendimento e também, posteriormente, como contribuição para a documentação do sistema alvo. O diagrama de casos de uso foi elaborado com a ajuda da ferramenta ArgoUML (ArgoUML, 2006) e é exibido na Figura 8. Na Figura 9 a especificação do caso de uso Cadastrar Pacientes engloba várias operações de um cadastro (inclusão, alteração, exclusão, consulta e impressão) e os

casos de teste são mostrados na Tabela 9. Em negrito estão as possíveis melhorias a serem realizadas, como por exemplo o atributo RG, que além de verificar se o tamanho máximo do campo foi ultrapassado, o sistema também poderia consistir se o RG é válido.

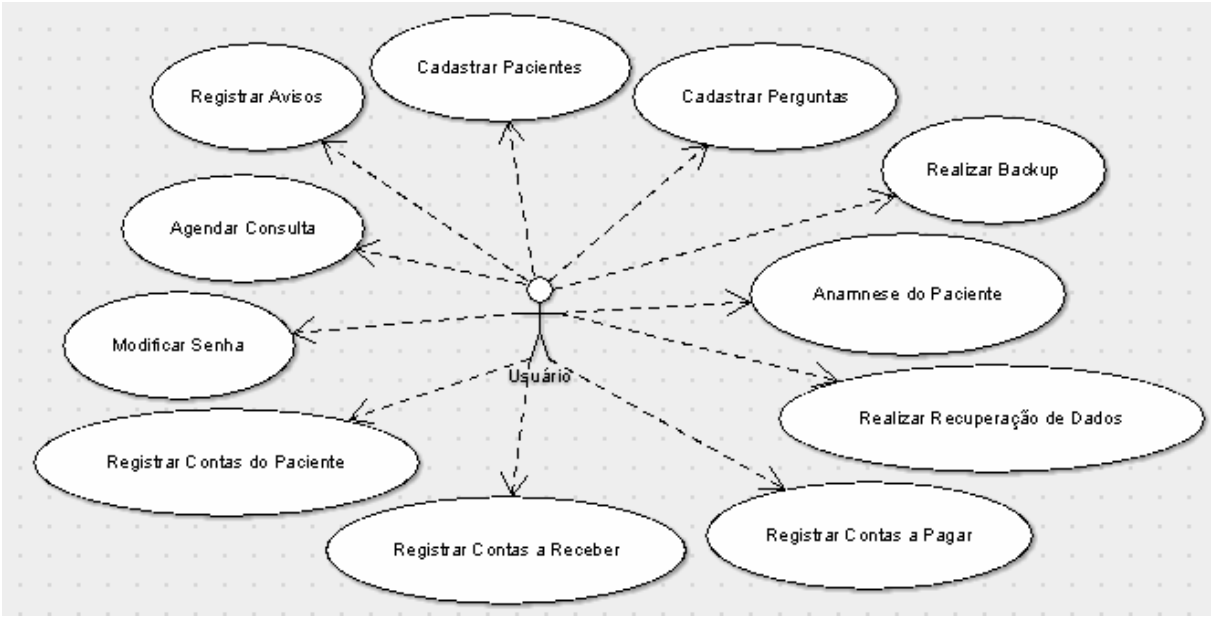

Figura 8 – Diagrama de Caso de Uso Cadastrar Pacientes

Caso de uso: CADASTRAR PACIENTES
Descrição: este caso de uso permite que o usuário realize o cadastro das pessoas que freqüentam a clínica.

1. Paciente fornece seu nome;
2. Paciente não cadastrado;

Fluxo Normal – Inclusão
2.1. Paciente fornece seus dados;
2.2. Gerar Código Paciente;
2.3. Criar Paciente;
2.4. Encerra caso de uso

Fluxo Alternativo - Alteração
2.5. Paciente já cadastrado
2.5.1. Paciente deseja alterar dados;
2.5.2. Paciente fornece dados a serem alterados;
2.5.3. Alterar Paciente;
2.5.4. Encerrar caso de uso

Fluxo Alternativo – Exclusão
2.5.5. Paciente será excluído;
2.5.6. Excluir Paciente;
2.5.7. Encerrar caso de uso.

Fluxo Alternativo – Impressão
2.5.8. Paciente será impresso;
2.5.9. Imprimir Paciente;
2.5.10. Encerrar caso de uso.

Figura 9 – Caso de Uso Cadastrar Pacientes

Tabela 9 – Casos de Teste

		Classes de Equivalência		**Casos de Testes**	
Atributos	**Descrição Banco**	**Classes Válidas**	**Classes Inválidas**	**Especificação dos Dados de Entrada**	**Saída Esperada**
Código ou Nome (na busca)	Texto 50	Texto ou vazio	----	Código ou Nome cadastrado	Paciente aparece na lista
				Vazio	Aparecem todos pacientes
Código do Paciente	Numeração automática	Gera Código	Não gera código	Verificar código	Código gerado automati/e
Data do Cadastro	Texto 12	Data Atual	Data ! = Atual	Verificar Data	Data Atual
Paciente	Texto 50	1 < = Texto < = 50	Texto < 1 Texto > 50	Texto < 50	Sucesso
				Texto = 50	Sucesso
				Texto > 50	Não Sucesso
Data de Nascimento	Texto 12	Texto < 12	Texto > 12	Texto < 12	Sucesso
				Texto = 12	Sucesso
				Texto > 12	Não Sucesso
		1 <= Dia <= 31	**Dia < 0 e Dia > 31**	**Dia = 10**	**Sucesso**
				Dia >31	**Não sucesso**
				Dia < 1	**Não sucesso**
				Dia = 31	**Sucesso depende do mês**
		1 <= Mês <= 12	**Mês < 1 e Mês > 12**	**Mês = 10**	**Sucesso**
				Mês > 12	**Não sucesso**
				Mês < 1	**Não sucesso**
				Mês = 12	**Sucesso**
Natural de	Texto 50	Texto < 50	Texto > 50	Texto < 50	Sucesso
				Texto = 50	Sucesso
				Texto > 50	Não Sucesso
Sexo (F/M)	Numero 1/0	Seleção de uma instância	Seleção de mais de uma instância	Seleção de mais de uma instância	Não Sucesso
RG	Texto 50	Texto < 50	Texto > 50	Texto < 50	Sucesso
				Texto = 50	Sucesso
				Texto > 50	Não Sucesso
		RG válido	**RG inválido**	**RG inválido**	**Não Sucesso**
CPF	Texto 30	Texto < 30	Texto > 30	Texto < 30	Sucesso
				Texto = 30	Sucesso
				Texto > 30	Não Sucesso
		CPF válido	**CPF inválido**	**CPF inválido**	**Não Sucesso**
Instrução	Seleção	Seleção de um registro	Seleção de mais de um registro	Seleção de mais de um registro	Não Sucesso
		Mostra todos os registros cadastrados	Não mostra todos os registros cadastrados	Mostra todos os registros cadastrados	Sucesso
Curso / Instituição	Texto 50	Texto < 50	Texto > 50	Texto < 50	Sucesso
				Texto = 50	Sucesso
				Texto > 50	Não Sucesso

6) A atividade "Desenvolver o Diagrama de Classes do Sistema Alvo" tem a finalidade de construir um esboço do diagrama de classes, considerando os requisitos priorizados pelo usuário para cada iteração.

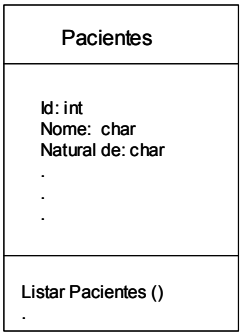

Figura 10 – Diagrama de Classes do sistema alvo

7) A atividade "Criar o sistema alvo no paradigma orientado a objetos" tem a finalidade de construir o novo sistema com a utilização do gerador GAwCRe (Pazin, 2004). Nessa exemplificação verificou-se que a função de Cadastro de Pacientes é atendida por meio do Padrão 1 – Identificar Pacientes, da SiGCli (Pazin, 2004). Na Figura 11 é mostrado como ficou a versão 0 desse cadastro gerado.

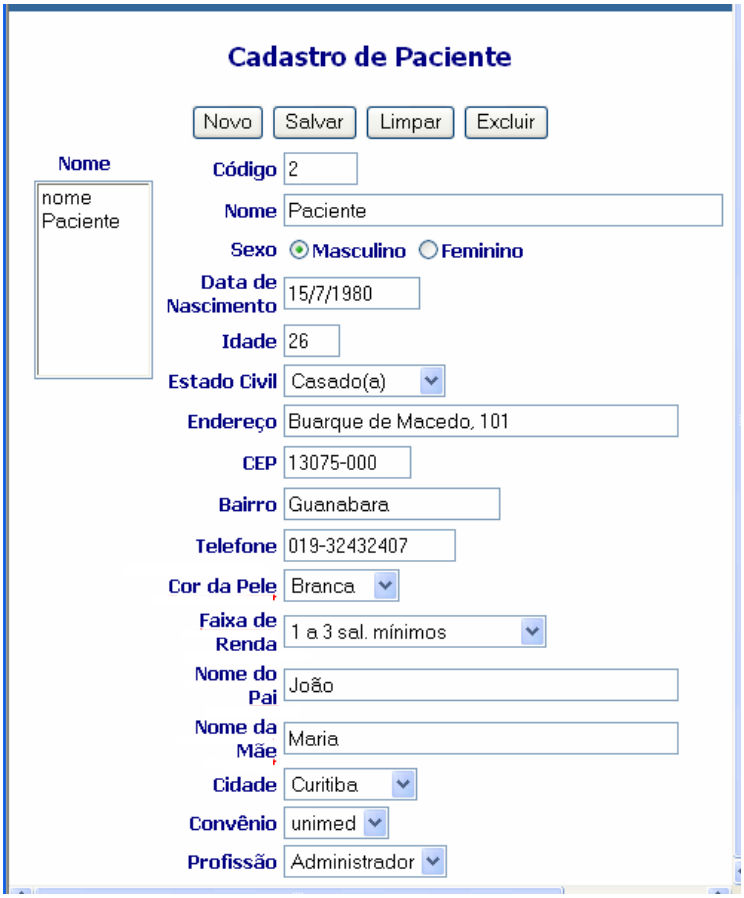

Figura 11 – Cadastro de Pacientes no sistema alvo – versão 0

8) A atividade "Executar os casos de teste no sistema alvo" visa assegurar que a funcionalidade do sistema legado está de acordo com a do sistema alvo gerado. Os casos de teste da Tabela 14 foram executados no sistema alvo e verificou-se que muitos deles encontraram erros, pelo fato dos atributos diferirem do sistema alvo, como por exemplo a inexistência do campo RG no sistema alvo. Foi realizada uma análise de quais campos estavam inadequados, de acordo com a Figura 12 e a Tabela 9.

Na Figura 12, o legado apresenta-se à esquerda e o sistema alvo à direita, sublinhados estão os atributos diferenciados no cadastro. Na Tabela 10, na coluna à esquerda estão os campos que faltaram no sistema alvo e à direita os que não são necessários.

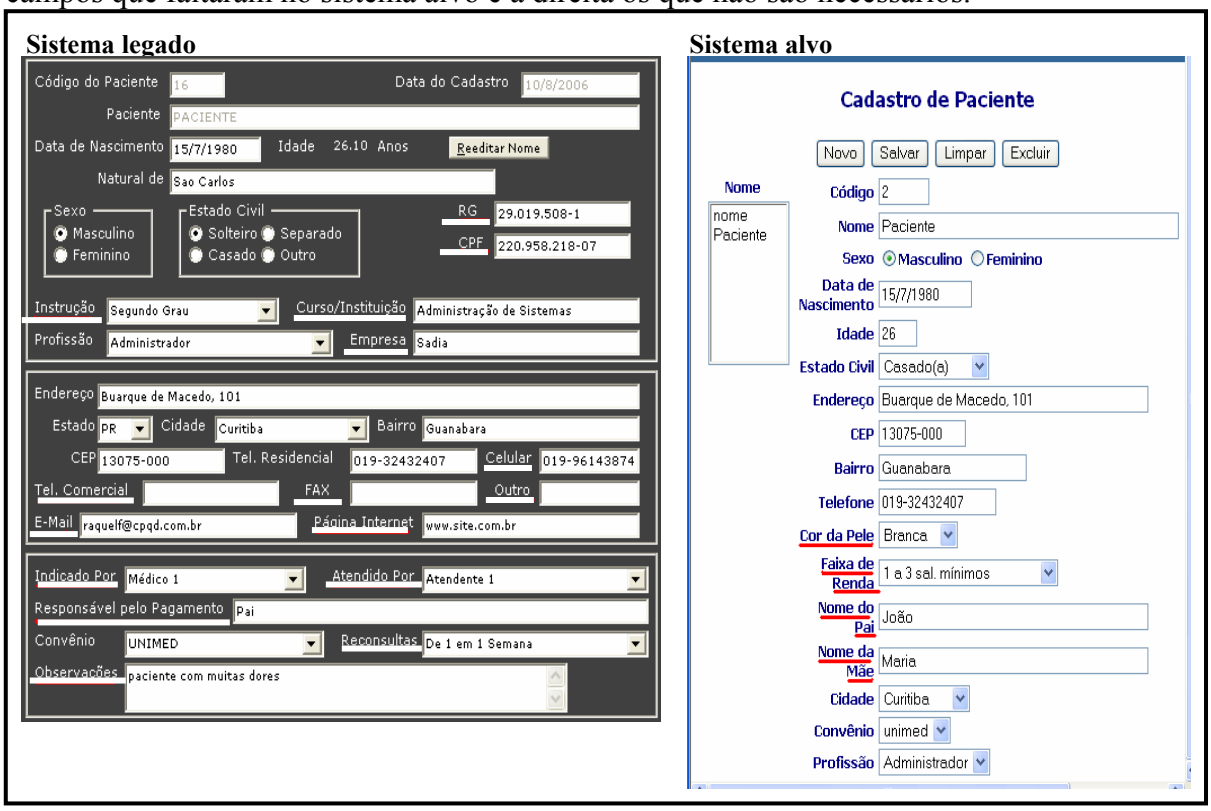

Figura 12 – Cadastro de Pacientes no sistema legado e alvo

Tabela 10 – Comparação dos atributos Cadastro de Pacientes

Sistema Alvo		
Faltou		**Não necessário**
Data de Cadastro	FAX	Cor da Pele
Natural de	Outro Tel.	Faixa Renda
RG	E-mail	Nome Pai
CPF	Pág. Internet	Nome Mãe
Instrução	Indicado por	
Curso	Atendido por	
Empresa	Responsável Pag/o	
Tel. Comercial	Reconsultas	
Celular	Obs	

Observou-se então que as funções que eram atendidas pelo gerador também deveriam sofrer alterações para atender aos requisitos do legado. Na atividade seguinte supriu-se as diferenças do sistema alvo construído seguindo os padrões SiGCli (Pazin, 2004).

9) Na atividade "Adaptar o sistema alvo" deve-se suprir as diferenças entre o legado e o alvo para que a reengenharia obtenha sucesso. Haviam duas maneiras para que essas modificações fossem realizadas: a) diretamente no sistema gerado ou b) no gerador. No ARA, com a utilização de *framework*, os requisitos específicos são implementados diretamente no sistema alvo, como já falado no capítulo anterior. Porém, de acordo com Pazin (2004), adaptações no gerador são possíveis, basicamente modificando-se a LMA, e mapeando para o documento XML.

Com essas informações as duas possibilidades foram realizadas no Cadastro de Pacientes para que uma análise pudesse verificar esforços e benefícios. Na Tabela 11, consta o passo realizado para a inclusão de um atributo por meio do gerador. Nesse caso, fez-se necessário somente, adicionar uma linha ao documento XML descrevendo esse atributo. Já na Tabela 12 é mostrado que quatro arquivos tiveram que ser modificados com vários ajustes em cada um.

Tabela 11 – Inclusão de Atributo no Gerador

Passos	Local	Gerador
1	XML	`<atributo tipo="String" tamanho="2" tipoCampo="text" nome="Empresa"> empresa</atributo>`

Tabela 12 – Inclusão de Atributo no Sistema Alvo

Passos	Local	Sistema Alvo (Modificações)
1	Paciente.java	**Inserir o atributo na sessão de declaração de variáveis** `private String empresa;`
2		**Inserir o atributo na declaração do construtor** `public Paciente(int codigoId,.,String empresa,.)` `{`
3		**Inserir o atributo no corpo do construtor** `this.empresa = empresa;`
4		**Inserir o atributo como coluna que será acessado no Banco de Dados** `this.colNames.addElement("empresa");`
5		**Criar uma entrada no Banco de Dados** `this.colValues.addElement(this.empresa);`
6		**Criar método set** `public synchronized void setempresa(String valor){` `if(valor.length()<=0){` `valor="";` `}`

		this.empresa = valor; Criar método get public String getempresa(){ if(this.empresa.length()<=0){ return ""; }else{ return this.empresa; } }
7	Paciente.jsp	**Inserir o atributo na função Carrega** function carrega(){ // exibe o valor selecionado . document.formulario.empresa='<%= rs.getString ("empresa")%>' .
8		**Inserir o atributo no formulário html** <form name= 'formulario' method='POST' action='paciente_resp.jsp'> . <tr> <td align='right'><p> Empresa </td> <td> <input type='text' name='idade' size='2'> </td> </tr>
9	Paciente_resp.jsp	**Inserir o atributo no formulário** <BODY> <form name='formulario' method='POST' action='empresa.jsp'> . <input type='hidden' name='Empresa'value='<%= request.getParameter ("Empresa")%>'>
10		**Inserir o atributo no construtor** <% try { Paciente obj = new Paciente(paciente.getcodigoId() ,., paciente.getempresa(), .);}
11	MySql ou ScriptTabelas.sql	**Inserir o atributo (como coluna) no Banco de Dados** Digitar a linha: Alter table Paciente add column empresa varchar (30) NULL; ou Inserir o atributo no arquivo ScriptTabelas.sql CREATE TABLE PACIENTE (. EMPRESA VARCHAR(30) NULL, .)

A inclusão de relatório é importante na reengenharia de legados, o relatório Listar Pacientes foi elaborado, no sistema alvo e no gerador para verificar novamente em qual haveria mais agilidade. Na Tabela 13 estão as modificações realizadas no gerador, ou seja, no documento XML. E na Tabela 14 estão as modificações em `paciente.java` e `menudados.jsp` e, também da criação de `listar_pacientes.jsp`, que são as alterações realizadas no sistema alvo.

Tabela 13 – Inclusão de relatório no Gerador

Passos	Local	Gerador
1	XML	Criação do método ListarPacientes `<metodo tipo="public" tipoRetorno="String" menu="relatorio" nome="Listar Pacientes">listarPacientes` `<parametro tipo="String" tamanho="10" tipoCampo="select" nome="Ordenar por">ordem` `<valor exibe="Código" usa="codigoId"></valor>` `<valor exibe="Nome" usa="descricao"></valor>` `</parametro>` `<corpo inicio="1"></corpo>` `<corpo pev="1">String html =""</corpo>` `<corpo>try{</corpo>` `<corpo pev="1">  String order = "ORDER BY " + ordem</corpo>` `<corpo pev="1">  ResultSet results = findall(order)</corpo>` `<corpo pev="1"> html= "<p align='center'><b>` `<font size='4' face='Tahoma' color='#000080'>"` `</corpo>` `<corpo pev="1"> html+="Listar Pacientes</font><` `/b></p>" </corpo>` `<corpo pev="1">html+="<TABLE BORDER=1 BGCOLOR='C0C0C0'` `align='center'>"</corpo>` `<corpo pev="1">  html+="<TH  BGCOLOR='white'><` `I>Código</I> </TH>"</corpo>` `<corpo pev="1"> html+="<TH BGCOLOR='white'><` `I>Nome< /I></TH>" </corpo>` `<corpo pev="1">  html+="<TR>"</corpo>` `<corpo> while(results.next()){</corpo>` `<corpo pev="1">html+="<TD ALIGN='center'>"+ results.` `getString` `("codigoId")+"<align='center'></TD>"</corpo>` `<corpo pev="1">html+="<TD ALIGN='center'>"+ results.` `getString` `("nome")+"<align='center'></TD>"</corpo>` `<corpo pev="1">html+="</TR>"</corpo>` `<corpo> } </corpo>` `<corpo>}catch (Exception e) {</corpo>` `<corpo pev="1">  e.printStackTrace()</corpo>` `<corpo pev="1">  html="Problema para listar os` `dados!"</corpo>` `<corpo>}</corpo>` `<corpo pev="1">return html</corpo>` `<corpo fim="1"></corpo>` `</metodo>`

Tabela 14 – Inclusão de relatório no Sistema Alvo

Passos	Local	Sistema Alvo
1	Paciente.java	**Criar o método listarPacientes** `public String listarPacientes (String ordem) {` `String html ="";` `try{` `String order = "ORDER BY " + ordem;` `ResultSet results = findall(order);` `html= "<p align='center'><b><font size='4'` `face='Tahoma' color='#000080'>";` `html+="Listar Pacientes</font></b></p>";` `html+="<TABLE BORDER=1 BGCOLOR='C0C0C0'` `align='center'>";` `html+="<TH BGCOLOR='white'> <I>Código</I> </TH>";` `html+="<TH BGCOLOR='white'> <I>Nome</I> </TH>";` `html+="<TR>";` `while(results.next()){` `html+="<TD ALIGN='center'>"+ results.getString` `("codigoId") + "<align='center'></TD>";` `html+="<TD ALIGN='center'>"+ results.getString` `("nome") + "<align='center'></TD>";` `html+="</TR>";` `}` `}catch (Exception e) {` `e.printStackTrace();` `html="Problema para listar os dados!";` `}` `return html;` `}// fim do método`
2	listar_pacientes.jsp	**Criar o arquivo listar_pacientes com o formulário que será exibido na funcionalidade** `<%@page import="teste_clinica.*"%>` `<%@page import="java.sql.*"%>` `<jsp:useBean id="paciente"` `class="teste_clinica.Paciente"/>` `<jsp:setProperty name="paciente" property="*" />` `<HTML>` `<HEAD>` `<TITLE>Listar Pacientes</TITLE>` `<%@ include file="menu.html" %>` `</HEAD>` `<BODY>` `<%` `ResultSet rs;` `String parametro = ""+request.getParameter("btn");` `try {` `if (parametro.equals("Imprimr")){` `out.println(paciente.listarPacientes(request.getPar` `ameter("ordem")));` `out.println("<a href='listar_pacientes.jsp'>voltar` `</a>");` `} else {` `out.println("<p align='center'><b><font size='4'` `face='Tahoma' color='#000080'>");` `out.println("Listar Pacientes</font></b></p>");` `out.println("<form name= 'formulario' method='POST'` `action='listar_pacientes.jsp'>");` `out.println("<table border='0' width='60%'` `align='center'>");`

		out.println("<tr>"); out.println("<td align='right'>"); out.println("<p>Ordenar por</td>"); out.println("<td>"); out.println("<select name='ordem'>"); out.println("<option value='codigoId'>Código</option>"); out.println("<option value='descricao'>Nome</option>"); out.println("</select>"); out.println("</td>"); out.println("</tr>"); out.println("<tr><td></td><td><input type='submit' name='btn' value='Imprimir'></td></tr>"); out.println("</table></form>"); } }catch(Exception e){out.println(e);} %></BODY> </HTML>
3	menudados. jsp	**Modificar o menudados.jsp adicionando o item Listar Pacientes** startMenu('mRelatorios', true, 0, 22, 200, subM); . addItem('Listar Pacientes', 'listar_pacientes.jsp', ''); .

