UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

*"GAwCRe: Um **G**erador de **A**plicações baseadas na **W**eb para o Domínio de **C**línicas de **Re**abilitação"*

Anderson Pazin

São Carlos – SP

2004

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

P348ga

Pazin, Anderson.
GAwCRe: um gerador de aplicações baseadas na web para o domínio de clínicas de reabilitação / Anderson Pazin. -- São Carlos : UFSCar, 2004.
160 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2004.

1. Geradores (programa de computador). 2. Família de produtos de software. I. Título.

CDD: 005.13 (20ª)

Dedico este trabalho ao meu pai João, que sempre me incentivou,
mas não está mais conosco para ver a realização
de mais uma etapa da minha vida;
à minha mãe Dirce, por sua presença, amor e carinho e
à minha noiva Marcela. Amo muito vocês!

Agradecimentos

Primeiramente a DEUS pelo dom da vida e por estar sempre presente guiando meus passos.

A minha orientadora Prof.ª Dr. Rosângela Aparecida Dellosso Penteado, pela confiança em me aceitar com aluno, pelos constantes ensinamentos transmitidos, orientação, amizade, paciência e companheirismo durante este período.

Aos meus familiares, em especial minha Mãe Dirce, meus irmãos Carlos, Gustavo e Ana, pelo carinho e companheirismo e por me amarem mais do que eu possa demonstrar que os amo.

A minha noiva Marcela, pelo amor, paciência e compreensão e por me motivar a enfrentar mais essa etapa da minha vida.

Aos Professores. Dr. Paulo César Masiero, Dr.ª Rosana Teresinha V. Braga e Dr. Fernão Stella R. Germano, pelas contribuições durante este trabalho.

A minha família Vicentina, conferência de São Miguel, pela amizade e constantes orações.

Aos meus amigos de vida Luiz Gustavo (Batavo), Piva, Rodrigo (Butcha), Tavinho, Walter, Piá, Bressan, Odair, pelo companheirismo.

Ao amigo Luiz Eduardo Bergamo, pela amizade e auxílios durante este trabalho.

Aos meus colegas de trabalho Luiz Fernando (Carneiro), Leandro, Cláudio, pela atenção e dedicação.

Aos meus amigos de mestrado Anderson Belgamo (Dinho), Adriano, Moraes, Pablo, Karina, Frank e demais companheiros de turma, pela amizade.

Aos amigos Valter e Ricardo (RAR), pela hospitalidade em sua república durante um semestre de estudos e pela constante alegria.

Ao amigo Luiz Fernando (Tuca) pela amizade e companhia durante as viagens.

Aos padres Pe. Afonso, Pe. Paulo, Pe. Tadeu, Pe. Osvaldo e os demais irmãos salesianos que acreditaram e confiaram em mim, permitindo-me estudar para aprimorar os meus conhecimentos.

A todos aqueles que participam da minha vida e tornaram este sonho possível.

As secretárias da Pós-Graduação Cristina e Mirian.

A Missão Salesiana de Mato Grosso pelo ambiente maravilhoso em suas casas.

Resumo

Linguagens de padrões definem um domínio, facilitando a modelagem de aplicações por desenvolvedores menos experientes. Com o uso de uma linguagem de padrões pode-se definir uma arquitetura genérica (*frameworks* ou geradores de aplicações) capaz de automatizar parte do processo de desenvolvimento de novas aplicações. Essa arquitetura pode ser representada por uma linguagem de modelagem de aplicações (LMA), utilizando os conceitos de linha de produtos de software, que permitem definir uma aplicação fazendo especificações em alto nível. Dentro desse contexto, este trabalho apresenta um gerador de aplicações baseadas na Web para um domínio de **Si**stemas de **G**estão de **Clí**nicas de Reabilitação (SiGCli), chamado GAwCRe (**G**erador de **A**plicações baseadas na **W**eb para **C**línica de **Re**abilitação) que permite instanciar aplicações usando uma LMA definida com base em uma linguagem de padrões. Para representar as informações referentes as LMA e a linguagem de padrões SiGCli foi elaborado um meta-modelo utilizado a linguagem XML. Assim, as informações (LMA) apresentadas na interface de instanciação das aplicações do gerador são criadas dinamicamente. Para o processo de geração dos produtos de software, são definidos gabaritos de código, dos produtos desejados, com pontos de substituição previamente definidos e que devem assumir os valores definidos no documento XML, segundo a especificação LMA da aplicação. O documento XML facilita a legibilidade da documentação da LMA e da linguagem de padrões e sua flexibilidade permite que outras linguagens de padrões possam ser mapeadas para a estrutura XML definida, possibilitando assim o reuso do gerador proposto para outros domínios.

Abstract

Pattern languages define a domain, easing application modeling by less experienced developers. With the use of a pattern language a generic architecture (frameworks or application generators) can be defined, capable of automating part of the new applications development process. That architecture can be represented by an Application Modeling Language (AML), using the concepts of software products line, which allow an application definition from high level specification. In that context, this work presents a Web based application generator for the Rehabilitation Clinic Management Systems domain (SiGCli; Sistemas de Gestão de Clínicas de Reabilitação, in portuguese), named GAwCRe (Web Application Generator for Rehabilitation Clinics; Gerador de Aplicações baseada Web para Clinícas de Reabilitação, in portuguese). Its instantiation for these applications is done using an AML that has been defined based on the SiGCli pattern language. A meta-model using the XML language has been made to represent AML and SiGCli information. Thus, the AML information presented in the applications generator instantiation interface is dynamically created. For the artifacts generation process, code templates are defined with substitution points previously established that have to assume the values defined in the XML document, according to the application AML specification. The XML document eases the AMl and SigCli documentation readability. With XML use, the GAwCRe applications generator has enough flexibility to support other pattern languages mapped to the XML structure defined, consequently allowing its reuse.

Sumário

Lista de Figuras

Lista de Quadros e Tabelas

CAPÍTULO 1

Introdução

1.1.Considerações Iniciais

Muitos sistemas sofrem amadurecimento no decorrer do seu ciclo de vida, o que os tornam mais confiáveis e seguros. Entretanto, em alguns casos, esses sistemas utilizam tecnologia considerada atual no início do projeto e que se torna obsoleta depois de algum tempo.

A existência de diversos sistemas, chamados de legados, desenvolvidos sem técnicas de engenharia de software e, muitas vezes, sem que um documento de requisitos tenha sido elaborado faz com que seus desenvolvedores busquem soluções para amenizar os constantes problemas de manutenção que enfrentam. Engenharia reversa pode ser utilizada nesses casos.

A engenharia reversa recupera informações de sistemas legados, em nível mais alto de abstração, por meio do estudo das suas funcionalidades, análise das estruturas de armazenamento de dados e das tomadas de decisão definidas em seu código fonte. Os documentos produzidos facilitam o entendimento do sistema e guiam o processo de engenharia avante, quando um processo de reengenharia for utilizado. A reengenharia utiliza esses documentos produzidos para apoiar a condução da fase de engenharia avante em que

novas tecnologias e linguagens de programação são escolhidas para a revitalização desses sistemas.

O reuso de software é uma atividade muito comum em empresas que o desenvolvem. Antes da orientação a objetos, o reuso era conseguido através da cópia do código fonte desejado, criado pelo próprio desenvolvedor, que o adaptava a um novo sistema que projetava (Braga, 2003). Com a orientação objetos, o reuso por meio de herança mostra-se mais eficiente na adaptação do código, mas é obtido apenas em pequena escala no processo de desenvolvimento do software. (Bosch et al., 1999[1] apud Braga,2003).

Os padrões de software surgem como uma alternativa para se fazer o reuso de uma forma mais completa, pois eles indicam não só o reuso de código, mas também o de análise, de *design*, de arquitetura e de processo de desenvolvimento (Re, 2002). Com o uso de padrões pode-se documentar diferentes soluções para problemas que ocorrem durante o processo de desenvolvimento de software, de modo que esses possam ser reutilizados em situações semelhantes, por outros desenvolvedores.

Uma linguagem de padrões é uma coleção organizada de padrões que cobrem um determinado domínio, em que o conjunto de padrões que a constitui pode ser aplicado em um todo ou parcialmente, para o desenvolvimento de uma aplicação.

Linhas de produto de software especificam uma maneira sistemática e previsível de realizar o reuso. Existem algumas contradições sobre linhas de produtos de software considerando que existem técnicas com características similares, tais como *frameworks*, padrões e engenharia de domínio, dentre outras. Na verdade, para o desenvolvimento de uma linha de produto de software essas técnicas devem ser aplicadas em conjunto. As linhas de produto de software devem ser vistas como uma forma de organização das técnicas de reuso conhecidas, permitindo às empresas gerenciarem o reuso de seus produtos (Gimenes; Travassos, 2002).

Uma forma de automatizar parte do processo de desenvolvimento de software é por meio de geradores de aplicações, que são ferramentas capazes de gerar aplicações a partir de especificações feitas em alto nível. Essas especificações podem ser documentadas por meio de uma Linguagem de Modelagem da Aplicação (LMA) que devem representar as abstrações (modelos) das aplicações (Weiss; Lai, 1999).

---

1 BOSH, J.; MOLIN, P.; MATTISSON, M.; BENGTSSON, P.; FAYAD, M. Framework problem and experiences in M. Fayad, R. Johnson, D. Schmidt. Building Application Frameworks: Object- Oriented Foundations of Framework Design, John Willey and Sons, p. 55–82, 1999.

1.2. Objetivo

Este trabalho tem por objetivo apresentar um **G**erador de **A**plicações baseadas na **W**eb para sistemas de gestão de **C**línicas de **Re**abilitação, denominado GAwCRe. As aplicações são especificadas, em alto nível, por meio de uma LMA definida neste trabalho, usando os conceitos de linhas de produto de software.

As definições da LMA foram elaboradas com base em uma linguagem de padrões de análise, chamada SiGCli (**Si**stemas de **G**estão de **Clí**nicas), que foi criada, também neste trabalho, para apoiar a modelagem de sistemas no domínio específico de clínicas de reabilitação.

O gerador GAwCRe utiliza a LMA e a SiGCli tanto para a geração das interfaces de instanciação das aplicações quanto para a geração dos seus artefatos. Os gabaritos de código, que definem os artefatos, são armazenados internamente ao código fonte do gerador possibilitando a geração dos *scripts:* de criação do banco de dados; das classes Java da aplicação com suas regras de negócio; e das interfaces de interação com os usuários. Cada um desses gabaritos possui partes comuns a todos os artefatos gerados e partes variáveis em que são atribuídos valores dinamicamente por meio de parâmetros, quando da geração do artefato, possibilitando assim, a criação artefatos distintos com o uso do mesmo gabarito.

As definições da LMA e da SiGCli são feitas em um documento XML para que o gerador GAwCRe possa obter as informações necessárias para instanciar e gerar as aplicações do domínio. A escolha da XML ocorreu devido à possibilidade de outras linguagens de padrões serem mapeadas para a estrutura XML definida para o gerador possibilitando assim, o reuso do gerador em outros domínios.

1.3. Motivação

A motivação para a realização deste trabalho deve-se às constantes pesquisas na área de engenharia de software, visando melhorar e facilitar o processo de desenvolvimento de software, utilizando técnicas de reuso e as pesquisas dos grupos de engenharia de software da UFSCar e da USP em reengenharia de sistemas. O uso de gerador de aplicações permite automatizar o processo de desenvolvimento de sistemas, em um determinado domínio, reusando, através de seus gabaritos de criação, produtos de software confiáveis, que já foram testados.

Nesses seis anos em que trabalho com o desenvolvimento de sistemas, me deparei várias vezes com a manutenção de sistemas desenvolvidos de forma aleatória e que possuíam várias partes reusadas, na realidade "copiadas", de outros sistemas. Assim, reengenharia foi uma das soluções propostas para manter e expandir esses sistemas. Com os estudos realizados e o conhecimento de outras técnicas, a proposta de criação de um gerador de aplicações foi considerada como a mais viável.

A escolha do domínio de sistemas de gestão de clínicas de reabilitação ocorreu devido à necessidade da instituição, Missão Salesiana de Mato Grosso, em migrar os seus sistemas desenvolvidos em ambiente ZIM para uma plataforma mais atual, como a tecnologia Oracle (2002). A quantidade de sistemas que essa instituição possui nesse domínio, motivou a escolha, pois assim, o gerador desenvolvido pode ser utilizado para gerar um conjunto maior de aplicações, o que justifica o esforço no seu desenvolvimento.

1.4. Organização do Trabalho

Esta dissertação está organizada da seguinte forma: O Capítulo 2 apresenta os assuntos relacionados que auxiliaram no desenvolvimento do trabalho. O Capítulo 3 mostra o processo de análise de domínio e de elaboração da linguagem de padrões para **Si**stemas de **G**estão de **Cl**ínicas de Reabilitação (SiGCli) que é à base do gerador de aplicações. A definição da Linguagem de Modelagem da Aplicação (LMA) que é usada para especificar as aplicações a serem geradas, bem como a arquitetura do gerador GAwCRe e das aplicações por ele geradas são apresentadas no Capítulo 4. Um exemplo de uso do gerador de aplicações gerando uma aplicação do domínio é mostrado no Capítulo 5. As considerações finais sobre o trabalho são apresentadas no Capítulo 6.

CAPÍTULO 2

Assuntos Relacionados

2.1. Considerações Iniciais

O desenvolvimento e a manutenção de sistemas computacionais, são atividades trabalhosas. Muitas empresas têm dificuldades em manter seus sistemas devido a alguns fatores: o desenvolvedor do software não está mais na empresa, dificultando a obtenção dos requisitos do negócio; o hardware ou o software do sistema não é mais suportado; o esforço da manutenção (tempo e custo) é grande, tais sistemas recebem o nome de sistemas legados

A engenharia reversa é uma fase do processo de reengenharia, na qual se obtêm, a partir do código ou documentação existente, as informações dos sistemas legados em um nível mais alto de abstração. A recuperação dessas informações possibilita a reconstrução de tais sistemas, através de um processo de reengenharia, utilizando-se novas tecnologias. Muitas abordagens são propostas na literatura para auxiliar no processo de engenharia reversa e reengenharia. (Penteado,1996), (Recchia, 2002), (Lemos,2002), (Demeyer et al. 2000).