Uma outra modificação realizada foi a criação de um novo cadastro, sendo que este, deveria ser selecionado em um cadastro já existente. Assim, foi construído o cadastro "Reconsulta" e adicionado no cadastro de Paciente. Na Tabela 15 é mostrada a modificação realizada no gerador e em 16 estão os passos macros para a realização desta alteração, já que esses passos já foram detalhados nas Tabelas 12 e 14. Isto significa que os seis passos macro serão subdivididos em vários passos quando executados.

Tabela 15 – Inclusão de Cadastro com dependência no Gerador

Passos	Local	Gerador
1	XML	**Criar a classe Reconsulta** <padrao numero="1" nome="Identificar Pacientes"> . <classe nome="Reconsulta" menu="cadastro" tipoInterface="1"> <atributo tipo="int" tamanho="3" tipoCampo="text" nome="Código" primaryKey="S">codigoId</atributo> <atributo tipo="String" tamanho="20" tipoCampo="lista" nome="Período">descricao</atributo> </classe>
2		**Inserir o atributo no cadastro de pacientes como associação** <padrao numero="1" nome="Identificar Pacientes"> . <associacao ocorrenciaMax="1" classe="Reconsulta">reconsulta</associacao> .

Tabela 16 – Inclusão de Cadastro com dependência no Atributo no Sistema Alvo

Passos Macros	Sistema Alvo
1	Criar o arquivo `Reconsultas.java`
2	Criar o arquivo `Reconsultas.jsp`
3	Criar o arquivo `ReconsultasResp.jsp`
4	Modificar o arquivo `menudados.jsp` inserindo o novo cadastro
5	Realizar os mesmos passos de Inserção de Atributos para inserir o atributo Reconsultas no cadastro de Pacientes.
6	Alteração Banco de Dados - insere a nova coluna Reconsultas e coloca como FK para Paciente ou Altera `scriptTabelas.sql` (não pode estar em produção)

Após as verificações das Tabelas de 11 a 16, optou-se por alterar o gerador, obtendo com isso agilidade e menor risco de erros. No caso do cadastro tratado neste capítulo, alguns atributos foram adicionados, outros retirados e ainda, modificados quanto ao tamanho. Na Figura 13, está a versão 1 do sistema alvo com as modificações realizadas no documento XML. A versão 1 do sistema legado encontra-se à esquerda, enquanto que, o sistema alvo, com os atributos que foram inseridos no sistema alvo sublinhados, está à direita.

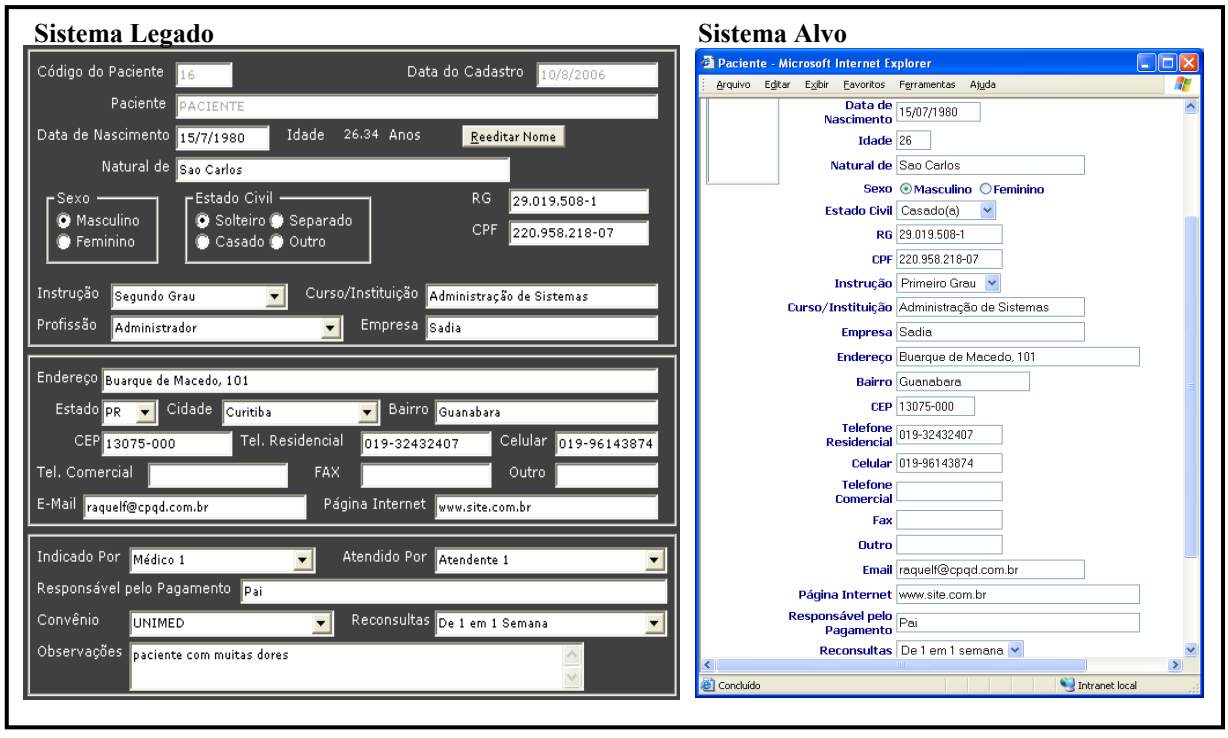

Figura 13 – Versão 1 com modificação da LMA

Após modificações realizadas no gerador, a documentação já elaborada também deve ser alterada para atender o sistema alvo, já que algumas funções podem diferenciar em algum ponto. Nesta atividade, os casos de teste, o diagrama de caso de uso e a especificação também são atualizados.

9) A atividade "Converter Banco de Dados" é necessária para que os dados do sistema legado sejam transferidos para o sistema alvo. Como o banco de dados do legado é Access, foi utilizada a ferramenta *MS Access to MySql* (MySql, 2006) que gera um *dump* da base. Na Tabela 17 são mostradas na primeira coluna as informações da **Tabela Pacientes** do sistema legado (retiradas do arquivo gerado pela ferramenta *MS Access to MySql*) e na segunda coluna as informações do sistema alvo (retiradas do arquivo gerado *ScriptsTabela.sql*). Na Figura 14, é mostrado o *dump* criado, porém para que esse seja executado corretamente alguns ajustes devem ser realizados: o nome da tabela deve ser o mesmo tanto no legado quanto no alvo; as colunas devem estar na mesma ordem. Para tal, o *dump* foi alterado de CadPacientes para Paciente e as colunas da tabela foram colocadas na mesma ordem que a do legado. Para isso, o XML foi modificado alterando a ordem dos atributos, e o arquivo ScriptTabelas.sql e o sistema alvo gerado novamente, ou seja, cada linha de *tag* "<atributo>... </atributo>" foi organizado de forma a ficar na mesma ordem que o *dump* gerado, conforme mostra a Figura 15.

Tabela 17 – Informações da Tabela de Pacientes do Sistema Legado e Alvo

Tabela de Pacientes do Legado	**Tabela de Pacientes do Sistema Alvo**
# # Table structure for table 'CadPacientes' # CREATE TABLE `**CadPacientes**` (`CodCli` INTEGER NOT NULL AUTO_INCREMENT, `Nome` VARCHAR(50), `Nascimento` VARCHAR(12), `Sexo` TINYINT, `RG` VARCHAR(50), `CPF` VARCHAR(50), `Profissao` VARCHAR(100), `Empresa` VARCHAR(100), `Endereco` VARCHAR(255), `Estado` VARCHAR(2), `Cidade` VARCHAR(50), `Bairro` VARCHAR(50), `TelResidencial` VARCHAR(50),	CREATE TABLE **PACIENTE** (CODIGOID INT(4) NULL, NOME VARCHAR(50) NULL, DATA_NASCIMENTO VARCHAR(12) NULL, IDADE INT(2) NULL, NATURAL_DE VARCHAR(50) NULL, SEXO VARCHAR(1) NULL, ESTADO_CIVIL VARCHAR(10) NULL, RG VARCHAR(50) NULL, CPF VARCHAR(50) NULL, INSTRUCAO VARCHAR(50) NULL, CURSO VARCHAR(50) NULL, EMPRESA VARCHAR(100) NULL, ENDERECO VARCHAR(255) NULL, BAIRRO VARCHAR(50) NULL,

`TelComercial` VARCHAR(50), `TelCelular` VARCHAR(50), `FAX` VARCHAR(50), `EMail` VARCHAR(100), `Pagina` VARCHAR(100), `CEP` VARCHAR(50), `RespPag` VARCHAR(100), `ProxConsulta` VARCHAR(50), `Obs` VARCHAR(255), `EstCivil` TINYINT, `Natural` VARCHAR(50), `Instrucao` VARCHAR(50), `Curso` VARCHAR(50), `Outro` VARCHAR(20), `Convenio` VARCHAR(30), PRIMARY KEY (`CodCli`)) TYPE=MyISAM;	CEP VARCHAR(50) NULL, TELEFONE VARCHAR(50) NULL, CELULAR VARCHAR(50) NULL, TEL_COMERCIAL VARCHAR(50) NULL, FAX VARCHAR(50) NULL, TEL_OUTRO VARCHAR(20) NULL, EMAIL VARCHAR(100) NULL, PAG_INTERNET VARCHAR(100) NULL, RESPONSAVEL_PAG VARCHAR(100) NULL, RECONSULTAS VARCHAR(50) NULL, OBSERVACOES VARCHAR(255) NULL, FK_PROFISSAO INT(100) NULL, FK_CIDADE INT(50) NULL, FK_CONVENIO INT(30) NULL, FK_ATENDENTE INT(50) NULL);

```
#
# Dumping data for table 'CadPacientes'
#

INSERT INTO `CadPacientes` VALUES (3, 'PACIENTE 3', '29/8/2006', 1, NULL, NULL, NULL, NULL,
NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL, NULL, NULL, 3, NULL, NULL, '1/8/2006', NULL, NULL, NULL, NULL, NULL, 0, NULL);
INSERT INTO `CadPacientes` VALUES (16, 'PACIENTE', '15/7/1980', 1, '29.019.508-1', '220.958.218-07',
'Administrador', 'Sadia', 'Buarque de Macedo, 101', 'PR', 'Curitiba', 'Guanabara', '019-32432407', NULL, '019-
96143874', NULL, 'raquelf@cpqd.com.br', 'www.site.com.br', '13075-000', 'Médico 1', 'Atendente 1', 'Pai', 'De
1 em 1 Semana', 'paciente com muitas dores', NULL, 3, 'Sao Carlos', 'Segundo Grau', '10/8/2006', NULL,
'Administração de Sistemas', NULL, 'UNIMED', NULL, 0, NULL);

                                    # 2 records
```

Figura 14 – Dados da Tabela Pacientes do sistema legado

```
<classe nome="Paciente" menu="cadastro" tipoInterface="1">
<atributo tipo="int" tamanho="4" tipoCampo="text" nome="Código"
primaryKey="S" requerido="S">codigoId</atributo>
<atributo tipo="String" tamanho="50" tipoCampo="lista" nome="Nome"
requerido="S"> nome </atributo>
<atributo tipo="String" tamanho="12" tipoCampo="text" nome="Data de
Nascimento"> data_nascimento</atributo>
<atributo tipo="String" tamanho="50" tipoCampo="text" nome="Natural
de">natural_de</atributo>
<atributo tipo="String" tamanho="1" tipoCampo="radio" nome="Sexo">sexo
<valor exibe="Masculino" grava="M" />
<valor exibe="Feminino" grava="F" />
</atributo> .....
<atributo tipo="String" tamanho="15" tipoCampo="text" nome="RG">rg
</atributo>
```

Figura 15 – XML com as tags <atributo> ordenadas

10) A atividade "Testar o Sistema Alvo" garante a qualidade do produto final. O sistema deve ser implantado somente após a execução novamente dos casos de teste, para que o usuário obtenha um sistema com qualidade.

3.3 Considerações Finais

Neste capítulo foram apresentadas as atividades realizadas durante o estudo de caso do sistema Psychologist, sendo que os demais sistemas estão apresentados no Apêndice 2. Este trabalho teve como base o processo de reengenharia ágil PARFAIT, com a diferença de que ao invés da utilização de um *framework*, o gerador de aplicações GAwCRe (Pazin, 2004) foi empregado.

Como primeiro ponto destaca-se a utilização de sistemas cujo domínio é o de clínicas médicas, pois sistemas para clínicas de reabilitação física, domínio do GAwCRe, não são encontrados na *Web* com facilidade. Pelo fato do domínio utilizado ser mais abrangente, nem todas as funções dos sistemas legados estudados puderam ser geradas diretamente. Como exemplo, tem-se a função anamnese do sistema Psychologist. Na SiGCli, essa função possui perguntas fixas (não cadastrais) voltadas para clínicas de reabilitação física que diferem das perguntas de uma clínica médica. Além disso, algumas funções foram consideradas como pertencentes a outros subdomínios, não suportados pela linguagem de padrões utilizada. Nesse contexto, enquadra-se a função Backup de Dados, do subdomínio Segurança de Dados.

Para a adaptação da primeira versão do sistema gerado a partir do sistema legado, verificou-se maior facilidade em modificar a especificação do XML contida no gerador e criar novamente a aplicação, do que alterar diretamente o sistema gerado. Assim, a cada iteração, as funções priorizadas são modificadas no gerador e uma nova versão do gerador é criada, até atingir a mesma funcionalidade do sistema legado. Assim, o gerador também passa por reengenharia, tendo-se no final do processo um novo gerador que atende a um determinado subdomínio de legado. A versão 0 desse gerador pode ser utilizada sempre que, um sistema legado possua funções nele implementadas e precisa passar por reengenharia.

Como o próprio ARA preconiza, ferramentas devem ser utilizadas para agilidade e qualidade da reengenharia. Na reengenharia realizada, cita-se como exemplo: a utilização da ferramenta CASE "ArgoUML", que facilitou a criação dos diagramas de caso de uso, o ambiente "Eclipse UML" que auxiliou na geração do diagrama de classes para o sistema alvo, e também a ferramenta "*Access to MySql"* para a migração dos dados de um sistema para outro.

CAPÍTULO 4 – PROCESSO ÁGIL DE REENGENHARIA COM GERADORES DE APLICAÇÃO

4.1 Considerações Iniciais

O Processo Ágil de Reengenharia com Geradores de Aplicações é uma adaptação do PARFAIT para atender o processo de reengenharia ágil utilizando geradores. É baseado no arcabouço ARA (Cagnin, 2005), atendendo à abordagem incremental e iterativa, reusando padrões de análise, utilizando técnicas ágeis e ferramentas para aumentar a agilidade, a qualidade e diminuir os riscos do processo.

O estudo de caso prospectivo do capítulo anterior teve a finalidade de verificar a utilização de geradores de aplicação no processo de reengenharia ágil PARFAIT. Obtêm-se assim, as adaptações necessárias para atender aos geradores de aplicação ao invés de *frameworks*.

Este capítulo está organizado da seguinte maneira: a Seção 4.2 mostra as fases e atividades que servem como guia na realização da reengenharia com geradores de aplicação; na Seção 4.3, uma comparação do PARFAIT quando utiliza *framework versus* gerador é realizada; na Seção 4.4 é apresentada a adaptação do ARA e na Seção 4.5 estão as considerações finais.

4.2 Fases e Atividades

Seguindo as diretrizes do PARFAIT (Cagnin, 2005), que utiliza abordagem incremental e iterativa, as fases para a realização do processo preconizado, são: Concepção, Elaboração, Construção e Transição. A seguir, são apresentadas essas fases e para cada uma é apresentada uma tabela com as seguintes colunas: a primeira contém uma descrição do que é a atividade, na segunda estão os objetivos que se deseja atingir com a atividade em questão, na terceira o responsável por ela e na quarta os artefatos que devem ser produzidos.

A fase de Concepção tem a finalidade de entendimento tanto do gerador de aplicação quanto do sistema legado, verificando-se por meio do domínio de ambos, se é possível a realização da reengenharia. Para essa fase as seguintes atividades são realizadas como mostrado na Tabela 18:

Tabela 18 – Fase de Concepção

Fase: **Concepção** Objetivo: Entendimento do domínio do gerador e do sistema legado, verificando-se os riscos de utilizar o gerador na reengenharia do sistema legado.			
Atividades	**Objetivos**	**Papéis**	**Artefatos produzidos / atualizados**
Familiarizar-se com o domínio do gerador (não obrigatória, caso o analista de sistema já possua familiaridade com o gerador disponível)	Analisar e entender o domínio ao qual o gerador pertence e verificar o funcionamento dos padrões da sua linguagem de padrões (SiGCLi) por meio da criação de sistemas alvo hipotéticos.	Analista de sistemas	
Observar o domínio do sistema legado em relação ao do gerador (obrigatória)	Identificar as características do sistema legado para verificar se ele pertence ao mesmo domínio do gerador ou possui as funções importantes/essenciais do domínio, a fim de utilizá-lo no projeto de reengenharia. Essa atividade também identifica se existem documentações do sistema legado.	Analista de Sistemas	Tabela contendo a listagem das funções do legado, verificando que padrões do gerador podem atender tais funções. Conforme Tabela 8 do capítulo anterior.

Na fase de Elaboração, cada função do sistema legado é estudada separadamente e de modo aprofundado a cada iteração, de acordo com priorização previamente estabelecida. As atividades da Tabela 19 são realizadas para a elaboração de casos de teste, modelo de caso de uso e especificação; aproveitando-se essa documentação na fase seguinte.

Tabela 19 – Fase de Elaboração

Fase: **Elaboração** Objetivo: Elaborar documentação para ajudar no entendimento do sistema legado e apoiar as próximas fases.			
Atividades	**Objetivos**	**Papéis**	**Artefatos produzidos / atualizados**
Desenvolver o diagrama de casos de uso e elaborar os casos de teste (obrigatória)	Entender os requisitos do sistema legado, priorizados para a iteração corrente, por meio da execução de casos de teste criados e identificados na atividade "Observar o domínio do sistema legado em relação ao do gerador".	Analista de sistemas, usuários	Documentação dos casos de teste, diagrama de casos de uso, documento de requisitos. Conforme Figuras 8, 9 e Tabela 10 do capítulo anterior.
Desenvolver o diagrama de classes do sistema alvo (obrigatória)	Obter o diagrama de classes parcial do sistema alvo gerado, por meio de ferramenta de reengenharia reversa.	Analista de Sistemas	Diagrama de classes do sistema. Conforme Figura 10 do capítulo anterior.

Na fase de Construção a cada iteração, os requisitos priorizados são atendidos pelo gerador. Após a construção, o diagrama de classes é construído e os testes são executados novamente sendo que, aqueles que não obtiverem sucesso indicam as adaptações a serem realizadas no gerador e na documentação. Modificações no XML implicam na geração do sistema alvo novamente. As atividades que compõem essa fase são apresentadas na Tabela 20.

Tabela 20 – Fase de Construção

Fase: **Construção** Objetivo: Criar o sistema alvo no paradigma orientado a objetos, relacionado aos requisitos do sistema legado, priorizados pelos usuários para a iteração corrente, e adaptá-lo para torná-lo funcionalmente compatível ao sistema legado.			
Atividades	**Objetivos**	**Papéis**	**Artefatos produzidos / atualizados**
Criar o sistema alvo no paradigma orientado a objetos (obrigatória)	Criar uma versão do sistema alvo no paradigma orientado a objetos usando o gerador, de acordo com os requisitos priorizados. A geração do sistema alvo é baseada nos padrões identificados na atividade "Observar o domínio do sistema legado em relação ao do	Programador	Sistema alvo. Conforme exemplo da Figura 10 do capítulo anterior.

	gerador".		
Executar os casos de teste no sistema alvo (obrigatória)	Executar os casos de teste do sistema legado no sistema alvo para observar se há diferença de comportamento, visando descobrir regras de negócio e requisitos específicos do sistema legado ou presentes no gerador e não requeridos pelo sistema legado.	Testador, usuários.	Resultado da execução dos casos de teste identificando as regras de negócio e requisitos específicos do legado. Conforme Figura 11 e Tabela 10 do capítulo anterior.
Adaptar o sistema alvo (obrigatória)	Regras de negócio, requisitos não cobertos pela linguagem de padrões ou fornecidos pelo gerador mas não requeridos pelo sistema legado.	Programadores	Sistema alvo, nova versão do gerador e documentação atualizada. Conforme Figura 12 do capítulo anterior.

Na fase de Transição, deve-se garantir toda documentação e qualidade do sistema alvo, como mostrado na Tabela 21. A migração de dados é realizada e os testes são executados novamente.

Tabela 21 – Fase de Transição

Fase: **Transição** Objetivo: Assegurar que o sistema alvo está pronto para ser disponibilizado a seus usuários e para ser implantado na empresa. A execução dessa fase, com exceção da primeira, pode ser feita somente depois que não houver mais requisitos do legado para serem considerados na reengenharia.			
Atividades	**Objetivos**	**Papéis**	**Artefatos produzidos / atualizados**
Converter a base de dados do sistema legado (obrigatória)	Migrar os dados do banco de dados do sistema legado para a do sistema alvo. Caso seja necessário modificar colunas, o XML é alterado e tem-se uma nova versão do gerador .	Administrador de banco de dados	Banco de dados atualizado do sistema alvo. Caso necessário, nova versão do gerador. Atualizado conforme Tabela 17 e Figuras 14, 15 do capítulo anterior.
Testar o sistema alvo (obrigatória)	Submeter o sistema alvo aos casos de teste documentados para avaliar a maturidade do produto.	Testador	Sistema alvo liberado para uso.

4.3 Comparação PARFAIT x Processo com Gerador

Nem todas as atividades do PARFAIT foram incorporadas ao processo ágil de reengenharia utilizando geradores de aplicação, por exemplo, a atividade "Elaborar o planejamento de reengenharia", não foi necessária porque o objetivo do estudo de caso foi verificar o comportamento do gerador ao longo do processo. Mesmo sem um planejamento efetivo, os requisitos foram priorizados pela sua importância ao domínio, conforme analisado na Tabela 8.

Já as atividades que foram executadas sofreram algumas adaptações devido às características do gerador serem diferentes das do *framework.* Essas alterações são mostradas na Tabela 22.

Tabela 22 – Adaptações do processo PARFAIT pela utilização de geradores

Fases	Atividade	Adaptações
Concepção	Familiarizar-se com o domínio do gerador.	Além dos objetivos preconizados no PARFAIT, aconselha-se a geração de alguns sistemas hipotéticos para: a) verificar o comportamento dos padrões e b) familizar com o processo com o gerador.
	Observar o domínio do sistema legado em relação ao do gerador.	O artefato para essa atividade deve ser uma tabela com as funções do sistema x padrões da SiGCli. Representando qual (is) padrão (ões) da SiGCli atenderá (rão) a cada função do legado.
Elaboração	Desenvolver o diagrama de casos de uso e elaborar os casos de teste.	No estudo de caso, a priorização foi realizada de acordo com a importância da função ao domínio. Mas na realização efetiva do processo deve-se levar em consideração a priorização feita pelo usuário.
	Desenvolver o diagrama de classes do sistema alvo	Da mesma forma.
Construção	Criar o sistema alvo no paradigma orientado a objetos	O sistema alvo é criado pelo gerador GAwCRe ao invés do *framework* GREN
	Executar os casos de teste no sistema alvo	Da mesma forma.
	Adaptar o sistema alvo	A adaptação do sistema alvo é realizada na especificação do XML do gerador, então a cada iteração o gerador passa por reengenharia e um novo sistema alvo é gerado. A cada nova reengenharia a ser realizada, a versão 0 do gerador é utilizada.
Transição	Converter a base de dados do sistema legado	Caso necessário, a especificação XML é alterada para atender a ordem em que estão as colunas do banco de dados do legado.
	Testar o sistema alvo	Da mesma forma.

4.4 Arcabouço de Reengenharia Ágil (ARA)

De acordo com o estudo de caso realizado e o processo ágil de reengenharia com geradores de aplicação descrito na Seção anterior, o arcabouço ágil pode ser empregado com geradores de aplicação. Na Figura 16 é apresentado o ARA (Cagnin, 2005), com algumas adaptações em relação ao original: a) a linguagem de padrões GRN (Braga; Masiero, 2002) é substituída pela linguagem de padrões SiGCli (Pazin, 2004), b) utiliza-se o gerador GAwCRe (Pazin, 2004) ao invés do *framework* GREN (Braga, 2003), c) para o controle de versão não é utilizada ferramenta, d) ferramenta de engenharia reversa é utilizada.

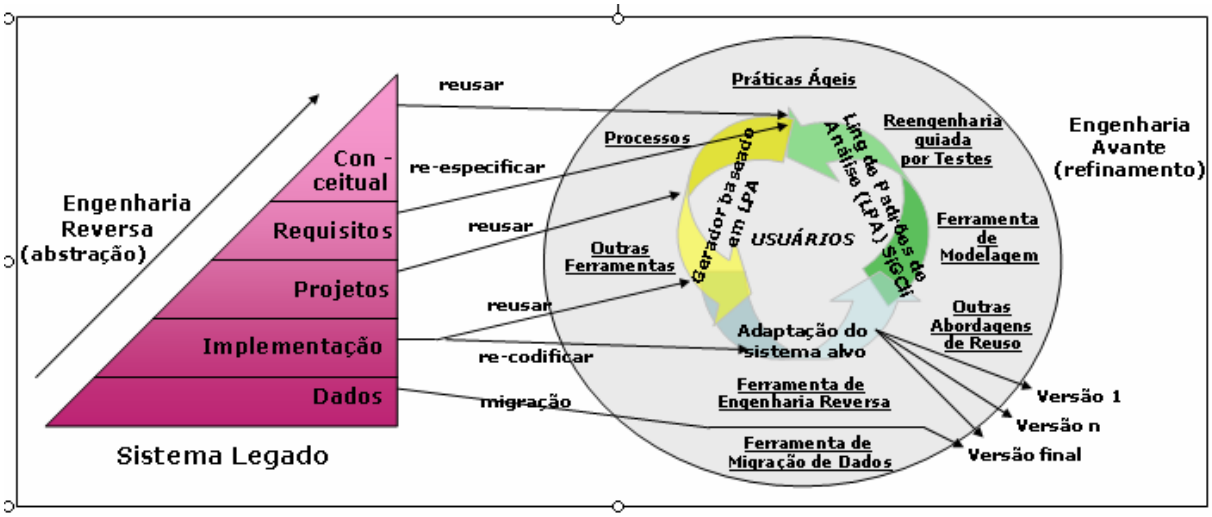

Figura 16 – ARA com Gerador

4.5 Considerações Finais

Neste capítulo foi mostrada a adaptação realizada para atender melhor a utilização de geradores de aplicação no processo PARFAIT (Cagnin, 2005). Um gerador possui características diferenciadas de um *framework*, por isso alterações fizeram-se necessárias ao longo do processo. Essa adaptação foi abstraída de um estudo de caso, mostrado no capítulo 3, realizado com quatro sistemas legados de clínicas médicas.

Algumas atividades não foram utilizadas, pelo fato de não serem necessárias quando se têm geradores ou por estarem relacionadas a atividades que referenciam a equipe de desenvolvimento. Porém, a maioria das atividades do PARFAIT foi seguida e obteve-se sucesso na sua realização com geradores. As principais alterações no processo são: (a) requisitos específicos são alterados na especificação XML, já que as modificações dos sistemas alvo mostraram-se mais ágeis dessa forma, (b) para cada sistema legado que passará pelo processo de reengenharia, a versão 0 do gerador modificado que atende a um subdomínio desse é utilizada. Assim, a linguagem de padrões SiGCli é sempre seguida alterando-se somente os requisitos específicos.

CAPÍTULO 5 – CONSIDERAÇÕES FINAIS

5.1 Considerações Iniciais

Este trabalho adaptou o Processo Ágil de Reengenharia apoiado por *Frameworks* PARFAIT (Cagnin, 2005), trocando *framework* por gerador de aplicação. Ocorreu a partir de um estudo prospectivo em quatro sistemas do domínio de clínicas médicas que passaram por reengenharia, utilizando o gerador de aplicação GAwCRe (Pazin, 2004) ao longo do processo PARFAIT. Apesar do GAwCre ser baseado na linguagem de padrões SiGCli do domínio de clínicas de reabilitação física e os sistemas legados serem do domínio de clínicas médicas, a reengenharia realiza-se com sucesso. Já que as principais funções para o funcionamento de uma clínica, que são: Cadastro do Paciente, Agendamento, Acompanhamento do Tratamento e Pagamento conforme estudo do capítulo 3 analisando a funcionalidade dos quatro sistemas, foram geradas pelo GAwCRe.

Essa adaptação é baseada no arcabouço ARA (Cagnin, 2005), cujas principais diretrizes são a) abordagem incremental e iterativa b) ferramentas para ajudar na elaboração dos artefatos c) padrões para o reuso de soluções já existentes em vários níveis de abstração d) práticas ágeis. Além dessas características, as atividades do PARFAIT servem como guia para a utilização do gerador na realização do processo. Porém, por se tratar de um gerador, o GAwCRe apresentou algumas diferenças em relação à utilização do *framework* GREN (Braga, 2003). O gerador possui a linguagem de padrões SiGCli especificada por meio de uma Linguagem de Modelagem de Aplicação (LMA) em um XML. O XML elaborado desta forma, demonstrou ser de fácil legibilidade por não necessitar de muito conhecimento prévio e, além disso, por meio de uma análise também realizada no capítulo 3, mostrou que as alterações são bem menores quando realizadas no gerador do que no sistema alvo. Adotando-se que, as alterações poderiam ser realizadas diretamente na especificação do XML, o gerador apresentou-se mais flexível às modificações, adaptando de forma mais ágil o sistema alvo.

Este capítulo está organizado da seguinte maneira: a Seção 5.2 mostra as contribuições deste trabalho; as limitações encontradas estão na Seção 5.3 e na Seção 5.4 estão os trabalhos futuros propostos.

5.2 Contribuições deste Trabalho

O trabalho realizado por Cagnin (2005) pôde ser utilizado amplamente, mesmo com a utilização de geradores de aplicação. Essa adaptação disponibiliza mais um meio de realização da reengenharia ágil, migrando sistemas legados procedimentais para sistemas alvo orientados a objetos com agilidade e qualidade.

Quando *frameworks* e geradores de aplicação são utilizados na reengenharia de sistemas, as características de ambos são passadas para o sistema alvo, já que o princípio de reutilização em vários níveis de abstração deve ser seguido. O *framework* GREN e o gerador GAwCRe utilizam linguagem de padrões assim, o sistema alvo é criado com base nos padrões de análise da linguagem de padrões em que foram concebidos, sendo o código fonte gerado na linguagem de programação em que eles foram implementados. O *framework* GREN (Braga, 2003) utilizado no processo PARFAIT é implementado na linguagem Smalltalk, que embora seja bastante difundida no meio científico, não é utilizada na prática profissional, além de não ser uma linguagem para *Web*, uma característica buscada nos sistemas atuais. Já o GAwCRe (Pazin, 2004) é construído em JAVA para *web*, uma linguagem amplamente aceita. Espera-se que com a utilização desse gerador, mais sistemas legados passem pelo processo ágil de reengenharia e acabem assim com seus problemas de manutenção, migrando para uma tecnologia mais nova.

O *framework* GREN exige do desenvolvedor conhecimento profundo de seu funcionamento e da linguagem *SmallTalk* para implementar os requisitos específicos do sistema legado. Já o gerador GAwCRe possui em seu código fonte os gabaritos necessários para realizar a substituição de valores especificados em XML; que com base em *tags* e atributos de *tags*, representa tanto a LMA, quanto a linguagem de padrões SiGCli. Assim, o conhecimento fica basicamente restrito à especificação XML, que por sua legibilidade torna as modificações mais fáceis de serem realizadas. Além de que, pela análise realizada das adaptações do sistema alvo, verificou-se que as alterações na especificação XML requerem bem menos passos do que na aplicação gerada.

Cagnin (2005) preocupa-se em seu trabalho com a verificação de quando o *framework* deve passar por reengenharia. Aconselha que, a construção de um requisito específico é realizada no sistema alvo, porém, quando esse requisito atinge mais de um sistema legado, uma análise deve ser realizada para verificar se ele não pertence ao domínio em questão. Assim, o *framework* engloba um novo requisito sempre que se comprova que ele é importante ao domínio, já que aparece em vários legados. Neste trabalho também houve essa

preocupação, porém pela flexibilidade encontrada na especificação XML, optou-se por fazer, sempre que possível, as modificações no gerador garantindo mais agilidade ao processo.

Seguindo a proposta do ARA para a utilização de ferramentas, sugere-se neste trabalho uma ferramenta de engenharia reversa para a criação do diagramas de classe do sistema alvo a cada iteração. Facilitando, assim o trabalho do programador e agilizando o processo.

A atividade "Desenvolver o diagrama de classes do sistema" poderia ter sido realizada utilizando-se uma ferramenta de engenharia reversa, como por exemplo a Omondo do Eclipse, para que o processo adquirisse maior agilidade. Porém optou-se por elaborar a cada iteração na fase de elaboração para obter uma maior visibilidade das funções a serem geradas no gerador do sistema legado.

Nem todas as atividades do PARFAIT foram utilizadas, já que durante o processo ágil de reengenharia com geradores de aplicação, não se fizeram necessárias. A atividade "Documentar as modificações realizadas no diagrama de classes", que tem como finalidade expressar as diferenças entre o diagrama de classe do framework e o diagrama adaptado para o legado em questão. Com o gerador de aplicações essas modificações no diagrama não são significativas já que para as próximas reengenharias sempre se utiliza a versão 0 do gerador, com o diagrama de classes correspondente ao da linguagem de padrões SiGCli, para realizar a próxima iteração.

5.3 Limitações do Trabalho

Algumas limitações foram encontradas ao longo deste trabalho. A primeira delas foi a inexistência de sistemas legados no domínio de clínicas de reabilitação física (sistemas específicos para clínicas de fisioterapia, terapia ocupacional e educação física) na *Internet* para a realização do estudo de caso prospectivo. Optou-se por sistemas que pertenciam a um domínio mais abrangente e, conseqüentemente, mais fáceis de se encontrar, o de clínicas médicas. Porém, o gerador não cria algumas funções dos legados, porque: a) não pertencem ao domínio restrito de clínicas de reabilitação física, por exemplo, o cadastro de anamnese do sistema Psychologist que é um cadastro genérico de perguntas e respostas, b) não atende a alguns subdomínios conexos, como por exemplo, o de Segurança de Dados, não gerando a função de Backup de Dados; apesar do gerador GAwCRe abranger funções do subdomínio conexo que cuidam da área financeira da clínica.

A extração de relatórios é importante em grande parte dos sistemas, por isso alguns relatórios foram implementados e outros especificados no gerador. Porém durante o estudo de caso prospectivo verificou-se que o gerador GAwCRe não está preparado para produzir relatórios que necessitem de informações de mais de uma tabela, ou seja, ele não consegue extrair informações de tabelas cruzadas. Para tal, o código fonte do gerador deve ser modificado, e assim pode ser utilizado em vários sistemas que passam por reengenharia com gerador, porém essa reengenharia do gerador não foi englobada neste trabalho. Dessa forma, é importante ressaltar que como essa função, outras podem existir e, implicar em mudanças no código fonte do gerador. Ressalta-se que, neste trabalho as adaptações necessárias não foram numerosas e complexas o suficiente para modificar a estrutura do gerador.

.

5.4 Trabalhos Futuros

A continuação deste trabalho poderá possibilitar que pontos deixados em aberto possam ser analisados de forma a consolidar as diretrizes aqui apresentadas. Algumas sugestões são:

- Verificar a possibilidade da linguagem de padrões SiGCli (Pazin, 2004) englobar subdomínios encontrados nos sistemas legados estudados;

- A partir da ampliação de domínio da SiGCli, atualizar também o gerador de aplicações GAwCRe (Pazin, 2004) para atender a esse domínio mais abrangente, possibilitando que os sistemas de mercado (como verificado no estudo de caso), possam passar por reengenharia com geradores, com maior facilidade;

- Alterar o GAwCRe para atender relatórios com informações em tabelas cruzadas;

- Verificar a existência de ferramentas que possam atender ao processo de reengenharia ágil com geradores de aplicação;

- Utilizar ferramenta de controle de versão para um melhor atendimento às práticas ágeis no processo ágil de reengenharia com geradores de aplicação.

REFERÊNCIAS BIBLIOGRÁFICAS

Abrahamsson et. al. (2003) — Abrahamsson, P.; Warsta, J.; Siponen, M.T.; Ronkainen, J. *New Diretctions on Agile Methods: a Comparative Analylis. In: ICSE'2003, 25th International Conference on Software Engineering*. 2003.

(Alexandre, 1979) — Alexander, C. *The Timeless Way of Building*. Oxford University Press. 1979.

(Ambler, 2001) — Ambler, S. Agile Documentation. Disponível em: http://www.agilemodeling.com/essays/agileDocumentation.htm. Acessado em 15/10/2004.

(Ambler, 2002) — Ambler, S. *Agile Modeling*, John Wiley and Sons, 2002.

(ArgoUML, 2006) — ArgoUML. Disponível em: www.argouml.tigris.org. Acessado em: 10/07/2006.

(Beck, 1999) — Beck, K. *Extreme Programming Explained: Embracing Change*, Addison Wesley, 1999.

(Beck, 2000) — Beck, K; Fowler, Martin. *Planning Extreme Programming*, Addison-Wesley, ISBN 0201710919, October 2000

(Beck et. al, 2001) — Beck, K; Beedle, M.; Bennekum, A.; Cockburn, A., Cunningham, W. et. al. Manifesto for agile development. 2001. Disponível em: http://www.agilemanifesto.org. Acesso em 10/12/2004.

(BioManager, 2006) — BioManager – Engenharia de Software. Disponível em: http://www.biomanager.com.br/. Acessado em 15/01/2006.

(Braga, 2002) — Braga, R. T. V.; Germano, F. S. R. e Masiero, P. C. GRN: Uma Linguagem de Padrões para Sistemas de Gestão de Recursos de Negócios - extensão para englobar o armazenamento de recursos e tratamento mais específico para pagamento de transações. Versão em Português Word Revisada em Julho/2002.

(Braga, 2003) — Braga, R. T. V. Um processo para a Construção e Instanciação de Frameworks baseado em uma Linguagem de Padrões para um Domínio Específico. Tese de Doutorado, ICMC/USP, São Carlos-SP, 2003.

(Cagnin, 2003a) — Cagnin, M. I., Maldonado, J. C., Germano, F. S., Chan, A., and Penteado, R. D. (2003a). Um estudo de caso de reengenharia utilizando o processo PARFAIT. In *SDMS'2003, Simpósio de Desenvolvimento e Manutenção de Software da Marinha*. Publicação em CD-ROM.

(Cagnin, 2003b) — Cagnin, M. I., Maldonado, J. C., Germano, F. S., and Penteado, R. D. (2003b). PARFAIT: *Towards a framework-based agile reengineering process. In ADC'2003, Agile Development Conference*, pages 22–31. IEEE.

(Cagnin,2004) — Cagnin, M.I.; Penteado, R.; Germano, F.; Maldonado, J.C. Evolução do PARFAIT: Um Processo de Reengenharia de Software Baseado em Framework. In: IV Simpósio de Desenvolvimento e Manutenção de Software da Marinha. CD-ROM, 12 pág., Rio de Janeiro-RJ.

(Cagnin,2005) Cagnin, M.I. PARFAIT: Uma Contribuição para a Reengenharia de Software baseada em Linguagens de Padrões e *Frameworks*. Tese de Doutorado – Instituto de Ciências Matemáticas e de Computação – ICMC / USP, São Carlos – SP, 2005.

(Cagnin,2006) Cagnin, M.I.; Maldonado, J. C. Uma Contribuição para a Reengenharia de Software baseada em Linguagens de Padrões e *Frameworks*. X Simpósio de Teses e Dissertações, Instituto de Ciências Matemáticas e de Computação, São Carlos-SP, Março, 2006

(Canfora et. al, 1995) Canfora, G.; Fasolino, A.R.; Tortorella, M.; *Towards reengineering in reuse reengineering processes*. IEEE Software. Páginas 17-20. 1995.