O reuso de software é uma atividade que permite melhorar a produtividade, a manutenibilidade e a qualidade tanto do software desenvolvido quanto do processo de

desenvolvimento. Padrões de software permitem que sejam evidenciadas as características comuns que podem ser reusadas em novos projetos e aplicações.

Este capítulo apresenta os assuntos pertinentes ao trabalho desenvolvido, como algumas técnicas, abordagens e ferramentas usadas. Ele está organizado da seguinte forma: a seção 2.2 aborda o ambiente ZIM, pois é nele que os sistemas legados considerados estão desenvolvidos. Engenharia reversa, reengenharia e algumas abordagens para auxiliar o processo de reengenharia de sistemas procedimentais para orientados a objetos são apresentadas na seção 2.3. A seção 2.4 trata dos padrões de software e de linguagem de padrões, sendo a GRN apresentada com maiores detalhes. A seção 2.5 apresenta *frameworks,* referenciando o mais relevante para este trabalho, o GREN. A seção 2.6 trata de geradores de aplicações. Na seção 2.7 são comentadas algumas abordagens para linhas de produto de software. A seção 2.8 apresenta as características da linguagem XML, que foi usada no desenvolvimento do gerador de aplicações proposto e na a seção 2.9 são feitas considerações finais.

2.2. O Ambiente ZIM

ZIM é um Sistema Gerenciador de Banco de Dados (SGBD) relacional que difere dos demais por permitir a implementação do Modelo Entidade-Relacionamento (MER) fiel ao modelo conceitual projetado. Ou seja, um relacionamento N:N entre duas entidades é implementado sem a necessidade de se criar uma nova entidade para mapear o relacionamento, como acontece normalmente com a maioria dos SGBD's. Além disso, possui uma linguagem de programação integrada ao seu dicionário de dados (Brown,1990). As seções seguintes descrevem características sobre o ambiente ZIM. A palavra objeto(s) usada nesta seção refere-se aos elementos criados nas entidades e relacionamentos do ambiente ZIM para uma aplicação.

2.2.1. Visão Geral do ZIM.

Uma aplicação ZIM consiste de uma base de dados e de uma coleção de programas, que são elaborados para permitir que os usuários da aplicação interajam com a base de dados (Brown, 1990).

O dicionário de dados é um repositório de informações que descreve todos os tipos de objetos necessários para o desenvolvimento de aplicações. O próprio dicionário de dados é

implementado como uma base de dados Entidade-Relacionamento eliminando a necessidade de uma linguagem de definição de dados distinta. A linguagem ZIM é empregada na manipulação do dicionário de dados exatamente da mesma forma que é empregada para manipular a base de dados em uma aplicação. Ela possui um conjunto de comandos que permitem manipular, isto é adicionar, eliminar e alterar os dados da base de dados da aplicação.

O ambiente ZIM além de possuir uma linguagem própria para a manipulação de uma base de dados no modelo entidade-relacionamento, possibilita o uso dos comandos no padrão SQL. Dessa forma os desenvolvedores podem utilizar os comandos ZIM ou os comandos do tipo SQL para ter acesso à base de dados. A Figura 2.1 apresenta o MER da estrutura do dicionário de dados ZIM. Em que as informações referentes a uma aplicação são armazenadas em entidades e relacionamentos desse modelo.

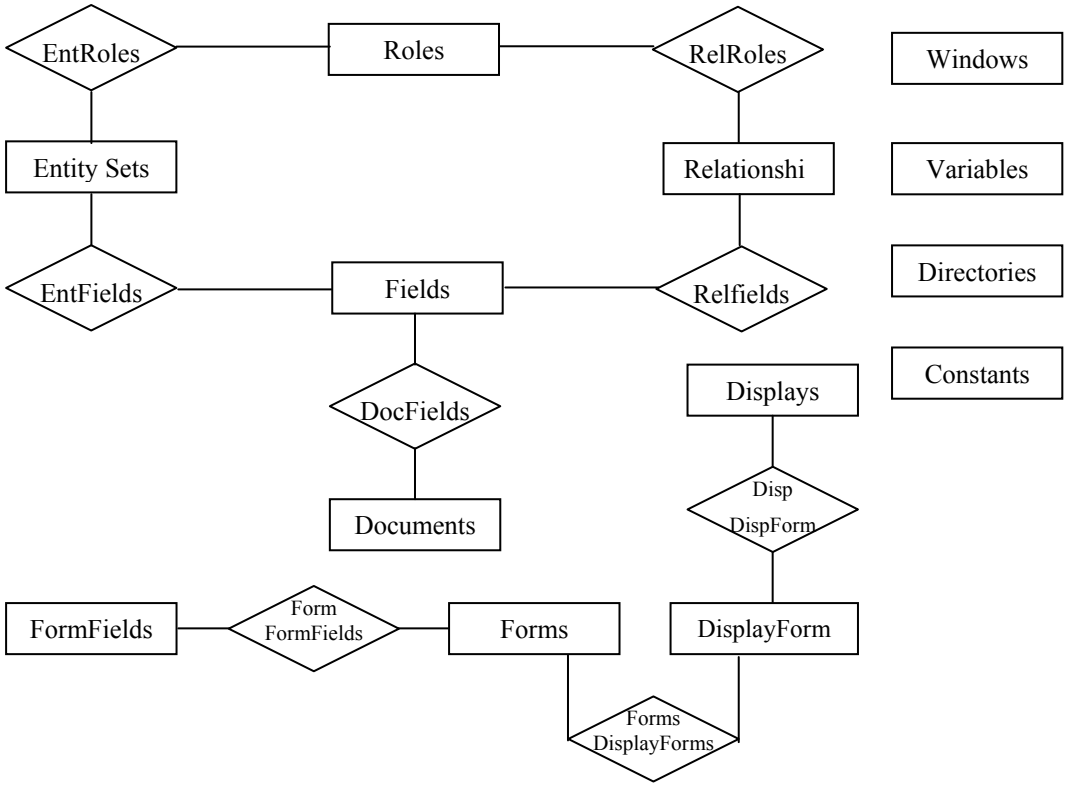


Figura 2.1 – MER da estrutura do dicionário de dados do ZIM

2.2.2. Características da Linguagem ZIM: Objetos e Operadores

No dicionário de dados, cada objeto criado pelo programador recebe um nome. A linguagem ZIM consiste de comandos capazes manipular esses objetos que são classificados

em três categorias, sendo que cada uma possui um conjunto de operadores associados como mostra o Quadro 2.1.

Quadro 2.1 – Tipos de objetos da base de dados ZIM

Objetos da base de dados	
Entidades(*Entitys Set*)	Uma coleção de dados organizados em forma tabular (tabela).
Campos(*Fields*)	São atributos usados em entidades, relacionamentos e documentos. Cada campo tem seus próprios atributos que definem suas características.
Relacionamentos (*Relationships*)	Representa uma interação entre entidades. Existem dois tipos: os que possuem campos de dados e os que não possuem.
Papéis (*Roles)*	Permitem o uso de nomes diferentes para o mesmo objeto em contexto diferentes. Um relacionamento "Casamento" é definido entre "Homens" e "Mulheres", que são papéis da entidade "Pessoas"
Conjuntos Nominados (*Named Sets*)	São "imagens" temporárias de ocorrências específicas da base de dados e podem conter dados de uma ou mais entidades e/ou relacionamentos.
Objetos de Interface com o usuário	
Janelas (*Windows*)	São ambientes independentes, separados, para interação com a aplicação a partir de um terminal físico.
Formulários (*Forms*)	Consiste de um ou mais campos de formulário que possuem atributos comuns
Campos de Formulário(*FormFields*)	Compõem um formulário.
Formulários de Entrada de Dados (*Displays)*	São conjuntos de formulários sobrepostos que agem como se fossem uma única imagem de apresentação na tela do seu terminal ou numa janela
Objetos de Programação	
Documentos (*Documents*)	Representam um arquivo texto no sistema operacional. Em geral são usados para armazenar o código fonte de programas escritos em ZIM
Variáveis (*Variable*)	Possuem um tipo de dado e comprimento e devem ser definidas no dicionário de dados antes do seu uso nos programas.
Constantes (*Constants*)	São objetos definidos no dicionário de dados com valores fixos e que podem ser usados durante a execução de um programa.

2.3. Engenharia Reversa e Reengenharia

O desenvolvimento de sistemas computacionais demanda esforços, pois envolve operações complexas e minuciosas. O levantamento de requisitos deve ser bem realizado para que todas as funcionalidades do sistema sejam obtidas. Em seguida, essas devem ser modeladas para posterior implementação em uma linguagem de programação que atenda às características do sistema. Na maioria dos sistemas existentes o processo de desenvolvimento não seguiu essas etapas e somente o código fonte é que permanece como documentação. Em alguns casos, os sistemas passaram por diversas etapas de manutenção sem que houvesse registro do que foi alterado, perdendo-se informações sobre as regras de negócio que norteiam esses sistemas. Outras vezes, os desenvolvedores desses sistemas já não se encontram mais na empresa e todas as modificações realizadas e as decisões tomadas no desenvolvimento do sistema são perdidas. Esses sistemas recebem o nome de legado independente do tempo de vida que tenham.

Engenharia reversa é um processo de análise de um sistema existente que identifica seus componentes e os representa em um nível mais alto de abstração (Chikofsky et al.,1990). Ela vem sendo aplicada em sistemas legados visando a sua futura reengenharia ou somente para representá-los de outra forma, com nível mais alto de abstração. A proposta da engenharia reversa é a elaboração de um modelo de projeto ou de análise, a partir do código fonte do sistema existente para facilitar o seu entendimento, explicitando as regras de negócio embutidas. Um dos objetivos para que se realize engenharia reversa em sistemas legados refere-se aos ganhos que podem ser obtidos com as melhorias realizadas no sistema e com as futuras manutenções. Após a engenharia reversa de um sistema as alterações tornam-se mais fáceis, reduzindo, dessa forma, os gastos na fase de manutenção (Penteado, 1996).

A reengenharia de software visa a contenção de custos da manutenção de sistemas, indicando a maximização de investimentos nos softwares já existentes, recriando-os, em vez de abandona-los ou de criar novos softwares. É comum que grande parte desses sistemas satisfaçam aos requisitos dos usuários, mas não disponham de documentação de análise e de projeto ou que essas estejam desatualizadas, dificultando manutenções de qualquer tipo, além de não utilizarem tecnologias/recursos atuais disponíveis. O processo de reengenharia de sistemas procedimentais para orientados a objetos é composto de: engenharia reversa + Δ + engenharia avante (Jacobson, 1991), onde Δ refere-se as decisões que o engenheiro de software toma quanto ao possível acréscimo de algumas funcionalidades ao sistema. Neste trabalho é considerado esse tipo de reengenharia.

2.3.1. As Abordagens para o Processo de Engenharia Reversa e Reengenharia

Muitas propostas são apresentas na literatura para processos de engenharia reversa e reengenharia. Penteado (1996) apresenta uma abordagem denominada Fusion/RE que permite recuperar um modelo de análise orientado a objetos a partir do sistema legado procedimental. Esse modelo pode ser utilizado tanto para tarefas de manutenção, como para iniciar o processo de reengenharia orientada a objetos. O método é genérico e pode ser utilizado em sistemas implementados em qualquer linguagem procedimental. Porém, para que sua aplicação seja otimizada deve-se especializá-lo criando diretrizes específicas para cada linguagem.

Cagnin (1999) utiliza o Fusion/RE em um processo de reengenharia de um sistema implementado em C para Java, utilizado padrões de persistência e banco de dados relacional. Camargo (2001) utiliza a abordagem Fusion/RE para a reengenharia de sistemas COBOL que são disponibilizados para Web.

A utilização da abordagem Fusion/RE, em estudos de casos, mostrou sua deficiência por se tratar de um processo seqüencial linear. Assim Recchia e Penteado (2002) propõe uma família de padrões, denominada FaPRE/OO (Família de Padrões para Reengenharia Orientada a Objetos), para a realização da reengenharia de sistemas legados orientados a procedimentos para orientados a objetos, especialmente para sistemas implementados em Clipper para Delphi com características orientadas a objetos. A FaPRE/OO é baseada nos passos do Fusion/RE, tornando o processo evolutivo e usando os modelos orientados a objetos da fase de análise da UML (UML, 2002). Essa família de padrões está dividida em quatro *clusters*. Os três primeiros guiam o processo de engenharia reversa e o último dá um enfoque para a engenharia avante.

Lemos (2002) propõe um aprimoramento da FaPRE/OO, definindo padrões para garantir a qualidade durante o processo de reengenharia num processo denominado PRE/OO (Processo de Reengenharia Orientado a Objetos), com foco em sistemas implementados em Delphi sem utilizar arquitetura orientada a objetos para sistemas Delphi em que essa arquitetura é usada. O processo é definido em sete *clusters*, compostos por vinte padrões, que atendem às seguintes etapas: planejamento do processo, engenharia reversa e engenharia avante, sendo dois *clusters* especialmente criados para a garantia de qualidade, sendo que um deles é aplicado durante todo o processo. Os *clusters* de garantia da qualidade foram elaborados a partir de KPAs (*Key Process Areas*) do nível 2 de maturidade do CMM (CMM, 2003).

Demeyer et al.(2000), propõem uma linguagem de padrões para conduzir engenharia reversa de sistemas orientados a objetos. Essa linguagem de padrões obtém as informações dos sistemas legados a partir do código fonte, da organização e de pessoas; conduzindo o processo de engenharia reversa no contexto de orientação a objetos e esses padrões serviram como base para os trabalhos de Recchia (2002) e Lemos (2002). Todas essas propostas visam facilitar o processo de engenharia reversa, de forma a documentar o que está sendo realizado.

Ré (2002) propõe um processo de engenharia reversa de sistemas através da análise de interface de sistemas Web. Para esse processo são escolhidos três sistemas-base, que são rastreados para a obtenção das principais funcionalidades dos sistemas. Esse processo possui quatro fases: Analisar Informações dos Sistemas-Base, Identificar as Páginas do Sistema-Base, Exercitar o Sistema-Base, Refinar os Requisitos.

Ramos et al. (2004) propõem uma abordagem, denominada *Aspecting,* para conduzir o processo de reestruturação de sistemas desenvolvidos no paradigma Orientado a Objetos para paradigma Orientado a Aspectos , com o intuito de separar os requisitos não funcionais, que ficam espalhados e entrelaçados no código fonte, dos interesses funcionais da aplicação.

2.4. Padrões de Software e Linguagem de Padrões

2.4.1. Padrões de Software

O reuso de software é uma atividade comum em empresas desenvolvedoras de software, devido à necessidade de melhorar a produtividade, a manutenibilidade e a qualidade tanto do software quanto do processo de desenvolvimento. Uma das formas para possibilitar o reuso é com padrões de software, pois eles podem ser usados em vários níveis de abstração (código, projeto, análise, arquitetura e processo de desenvolvimento) (Ré,2002).

O conceito de padrões surgiu na década de 70 com o arquiteto Chirstopher Alexander (Alexander, 1979[3]; Alexander et al, 1977[4] apud Braga, 2003). Segundo Alexander, cada padrão descreve a solução de um problema que pode ocorrer inúmeras vezes em um ambiente. Essa solução é utilizada diversas vezes, não necessariamente da mesma forma. Apesar de Alexander atuar na área de arquitetura, o mesmo conceito pode ser empregado no processo de desenvolvimento de software.

---

[3] ALEXANDER, C. The timeless way of building. New York: Oxford University Press, 1979.
[4] ALEXANDER, C.; ET AL. A pattern language. New York: Oxford University Press, 1977.

Padrões de software descrevem as soluções de sucesso para os problemas de desenvolvimento de software que geralmente ocorrem em determinados contextos, com a finalidade de auxiliar os desenvolvedores menos experientes com os conhecimentos adquiridos dos mais experientes (Fayad et. al 1999). Dessa forma desenvolvedores novatos podem agir como um especialista (Gamma et. al, 1995). Um padrão pode ser descrito como um esquema composto por três elementos:

- Contexto: que é a situação que origina o problema;
- Problema: que é a dificuldade encontrada no desenvolvimento; e
- Solução: que é a maneira comprovada e consistente para solucionar o problema.

Projetistas que conhecem determinados padrões podem aplicá-los imediatamente como solução para problemas sem ter que redescobri-las (Gamma et al.,1995). Com o intuito de facilitar o reuso dos padrões de software, pode-se classificá-los em diversas categorias, porém tal classificação não é rigorosa pois podem existir padrões que se encaixam em mais de uma categoria (Ré, 2002):

- Padrões de processo: definem soluções para os problemas encontrados nos processos envolvidos na engenharia de software, por exemplo, desenvolvimento, controle de configuração, testes, etc;
- Padrões arquiteturais: expressam o esquema ou organização estrutural fundamental de sistemas de software ou hardware;
- Padrões de análise: descrevem soluções para problemas de análise de sistemas, embutindo conhecimento sobre o domínio de aplicação específico;
- Padrões de projeto: definem soluções para problemas de projeto de software;
- Padrões de programação: descrevem soluções de programação particulares de uma determinada linguagem ou regras gerais de estilo de programação;

Braga (2003) cita alguns formatos, existentes na literatura, que podem ser adotados para descrever os padrões:

- Formato de Portland: inspirado no formato de Alexander, formado por apenas três elementos: a) nome do padrão, b), um parágrafo contendo a contextualização e a descrição do problema, e c) um parágrafo contendo a solução do problema, as conseqüências, as colaborações.

- Formato de GoF[5]: cada padrão contém os seguintes elementos: nome do padrão; intenção ou motivação; aplicabilidade; estrutura; participantes; colaborações; conseqüências; implementação; código exemplo; usos conhecidos e padrões relacionados.
- Formato de Appleton: tenta padronizar os formatos dentre os diversos utilizados por outros autores. Assim, cada padrão possui os seguintes elementos: nome; problema; contexto; influências; solução (a qual pode ter subseções estrutura, participantes, dinâmica, implementação e variantes); exemplos; contexto resultante; justificativa; padrões relacionados e usos conhecidos.
- Formato proposto por Demeyer et al. (2000): é composto por a) Nome; b) Intuito; c) Contexto (Influências); d) Problema; e) Solução; f) Avaliação; g) Justificativa; h) Usos Conhecidos; i) Padrões Relacionados.

2.4.2. Linguagem de Padrões

Uma linguagem de padrões é uma coleção estruturada de padrões que se apóiam entre si para transformar requisitos e restrições numa arquitetura (Coplien, 1998). Os padrões que a constituem devem abranger todos os aspectos importantes de um determinado domínio e pelo menos um padrão deve estar disponível para cada aspecto da construção e da implementação de um sistema de software. 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[6] apud Braga, 2003).

Uma linguagem de padrões pode auxiliar a subdivisão de problemas gerais com soluções complexas em problemas menores e relacionados, de forma a facilitar a solução. Uma característica importante para as linguagens de padrões é que cada padrão pode ser usado de forma isolada ou com um certo número de padrões da linguagem, com isso um único padrão é considerado útil mesmo se a linguagem não for aplicada na sua totalidade.

Braga et al.(1998, 1999) e (Braga,2003) definem uma linguagem de padrões para Gestão de Recursos de Negócios (GRN), apresentada com maiores detalhes na seção 2.4.3. Na seção 2.4.4 são apresentadas outras linguagens de padrões.

---

[5] A sigla GoF significa Gang of Four, apelido dado aos quatro autores do livro "Design Patterns" (Gamma et al.,1995)

[6] BRUGALI, D.; MENGA, G.; AARSTEN, A. A case study for flexible manufacuring systems in M. Fayad, R. Johnson. Domain-Specific Application Frameworks: Frameworks Experience by Industry, John Willey and Sons, p. 85–99, 2000.

2.4.3. A Linguagem de Padrões GRN (Gestão de Recursos de Negócios)

A Linguagem de Padrões para Gestão de Recursos de Negócios (GRN), é composta por quinze padrões de análise, sendo alguns deles aplicações ou extensões de padrões existentes na literatura. Essa linguagem auxilia desenvolvedores menos experientes no desenvolvimento de aplicações que tratam de gestão de recursos de negócios, ou seja, nas quais existe a necessidade de registrar transações de aluguel, comércio ou manutenção de recursos. Na Figura 2.2, pode-se observar os relacionamentos e dependências entre os padrões existentes. As linhas mais espessas indicam os principais padrões da linguagem: LOCAR O RECURSO, COMERCIALIZAR O RECURSO e MANTER O RECURSO.

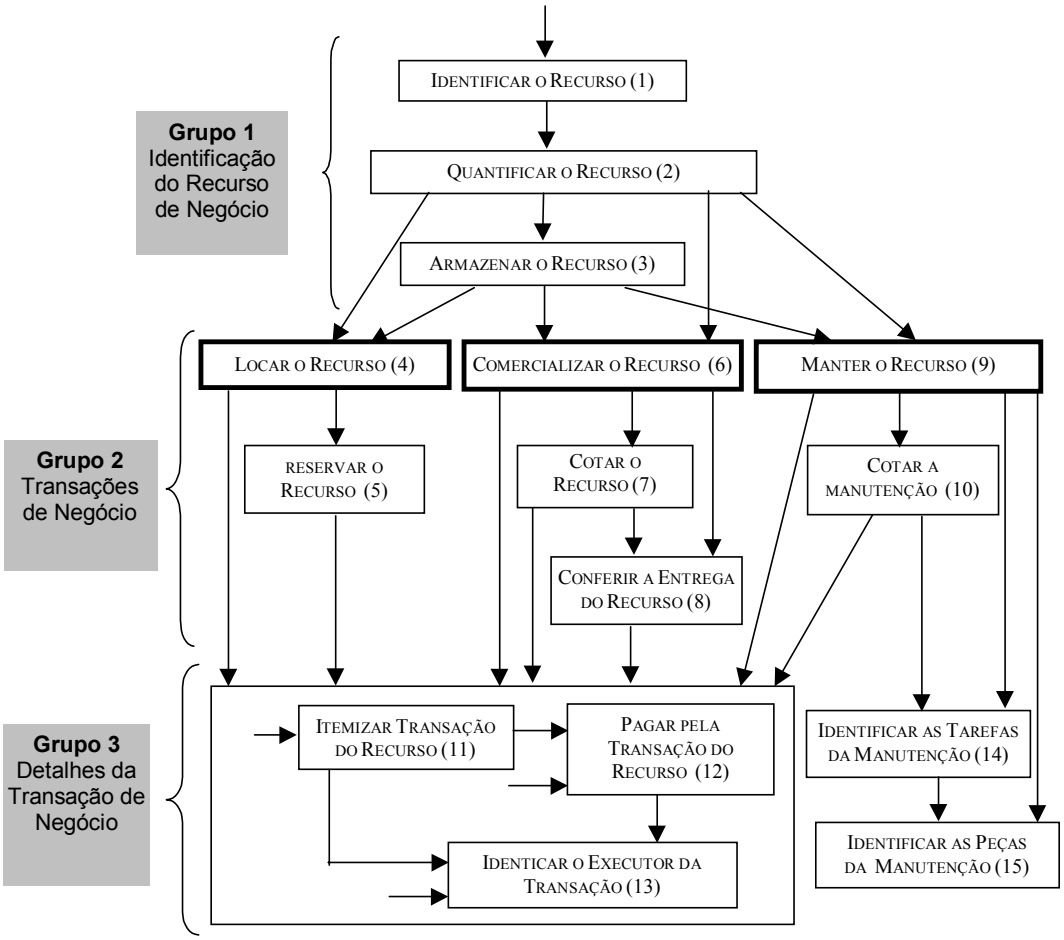


Figura 2.2 – Relacionamento entre os padrões da linguagem GRN (Braga,2003)

Os padrões estão agrupados de acordo com seu propósito. O primeiro grupo é composto por três padrões: IDENTIFICAR O RECURSO, QUANTIFICAR O RECURSO e ARMAZENAR O RECURSO, que tratam da identificação e possível qualificação, quantificação e armazenagem dos recursos gerenciados pelo negócio. No segundo grupo

estão os padrões relacionados à manipulação dos recursos de negócio pelo sistema. Existem sete padrões nesse grupo: LOCAR O RECURSO, RESERVAR O RECURSO, COMERCIALIZAR O RECURSO, COTAR O RECURSO, CONFERIR A ENTREGA DO RECURSO, MANTER O RECURSO e COTAR A MANUTENÇÃO. O terceiro grupo possui cinco padrões que cuidam de detalhes das transações efetuadas com o recurso. Os três primeiros, ITEMIZAR TRANSAÇÃO DO RECURSO, PAGAR PELA TRANSAÇÃO DO RECURSO e IDENTIFICAR O EXECUTOR DA TRANSAÇÃO, são aplicáveis a quaisquer das transações do grupo 2. Os dois últimos, 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.

2.4.4. Outros Exemplos de Linguagens de Padrões

A linguagem de padrões G++, proposta por Aarsten et al. (1995[7] apud Ré 2002) é voltada para manufatura integrada ao computador. Essa linguagem aborda o projeto de sistemas de informação concorrentes e, provavelmente distribuídos, com aplicações na manufatura integrada ao computador. Com isso pretende-se aumentar o reuso de projetos orientados a objetos desde o nível de componentes até o nível arquitetural, oferecendo um modelo conceitual para arquiteturas concorrentes e distribuídas.

A linguagem de padrões chamada "*Caterpillar's Fate: A Pattern Language for Transformation from Analysis to Design*", proposta por Kerth[8] (1995 apud Ré 2002) é usada para apoiar a transformação dos documentos de análise em um projeto inicial de software. A finalidade da linguagem é captar a experiência adquirida durante o desenvolvimento de soluções de projeto a partir de modelos de análise, documentando o que deve ser feito durante a transição.

A linguagem de padrões "*The CHECKS Pattern Language of Information Integrity* ", apresentada por Cunningham[9] (1995 apud Braga, 2003), mostra como separar os dados válidos dos inválidos, durante a entrada de dados, assegurando que um menor número de dados inválidos sejam registrados.

A Linguagem de Padrões LV (Linguagem de Padrões para Leilões Virtuais) proposta por Ré et al. (2001), aplica-se ao desenvolvimento de sistemas para gestão de vendas por

---

[7] AARSTEN, A.; ELIA, G.; MENGA, G. *Pattern languages of program design*, capítulo G++ : A pattern Language for Computer-Integrated Manufactoring. In: Coplien e Schmidt (1995), p. 91–118, 1995.

[8] KERTH, N. *Pattern languages of program design*, capítulo Caterpillar's Fate: A pattern Language for Transformation from Analysis to Design. In: Coplien e Schmidt (1995), p. 293–320, 1995.

intermédio de leilões virtuais. Ela é constituída de 10 padrões e pode ser considerada como uma extensão da GRN, pois trata de uma transação (leilão) não coberta pela GRN.

2.5. Frameworks

Framework é uma infra-estrutura genérica, baseada em um domínio, que pode ser adaptada para solucionar problemas específicos desse domínio, servindo como um modelo para a construção de aplicações através da especificação das classes e das colaborações entre elas.

Existem diversas definições sobre *frameworks,* entre elas: a de Roberts e Johnson (1998) que definem *framework* como um projeto, reutilizável em todo ou parte de um sistema, que é representado por um conjunto de classes abstratas e por um modelo que representa as interações de suas instâncias; a de Gamma et al. (1995) que definem *framework* como um conjunto de classes relacionadas que suportam a reutilização em projetos com classes específicas do mesmo domínio.

Um *framework* funciona como um molde para a construção de aplicações ou subsistemas, dentro de um domínio. Todas as aplicações construídas a partir de um mesmo *framework* apresentam características similares, diferenciando-se em seu comportamento, que varia conforme a necessidade da aplicação. Isto torna as aplicações desenvolvidas, a partir do *framework*, mais fáceis de se manter e mais consistentes para os usuários, que não precisam aprender diferentes aplicações. Internamente ele possui em sua estrutura partes variáveis e fixas. As partes variáveis são chamadas de *hot-spots* e as partes fixas são chamadas de *frozen-spots*.

Um *framework* permite além do reuso de código, também o reuso de projeto, liberando o desenvolvedor de cuidar dos aspectos comuns daquele domínio de aplicação, aumentando assim a sua produtividade (Johnson; Foote,1998).