(Cleaveland, 1988) Cleaveland, J. C. *Building Application Generators*. IEEE Software, 25-33, Jul. 1988

(Chikofsky, 2000) Chikofsky, J. E.; Cross, J. H. *Reverse Engineering and Design Recovery: A Taxonomy*. IEEE Software. Volume 7, número 1, páginas 13-17. 1990.

(CMM, 2005) *Capability Maturity Model for Software* – Disponível em: http://www.sei.cmu.edu/cmm/cmm.html. Acesso em: 24/10/2004.

(CMMI, 2005) *Capability Maturity Model for Software Integrated* – Disponível em: http://www.sei.cmu.edu/cmm/cmms/cmms.integration.html. Acesso em 15/01/2005.

(CMMI-SW, 2002a) *CMMI for Software Engineering, Version 1.1, Continuous Representation*, CMU/SEI-2002-TR-028.2002 Disponível em: http://www.sei.cmu.edu/publications/documents/02.reports/02tr028.html. Acesso em: 15/01/2005.

(CMMI-SW, 2002b) CMMI for Software Engineering, Version 1.1, Staged Representation. CMU/SEI-2002-TR-029, 2002. Disponível em: http://www.sei.cmu.edu/publications/documents/02.reports/02tr029.html. Acesso em: 15/01/2005.

(Cockburn, 2001) Cockburn, A. *Agile Software Development*, Addison-Wesley, ISBN 0201699699, December 2001

(Cohn, 2004) Cohn M. *User Stories Applied : For Agile Software Development*. Addison-Wesley. 2004

(Coplien, 1998) Coplien, J. O. *Software design patterns: Common questions and answers* in L. Rising – The Patterns Handbook: Techniques, Strategies, and Applications, Cambridge University Press, p. 311–320, 1998.

(Demeyer et. al., 1999) Demeyer, S.; Ducasse, S; Nierstrasz, *O. A Pattern Language for Reverse Engineering. Proceedings of the 4th European 5th European Conference on Pattern Languages of Programming and Computing*. Paul Dyson (Ed.) Universitätsverlag Konstanz GmbH, Konstanz, Germany. July 1999.

(Demeyer et. al., 2000) Demeyer, S.; Ducasse, S; Nierstrasz, O. *A Pattern Language for Reverse Engineering. Proceedings of the 5th European Conference on Pattern Languages of Programming and Computing*. Andreas Rüping (Ed.). 2000.

(Durscki et. al, 2004) Durscki R. C.; Spinola M. M.; Burnett, R. C., Reinehr, S. S. Linhas de Produto de Software: riscos e vantagens de sua Implantação. VI Simpósio Internacional de Melhoria de Processos de Software. São

	Paulo. 2004.
(D&D Consultoria, 2006)	D&D Consultoria. Disponível em: http://www.dedconsultoria.com.br/. Acessado em 20/02/2006.
(Eclipse, 2006)	Eclipse - Disponível em: www.eclipse.org. Acessado em: 10/07/2005.
(Fayad et. al, 1999)	Fayad,M. E.; Johnson, R.; Schmidt, D. C. *Building application frameworks: Objectoriented foundations of framework design*. John Wiley & Sons, 1999.
(Franca; Staa, 2001)	França, L. P. A.; Staa, A. V. Geradores de Artefatos: Implementação e Instanciação de *Frameworks*. In: Anais do XV SBES-2001- Simpósio Brasileiro de Engenharia de Software. Rio de Janeiro. 302-315 p., 2001.
(Gamma, 1995)	Gamma, E., Helm, R., Johnson, R., Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented software*. Addison-Wesley, 1995.
(ISO, 1995)	ISO. NBR ISSO/IEC 12207:1995. Tecnologia da Informação – processos de ciclo de vida de software. Associação Brasileira de Normas Técnicas, 1995.
(Johnson, R. E.; Foote, B., 1998)	Johnson, Ralph E.; Foote B. *Designing Reusable Classes. Journal of Object Oriented Programming* – JOOP, 1(2):22-35, Junho/Julho 1988.
(Kruchten, 2001)	Kruchten, P. Agility with the RUP. Cutter IT Journal, 2001..
(Lemos, 2002)	Lemos, G. S. PRE/OO – Um processo de Reengenharia Orientado a Objetos com Ênfase na Garantia da Qualidade. Dissertação de Mestrado – Programa de Pós Graduação em Ciência da Computação – Universidade Federal de São Carlos, São Carlos – SP, 2002.
(Lycett et. al., 2003)	Lycett, M; Macredie, R. D.; Patel, C.; Paul R. J. *Migrating Agile Methods to Standardized Development Practice*. In: Computer, pp. 79-85. IEEE Computer Society. Junho de 2003.
(Mangan et. al., 2004)	Mangan M. A. S.; Vargas P. K.; Azzolin D. Orientação a Objetos. Disponível em: http://www.portaljava.com.br/home/modules.php?name=Content&pa=showpage&pid=49. Acessado em:18/03/42005.
(Martin, 2001)	Martin, Robert C. *Agile Process. 2001.*
(Medical Office, 2006)	Site do sistema Medical Office. Disponível em www.medicaloffice.com.br. Acessado em: 20/02/2006.
(MySql, 2006)	MySql. Disponível em: www.mysql.com. Acessado em: 15/08/2006.
(Neighbors, 1984)	Neighbors, J.M. *The Draco approach to Constructing Software from Reusable Components. IEEE Transactions on Software Engineering*. v.se-10, n.5, pp.564-574, September, 1984.
(Newkirk J; Martin R. C., 2001)	Newkirk, J; Martin R. C. *Extreme Programming in Practice*, Addison Wesley, 2001
(Omondo, 2006)	Omondo – Eclipse UML. Disponível em www.omondo.com. Acessado em: 05/09/2006.
(Palmer, S. R.; Felsing, J. M., 2002)	Palmer, S. R.; Felsing, J. M. *A Practical Guide to Feature-Driven Development. Upper Saddle River, MJ,* Prentice-Hall. 2002
(Paulk et. al, 1993)	Paulk, M. C.; Curtis, B.; Chrissis, M. B.; Weber, C. V. Capability Maturity Model for Software, Version 1.1. Software Engineering Institute, CMU/SEI-93-TR-24. 1993. Disponível em:

http://www.sei.cmu.edu/cmm/cmm.html. Acesso em: 24/10/2004.

(Paulk, 2001) Paulk, M. C. XP from a CMM Perspective, November/December 2001 issue of IEEE Software

(Pazin, 2004) Pazin, A. Um Gerador de Aplicações para o Domínio de Clínicas de Reabilitação. Dissertação de Mestrado – Programa de Pós Graduação em Ciência da Computação – Universidade Federal de São Carlos, São Carlos – SP, 2004.

(Pazin et. al, 2004) Pazin, A., Penteado, R. A. D., Masiero, P.C. SiGCli: *A Pattern Language for Rehabilitation Clinics Management*. : 4ª Conferencia Latino-Americana em Linguagem de Padrões para Programação (SugarLoafPlof), Porto das Dunas – CE, Brasil, 2004.

(Prado, 1992) Prado, A.F.; "Estratégia de Engenharia de Software Orientada a Domínios". Rio de Janeiro, 92. Tese de Doutorado. Pontificia Universidade Católica, p.333.

(Rational, 2004) Rational Corporation. *Unified Modeling Language.* http://www.rational.com/uml – Acessado em 12/09/2004.

(Ré, 2002) Ré, R.; Braga, R. T.V.; Masiero, P. C. *A pattern language for online auctions. In: 8th Pattern Languages of Programs Conference* (PloP'2001), Monticello – IL, USA, 2001.

(Recchia, 2002) Recchia, E. L. Engenharia Reversa e Reengenharia Baseadas em Padrões. Dissertação (Mestrado em Ciência da Computação). Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de São Carlos – São Carlos/SP. Junho 2002.

(Recchia et. al, 2002) Recchia, E. L.; Penteado, R. A. D. *FaPRE/OO*: Uma Família de Padrões para Reengenharia Orientada a Objetos de Sistemas Legados Procedimentais.
SugarloafPLoP2002 – The Second Latin American Conference on Pattern Languages of Programming. Itaipava, Rio de Janeiro. Agosto, 2002.

(Rizzo, 2005) Rizzo, J. A. Rizzo. Validação de Processos Ágeis de Reengenharia e de Geradores de Aplicações
Relatório de Iniciação Científica. PIBIC/CNPq-UFSCar. 2005

(Rocha et.al, 2001) Rocha, A. R. C.; Maldonado, J. C.; Weber, K. C. Qualidade dos produtos de software: teoria e prática. São Paulo: Prentice Hall, 2001.

(Rosenberg L. H., 2004) Rosenberg L. H. http://satc.gsfc.nasa.gov/support/reengrpt.pdf. Acessado em 10/11/2004.

(Schwaber,1995) Schwaber, K. "*SCRUM Development Process", OOPSLA'95 Workshop on Business Object Design and Implementation. Springer-Verlag.* (1995).

(Shafer, 1991) Shafer, D. *Practical Smalltalk : using Smalltalk. Springer-Verlag*, 1991.

(Smaragdakis; Batory, 1998) Smaragdakis Y.; Batory D. *Application Generatos. Department of Computer Sciences. The Universit f Texas at Austin.* 1998

(SPICE, 1998) SPICE. ISO/IEC 15504. *Software Process Assessment, Parts 1-9. Technical Report*. 1998.

(Taligent, 1993) Taligent, I. *Leveraging object-oriented frameworks*. 1993. White-paper. Disponível em: <http://www.ibm.com/java/education/oobuilding/index.htm>.

(Tesche & Tesche & Vasconcelos – Fornecendo Soluções em Software.

Vasconcelos, 2006) Disponível em: http://www.tvsistemas.com.br/. Acessado em 20/02/2006.

(XML, 2006) EXtensible Markup Language – Disponível em: http://www.w3schools.com/xml/. Acessado em 10/06/2006.

(Weiss, 1999) Weiss, D.; Lai, C. T. R. *Software Product-Line Engineering: a family-based software development process. Ed. Addison Wesley*, 1999.

(Wells, 2003) Wells, D. *Donwt Solve a Problem Before You Get to It. In: IEEE Software, pp. 45-47. IEEE Computer Society*. Maio/Junho de 2003.

APÊNDICE I – LINGUAGEM DE PADRÕES SIGCLI

A linguagem de padrões SiGCli é composta por 9 padrões, de acordo com Figura 17, aborda o domínio de clínicas de reabilitação, incluindo clínicas de fisioterapia, terapia ocupacional e educação física.

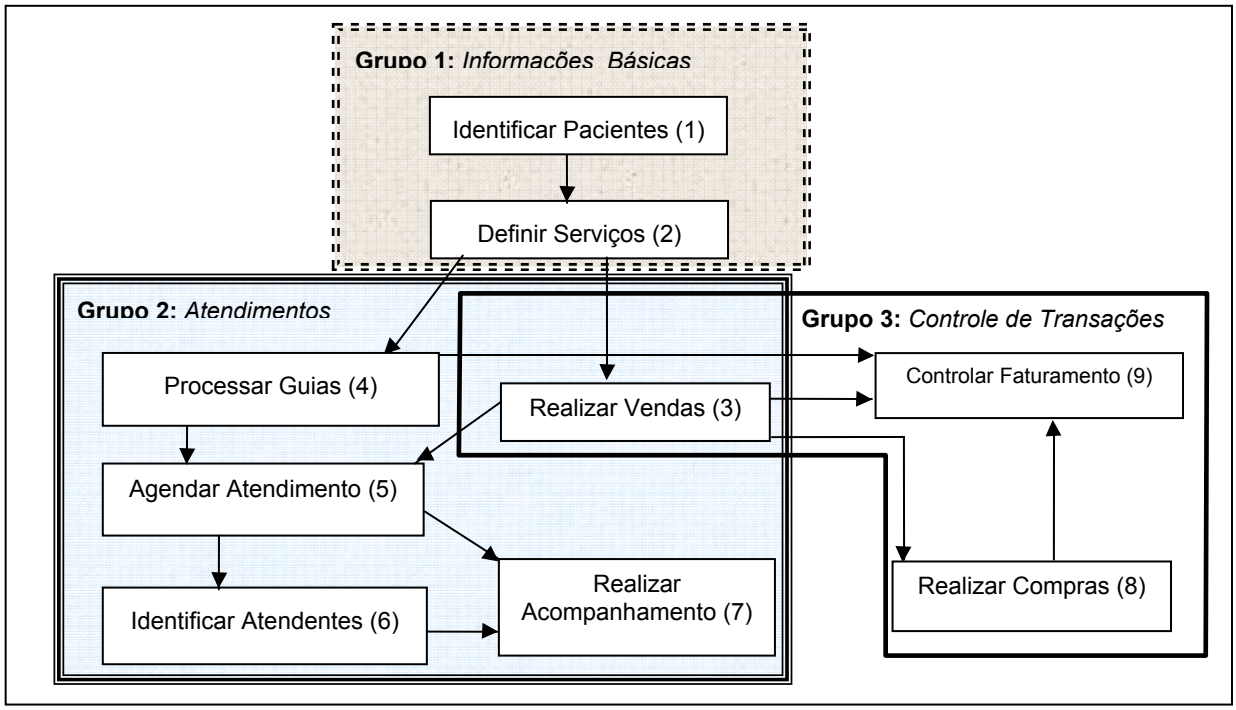

Figura 17 – SiGCli (Pazin, 2004)

Os padrões estão agrupados em 3 grupos, de acordo com seu contexto. Sendo que, o grupo 1 atende as informações básicas das clínicas, o grupo 2 trata o gerenciamento do atendimento e o grupo 3 realiza o controle financeiro.

Os padrões são apresentados a seguir, com o seguinte formato, de acordo com Pazin et. al (2004): Objetivo, Padrões Usados na GRN, Outros Padrões Relacionados, Estrutura do Padrão, Equivalências Adotadas, Classes Adicionais (não mapeadas na GRN).

Padrão 1: Identificar Pacientes

Objetivo

As pessoas que buscam tratamento em uma clínica de reabilitação, chamadas de pacientes, precisam ser identificadas e registradas no sistema por meio deste padrão.

Padrões Usados da GRN

Na Tabela 23 são apresentados os padrões da GRN usados para a definição do padrão Identificar Pacientes.

Tabela 23 – Padrões da GRN usados para definir o padrão *Identificar Pacientes (Pazin, 2004)*

Nº	*Padrão*	*Variante*
1	Identificar o Recurso	Múltiplos tipos de recursos
2	Quantificar o Recurso	Recurso Simples

Outros Padrões Relacionados

Type Object (Johnson;Woolf, 1998) e Observation (Fowler,1997).

Estrutura do Padrão

Na Figura 18 é apresentado o diagrama de classes para o padrão Identificar Pacientes.

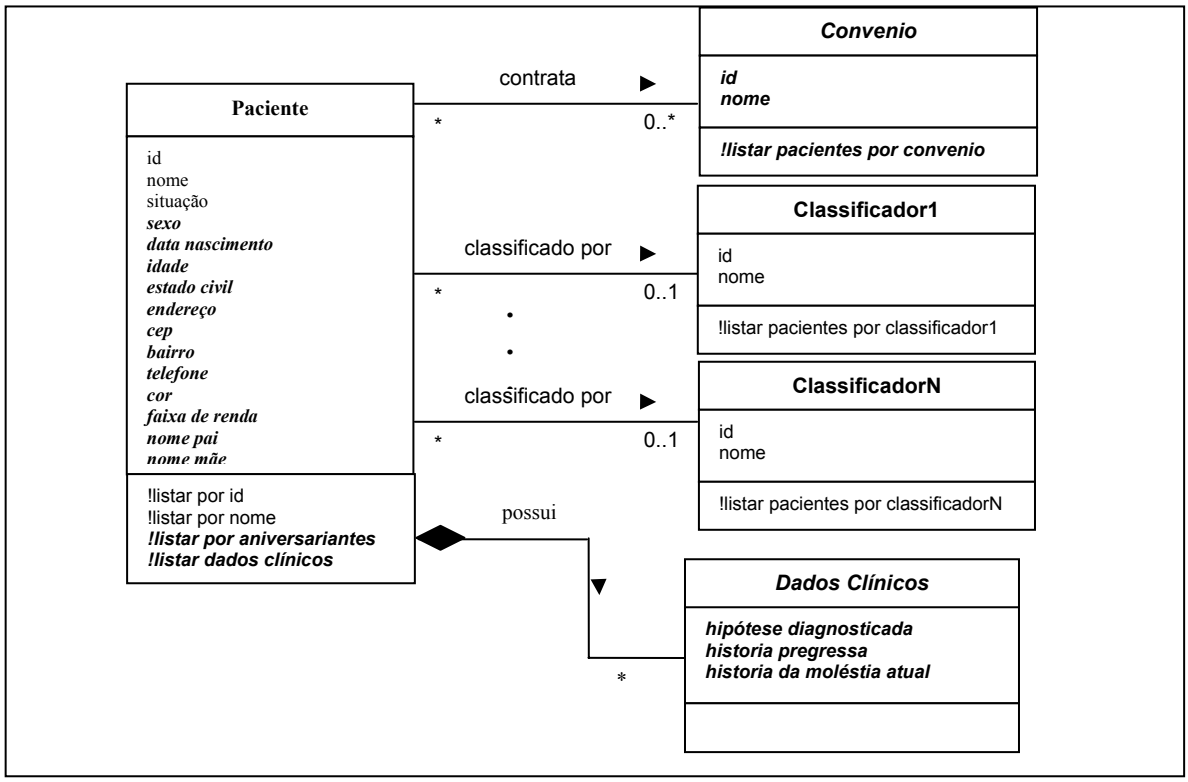

Figura 18 – Diagrama de classes para o padrão Identificar Pacientes (Pazin, 2004)

Equivalências Adotadas

Na Tabela 24 são apresentadas as equivalências adotadas entre as classes da SiGCli com as classes da GRN para definir o padrão Identificar Pacientes.

Tabela 24 – Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão *Identificar Pacientes* (Pazin, 2004)

Classes (SiGCli)	*Padrões GRN*	*Classes*
Paciente	(1) Identificar o Recurso e	Recurso
Classificador ...	(2) Quantificar o Recurso	Tipos de Recurso

Classes adicionais (não mapeadas na GRN)

Convênio e Dados Clínicos.

Padrão 2: Definir Serviços

Objetivo

Os serviços realizados pela clínica também devem ser identificados e armazenados, para que o paciente possa utilizar um determinado serviço (tratamento) oferecido pela clínica. Caso haja necessidade, pode-se classificar o serviço por tipo de serviço.

Padrões Usados da GRN

Na Tabela 25 são apresentados os padrões da GRN usados para a definição do padrão Definir Serviços.

Tabela 25 – Padrões da GRN usados para definir o padrão Definir Serviços (Pazin, 2004)

Nº	*Padrão*	*Variante*
1	Identificar o Recurso	Múltiplos tipos de recursos
2	Quantificar o Recurso	Recurso Simples

Outros Padrões Relacionados

Type Object (Johnson;Woolf, 1998).

Estrutura do Padrão

Na Figura 19 é apresentado o diagrama de classes para o padrão Definir Serviços

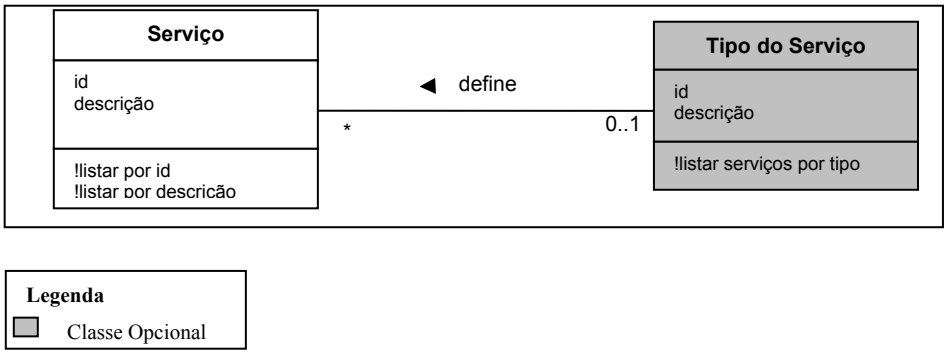

Figura 19 – Diagrama de classes para o padrão Definir Serviços (Pazin, 2004)

Equivalências Adotadas

Na Tabela 26 são apresentadas as equivalências adotadas entre as classes da SiGCli com as classes da GRN para definir o padrão Definir Serviços.

Tabela 26 – Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão *Definir Serviços (Pazin, 2004)*

Classes (SiGCli)	*Padrões GRN*	*Classes*
Serviço	(1) Identificar o Recurso e	Recurso
Tipo do Serviço	(2) Quantificar o Recurso	Tipos de Recurso

Classes adicionais (não mapeadas na GRN)

Não há classes adicionais.

Padrão 3: Realizar Vendas

Objetivo

Uma clínica pode comercializar serviços e produtos. Os serviços são realizados pela clínica de um agendamento prévio. Um produto, quando comercializado, passa da clínica ao poder do paciente. Estes produtos devem então, ser identificados e previamente cadastrados no sistema, para que seja possível fazer seu controle. Desta forma o gerenciamento comercial da clínica é realizada.