Uma característica muito importante dos *frameworks* é a inversão de controle. Em um *framework* bem projetado, o desenvolvedor implementa somente o código dos métodos das classes específicas que corresponderão à sua aplicação, sendo responsabilidade do *framework* chamar esse código quando for necessário. Já em bibliotecas de classes orientadas a objetos ou procedimentais, além do código da aplicação propriamente dito, o desenvolvedor

---

[9] CUNNINGHAM, W. *Pattern languages of program design*, capítulo The CHECKS Pattern Language of Information Integrity. In: Coplien e Schmidt (1995), p. 145–155, 1995.

deve especificar o fluxo de execução, a estrutura do programa, etc. Essa inversão de controle torna o *framework* uma infra-estrutura extensível, pois os métodos elaborados pelo desenvolvedor especializam os algoritmos genéricos definidos nesse *framework* para uma aplicação específica (Johnson; Foote,1998).

2.5.1. O *Framework* GREN (Gestão de REcursos de Negócios)

Com base na Linguagem de Padrões GRN, foi construído o *Framework* GREN (**G**estão de **RE**cursos de **N**egócios) (Braga; Masiero, 2002). Ele permite criar aplicações no domínio de sistemas de gestão de recursos de negócios e foi desenvolvido em linguagem *Smalltalk*. A Figura 2.3 mostra a arquitetura do *Framework* GREN.

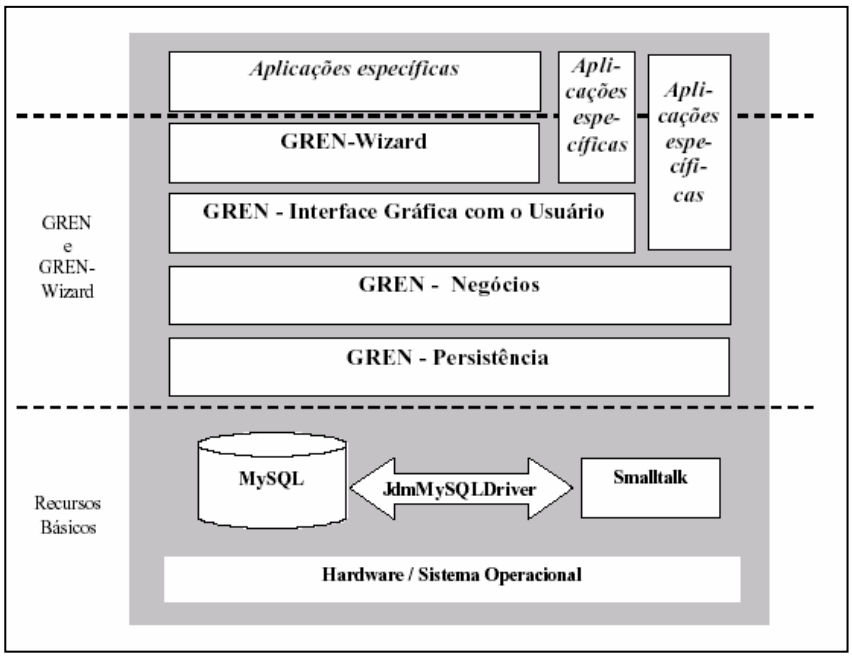


Figura 2.3 – Arquitetura do Framework GREN (Braga,2003)

A camada de *GREN-Persistência* possui classes para cuidar da conexão com a base de dados, gerenciamento dos identificadores dos objetos e persistência dos objetos. A camada de *GREN-Negócios* comunica-se com a camada de *GREN-Persistência* sempre que for necessário armazenar permanentemente um objeto. Dentro da camada de *GREN-Negócios* existem diversas classes derivadas diretamente dos padrões de análise que compõem a GRN, ou seja, as classes e relacionamentos contidos em cada padrão possuem a implementação correspondente nessa camada. A camada de *GREN-Interface Gráfica com o Usuário* contém as classes responsáveis pela entrada e saída de dados, como formulários de interface, janelas e menus, que permitem a interação do usuário final com o sistema. Essa camada comunica-se

com a de *Negócios* para obtenção de objetos a serem mostrados na interface com o usuário. A camada *GREN-Wizard*, encontra-se acima da camada de interface gráfica com o usuário, pois utiliza todas as demais camadas para realizar a instanciação de aplicações de forma gráfica com base na GRN.

As aplicações específicas podem ser instanciadas a partir da camada de *GREN-Interface Gráfica com o Usuário*, usando (por meio de herança ou referência a objetos) classes de todas as camadas do GREN, ou com base na camada de *GREN-Negócios*, caso a camada *GREN-Interface Gráfica com o Usuário* não seja reutilizada a partir do GREN, mas implementada separadamente, ou ainda do *GREN-Wizard*, que se encarrega de fazer a comunicação com as demais camadas do GREN.

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.

2.5.2. Outros Exemplos de *Frameworks*

A maioria dos *frameworks* existentes aplica-se a domínios técnicos (ou básicos), tais como interfaces com o usuário ou distribuição. O Model-View-Controller (MVC, 2002) (MVC, 2004) (MVC, 2004a) foi o primeiro *framework* amplamente utilizado. Implementado inicialmente em Smalltalk-80, conta com implementações em todas as versões existentes de *Smalltalk*,, bem como em outros ambientes de implementação. O MVC é usado como interface com o usuário, tendo mostrado a adequação da orientação a objetos para implementação de interfaces gráficas com o usuário. Outro exemplo é o *HotDraw* (HotDraw, 2003), que é um *framework* gráfico bidimensional para editores de desenho estruturado, escrito no VisualWorks *Smalltalk*. Ele tem sido usado para criar diversos editores diferentes.

Muitos *frameworks* de aplicação não são de domínio público, como o IBM San Francisco para domínios específicos de negócio. Seu objetivo é auxiliar a criação de aplicações de negócio que sejam apropriadas (para uma grande variedade de aplicações) para esse domínio. Ele utiliza padrões de projeto para gerenciar e organizar sua documentação, além de se beneficiar das características que os padrões proporcionam (Carey et al., 2000[10] apud Braga, 2003). O San Francisco apóia aplicações de gerenciamento de armazenamento, gerenciamento de pedidos, livro razão, conta a pagar, contas a receber etc.

O *framework* OmniBuilder (Omnisphere, 2003), gerencia o ciclo de vida completo das aplicações, captando o modelo de objetos, incluindo regras de negócios, requisitos e casos de uso. Ele gera aplicações completas a partir do Modelo de Negócios, que pode ser fornecido manualmente durante a análise ou pode ser criado utilizando os Gabaritos de Negócios (*Business Templates*).

O *Framework* Qd+ foi desenvolvido com base na Linguagem de Padrões LV (Ré,2002) com o processo definido por Braga e Masiero (2002). Ele permite instanciar aplicações para o domínio de Leilões Virtuais e pode ser considerado uma extensão do *Framework* GREN. Uma diferença entre os *frameworks* GREN e o Qd+ é que o Qd+ tem uma interface web, usando a arquitetura de três camadas (3-*tier*), na qual existe um navegador na máquina cliente, que corresponde à camada de apresentação, um servidor de Web ou servidor HTTP, que corresponde à camada de aplicação, e um servidor de banco de dados, que corresponde à camada de persistência.

2.6. Geradores de Aplicações

Durante o processo de desenvolvimento de sistemas, o reuso é uma das atividades mais realizadas pelo engenheiro de software, sendo que na maioria das vezes, esse consiste de repetidas operações de "cortar/colar/modificar". A qualidade do artefato produzido depende da quantidade de vezes que as operações de reuso são realizadas. Quanto mais adaptações são necessárias, pior é a qualidade do produto final, pois essas operações repetitivas e manuais levam a possíveis inserções de erros (Franca, 2000). A eliminação desses erros pode ser minimizada com a automatização das operações envolvidas no reuso. A automatização transforma as atividades do desenvolvimento de software em especificações em alto nível, declarando o quê a aplicação deve fazer de uma forma menos complexa. As aplicações que permitem especificações de um produto, em um nível mais alto de abstração, são denominadas de geradores de aplicações

Geradores de aplicações são ferramentas de software que conseguem automatizar parte de um processo rotineiro na atividade de desenvolvimento de software, acelerando o processo de implementação, transformando especificações de alto nível em produtos da aplicação.

---

[10] CAREY, J.; CARLSON, B.; GRASER, T. *SanFrancisco design patterns: Blueprints for business software*. Addison-Wesley, 2000.

Geradores de aplicações podem ser considerados como compiladores para uma linguagem de um domínio específico (Smaragdaski; Batory, 1998). Essa linguagem pode ser elaborada após a análise de domínio e pode ser representada através de uma linguagem de padrões, por exemplo.

O uso de geradores de aplicações só é justificado em uma organização quando o reuso, de um determinado produto de software, é realizado em grande escala (Pfleeger, 1998[11] apud Franca 2000). Isto é, quando se possui grande quantidade de softwares para um mesmo domínio (família de software).

Os principais benefícios no uso de geradores são (Franca,2000):

- **Aumento de produtividade**: pois a maior parte do produto gerado corresponde a trechos fixos referentes a detalhes de implementação, isso proporciona uma elevação da produtividade da equipe de desenvolvimento.
- **Redução do tempo para o mercado**: o tempo de desenvolvimento de um novo produto limita-se ao tempo de sua especificação
- **Prototipação**: novos produtos são gerados alterando-se as especificações fornecidas ao gerador, facilitando assim o desenvolvimento de novas versões e possibilitando ao desenvolvedor conduzir diversos experimentos para determinar o produto mais adequado ao usuário.
- **Qualidade no produto gerado**:uma vez que o funcionamento do gerador esteja correto, o produto gerado também estará, sendo que possíveis problemas no produto estão associados a problemas na especificação fornecida ao gerador

Para se construir um gerador de aplicações deve-se entender a estrutura comum (similaridades) e variável (variabilidades) do domínio específico. O modelo do domínio pode ser visto como um guia para se fazer à análise das similaridades desses sistemas, mas geralmente o estudo das variabilidades não pode ser feito de uma forma sistemática.

As variações podem ser entendidas pelo conhecimento do engenheiro de software sobre o domínio desejado. Torna-se necessário entender as similaridades e as variabilidades para que se possa projetar uma ferramenta de geração poderosa, que cubra o máximo possível do domínio. A necessidade de se modelar as similaridades e as variabilidades do sistema surge naturalmente no decorrer da evolução do processo de desenvolvimento, e torna-se muito importante no reuso (Jarzabeck, 1995).

---

[11] PFLEEGER, S.L.; *Software Engineering Theory and Practice*; Prentice-Hall; NJ; 1998

Duas abordagens podem ser adotadas com relação ao desenvolvimento de um gerador de aplicações. A primeira consiste na construção de um gerador mais genérico, que possui grande quantidade de parâmetros, permitindo assim uma especificação variável para produto final a ser produzido. Essa abordagem é de difícil implementação, uma vez que a estrutura do gerador se torna bastante complexa. A segunda abordagem consiste na simplificação da construção de geradores específicos, caso o gerador não seja capaz de gerar um produto para um domínio específico, um outro deve ser desenvolvido para aquele domínio, isso torna a estrutura do gerador mais simples de ser implementada (Franca,2000).

2.7. Linhas de Produtos de Software

Um software deve atender às necessidades do usuário, ser confiável, ter alta usabilidade e manutenibilidade, ter desenvolvimento rápido com baixo custo e ser competitivo no mercado (Weiss; Lai,1999). Para se obter software com qualidade e que seja economicamente viável é necessário um conjunto sistemático de processos, técnicas e ferramentas. A técnica mais relevante em todo esse conjunto é o reuso. Reutilizando-se partes especificadas, desenvolvidas e já testadas, ganha-se em tempo de desenvolvimento e confiabilidade do produto. Muitas técnicas para o reuso foram propostas durante os últimos anos, tais como: *frameworks*, padrões, engenharia de domínio. Entretanto, o que falta nesse contexto é uma maneira precisa de realizar o reuso (Gimenes; Travassos, 2002). Dessa forma, linhas de produtos de software surgem como uma abordagem para construção sistemática de software baseado em uma família de produtos.

O conceito de família de produtos é muito utilizado em outros ramos da indústria: nos setores automobilístico, aeroespacial, componentes eletrônicos entre outros, em que um processo de produção é definido e seguido até a construção do produto final. Embora software seja desenvolvido de uma maneira completamente diferente de carros, motos, aviões ou computadores, os engenheiros de software podem utilizar estratégias similares para o seu desenvolvimento.

2.7.1. Família de Produtos de Software e Linha de Produtos de Software.

Família de produtos de software refere-se a um conjunto de produtos de software com características suficientemente similares para permitir a definição de uma infra-estrutura

comum de estruturação dos itens que compõem os produtos e a parametrização das diferenças entre esses produtos (Gimenes; Travassos, 2002).

Uma linha de produto de software envolve um conjunto de aplicações similares dentro de um domínio que podem ser desenvolvidas a partir de uma arquitetura genérica comum, a arquitetura da linha de produto, e um conjunto de componentes que povoam a arquitetura. Tem por finalidade identificar as semelhanças e as diferenças entre os produtos (artefatos) de software durante todo o processo de desenvolvimento da linha de produto, definindo as tomadas de decisão para que a adaptação dos componentes para geração de produtos específicos possa ser realizada. Uma linha de produto de software define:

- Um conjunto de aplicações com características comuns, pertencentes a um determinado domínio e que podem ser desenvolvidas a partir de uma arquitetura genérica básica;
- A arquitetura da linha de produto e;
- Um conjunto de componentes, previamente desenvolvidos, que constituem a arquitetura.

2.7.2. Elementos de uma Linha de Produtos de Software.

A construção de uma linha de produto de software está dividida em três atividades essenciais, segundo Clements e Northrop (2001[12] apud Gimenes; Travassos, 2002), que são:

- Desenvolvimento do núcleo de artefatos (engenharia de domínio);
- Desenvolvimento do produto (engenharia de aplicação);
- Gerenciamento da linha de produto;

A Figura 2.4 representa o modelo do processo de desenvolvimento de uma linha de produto de software. Com dois ciclos distintos, porém relacionados.

O primeiro ciclo mostra o desenvolvimento do núcleo de artefatos, denominado de engenharia de domínio, cujas fases são: Análise de Domínio, Desenvolvimento da Arquitetura da Linha de Produto e Desenvolvimento dos Componentes Reutilizáveis. Esse ciclo produz um Modelo de Domínio, uma Arquitetura da Linha de Produto, um conjunto de Componentes Reutilizáveis e geradores de software para a linha de produtos.

---

[12] CLEMENTS, P., NORTHROP, L. Software Product Lines: Practices and Patterns, SEI Series in Software Engineering, Addison-Wesley. 563 p, 2001

Dentre todos os artefatos existentes em uma linha de produto de software, a Arquitetura da Linha de Produto é a que possui o papel principal, pois será a infra-estrutura central de desenvolvimento, utilizada na construção dos membros da família. Ela representa o maior investimento de todo o processo de definição da linha de produto. Dessa forma, antes de se decidir pela implantação de uma linha de produtos, deve-se analisar se o investimento necessário no seu desenvolvimento ocasionará o retorno esperado (Weiss; Lai, 1999).

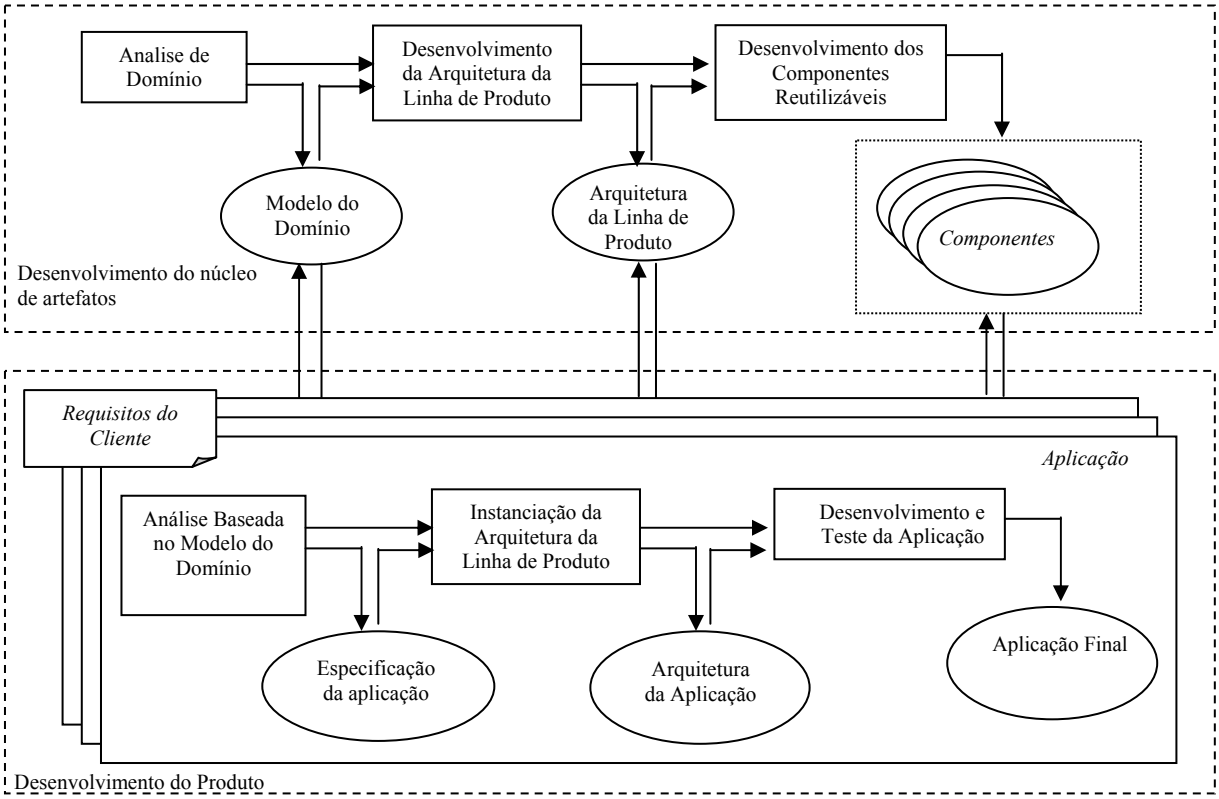


Figura 2.4 – O modelo do processo de desenvolvimento da linha de produto

A arquitetura de uma linha de produtos pode ser definida através de uma Linguagem de Modelagem da Aplicação (LMA) (em inglês *Application Modeling Language- AML*) que deve ser usada para especificar os membros da família. A LMA é definida como uma linguagem para enfatizar que as especificações escritas em uma linguagem devem representar modelos. A arquitetura então deve ser capaz de analisar as especificações escritas nessa LMA (Weiss; Lai,1999). A implementação da LMA pode ser feita por um *framework* ou um por gerador de aplicações específico para o domínio desejado.

Outros artefatos a serem considerados pela linha de produtos de software são os componentes gerados pelo núcleo de artefatos. Tais componentes são desenvolvidos para que

possam ser unidades independentes de software, podendo ser reutilizados por meio de suas interfaces, que devem estar bem definidas. Eles têm papel importante no desenvolvimento de novas aplicações, pois são elementos já testados em aplicações que serão reutilizados no desenvolvimento de novas aplicações.

O segundo ciclo da Figura 2.4 trata do desenvolvimento do produto, denominado de engenharia de aplicação. Nesse ciclo são utilizados os artefatos gerados pelo núcleo de artefatos e são realizadas três atividades essenciais:

- Análise Baseada no Modelo do Domínio, que é utilizada para definição das variabilidades da aplicação, refinando as informações específicas sobre a aplicação a ser desenvolvida.
- Instanciação da Arquitetura da Linha de Produtos que deve ser feita através da arquitetura especificando-se o produto segundo a LMA definida para o domínio.
- Desenvolvimento e Testes da Aplicação que se refere à utilização dos componentes existentes a fim de desenvolver o produto final e realizar os testes com a aplicação baseados nos Requisitos dos Clientes.

A Especificação da Aplicação, Arquitetura da Aplicação e a Aplicação Final são os artefatos produzidos por esse ciclo. Outro fator importante na fase de desenvolvimento do produto é o gerenciamento do produto. Grande parte do sucesso de uma linha de produto depende do compromisso da organização em construí-la e mantê-la. Assim, é importante que a organização estabeleça um plano de adoção da linha de produtos que descreva o estado desejado e as estratégias para atingir este estado. Para tal é necessário prover a equipe responsável pela linha de produto com recursos e garantir que as atividades da linha de produtos sejam coordenadas e supervisionadas. Deve haver um sincronismo entre o grupo que desenvolve os artefatos e o grupo que gera os produtos. Deve também ser garantido o apoio ao desenvolvimento e à evolução dos artefatos da linha.

2.7.3. Abordagens para o Desenvolvimento de Linhas de Produtos.

Nesta seção são apresentadas algumas abordagens existentes no mercado para o desenvolvimento de uma linha de produto de software.

PLP (Product Line Practice) do SEI/CMU

A iniciativa PLP (SPL, 2002) foi elaborada pelo *Software Engineering Institute* (SEI) e tem como objetivo o desenvolvimento de uma linha de produto de software com atividades

relacionadas ao desenvolvimento de artefatos centrais e ao desenvolvimento e gerenciamento de produtos que se utilizam desses artefatos. Para isso são definidas áreas de trabalho (*Practice Area*) com planos de trabalho e métricas associadas que auxiliam o acompanhamento da execução e avaliação dos trabalhos realizados. A estratégia PLP define três áreas de trabalho: engenharia de software, gerenciamento técnico e gerenciamento organizacional. É uma descrição de abordagem similar ao CMM (CMM,2003).

Synthesis

Elaborada pelo Software Productivity Consortium (PL, 2002) essa abordagem descreve uma forma de construção de sistemas através de instâncias de sistemas que possuem descrições parecidas. Quatro princípios definem essa abordagem:

- Família de programas: os programas devem ser desenvolvidos, não como artefatos únicos, mas como instâncias de uma família de programas similares.
- Processo iterativo: o processo de desenvolvimento de produtos de software é definido como uma repetição de atividades de produção e utilização.
- Especificações: as propriedades verificáveis devem ser descritas considerando-as para um produto membro ou um conjunto de produtos membros a serem produzidos.
- Reutilização baseada em abstrações: a família de produtos deve ser representada como um conjunto de produtos membros adaptáveis e também para derivar instâncias de cada produto membro e produzir um sistema produto em particular.

FAST

A análise de características comuns é um aspecto marcante da utilização de FAST (Family-oriented Abstraction, Specification and Translation). O FAST é um padrão de processo de produção de software que objetiva resolver os problemas entre uma produção rápida e uma engenharia cuidadosa. Ele está organizado em três sub-processos (Weiss; Lai, 1999):

- Qualificação do Domínio: cujo propósito é identificar a família merecedora de investimento.
- Engenharia de Domínio: cujo propósito é investir em facilidades para produção de membros da família.

- Engenharia de Aplicação: que tem por objetivo fazer uso das facilidades desenvolvidas para uma rápida produção dos membros da família.

Cada um desses sub processos possui padrões característicos para as atividades. A Qualificação do Domínio consiste de uma análise econômica da família uma vez que será necessário um alto investimento para produção dessa família, analisando-se qual delas possui um grande número de membros instanciáveis. A Engenharia de é o processo que exige o maior investimento, pois nessa fase será desenvolvida a arquitetura que facilitará o processo de instanciação dos membros da família. A Engenharia de Aplicação se beneficia dos produtos e artefatos gerados pela Engenharia de Domínio para uma rápida produção de membros da família.

2.8. A linguagem XML (*eXtensible Markup Language*)

XML é uma linguagem que faz parte do subconjunto de uma outra linguagem a SGML (Standard Generalized Markup Language) (W3C,2004), que permite o armazenamento de informação de forma muito bem estruturada e organizada. Uma das principais vantagens do XML em relação ao HTML é que o desenvolvedor pode criar suas próprias *tags*, sem se prender às pré-determinadas como é o caso do HTML.

Com a flexibilidade do XML existem seis tipos de marcações que podem ocorrer em um documento XML (Pitts-Moultis; Kirk, 2000):

- *Elementos*: são as formas mais comuns de marcação. Delimitados pelos sinais de menor (<) e maior (>), a maioria dos elementos identificam a natureza do conteúdo que envolve. A marcação que determina o término de um elemento é indicada pelo caracter `/`. Os elementos podem possuir atributos.
- *Referências a Entidades*: devem ser usadas para introduzir caracteres que são reservados da linguagem XML, por exemplo, O sinal de menor, < , identifica o início de uma marca de elemento. Para inserir estes caracteres em um documento como conteúdo é necessário fazer referência à entidade, por exemplo `&It`. Correspondentemente, o sinal de maior é referenciado por `&Dt`
- *Comentários:* iniciam com `<!--`e terminam com `-->`. Os comentários podem conter qualquer dado, exceto a literal `"--"`. Podem-se colocar comentários entre

marcas em qualquer lugar em seu documento. Comentários não fazem parte de um conteúdo textual de um documento XML.

- *Instruções de Processamento:* são formas de fornecer informações a uma aplicação. Assim como os comentários, elas não são textualmente parte de um documento XML.
- *Seções Marcadas:* define uma seção onde o analisador XML deve ignorar os caracteres de marcação. É definida pela marcação `<![CDATA[` *conteúdo da seção* `]]>`,
- *Declarações de Tipos de Documento:* permite a definição das regras de um documento XML. Há quatro tipos de declarações em XML: Declarações de Tipos de Elementos, Declarações de Listas de Atributos, Declarações de Entidades e Declarações de Notações.

2.8.1. Parser DOM XML para a Linguagem Java

Um documento XML é representado por nós em uma árvore de estruturas de objetos. Cada nó representa uma parte do documento XML, incluindo atributos, elementos e entidades. Essa flexibilidade permite armazenar informações em um documento XML que possam ser recuperadas posteriormente usando *parsers* específicos para esse propósito (Cleaveland, 2001).

O DOM (*Document Object Model*) é uma plataforma e uma interface de linguagem neutra, definida pela W3C (W3C, 2004), que permite que programas e *scripts* acessem dinamicamente e atualizem o conteúdo, a estrutura e o estilo dos documentos XML. Os documentos podem ser processados e os resultados do processamento podem ser retornados a quem solicitou a análise da estrutura do documento (DOM, 2004).

Para que um programa Java acesse um documento XML, por exemplo, o DOM prove uma interface padrão para localizar e manipular informações através da árvore de objetos do documento XML, permitindo a navegação por entre os nós do documento (Cleaveland, 2001). O parser DOM cria a estrutura de objetos de um documento XML na memória. Para isso é necessário o uso de um *parser* que seja capaz de ler um arquivo XML. Existem vários *parsers* atualmente que podem ser usados para esse objetivo, tais como Oracle (Oracle, 2003), IBM's alphaWorks (IBM, 2003), Apache (Apache, 2003) e Sun (SUN, 2003).

Além do DOM existem também outros *parsers* como o SAX (*Simple API for XML*) que pode ser usado para criar customizações no parser XML, definindo-se quais itens do

documento XML devem ser analisados. Para este trabalho optou-se pelo uso do parser DOM, devido à facilidade para a manipulação do documento XML.

2.9. Considerações Finais

Este capítulo apresentou a maioria dos conceitos utilizados neste trabalho. A engenharia reversa e a reengenharia permitem a reconstrução de sistemas legados, obtendo-se as informações sobre sistema em um nível mais alto de abstração (engenharia reversa), e os reconstruindo com a utilização de novas tecnologias/recursos (reengenharia). O processo de engenharia reversa é usado neste trabalho com o propósito de se fazer à análise de domínio dos sistemas legados desenvolvidos na linguagem ZIM.

Linguagens de padrões são conjuntos de padrões que interagem com a finalidade de transformar os requisitos e restrições em uma arquitetura. Esses padrões devem abranger todos os aspectos importantes para um determinado domínio. As linguagens de padrões definem as diretrizes que explicam como e quando utilizá-los. Linguagens de padrões são utilizadas neste trabalho com o intuito de auxiliar o desenvolvimento de uma linguagem de padrões para um domínio específico, o de clínicas de reabilitação, denominada de SiGCli (Sistemas de Gestão de Clínicas de reabilitação).

Um *framework* é um projeto genérico em um domínio que pode ser adaptado a aplicações específicas, servindo como um molde para a construção de aplicações. Todas as aplicações construídas a partir de um mesmo *framework* apresentam a mesma estrutura, diferenciando-se em seu comportamento.

Geradores de aplicações provêm projetos genéricos que são construídos por meio de parametrização de componentes, dessa forma eles evitam o alto número de problemas e permitem o aumento de produção. A limitação da abordagem que faz uso de geradores é que o reuso é restrito a um certo problema, no qual o domínio do gerador se aplica.