Padrões Usados da GRN

Na Tabela 27 são apresentados os padrões da GRN usados para a definição do padrão Realizar Vendas.

Tabela 27 – Padrões da GRN usados para definir o padrão *Realizar Vendas (Pazin, 2004)*

Nº	*Padrão*	*Variante*
1	Identificar o Recurso	Default
2	Quantificar o Recurso	Recurso Mensurável
6	Comercializar o Recurso	Venda sem origem
11	Itemizar Transação do Recurso	Default

Estrutura do Padrão

Na Figura 20 é apresentado o diagrama de classes para o padrão Realizar Vendas.

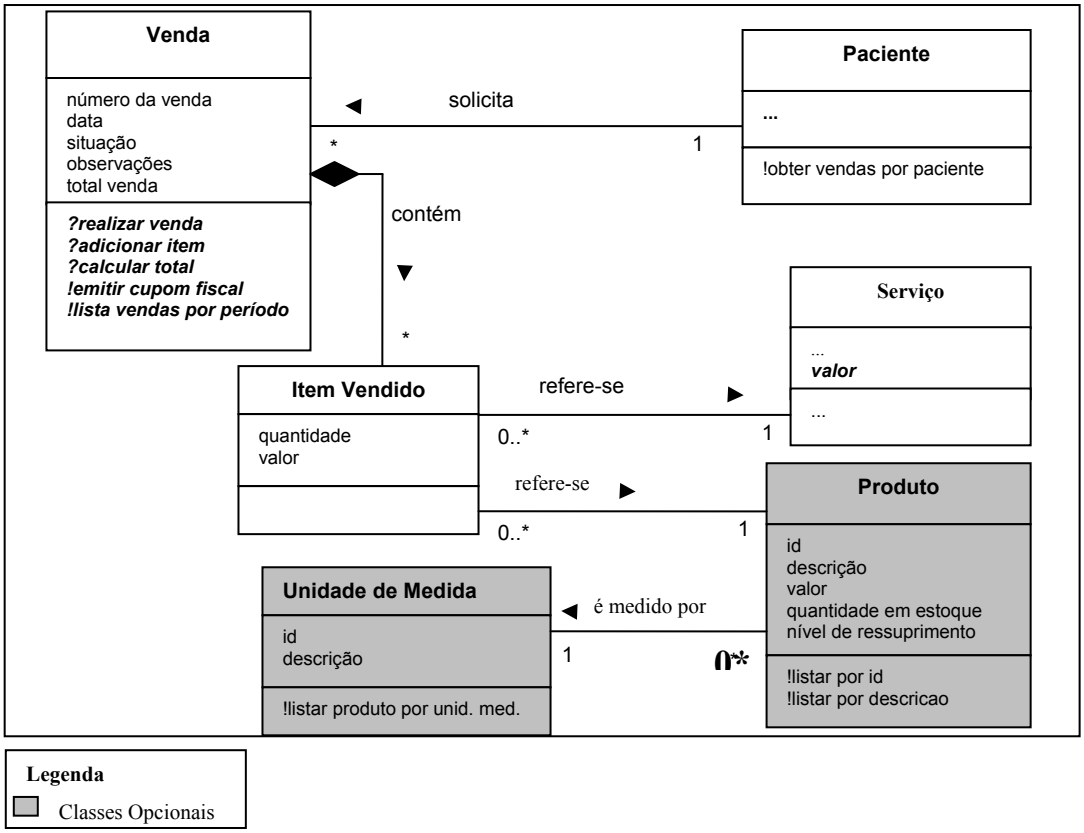

Figura 20 – Diagrama de classes para o padrão Realizar Vendas (Pazin, 2004)

Equivalências Adotadas

Na Tabela 28 são apresentadas as equivalências adotadas entre as classes da SiGCli com as classes da GRN para definir o padrão Realizar Vendas.

Tabela 28 – Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão Realizar Vendas

Classes (SiGCli)	*Padrões GRN*	*Classes*
Produto	(1) Identificar o Recurso e (2) Quantificar o Recurso	Recurso
Unidade de Medida		Unidade de Medida
Paciente	(6) Comercializar o Recurso	Destino
Venda		Comercialização do Recurso
Serviço / Produto		Recurso
Venda	(11) Itemizar Transação do Recurso	Transação do Recurso
Item Vendido		Item de transação
Serviço / Produto		Recurso

Classes adicionais (não mapeadas na GRN)

Não há classes adicionais.

Padrão 4: Processar Guias

Objetivo

Guias de convênios são encaminhadas por médicos para a realização do tratamento de seus pacientes. Por meio delas, uma clínica pode realizar atendimentos aos pacientes, possibilitando o atendimento e agendamento dos pacientes na clínica.

Padrões Usados da GRN

Na Tabela 29 é apresentado o padrão da GRN usados para a definição do padrão Processar Guias.

Tabela 29 – Padrões da GRN usados para definir o padrão Processar Guias (Pazin, 2004)

Nº	*Padrão*	*Variante*
9	Manter Recurso	Default

Estrutura do Padrão

Na Figura 21 é apresentado o diagrama de classes para o padrão Processar Guias.

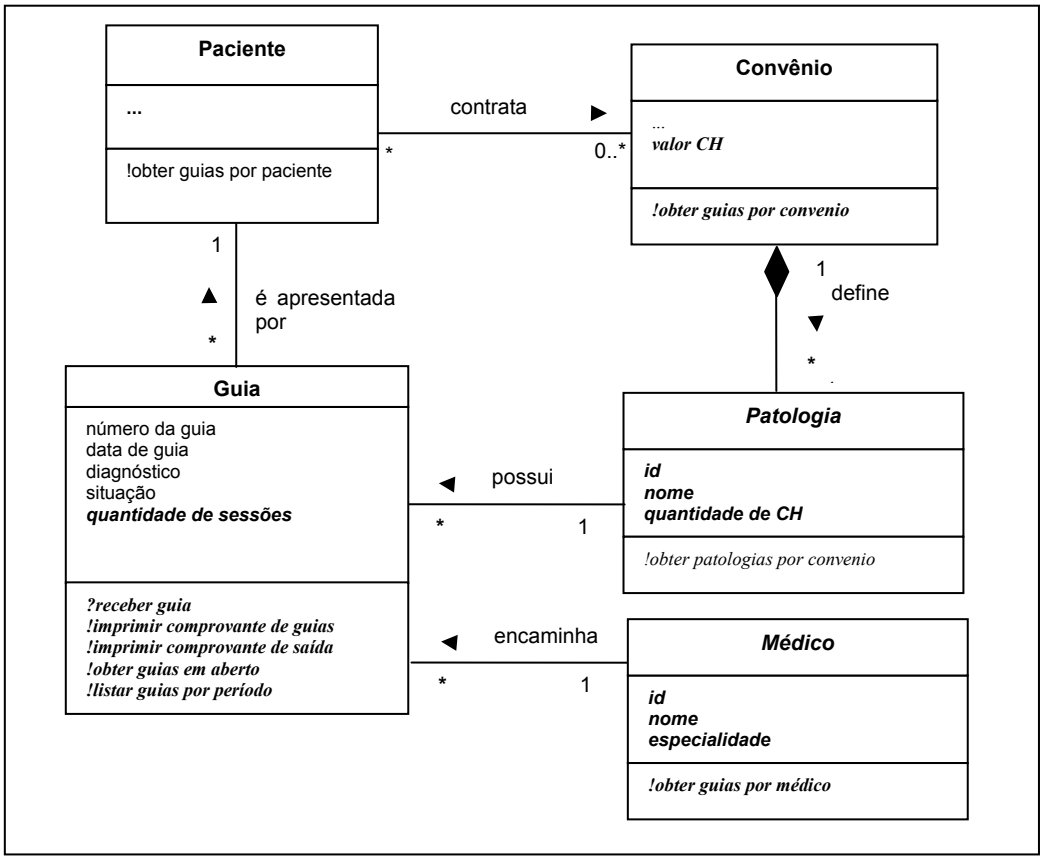

Figura 21 – Diagrama de classes para o padrão Processar Guias (Pazin, 2004)

Equivalências Adotadas

Na Tabela 30 são apresentadas as equivalências adotadas entre as classes da SiGCli com as classes da GRN para definir o padrão Definir Serviços.

Tabela 30 – Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão Processar Guias (Pazin, 2004)

Classes (SiGCli)	*Padrão GRN*	*Classes*
Paciente	(9)Manter Recurso	Recurso
Guia		Manutenção do Recurso
Convênio		Origem

Classes adicionais (não mapeadas na GRN)

Patologia, Médico

Padrão 5: Agendar Atendimentos

Objetivo

A agenda de uma clínica é imprescindível para o controle de seus atendimentos. Um paciente só pode ser atendido pela clínica quando tiver as sessões agendadas.

Padrões Usados da GRN

Na Tabela 31 é apresentado o padrão da GRN usados para a definição do padrão Agendar Atendimentos.

Tabela 31 – Padrões da GRN usados para definir o padrão Agendar Atendimentos (Pazin, 2004)

Nº	*Padrão*	*Variante*
14	Identificar Tarefas de Manutenção	Default

Estrutura do Padrão

Na Figura 22 é apresentado o diagrama de classes para o padrão Agendar Atendimento.

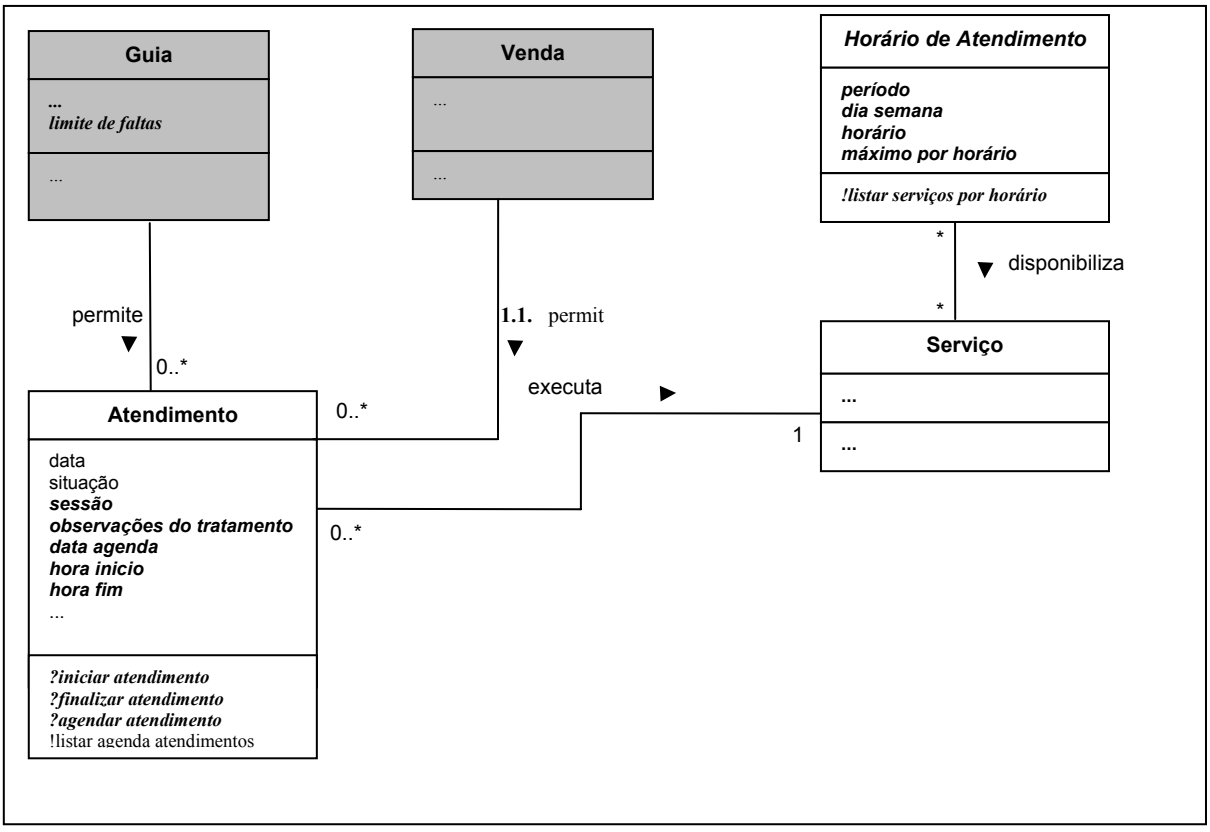

Figura 22 – Diagrama de classes para o padrão Agendar Atendimentos (Pazin, 2004)

Equivalências Adotadas

Na Tabela 32 são apresentadas as equivalências adotadas entre as classes da SiGCli com as classes da GRN para definir o padrão Agendar Atendimento.

Tabela 32 – Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão Agendar Atendimento (Pazin, 2004)

Classes (SiGCli)	*Padrão GRN*	*Classes*
Venda	(14) Identificar Tarefas de Manutenção	Comercialização do Recurso
Guia		Manutenção do Recurso
Atendimento		Tarefa de Manutenção

Classes adicionais (não mapeadas na GRN)

Horário de Atendimento e Serviço.

Padrão 6: Identificar Atendentes

Objetivo

Os atendentes são profissionais que trabalham em um clinica, podem ser: fisioterapeutas, terapeutas ocupacionais, professores de educação física, estagiários entre outros.

Padrões Usados da GRN

Na Tabela 33 é apresentado o padrão da GRN usados para a definição do padrão Identificar Atendentes.

Tabela 33 – Padrões da GRN usados para definir o padrão Identificar Atendentes (Pazin, 2004)

Nº	*Padrão*	*Variante*
13	Identificar o executor da Transação	Default

Estrutura do Padrão

Na Figura 23 é apresentado o diagrama de classes para o padrão Identificar Atendentes.

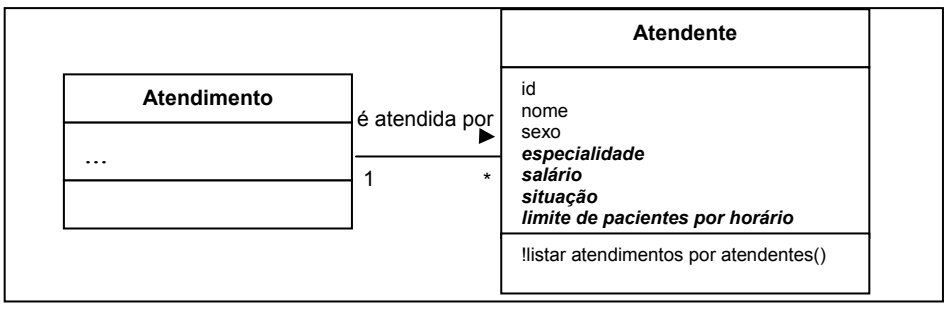

Figura 23 – Diagrama de classes para o padrão Identificar Atendentes (Pazin, 2004)

Equivalências Adotadas

Na Tabela 34 são apresentadas as equivalências adotadas entre as classes da SiGCli com as classes da GRN para definir o padrão Identificar Atendentes.

Tabela 34 – Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão Identificar Atendentes (Pazin, 2004)

Classes (SiGCli)	*Padrão GRN*	*Classes*
Agenda	(13) Identificar o executor da Transação	Tarefa de Manutenção
Atendente		Executor da Tarefa

Classes adicionais (não mapeadas na GRN)

Não há classes adicionais.

Padrão 7: Realizar Acompanhamento

Objetivo

O armazenamento dos tratamentos realizados nos pacientes, faz-se necessário para que se tenha um histórico da evolução dos pacientes. Com esses dados os atendentes podem avaliar qual o melhor tratamento para uma determinada lesão.

Padrões Usados da GRN

A GRN não prevê o acompanhamento detalhado de seus recursos, assim sendo nenhum padrão da GRN é usado.

Estrutura do Padrão

Uma clínica pode realizar avaliações neurológica, cardiológica, ortopédica, nutricional, anamnese, dentre outras. Essas avaliações específicas devem ser classes filhas (especializações) da classe avaliação. Na Figura 24 é apresentado o diagrama de classes para o padrão Realizar Acompanhamento.

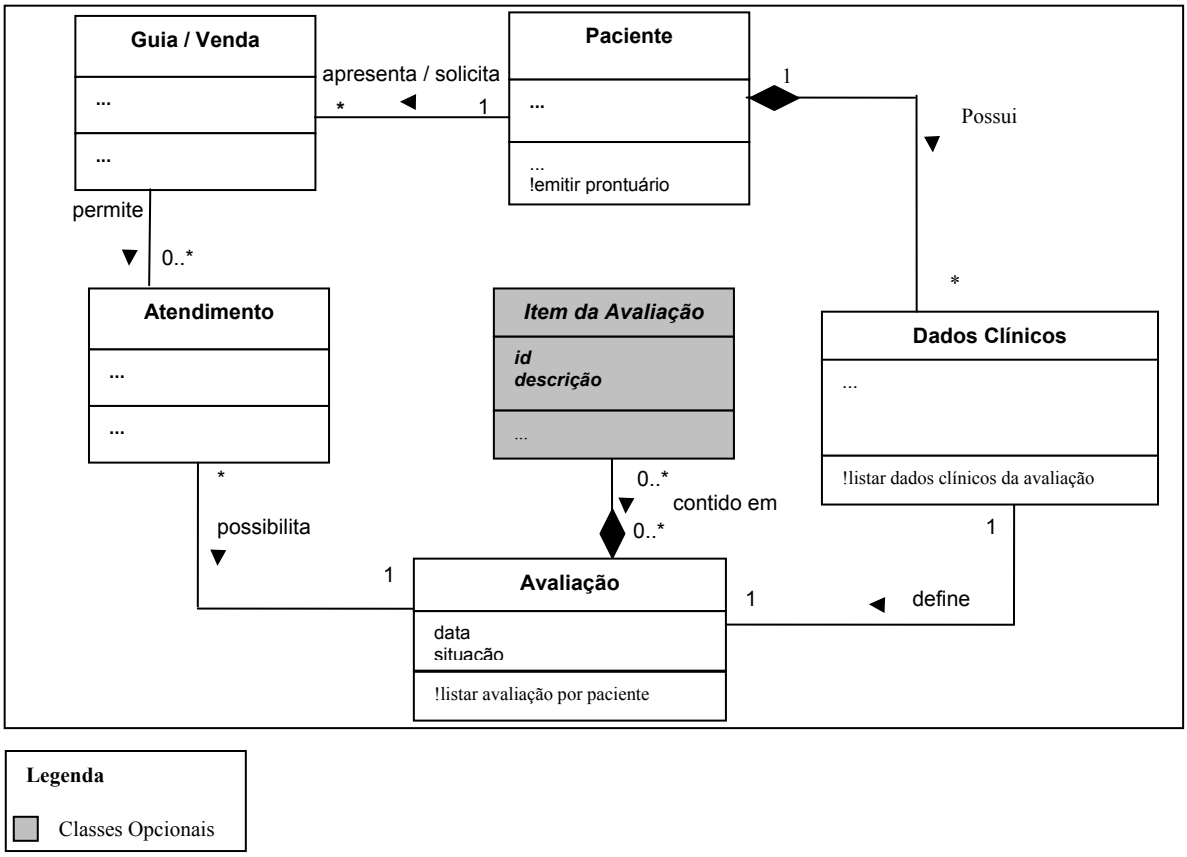

Figura 24 – Diagrama de classes para o padrão Realizar Acompanhamento (Pazin, 2004)

Equivalências Adotadas

Não há equivalências entre as classes.

Classes adicionais (não mapeadas na GRN)

Item da Avaliação, Dados Clínicos, Avaliação, Atendimento, Paciente, Guias, Vendas.

Padrão 8: Realizar Compras

Objetivo

Os produtos que são comercializados, devem ser armazenados para controle da clínica. Esses produtos são comercializados por meio de um pedido a um fornecedor responsável pela sua entrega.

Padrões Usados da GRN

Na Tabela 35 são apresentados os padrões da GRN usados para a definição do padrão Realizar Compras.

Tabela 35 – Padrões da GRN usados para definir o padrão Realizar Compras (Pazin, 2004)

Nº	*Padrão*	*Variante*
6	Comercializar o Recurso	Default
11	Itemizar a Transação do Recurso	Default

Estrutura do Padrão

Na Figura 25 é apresentado o diagrama de classes para o padrão Realizar Compras.