Franca (2000) usa o termo gerador de artefatos ao invés de gerador de aplicações, isso porque ele considera que uma aplicação é um tipo de artefato que será entregue ao cliente final, mas existem outros artefatos que são gerados durante o processo de geração tais como *scripts* de banco, documentação, etc. Um gerador então é uma ferramenta que produz um artefato a partir de sua especificação. Os resultados obtidos, artefatos, podem ser programas, módulos, documentação, arquivos de configuração,dentre outros.

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 da aplicação por meio do gerador (Franca; Staa, 2001).

Linhas de produto de software são vistas como uma área muito promissora, pois oferecem uma maneira sistemática, planejada e prática de reutilização de software. Uma característica importante no enfoque de linha de produtos é a forma como ela se propõe a resolver problemas, redução de custo e tempo de produção, aumentando assim a sua produtividade. A efetiva aplicação do enfoque de linha de produtos está atualmente favorecida por um lado pelo amadurecimento das técnicas de engenharia de software e por outro pela disponibilidade de algumas tecnologias de apoio. Existem duas formas de implementar a arquitetura de uma linha de produto de software: usando *frameworks* ou usando geradores de aplicações. Linha de produto de software foi a abordagem usada, neste trabalho, para definir as especificações do domínio em alto nível, com base na linguagem de padrões SIGCli, definiu-se a arquitetura da linha de produto através de uma LMA. Essa LMA é usada na construção do gerador GAwCRe.

A linguagem XML é um padrão de representação de informações, podendo ser usada para representar qualquer domínio ou aplicação. A flexibilidade do XML permitiu definir toda Linguagem de Padrão SiGCli em um único arquivo XML, assim usando o parser DOM é possível elaborar um gerador, em linguagem Java, que seja capaz de ler as definições XML e gerar a aplicação com base nessas informações.

CAPÍTULO 3

A Análise de Domínio e a Elaboração de Linguagem de Padrões SiGCli

3.1.Considerações Iniciais

A engenharia de domínio é a atividade de coletar, organizar e armazenar experiências no desenvolvimento de sistemas ou de partes específicas de sistemas em um domínio particular, permitindo o reuso de recursos para a construção de novos sistemas. Domínio refere-se à área do conhecimento caracterizada por um conjunto de conceitos e terminologias entendidas por profissionais dessa área (Cleveland, 2001). Engenharia de domínio é subdividida em três atividades(Czarnecki; Eisenecker, 2000):

- **Análise de domínio:** que tem por objetivo selecionar e definir o domínio, coletando informações coerentes para modelá-lo, representando as variabilidades e similaridades existentes entre os sistemas desse domínio.
- **Projeto do domínio:** seu propósito é desenvolver a arquitetura para a família de sistemas no domínio e projetar um plano de produção, definindo os passos a serem seguidos para a construção dos produtos.
- **Implementação do domínio:** envolve a implementação da arquitetura, componentes e o plano de produção usando tecnologia apropriada.

O modelo do domínio de uma aplicação deve representar a grande maioria das funcionalidades presentes nas aplicações, sendo necessário observar as soluções mais comumente aplicadas para resolver os problemas desse domínio. Ele pode ser representado por um conjunto de elementos tais como diagramas de classes, diagramas de estado, diagramas de caso de uso, etc (Ré,2002). Para este trabalho a representação usada é uma de linguagem de padrões.

Esse capítulo descreve o processo utilizado para a realização da análise do domínio usando como base sistemas legados desenvolvidos em ZIM. Após a análise de domínio, uma linguagem de padrões para Sistemas de Gestão de Clínicas de Reabilitação (SiGCli) foi elaborada. A seção 3.2 apresenta a descrição dos sistemas usados como base para o processo de análise de domínio. Na seção 3.3 é descrito o processo de análise de domínio e na seção 3.4 o processo de elaboração da linguagem de padrões. A seção 3.5 discute como os padrões da SiGCli foram elaborados com base nos padrões da GRN e na seção 3.6 apresenta as considerações finais sobre o capítulo.

3.2. Descrição dos Sistemas Atuais

O processo de análise de domínio foi realizado através do estudo de três sistemas no domínio de clínicas de reabilitação física (fisioterapia, terapia ocupacional e educação física). Nas seções 3.2.1, 3.2.2 e 3.2.3 tem-se as descrições dos sistemas das Clínicas de: Fisioterapia (CliFisio), Terapia Ocupacional (CliTO) e Educação Física (CliEF) respectivamente. Os dois primeiros, CliFisio e CliTO, são utilizados nas Clínicas do Centro de Reabilitação Física Dom Bosco, pertencente à Missão Salesiana de Mato Grosso – Salesiano de Lins - SP, e foram desenvolvidos pela própria instituição. O terceiro, CliEF, ainda não foi desenvolvido e sua especificação foi baseada nas necessidades dos usuários e no conhecimento do funcionamento de outros sistemas para esse domínio.

3.2.1. O Sistema da Clínica de Fisioterapia (CliFisio)

O CliFisio tem a finalidade de controlar todo o atendimento realizado em pacientes que procuram tratamento fisioterápico. Quando um paciente inicia o tratamento na clínica, a recepcionista realiza o seu cadastro, classificando-o segundo alguns atributos: sexo, cor da pele, grau de escolaridade, faixa de renda, faixa etária, etc.

Com o paciente cadastrado, é aberta uma guia de consulta para que ele possa iniciar o tratamento. Na guia de consulta constam informações sobre o convênio do paciente, número de sessões a ser realizadas, que varia de acordo com o convênio; o nome do médico responsável pelo encaminhamento do paciente e a patologia por ele diagnosticada. Para cada convênio existe um código de classificação da patologia com a quantidade de coeficiente de honorário (chs), correspondente a cada uma, usados para efetuar o pagamento dos tratamentos. Após a guia ser aberta, é realizado o agendamento das sessões.

As sessões são agendadas conforme a disponibilidade de horário dos estagiários e do paciente. O mesmo estagiário acompanha o paciente durante todas as sessões de uma mesma guia. Dessa forma, o paciente fica vinculado a um determinado estagiário, mas caso ocorra algum problema com ele, um outro pode atender o paciente. Após ter o seu horário agendado, o paciente é informado do inicio do tratamento.

A cada sessão a recepcionista confirma a presença do paciente no sistema. Essa confirmação tem dois objetivos: a) a simples confirmação de presença, pois se o paciente faltar a três sessões precisará de uma nova guia para dar continuidade ao tratamento; e b) para notificar ao estagiário, por meio de um terminal de consulta, da presença dos pacientes para que a sessão possa ser iniciada.

A cada guia do paciente, na primeira sessão, são armazenados os seus dados clínicos e é realizada a avaliação do seu estado pelo estagiário responsável. Nos dados clínicos são armazenadas as hipóteses diagnosticadas, a história pregressa do paciente, a história da moléstia atual e outras informações adicionais como: se o raio X da lesão foi apresentado (caso possua), dias de imobilização, se fez ou faz uso de algum medicamento, etc. A avaliação do estado do paciente é realizada segundo alguns critérios pré-definidos pelos fisioterapeutas responsáveis pela clínica. Essas avaliações estão divididas em áreas conforme o tratamento a ser realizado, por: ortopedia, hidroterapia, pneumologia, cardiologia, neurologia. Com exceção da avaliação neurológica, todas as outras são realizadas apenas na primeira sessão de cada guia apresentada pelo paciente.

A cada sessão são armazenadas as evoluções do paciente e a avaliação da sessão. Quanto à evolução são analisados alguns pontos pré-definidos como: limiar de dor, ausculta pulmonar, entre outros. Na avaliação da sessão o estagiário descreve o tratamento realizado com paciente comentando os seus pontos positivos e negativos. Uma escala de valores representando a intensidade de dores que um paciente sente é usada para avaliar o tratamento que está sendo aplicado. Assim, um valor é atribuído a cada avaliação e, posteriormente, é

comparado com os outros, verificando-se a evolução obtida pelo paciente. Somente após a avaliação da sessão do paciente, é confirmada a sua saída da agenda diária do estagiário.

Depois de finalizadas as sessões agendadas pode ser necessário dar continuidade ao tratamento, para isso o paciente deve apresentar uma nova solicitação, por meio de uma guia. A nova guia pode ser aberta com duas situações diferentes:

- Dando continuidade ao tratamento, nesse caso é necessário que o paciente permaneça com o mesmo estagiário.
- Iniciando um novo tratamento, nesse caso não é necessário ser o mesmo estagiário, pois o paciente apresenta uma nova patologia.

O sistema permite a consulta a tratamentos anteriormente realizados pelos pacientes, com isso os estagiários podem avaliar os tratamentos já aplicados a diagnósticos similares aos que estão atendendo. Um levantamento estatístico sobre os tratamentos realizados também pode ser obtido.

Funcionalidades do Sistema

As principais funcionalidades que compõem esse sistema são:

- Cadastro de: pacientes, dados clínicos, avaliações, novas guias, agendas, feriados, usuários, avaliação da sessão do pacientes, evolução dos pacientes. Existem alguns dados adicionais que auxiliam no gerenciamento da clínica, como: escolaridade do paciente, faixa de renda, cor, medicamentos usados pelos pacientes, testes especiais utilizados em tratamentos, convênios, ausculta pulmonar, ritmo respiratório, manobras realizadas em tratamentos, estado psicológico, antecedentes familiares, patologias congênitas, sinais de enfarto, dor muscular, dor articular, classificação cardiológica, estagiários e patologias. Esses dados facilitam os levantamentos estatísticos necessários.
- Rotinas de Processamento: re-agendar pacientes, trocar de estagiários (mudança que acontece a cada 3 meses), trocar de estagiários por paciente (troca que acontece quando um paciente vai ser atendido por outro estagiário), cancelar agenda dos pacientes por falta (pacientes que tiveram três faltas com horário agendados) e atualização dos horários dos estagiários.
- Rotinas diárias: verificar o horário dos estagiários, visualizar guias para a avaliação diária (estas guias que se tornam disponíveis depois que a atendente

recepciona o paciente) e alterar tratamento. Dentro das rotinas diárias existem dois subgrupos que são os das rotinas de recepção de pacientes e os das rotinas dos estagiários. Nas rotinas de recepção dos pacientes, existem as seguintes funções: recepção de pacientes, consulta de diagnósticos dos pacientes, consulta de situação das guias, horários dos estagiários, reagendamento de paciente, troca de estagiários e alguns relatórios como: as fichas de assinatura dos pacientes (necessária para a comprovar que o paciente está freqüentando a clínica), estatística de atendimentos por convênio, ficha de atendimento, relatório para avaliação dos estagiários. Nas rotinas de estagiários tem-se: a agenda do estagiário, guias para a evolução da sessão, tratamentos realizados anteriormente e reavaliações dos pacientes.

3.2.2. O Sistema da Clínica de Terapia Ocupacional (CliTO)

O CliTO tem a finalidade de controlar todo o atendimento realizado a pacientes que procuram tratamento em Terapia Ocupacional. Esse sistema possui muitas funcionalidades similares ao CliFisio quanto:

- Ao cadastro de pacientes;
- À abertura de guias;
- Ao agendamento das sessões;
- À realização de cada sessão;
- Ao acompanhamento do tratamento;
- À finalização das sessões;
- À consulta de tratamentos realizados.

As diferenças existentes são:

- A cada novo paciente da clínica é feita uma entrevista inicial, denominada anamnese, para relatar as informações importantes que auxiliam na definição dos tratamentos. A anamnese está dividida em duas áreas (adulto e infantil) seguindo parâmetros pré-definidos pelos terapeutas. Na anamnese infantil são coletadas informações sobre o paciente, desde o nascimento até a presente data. Na anamnese adulto são coletadas informações sobre o paciente desde a lesão até a presente data.

- Na primeira sessão de cada guia do paciente, são armazenados os dados clínicos do paciente e é realizada a avaliação do seu estado, pelo estagiário responsável. Nos dados clínicos são armazenadas as hipóteses diagnosticadas, a história pregressa do paciente, a história da moléstia atual e outras informações adicionais como, se apresentou raio X da lesão (caso possua), dias de imobilização, se faz uso de algum medicamento, etc. A avaliação do estado do paciente é realizada segundo alguns critérios pré-definidos pelos terapeutas ocupacionais responsáveis pela clínica e divididas em infantis e adultas. As avaliações infantis são as: neurológicas, psicomotora, TDE (Teste de Desempenho Escolar), física e portage. Para a avaliação adulta existe apenas a avaliação física. Todas essa avaliações são realizadas em datas aleatórias, dependendo da necessidade da evolução do paciente.

Funcionalidades do Sistema

As principais funcionalidades que compõem esse sistema são:

- Cadastro de: pacientes, dados clínicos, avaliações, novas guias, agendas, feriados e usuários. Existem alguns dados adicionais que auxiliam no gerenciamento da clínica, como os dados de : escolaridade do paciente, taxa de renda, cor, medicamentos usados pelos pacientes, testes especiais utilizados em tratamentos, convênios, escolas, estagiários e patologias. Esses dados facilitam os levantamentos estatísticos necessários.
- Rotinas de Processamento: re-agendar pacientes, troca de estagiários (mudança que acontece a cada 3 meses), troca de estagiários por paciente (troca que acontece quando um paciente vai ser atendido por outro estagiário), cancelamento pacientes por falta (pacientes que tiveram três faltas com horário agendados) e atualização dos horários dos estagiários.
- Rotinas diárias: verificar o horário dos estagiários. Dentro das rotinas diárias existem dois subgrupos que são o das rotinas de recepção de pacientes e o das rotinas dos estagiários. Nas rotinas de recepção dos pacientes existem as seguintes funções: recepção de pacientes, horários dos estagiários, re-agendamento de paciente, troca de estagiários e alguns relatórios como: as ficha de assinatura dos pacientes (necessária para a comprovar que o paciente está freqüentando a clínica), estatística de atendimentos por convênio, ficha de

atendimento e relatório para avaliação dos estagiários. Nas rotinas de estagiários tem-se: a agenda do estagiário, a evolução da conduta do paciente, e avaliações dos pacientes e as entrevistas iniciais (anamnese adulto e infantil).

3.2.3. O Sistema da Clínica de Educação Física (CliEF)

O CliEF tem por finalidade controlar o atendimento a pacientes que procuram a clínica com para se reabilitar de alguma lesão ou realizar atividades físicas.

Quando um paciente procura a clínica para realizar uma atividade física, a recepcionista realiza o seu cadastro e a devida matrícula nas atividades desejadas. Em seguida, é feito o agendamento das atividades a serem realizadas. Para cada atividade há um estagiário responsável, dependendo do horário de sua realização.

Na matrícula é feito o agendamento da avaliação (física, postural, nutricional,...) do paciente. Na primeira avaliação é feita uma entrevista inicial (anamnese) para que o estado atual do paciente seja descrito.

O sistema deve controlar o pagamento de mensalidades por parte dos pacientes, devendo identificar aquele que deixou de efetuá-lo para que seja impedido de freqüentar as aulas na clínica.

Funcionalidades do sistema

- Cadastros de: atividades, clientes, locais de realização de atividades e impedimentos, estagiários.
- Controles: Agendas dos clientes, horário das atividades e mensalidades
- Avaliações: Física, Postural, Anamnese, etc.
- Relatórios: Aniversariantes do mês, clientes em débito e avaliações.
- Deseja-se realizar vendas de produtos específicos, controle de contas a pagar e contas a receber e controle de estoque de produtos específicos.

3.3.O Processo de Análise de Domínio

O passo inicial para a elaboração de uma linguagem de padrões é a definição de um modelo de domínio capaz de representar as funcionalidades presentes na maioria das aplicações desse domínio alvo. Uma das formas de obter informações sobre as

funcionalidades desse domínio é por meio de um processo de engenharia reversa de alguns sistemas já existentes. Durante esse processo são criados modelos intermediários que representam cada sistema para em seguida realizar um processo de generalização desses modelos, que resultará no modelo de análise de domínio (Ré et al. 2001).

A Figura 3.1 ilustra o processo de criação do modelo do domínio da aplicação proposto por Ré (2002) usando a notação SADT (Ross, 1977). Os retângulos representam as atividades que devem ser realizadas, as entradas e saídas dessas atividades são representadas por setas à direita e à esquerda dos retângulos respectivamente os controles são as setas na parte superior dos retângulos, representam as informações necessárias para a realização das atividades, e os mecanismos, setas na parte inferior dos retângulos, representam os sistemas existentes que serão utilizados.

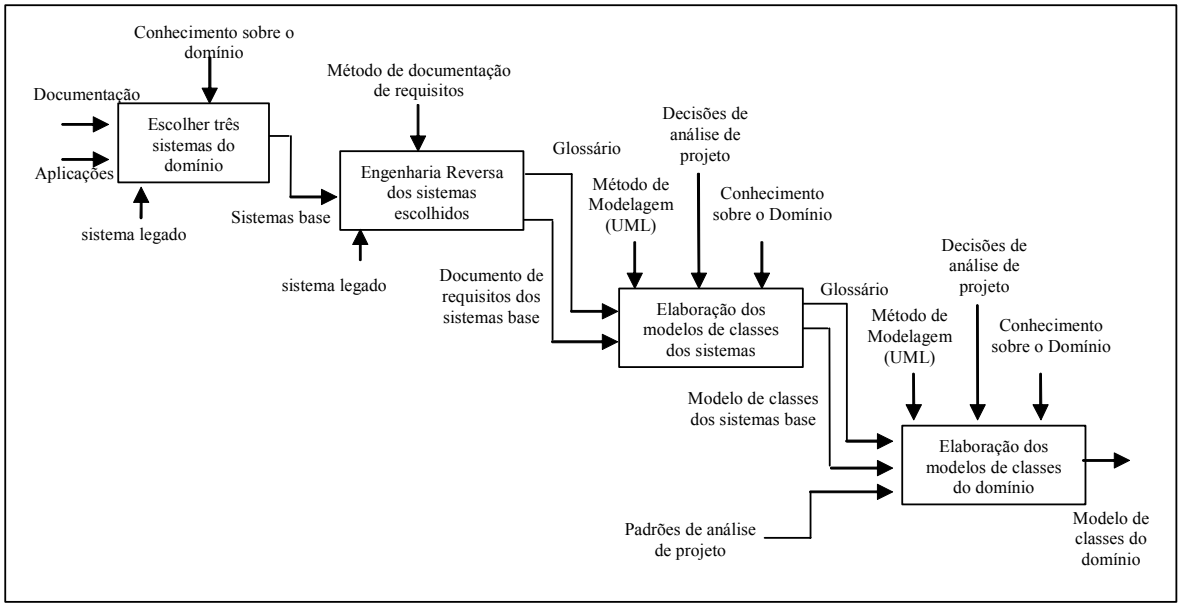


Figura 3.1 – Processo de criação do modelo do domínio da aplicação proposto por Ré (2002)

Para este trabalho, essas atividades foram parcialmente modificadas e dois apoios computacionais foram elaborados para auxiliá-las. A Figura 3.2 apresenta o processo de criação do modelo de domínio da aplicação aqui utilizado com alterações realizadas a partir do processo proposto por Ré (2002) exibido na Figura 3.1. As atividades dos passos 1 e 3, deste processo, são as mesmas que as dos passos 1 e 4 propostos por Ré, enquanto que as atividades dos passos 2 e 3 de Ré são englobadas em uma única atividade .

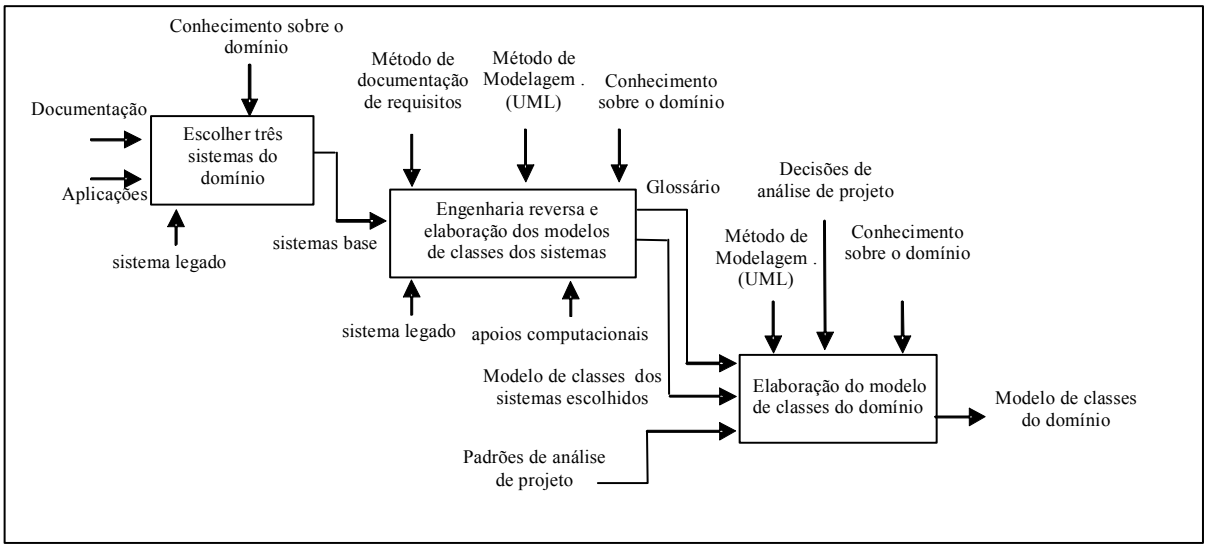


Figura 3.2 – Processo de criação do modelo do domínio.

3.3.1. Passo 1: Escolher três sistemas do domínio

Trata da escolha de quais sistemas servirão de base para o estudo do domínio. Roberts e Johnson (1998) sugerem que o desenvolvimento de um *framework* deve acontecer após o desenvolvimento de três aplicações concretas que irão fornecer conhecimento suficiente para permitir, de forma gradativa, a generalização dessas aplicações. Embora o objetivo deste trabalho não seja a elaboração de um *framework*, esses conceitos podem ser aplicados para a análise de domínio.

Para este trabalho foram utilizados sistemas legados desenvolvidos em linguagem ZIM, e não os SibWebs como proposto originalmente pelo processo de Ré(2002). Dois dos três sistemas escolhidos como sistemas base são utilizados pela no Centro de Reabilitação Física Dom Bosco, pertencente às Faculdades Salesianas de Lins-SP, como apresentados anteriormente: CliFisio, é usado para gerenciar a clínica de fisioterapia e o CliTO é usado para gerenciar a clínica de terapia ocupacional. O terceiro sistema escolhido, CliEF, foi elaborado com base em um levantamento de requisitos da clínica de educação física denominado.

3.3.2. Passo 2: Engenharia reversa e elaboração dos modelos de classes dos sistemas.

O principal objetivo deste passo é encontrar e documentar, da forma mais completa possível, as funcionalidades dos sistemas legados escolhidos executando todas as atividades apresentadas na Figura 3.3, adaptada de Ré(2002). A fase de engenharia reversa deve ser realizada completamente para cada sistema escolhido.

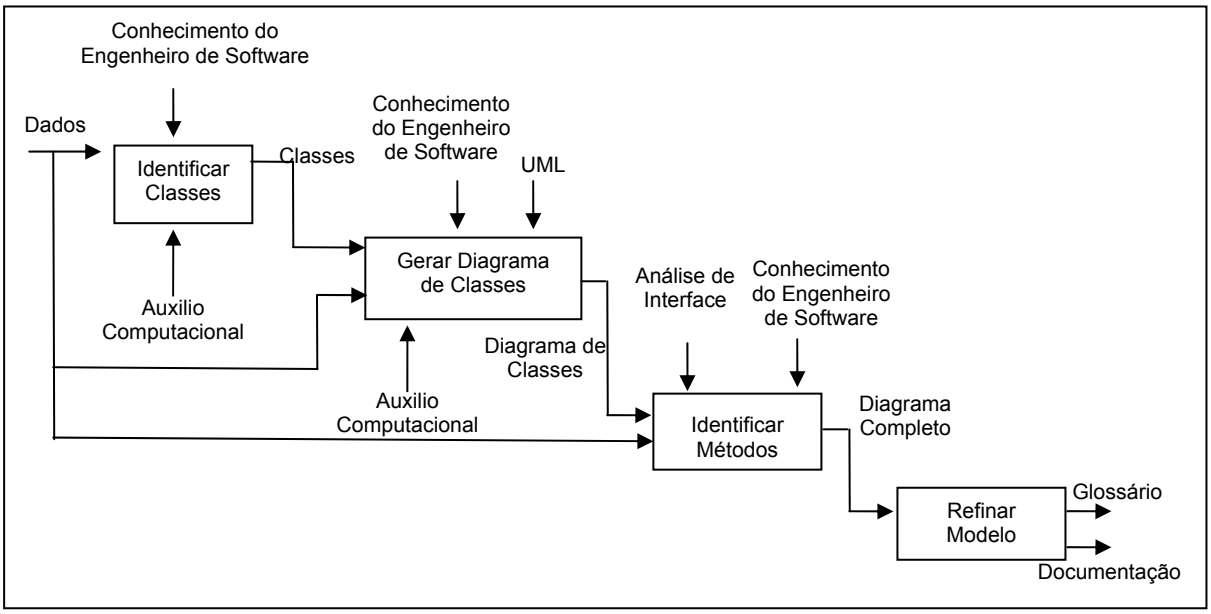


Figura 3.3 – Processo de engenharia reversa dos sistemas escolhidos desenvolvidos em ZIM

Neste trabalho esse passo difere do proposto por Ré (2002), pois trata da engenharia reversa de sistemas desenvolvidos em ZIM a partir do código fonte e das informações disponíveis no dicionário de dados da aplicação, não considerando apenas a análise das interfaces dos sistemas como é feito originalmente. Outra característica importante desse passo é a possibilidade de utilizar a infra-estrutura do banco de dados ZIM para revitalizar as informações pertinentes às aplicações. Dessa forma, dois apoios computacionais foram elaborados em linguagem ZIM, durante este trabalho, para auxiliar o processo de engenharia reversa: `Gera Java` e `Gera SQL`. As duas primeiras atividades, *Identificar Classes* e *Gerar Diagrama de Classes*, do processo de engenharia reversa mostrado na Figura 3.3 são realizadas com esses apoios.

Na primeira atividade, *Identificar Classes*, com o auxilio de `Gera Java`, pode-se identificar as classes dos sistemas legados candidatas a compor o diagrama de classes de um modelo totalmente orientado a objetos. Cada entidade da aplicação ZIM é convertida em uma classe e definida em um único arquivo escrito em linguagem Java (`classes.java`). Esse arquivo contém as definições de todas as classes e seus respectivos atributos, bem como, os métodos construtores de cada classe.

Como todas as classes são geradas em um único arquivo cabe ao engenheiro de software separá-las em arquivos distintos, nomeando-os de acordo com o nome da classe que ele referencia. A partir desses arquivos pode-se usar uma ferramenta CASE que possua o recurso de engenharia reversa, como por exemplo, Rational Rose (Rose, 2003), para auxiliar

na construção de um diagrama de classes usando as classes Java geradas a partir do código legado. A Figura 3.4 ilustra o processo descrito para a atividade *Identificar Classes* (Figura 3.3).

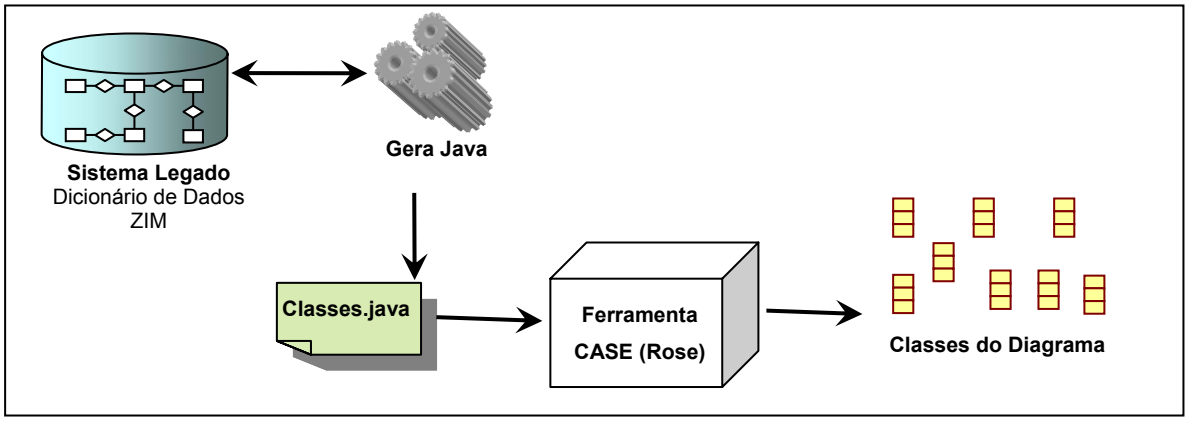


Figura 3.4 – Identificando as classes com o auxilio computacional Gera Java

A segunda atividade do processo de engenharia reversa é *Gerar o Diagrama de Classes* e definir os relacionamentos entre as classes geradas anteriormente. Esses relacionamentos devem ser estabelecidos manualmente, a partir do conhecimento do engenheiro de software sobre o sistema legado usando o código fonte e a documentação obtida por meio do apoio computacional `Gera SQL`.