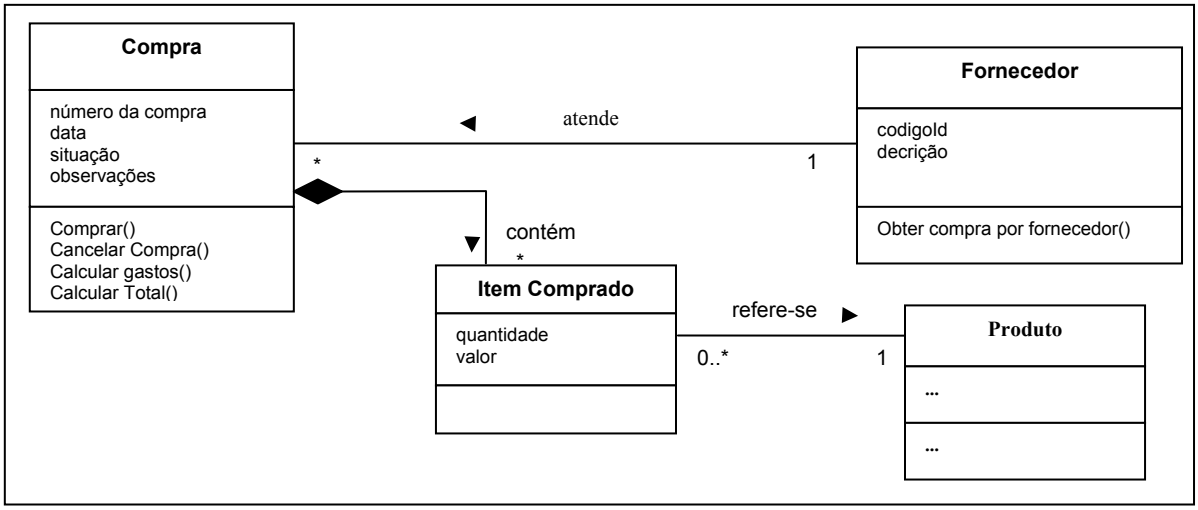

Figura 25 – Diagrama de classes para o padrão Realizar Compras (Pazin, 2004)

Equivalências Adotadas

Na Tabela 36 são apresentadas as equivalências adotadas entre as classes da SiGCli com as classes da GRN para definir o padrão Realizar Compras.

Tabela 36 – Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão Realizar Compras (Pazin, 2004)

Classes (SiGCli)	*Padrão GRN*	*Classes*
Fornecedor	(6) Comercializar o Recurso	Origem
Compra		Comercialização do Recurso
Produto		Recurso
Compra	(11) Itemizar Transação do Recurso	Transação do Recurso
Item Comprado		Item de transação
Produto		Recurso

Classes adicionais (não mapeadas na GRN)

Não há classes adicionais.

Padrão 9: Controlar Faturamento

Objetivo

Os serviços e produtos comercializados pela clínica devem ser armazenados para posterior análise e controle do seu faturamento.

Padrões Usados da GRN

Na Tabela 37 é apresentado o padrão da GRN usados para a definição do padrão Controlar Faturamento.

Tabela 37 – Padrões da GRN usados para definir o padrão Controlar Faturamento (Pazin, 2004)

Nº	*Padrão*	*Variante*
12	Pagar pela Transação do Recurso	Default

Estrutura do Padrão

A Figura 26 apresenta o diagrama de classes para o padrão Controlar Faturamento.

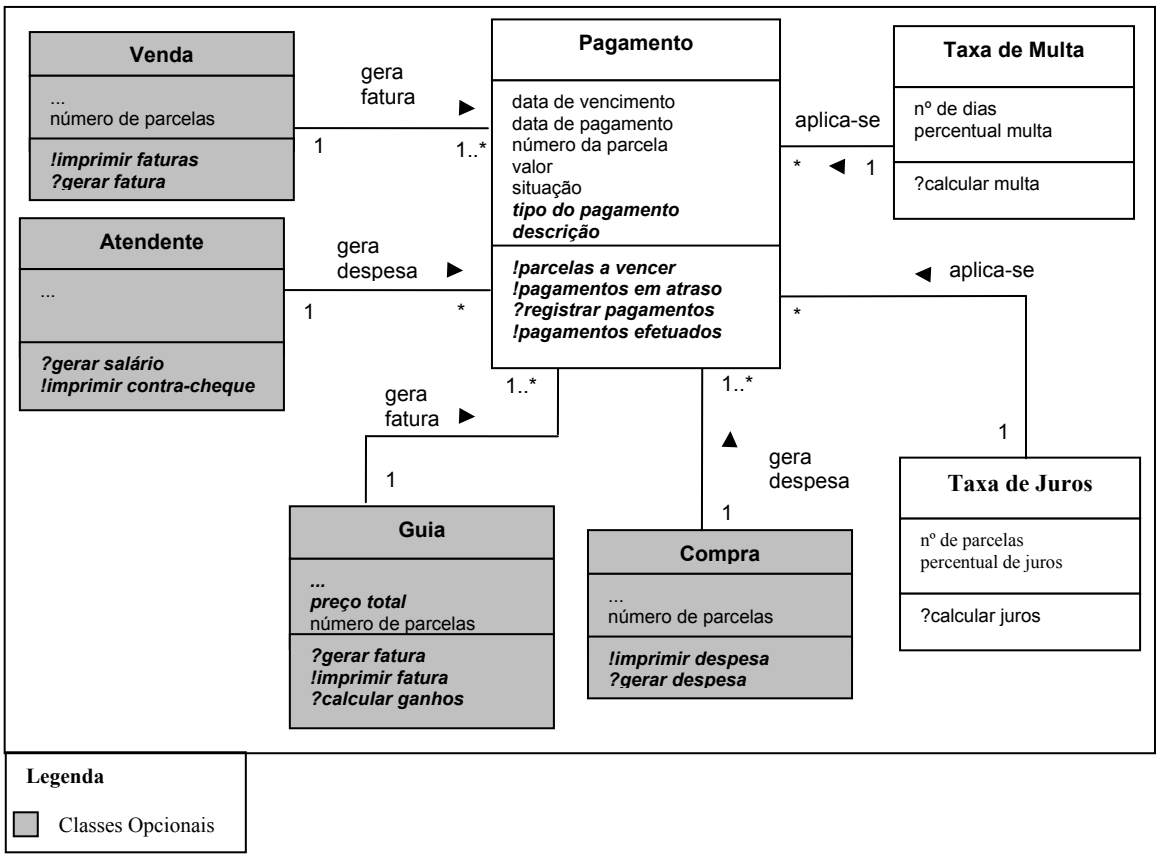

Figura 26 – Diagrama de classes para o padrão Controlar Faturamento (Pazin, 2004)

Equivalências Adotadas

Na Tabela 38 são apresentadas as equivalências adotadas entre as classes da SiGCli com as classes da GRN para definir o padrão Controlar Faturamento.

Tabela 38 – Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão Controlar Faturamento

Classes (SiGCli)	***Padrão GRN***	***Classes***
Venda	(12) Pagar pela Transação do Recurso	Transação do Recurso
Guia		Transação do Recurso
Compra		Transação do Recurso
Pagamento		Pagamento
Taxa de Juros		Taxa de Juros
Taxa de Multa		Taxa de Multa

Classes adicionais (não mapeadas na GRN)

Atendente.

APÊNDICE 2 – ESTUDO DE CASO DOS OUTROS SISTEMAS

1. Introdução

Este apêndice contém a documentação referente aos estudos de caso realizados com os sistemas "Clínicas Integradas", "CIC" e "Medical Office". Os artefatos gerados estão de acordo com o processo ágil de reengenharia com geradores de aplicação, descrito no capítulo 4 e são os seguintes: tabela de comparação das funções do legado com os padrões da SiGCli, diagrama de casos de uso, caso de uso, caso de teste e diagrama de classes. Além disso, mostra a versão 0 do sistema alvo e, em seguida, a versão 1 já com as adaptações necessárias. Na Seção 2 estão os artefatos do sistema Clínicas Integradas, o sistema CIC tem seus documentos apresentados na Seção 3 e na Seção 4 o Medical Office têm seus documentos mostrados.

2. CLÍNICAS INTEGRADAS

2.1. Observar o domínio do sistema legado em relação ao gerador

Objetivo da Atividade: identificar as características do sistema legado para verificar se ele pertence ao mesmo domínio do gerador ou possui as funções importantes / essenciais do domínio, a fim de utilizá-lo no projeto de reengenharia. Essa atividade também identifica se existem documentações do sistema legado.

Artefatos Produzidos: Tabela 39 contendo a listagem das funções do legado, verificando que padrões do gerador podem atender tais funções.

Tabela 39 – Funções Legado x Padrões SiGCli

Clínicas Integradas			**Gerador**
Menu	**Sub-Menu**	**Funções**	**SiGCli**
Arquivo		Pacientes	**1- Identificar Pacientes**
		Pacientes Inativos	**1- Identificar Pacientes**
		Agenda de Consultas	**5- Agendar Atendimento** **3- Realizar Vendas**
		Pedidos	**8 – Realizar Compra**
		Financeiro (contas pagas/recebidas)	**9 – Controlar Faturamento**
		Receita	
		Atestado	
		Recibo	
		Agenda de Telefones	
		Contrato	
		Orçamento	
Imprimir		Agendamentos	
	Cartões	Aniversário / Pacientes Ativos e Inativos	
		Natal / Pacientes Ativos e Inativos	
		Páscoa / Pacientes Ativos e Inativos	
	Financeiro	Fechamento de Dia	**9 – Controlar Faturamento**
	Pacientes	Ativos	**1- Identificar Pacientes**
		Inativos	**1- Identificar Pacientes**
	Carta para Paciente	Ativo	
		Inativo	
		Retorno	
		Envelopes	
		Pedidos	
		Agenda de Telefones	
		Contratos	
		Orçamentos	
Manutenção		Backup	
		Histórico de Erros	

2.2. Desenvolver o diagrama de casos de uso e elaborar os casos de testes

Objetivo da Atividade: Entender os requisitos do sistema legado, priorizados para a iteração corrente, por meio da execução de casos de teste criados e identificados na atividade "Observar o domínio do sistema legado em relação ao do gerador".

Artefatos Produzidos: Documentação dos casos de teste (Tabela 40), diagrama de casos de uso (Figuras 27 e 28) e documento de requisitos (Figura 29).

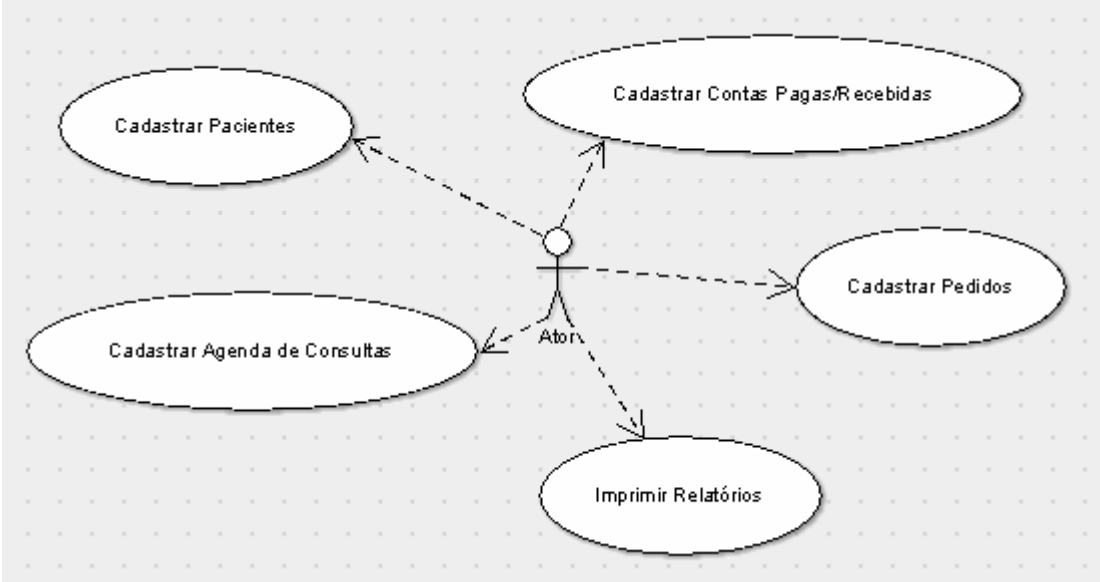

Figura 27 – Diagrama de casos de uso (ucs atendidos pela SiGCli)

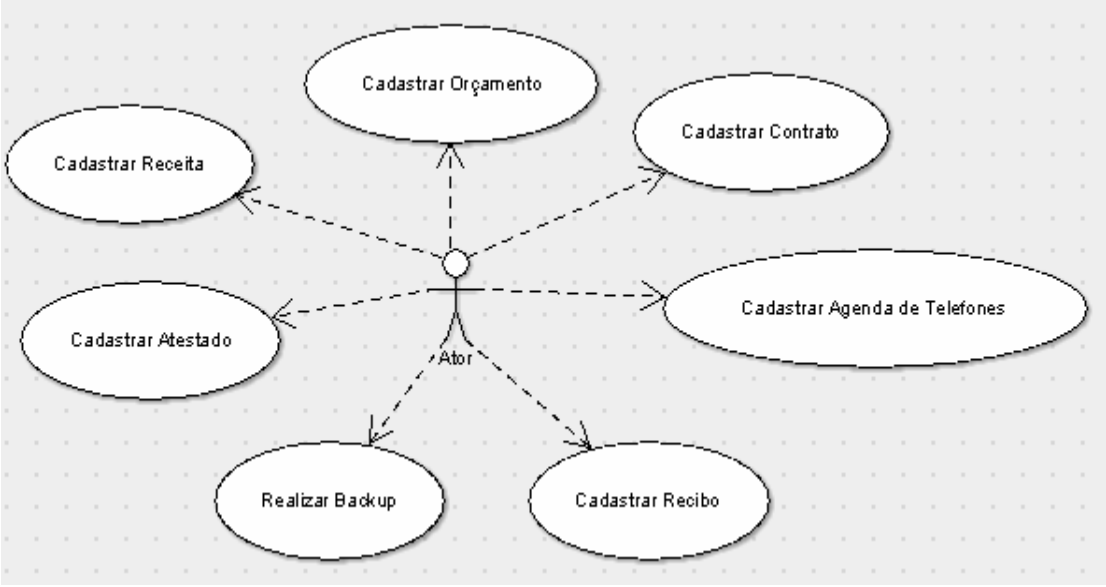

Figura 28 – Diagrama de casos de uso (ucs não atendidos pela SiGCli)

Caso de uso: CADASTRAR PACIENTES
Descrição: este caso de uso permite que o usuário realize o cadastro das pessoas que freqüentam a clínica.

1. Paciente fornece seu nome;
2. Paciente não cadastrado;

Fluxo Normal – Inclusão
2.1. Paciente fornece seus dados;
2.2. Gerar Código Paciente;
2.3. Encerra caso de uso

Fluxo Alternativo - Alteração
2.4. Paciente já cadastrado
2.4.1. Paciente deseja alterar dados;
2.4.2. Paciente fornece dados a serem alterados;
2.4.3. Alterar Paciente;
2.4.4. Encerrar caso de uso

Fluxo Alternativo – Exclusão
2.4.5. Paciente será excluído;
2.4.6. Excluir Paciente;
2.4.7. Encerrar caso de uso.

Figura 29 – Caso de Uso Cadastrar Pacientes

Tabela 40 – Casos de Teste

		Classes de Equivalência		Casos de Testes	
Atributos	**Descrição Banco**	**Classes Válidas**	**Classes Inválidas**	**Casos de Teste**	**Saída Esperada**
Nome	Texto 30	Texto < 30	Texto > 30	Texto < 30	Sucesso
				Texto = 30	Sucesso
				Texto > 30	Não Sucesso
Endereço	Texto 50	Texto < 50	Texto > 50	Texto < 50	Sucesso
				Texto = 50	Sucesso
				Texto > 50	Não Sucesso
Bairro	Texto 12	Texto < 12	Texto > 12	Texto < 12	Sucesso
				Texto = 12	Sucesso
				Texto > 12	Não Sucesso
CEP	Texto 12	Texto < 12	Texto > 12	Texto < 12	Sucesso
				Texto = 12	Sucesso
				Texto > 12	Não Sucesso
Cidade	Texto 50	Texto < 50	Texto > 50	Texto < 50	Sucesso
				Texto = 50	Sucesso
				Texto > 50	Não Sucesso
Telefone	Texto 12	Texto < 12	Texto > 12	Texto < 12	Sucesso
				Texto = 12	Sucesso
				Texto > 12	Não Sucesso
Sexo	Texto 50	Texto < 50	Texto > 50	Texto < 50	Sucesso
				Texto = 50	Sucesso
				Texto > 50	Não Sucesso
Data Nasc	Texto 10	Texto < 10	Texto > 10	Texto < 10	Sucesso
				Texto = 10	Sucesso

				Texto > 10	Não Sucesso
Data Entrada	Texto 20	Texto < 20	Texto > 20	Texto < 20	Sucesso
				Texto = 20	Sucesso
				Texto > 20	Não Sucesso
Indicação	Texto 50	Texto < 50	Texto > 50	Texto < 50	Sucesso
				Texto = 50	Sucesso
				Texto > 50	Não Sucesso
Convênio	Texto 30	Texto < 30	Texto > 30	Texto < 30	Sucesso
				Texto = 30	Sucesso
				Texto > 30	Não Sucesso
Especialista	Seleção	Seleção de um registro	Seleção de mais de um registro	Seleção de mais de um registro	Não Sucesso
		Mostra todos os registros cadastrados	Não mostra todos os registros cadastrados	Mostra todos os registros cadastrados	Sucesso
Dr(a)	Texto 50	Texto < 50	Texto > 50	Texto < 50	Sucesso
				Texto = 50	Sucesso
				Texto > 50	Não Sucesso

Criar o sistema alvo no paradigma orientado a objetos

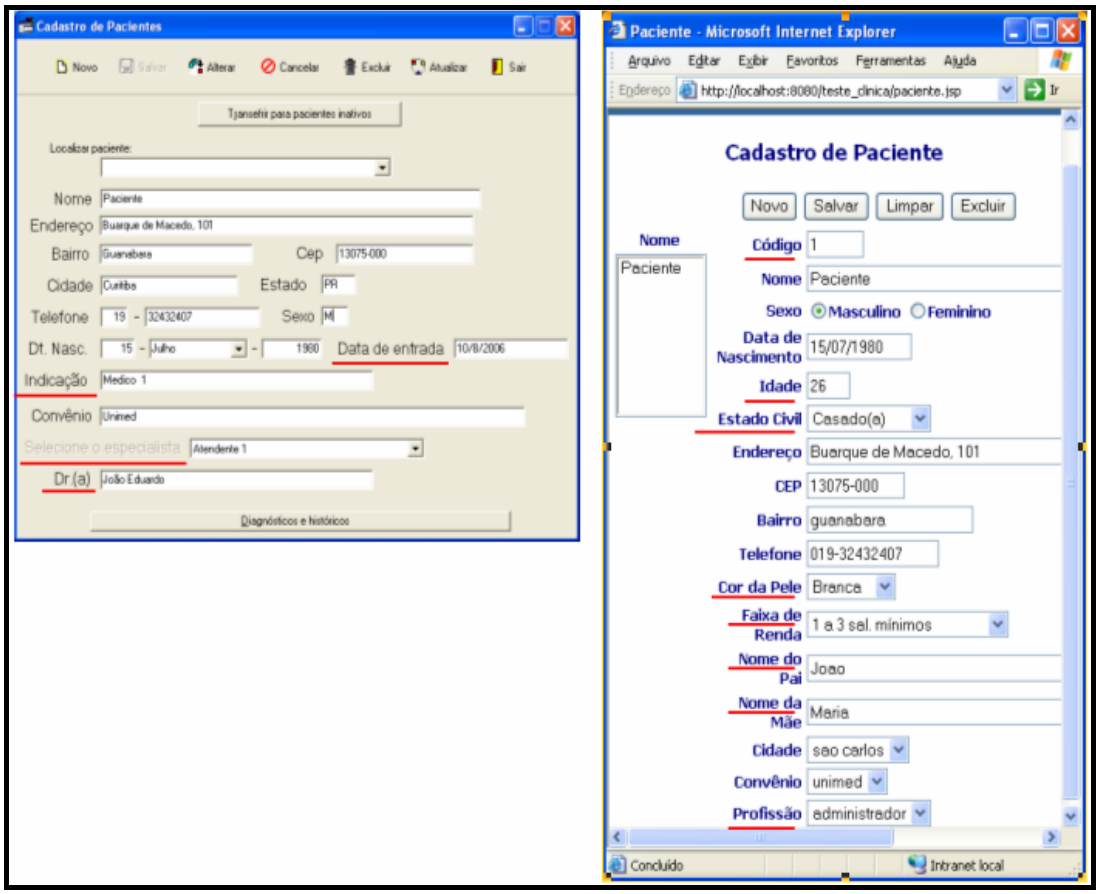

Figura 30 – Versão 0 do Sistema Alvo

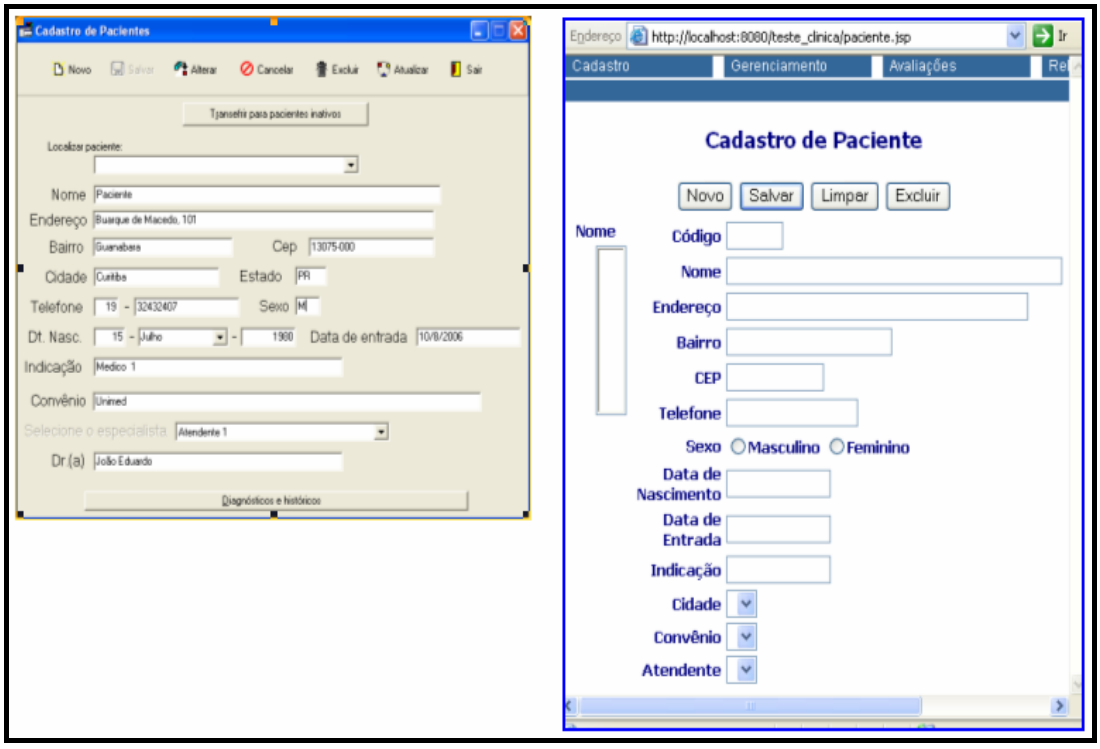

Figura 31 – Versão 1do Sistema Alvo

Desenvolver o Diagrama de Classes do Sistema Alvo

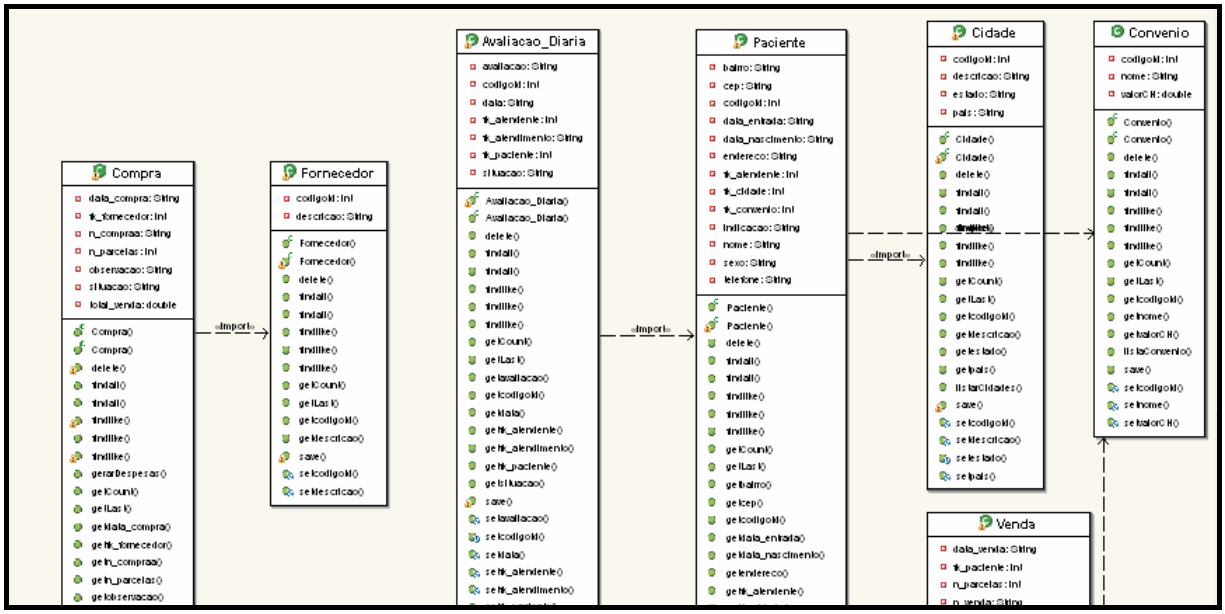

Figura 32 – Diagrama de Classes

3. CIC

Observar o domínio do sistema legado em relação ao gerador

Tabela 41 – Funções Legado x Padrões SiGCli

CIC			**Gerador**
Menu	**Sub-Menu**	**Funções**	**SiGCli**
Consultório	Cadastros	Pacientes	**1- Identificar Pacientes**
		Atendentes	**6- Identificar Atendentes**
		Convênios	**1- Identificar Pacientes**
	Financeiro	Contas a Receber	**9 – Controlar Faturamento**
	Agenda		**5- Agendar Atendimento** **3- Realizar Vendas**
	Tabelas	Procedimentos	**7 – Realizar Acompanhamento**
		Remédios	
		CEP	
Atendimento		Atendimento ao Paciente	**6- Identificar Atendentes** **7- Realizar Acompanhamento** **2- Definir Serviços**
		Controle de Exames	
		Receituário	
		Atestado Médico	
Consultas	Pacientes	Consulta Expressa	**1- Identificar Pacientes**
		Consulta Aniversariantes	
		Consulta por Faixa Etária	
		Consulta por Cidade	
	Agenda	Por Data	
		Por Médico	
		Por Paciente	
		Convênios	**1- Identificar Pacientes**
		Procedimento	
		Remédios	
	Contas a Receber	Por Data	
		Por Paciente	
		Por Convênio	
Diversos		Troca de Usuário	
		Ajuda do Sistema	
		Parâmetros do Sistema	
		Manutenção	
		Backup	
		Calculadora	
		Resumo do Dia	

Desenvolver o diagrama de casos de uso e elaborar os casos de testes

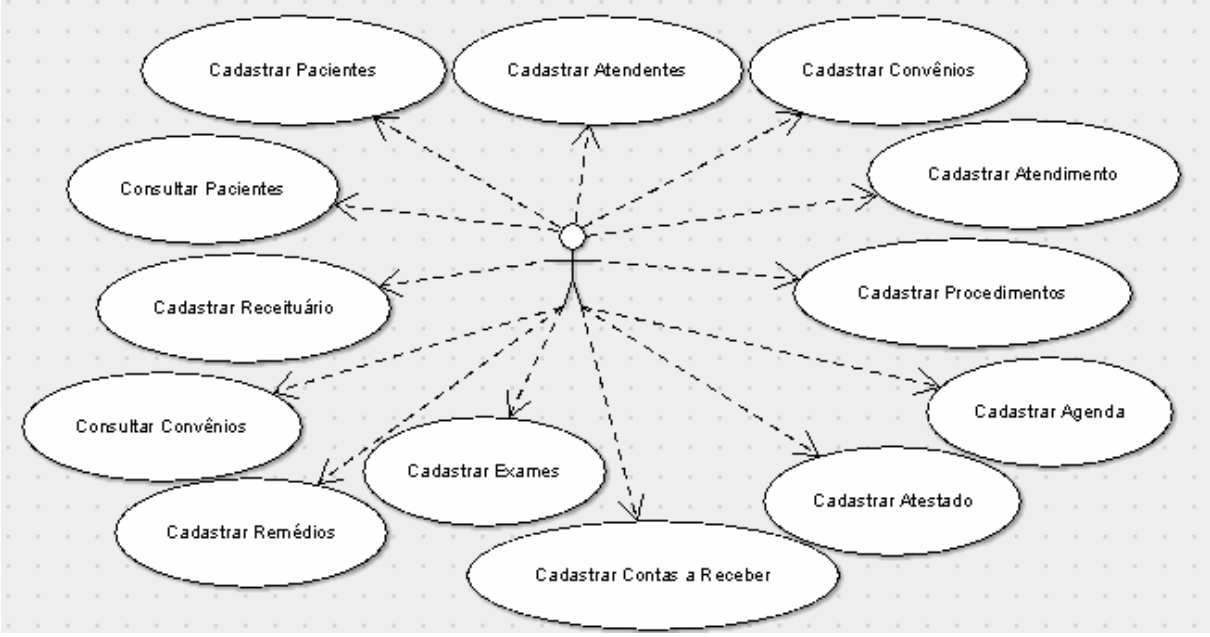

Figura 33 – Diagrama de Casos de Uso

Caso de uso: CADASTRAR PACIENTES
Descrição: este caso de uso permite que o usuário realize o cadastro das pessoas que freqüentam a clínica.

1. Paciente fornece seu nome;
2. Paciente não cadastrado;

Fluxo Normal – Inclusão
2.1. Paciente fornece seus dados;
2.2. Gerar Código Paciente;
2.3. Encerra caso de uso

Fluxo Alternativo - Alteração
2.4. Paciente já cadastrado
2.4.1. Paciente deseja alterar dados;
2.4.2. Paciente fornece dados a serem alterados;
2.4.3. Alterar Paciente;
2.4.4. Encerrar caso de uso

Fluxo Alternativo – Exclusão
2.4.5. Paciente será excluído;
2.4.6. Excluir Paciente;
2.4.7. Encerrar caso de uso.

Figura 34 – Casos de Uso

Tabela 42 – Casos de Teste

		Classes de Equivalência		Casos de Testes	
Atributos	**Descrição Banco**	**Classes Válidas**	**Classes Inválidas**	**Casos de Teste**	**Saída Esperada**
Cód Paciente	Numérico	Gerado pelo sistema	Não gerado	Numero seqüencial gerado	Sucesso
Nome	Texto 40	Texto < 40	Texto > 40	Texto < 40	Sucesso
				Texto = 40	Sucesso
				Texto > 40	Não Sucesso
CEP	Texto 9	Texto < 9	Texto > 9	Texto < 9	Sucesso
				Texto = 9	Sucesso
				Texto > 9	Não Sucesso
Endereço	Texto 40	Texto < 40	Texto > 40	Texto < 40	Sucesso
				Texto = 40	Sucesso
				Texto > 40	Não Sucesso
Numero	Texto 5	Texto < 5	Texto > 5	Texto < 5	Sucesso
				Texto = 5	Sucesso
				Texto > 5	Não Sucesso
Complemento	Texto 8	Texto < 8	Texto > 8	Texto < 8	Sucesso
				Texto = 8	Sucesso
				Texto > 8	Não Sucesso
Bairro	Texto 15	Texto < 15	Texto > 15	Texto < 15	Sucesso
				Texto = 15	Sucesso
				Texto > 15	Não Sucesso
Cidade	Texto 20	Texto <20	Texto > 20	Texto < 20	Sucesso
				Texto = 20	Sucesso
				Texto > 20	Não Sucesso
Estado	Texto 2	Texto < 2	Texto > 2	Texto < 2	Sucesso
				Texto = 2	Sucesso
				Texto > 2	Não Sucesso
Tel. Residência	Texto 13	Texto < 13	Texto > 13	Texto < 13	Sucesso
				Texto = 13	Sucesso
				Texto > 13	Não Sucesso
Tel. Recado	Texto 13	Texto < 13	Texto > 13	Texto < 13	Sucesso
				Texto = 13	Sucesso
				Texto > 13	Não Sucesso
Tel. Celular	Texto 13	Texto < 13	Texto > 13	Texto < 13	Sucesso
				Texto = 13	Sucesso
				Texto > 13	Não Sucesso
CPF	Texto 14	Texto < 14	Texto > 10	Texto < 10	Sucesso
				Texto = 10	Sucesso
				Texto > 10	Não Sucesso
RG	Texto 15	Texto < 20	Texto > 14	Texto < 14	Sucesso
				Texto = 14	Sucesso
				Texto > 14	Não Sucesso
Estado Civil	Seleção	Seleção de um registro	Seleção de mais de um registro	Seleção de mais de um registro	Não Sucesso
		Mostra todos os registros cadastrados	Não mostra todos os registros cadastrados	Mostra todos os registros cadastrados	Sucesso
Nascido em	Texto 30	Texto < 30	Texto > 30	Texto < 30	Sucesso
				Texto = 30	Sucesso

				Texto > 30	Não Sucesso
Idade	Numérico 3	Texto < 3	Texto > 3	Texto < 3	Sucesso
				Texto = 3	Sucesso
				Texto > 3	Não Sucesso
Descrição	Texto 40	Texto < 40	Texto > 40	Texto < 40	Sucesso
				Texto = 40	Sucesso
				Texto > 40	Não Sucesso
Observação	Texto 30	Texto < 30	Texto > 30	Texto < 30	Sucesso
				Texto = 30	Sucesso
				Texto > 30	Não Sucesso
Indicação	Texto 40	Texto < 40	Texto > 40	Texto < 40	Sucesso
				Texto = 40	Sucesso
				Texto > 40	Não Sucesso
Pai	Texto 40	Texto < 50	Texto > 50	Texto < 50	Sucesso
				Texto = 50	Sucesso
				Texto > 50	Não Sucesso

Criar o sistema alvo no paradigma orientado a objetos

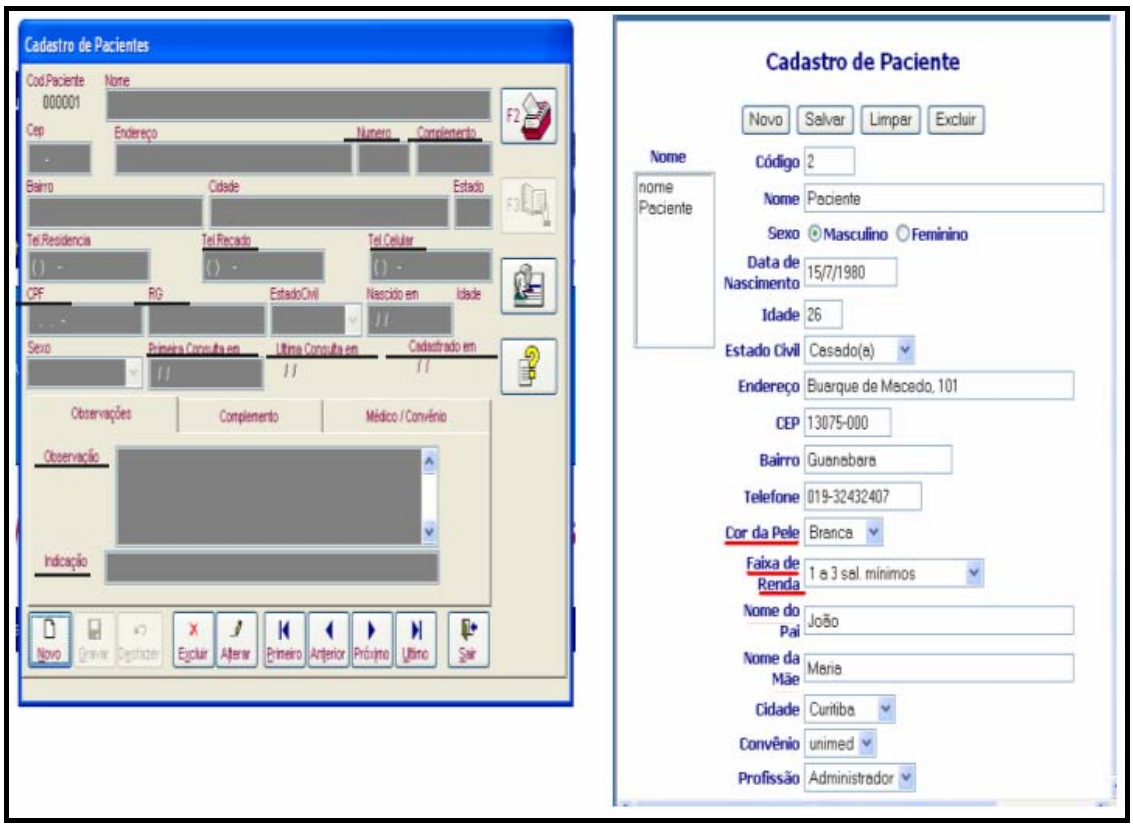

Figura 35 – Versão 0 do Sistema Alvo

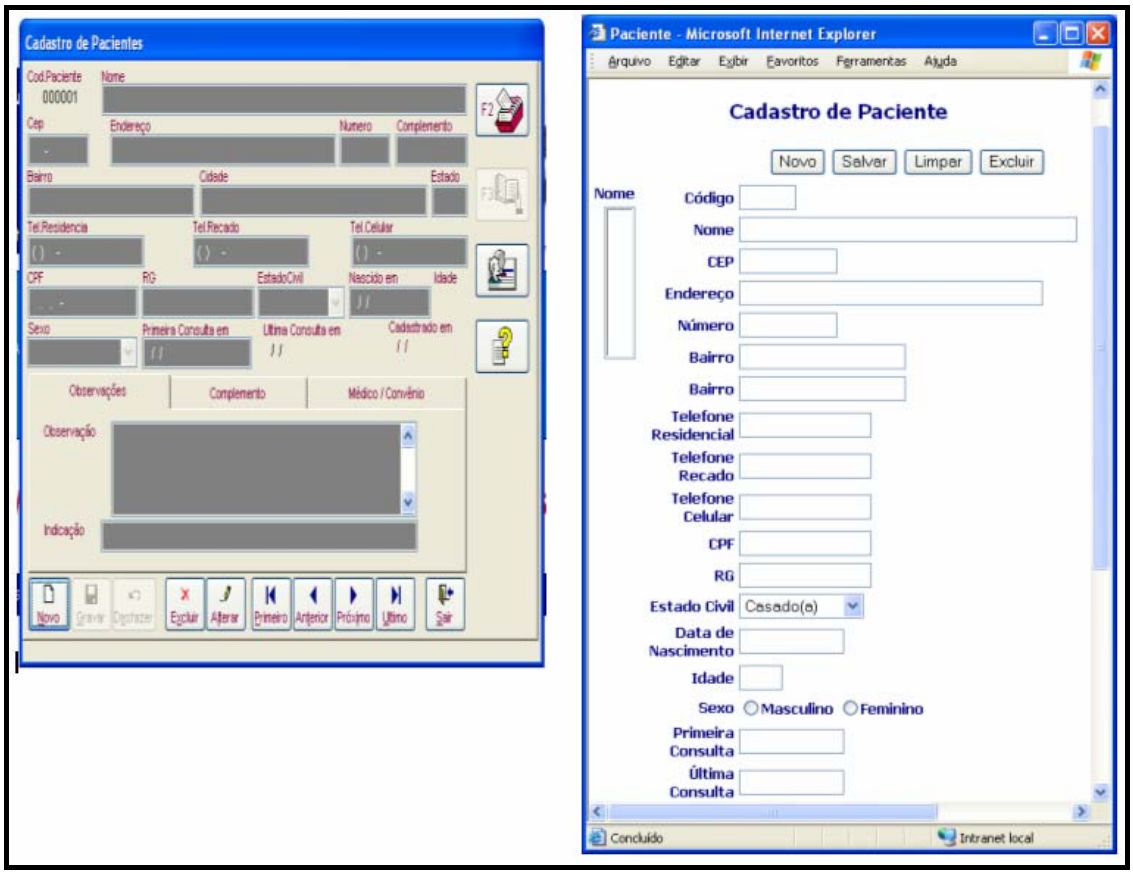

Figura 36 – Versão 1 do Sistema Alvo

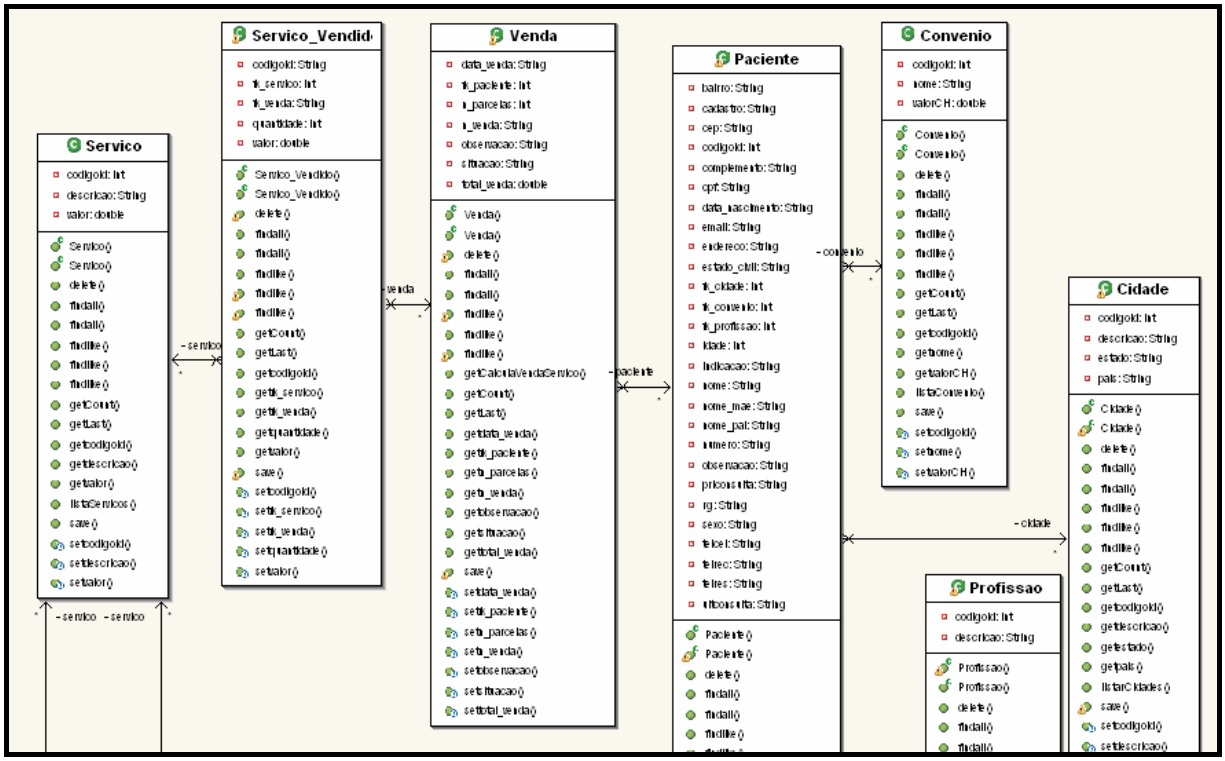

Figura 37 – Diagrama de Classes do Sistema Alvo

4. MEDICAL OFFICE

Observar o domínio do sistema legado em relação ao gerador

Tabela 43 – Funções Legado x Padrões SiGCli

Medical Office			**Gerador**
Menu	**Sub-Menu**	**Funções**	**SiGCli**
Arquivos		Pacientes	**1- Identificar Pacientes**
		Prontuário	**6- Identificar Atendentes** **7- Realizar Acompanhamento** **2- Definir Serviços**
	Agendas	Consultas	**5- Agendar Atendimento** **3- Realizar Vendas**
		Compromissos	
		Recados	
		Espera	
		Secretárias	
		Telefônica	
	Tabelas	Convênios	**1- Identificar Pacientes**
		Indicações	
		Especialidades	
		Diagnósticos	
		Trat Clinico	**7- Realizar Acompanhamento**
		Trat Cirurgico	**7- Realizar Acompanhamento**
		Exames	
		Remédios	
		Dietas	
		Orientações	
		Anotações	
		Licença	
		Médicos	
	Usuários	Manutenção	
		Alterar Senha	
		Trocar Senha	
	Importar	CID	
		Trat. Clinico AMB	
		Trat. Cirúrgico AMB	
		Exames AMB	
		Remédios	
Impressões		Agendas	**5- Agendar Atendimento** **3- Realizar Vendas**
		Prontuários	
		Sol. Exame	
		Receita	
		Dieta	
		Orientação	
		Laudo/Rel. Médico	
	Recibos	Impressão	
		Listagem	
		Mov. Financeiro	
		Listagem	
		Etiquetas	

Utilitários	Copia BD	
	Recupera BD	
	Compactar/Reparar BD	
	Temporário BD	
	Atualiza BD	
	Mensagem	
	Cadastra Recado	
	Data/Hora	
	Calculadora	

Desenvolver o diagrama de casos de uso e elaborar os casos de testes

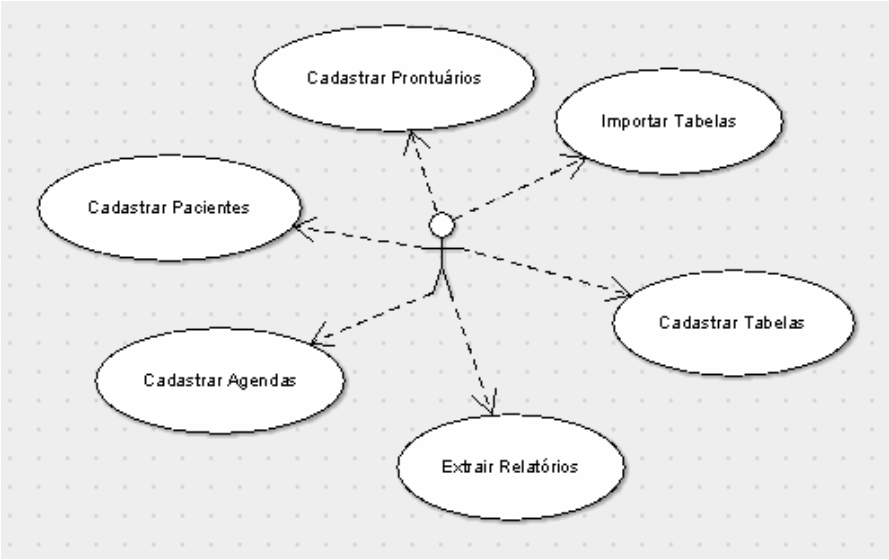

Figura 38 – Diagrama de Classes do Sistema Alvo

Caso de uso: CADASTRAR PACIENTES
Descrição: este caso de uso permite que o usuário realize o cadastro das pessoas que freqüentam a clínica.

1. Paciente fornece seu nome;
2. Paciente não cadastrado;

Fluxo Normal – Inclusão
2.1. Paciente fornece seus dados;
2.2. Gerar Código Paciente;
2.3. Encerra caso de uso

Fluxo Alternativo - Alteração
2.4. Paciente já cadastrado
2.4.1. Paciente deseja alterar dados;
2.4.2. Paciente fornece dados a serem alterados;
2.4.3. Alterar Paciente;
2.4.4. Encerrar caso de uso

Fluxo Alternativo – Exclusão
2.4.5. Paciente será excluído;
2.4.6. Excluir Paciente;
2.4.7. Encerrar caso de uso.

Figura 39 – Caso de Uso

Tabela 44 – Casos de Teste

		Classes de Equivalência		**Casos de Testes**	
Atributos	**Descrição Banco**	**Classes Válidas**	**Classes Inválidas**	**Casos de Teste**	**Saída Esperada**
Cód Paciente	Numero 10	Numero < 10	Numero > 10	Num < 10	Sucesso
				Num = 10	Sucesso
				Num > 10	Não Sucesso
Data de Cadastro	Texto 20	Texto < 20	Texto > 20	Texto < 20	Sucesso
				Texto = 20	Sucesso
				Texto > 20	Não Sucesso
Médico	Seleção	Seleção de um registro	Seleção de mais de um registro	Seleção de mais de um registro	Não Sucesso
		Mostra todos os registros cadastrados	Não mostra todos os registros cadastrados	Mostra todos os registros cadastrados	Sucesso
Nome	Texto 40	Texto < 40	Texto > 40	Texto < 40	Sucesso
				Texto = 40	Sucesso
				Texto > 40	Não Sucesso
Estado Civil	Seleção	Seleção de um registro	Seleção de mais de um registro	Seleção de mais de um registro	Não Sucesso
		Mostra todos os registros cadastrados	Não mostra todos os registros cadastrados	Mostra todos os registros cadastrados	Sucesso
Sexo	Seleção	Seleção de um registro	Seleção de mais de um registro	Seleção de mais de um registro	Não Sucesso
		Mostra todos os registros cadastrados	Não mostra todos os registros cadastrados	Mostra todos os registros cadastrados	Sucesso
Cor	Seleção	Seleção de um registro	Seleção de mais de um registro	Seleção de mais de um registro	Não Sucesso
		Mostra todos os registros cadastrados	Não mostra todos os registros cadastrados	Mostra todos os registros cadastrados	Sucesso
Data de Nascimento	Texto 8	Texto < 8	Texto > 8	Texto < 8	Sucesso
				Texto = 8	Sucesso
				Texto > 8	Não Sucesso
Naturalidade	Texto 15	Texto < 15	Texto > 15	Texto < 15	Sucesso
				Texto = 15	Sucesso
				Texto > 15	Não Sucesso
Profissão	Texto 20	Texto <20	Texto > 20	Texto < 20	Sucesso
				Texto = 20	Sucesso
				Texto > 20	Não Sucesso
CPF	Texto 30	Texto < 30	Texto > 30	Texto < 30	Sucesso
				Texto = 30	Sucesso
				Texto > 30	Não Sucesso
RG	Texto 30	Texto < 30	Texto > 30	Texto < 30	Sucesso
				Texto = 30	Sucesso
				Texto > 30	Não Sucesso
Responsável	Texto 13	Texto < 13	Texto > 13	Texto < 13	Sucesso
				Texto = 13	Sucesso

				Texto > 13	Não Sucesso
Endereço	Texto 13	Texto < 13	Texto > 13	Texto < 13	Sucesso
				Texto = 13	Sucesso
				Texto > 13	Não Sucesso
Bairro	Texto 14	Texto < 14	Texto > 10	Texto < 10	Sucesso
				Texto = 10	Sucesso
				Texto > 10	Não Sucesso
Cidade	Texto 15	Texto < 20	Texto > 14	Texto < 14	Sucesso
				Texto = 14	Sucesso
				Texto > 14	Não Sucesso
Estado	Seleção	Seleção de um registro	Seleção de mais de um registro	Seleção de mais de um registro	Não Sucesso
		Mostra todos os registros cadastrados	Não mostra todos os registros cadastrados	Mostra todos os registros cadastrados	Sucesso
CEP	Texto 30	Texto < 30	Texto > 30	Texto < 30	Sucesso
				Texto = 30	Sucesso
				Texto > 30	Não Sucesso
Idade	Numérico 3	Texto < 3	Texto > 3	Texto < 3	Sucesso
				Texto = 3	Sucesso
				Texto > 3	Não Sucesso
Telefone 1	Texto 30	Texto < 30	Texto > 30	Texto < 30	Sucesso
				Texto = 30	Sucesso
				Texto > 30	Não Sucesso
Telefone 2	Texto 30	Texto < 30	Texto > 30	Texto < 30	Sucesso
				Texto = 30	Sucesso
				Texto > 30	Não Sucesso
Fax	Texto 30	Texto < 30	Texto > 30	Texto < 30	Sucesso
				Texto = 30	Sucesso
				Texto > 30	Não Sucesso
Email	Texto 30	Texto < 30	Texto > 30	Texto < 30	Sucesso
				Texto = 30	Sucesso
				Texto > 30	Não Sucesso
Indicação	Texto 40	Texto < 40	Texto > 40	Texto < 40	Sucesso
				Texto = 40	Sucesso
				Texto > 40	Não Sucesso
Convênio	Texto 30	Texto < 50	Texto > 50	Texto < 50	Sucesso
				Texto = 50	Sucesso
				Texto > 50	Não Sucesso
Registro	Texto 40	Texto < 40	Texto > 40	Texto < 40	Sucesso
				Texto = 40	Sucesso
				Texto > 40	Não Sucesso
Validade	Texto 40	Texto < 40	Texto > 40	Texto < 40	Sucesso
				Texto = 40	Sucesso
				Texto > 40	Não Sucesso
Observação	Texto 100	Texto < 100	Texto > 100	Texto < 100	Sucesso
				Texto = 100	Sucesso
				Texto > 100	Não Sucesso

Criar o sistema alvo no paradigma orientado a objetos

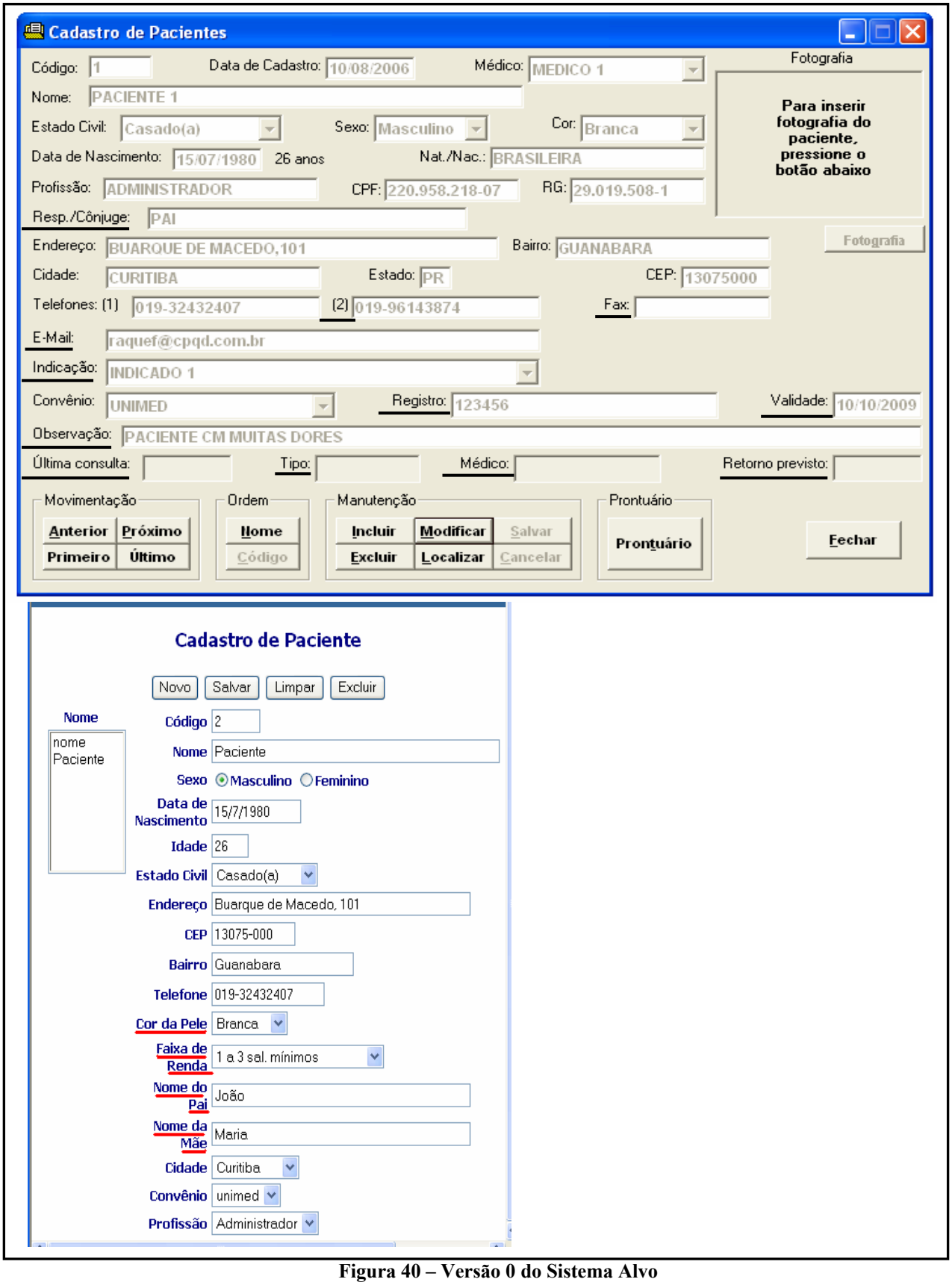

Figura 40 – Versão 0 do Sistema Alvo

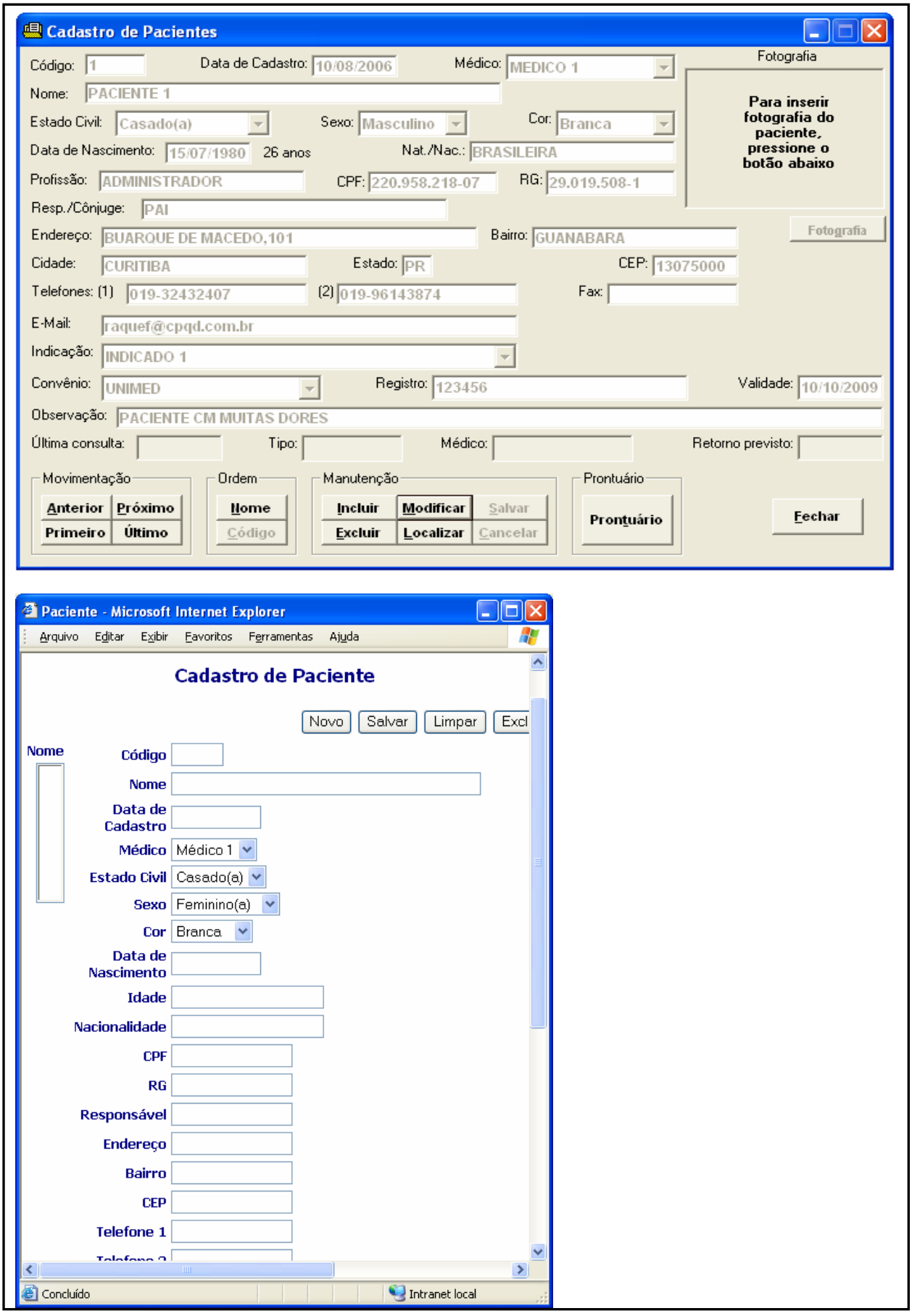

Figura 41 – Versão 1 do Sistema Alvo

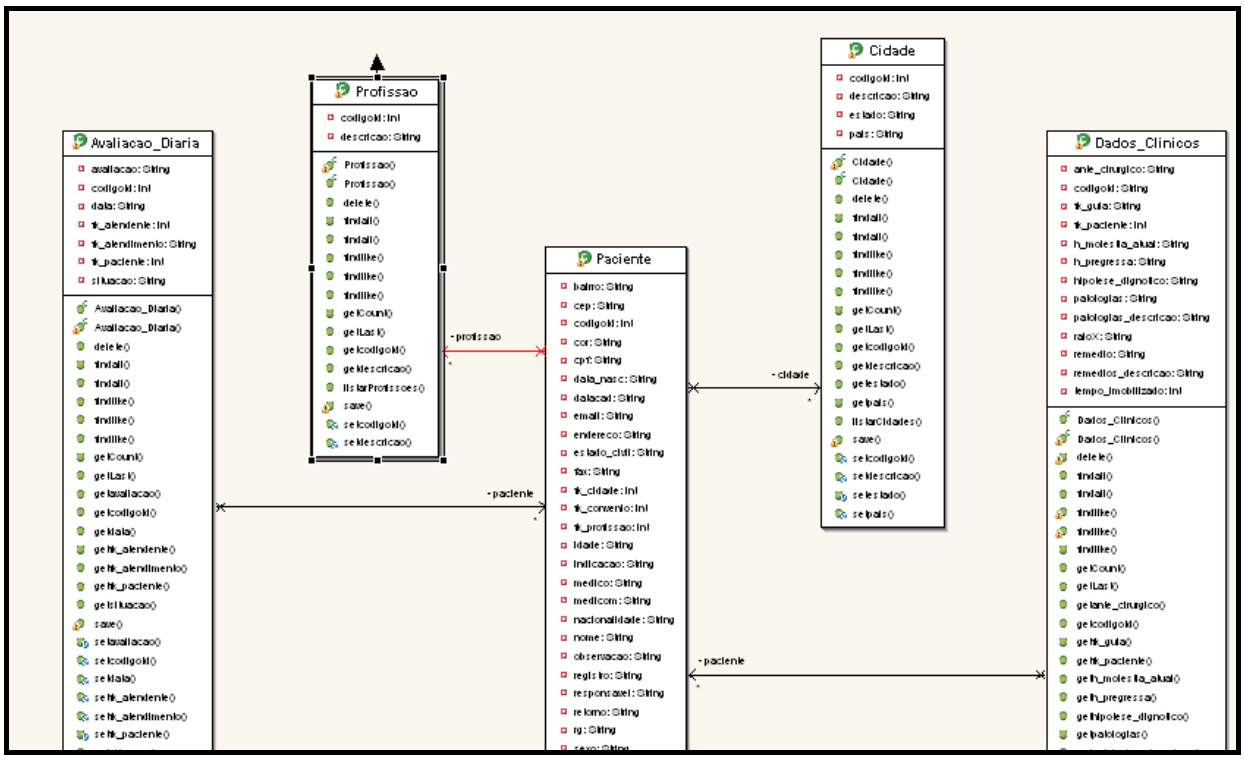

Figura 42 – Diagrama de Classes do Sistema Alvo