A `Gera SQL` lê o dicionário de dados da aplicação ZIM, identificando as entidades e os relacionamentos existentes. Dessa forma, é possível identificar quais são as tabelas existentes para gerar o *script* SQL de criação das tabelas, gerando assim o arquivo `Tabelas`. Além desse arquivo principal, também são gerados outros dois arquivos o `fldsdoc` e o `relsdoc`. O arquivo `fldsdoc` (*fields document*) é utilizado para auxiliar a identificação de quais campos são as possíveis chaves primárias das entidades, uma vez que o ZIM não deixa isso explícito na definição do dicionário de dados. O arquivo `relsdoc` (*relationship document*) armazena os relacionamentos explícitos, retirados da entidade `rels`, com o nome e a condição de relacionamento. Com esses arquivos é possível alterar o *script* gerado no arquivo `Tabelas`, criando-se as chaves primárias e estrangeiras. Esse processo é explicado com maiores detalhes em (Pazin; Penteado, 2003).

Da mesma forma que é possível definir as chaves primárias e estrangeiras com o auxílio dos documentos gerados pela `Gera SQL`, é possível também estabelecer os relacionamentos entre as classes candidatas geradas pela `Gera Java`. O arquivo `Tabelas`, depois de editado, é muito útil, pois nele estão contidas as definições das chaves estrangeiras

(*foreign keys*) que representam os relacionamentos entre as tabelas e conseqüentemente as associações entre as classes candidatas. O engenheiro de software pode completar o diagrama de classes, iniciado na atividade anterior a essa, colocando os relacionamentos manualmente. A Figura 3.5 ilustra o processo de criação dos relacionamentos do diagrama de classes.

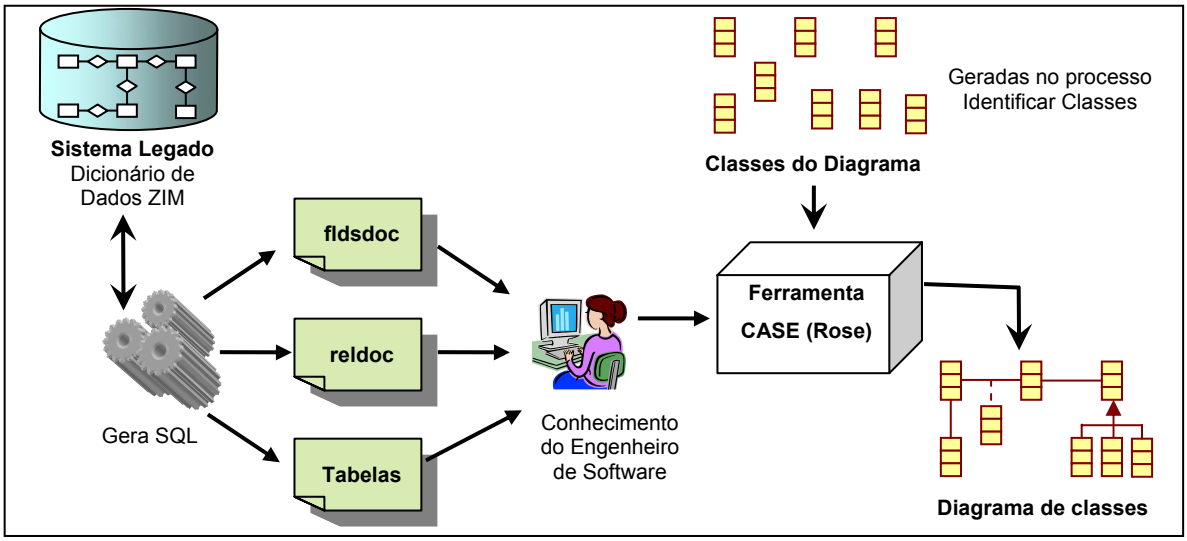


Figura 3.5 – Gerando o Diagrama de Classes

A terceira atividade do processo de engenharia reversa é *Identificar Métodos*. Essa atividade depende muito do conhecimento do engenheiro de software sobre o funcionamento do sistema e é decorrente, principalmente, da análise de interface e do código fonte da aplicação. Não há uma forma específica para realizar tal atividade, mas esse processo pode ser auxiliado por algumas diretrizes propostas na literatura como já foi discutido na seção 2.3.1.

A última atividade do processo de engenharia reversa é *Refinar Modelo* para torná-lo consistente e não ambíguo e ser representado em um nível mais alto de abstração. A elaboração de um glossário dos termos técnicos, bem como a definição dos requisitos, são produtos obtidos também nessa fase. O processo de engenharia reversa apresentado nesta seção e utilizado neste trabalho é o exibido na Figura 3.6. O modelo de classes do sistema de clinicas de fisioterapia (CliFisio) gerado após utilizar esse processo de engenharia reversa é apresentado na Figura 3.7.

Ré (2002) elabora o modelo de classes do sistema com o auxílio de uma linguagem de modelagem orientada a objetos em um passo posterior ao de engenharia reversa. Neste trabalho, a criação do modelo de classes representando o sistema legado é obtida neste passo com o auxílio de `Gera Java` e de `Gera SQL`.

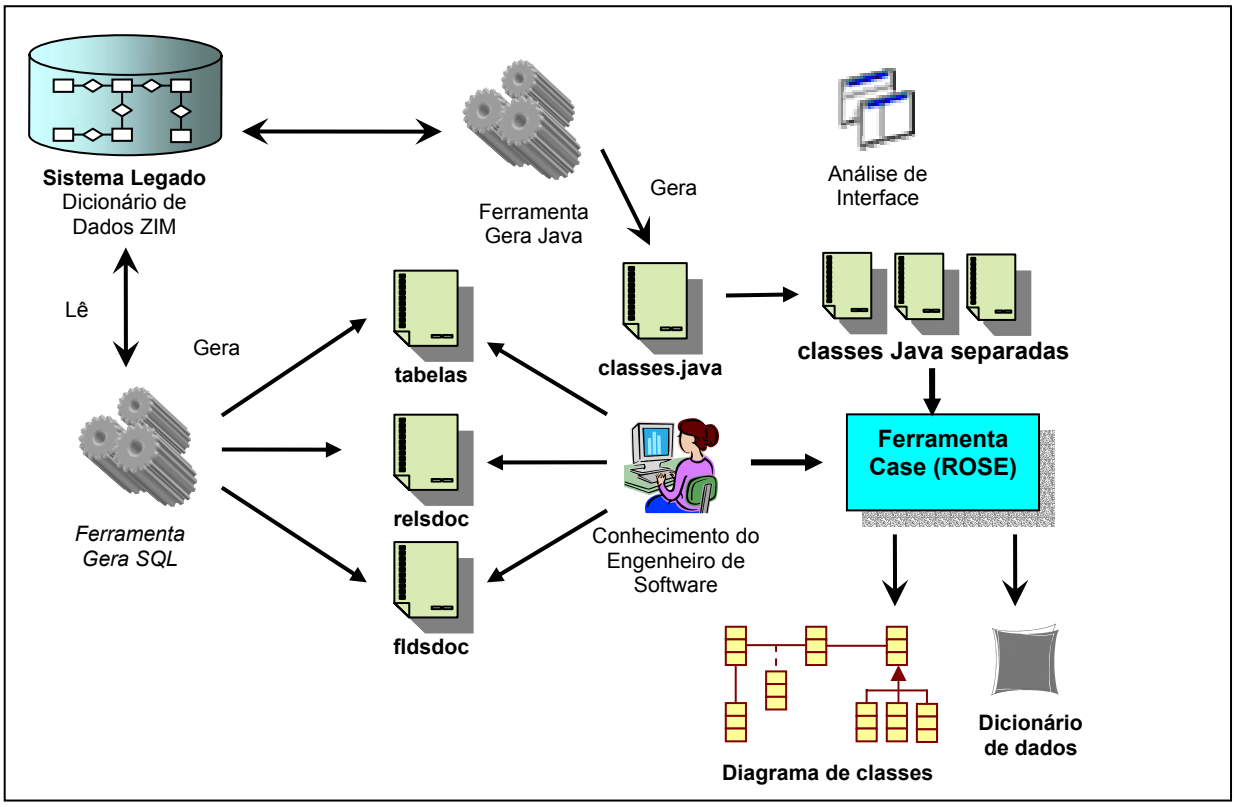


Figura 3.6 – O processos de engenharia reversa

3.3.3. Passo 3: Elaboração do Modelo de Classes do Domínio

Este passo visa a elaboração de um modelo de classes que represente as funcionalidades encontradas nos sistemas escolhidos ou suas principais funções. O engenheiro de software deve se concentrar na análise do conjunto dos sistemas escolhidos para elaborar esse modelo, assim há necessidade de consultar produtos intermediários das atividades anteriores, como por exemplo, a especificação de requisitos ou o dicionário de dados.

Embora os sistemas escolhidos pertençam ao mesmo domínio, pode ser que existam funções similares, funções que diferem de alguns detalhes e outras que são totalmente distintas. Considerando tais características, a elaboração do modelo de classes do domínio depende da compreensão das funcionalidades desses sistemas, para que se determine o grau de similaridade entre eles.

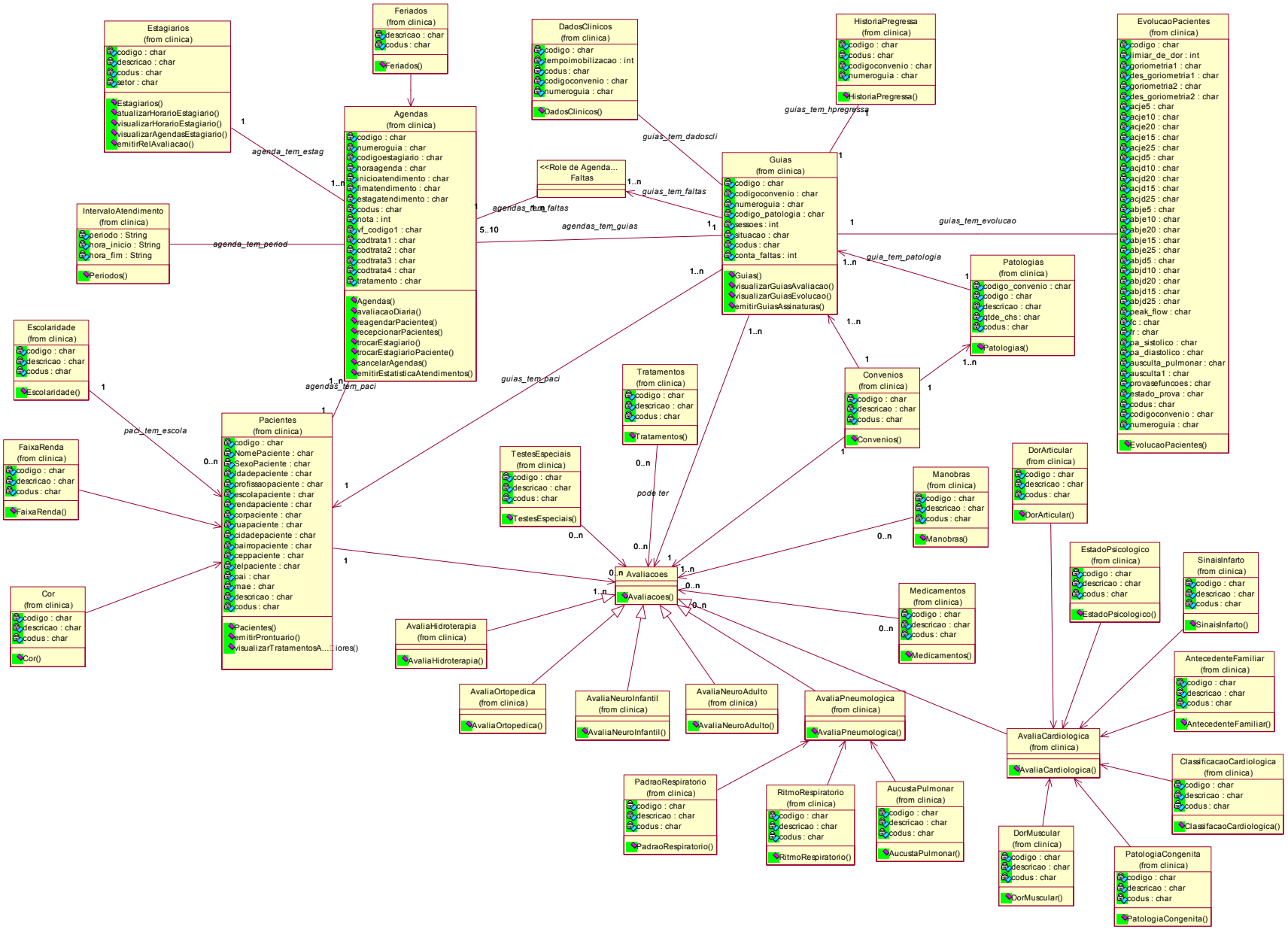


Figura 3.7 – Diagrama de classes do sistema da clínica de fisioterapia, gerado após o processo de engenharia reversa

A análise de similaridades é um conceito aplicado em linha de produtos de software em que se define o que é comum aos membros de uma família (Weiss; Lai, 1999). É uma atividade subjetiva que depende do conhecimento do engenheiro de software sobre o domínio do sistema para as constantes tomadas de decisão quanto a permanência ou não de um elemento no modelo. Um critério que deve ser utilizado para essa decisão é a observância de quanto um determinado elemento enriquecerá ou complementará o domínio. Ré (2002) define cinco sub-passos para auxiliar o processo de criação do modelo de classes do domínio, exibido na Figura 3.8. São eles:

1. *Identificar Classes*: para classificar as classes segundo o grau de similaridade existente entre elas nos sistemas escolhidos, observando-se suas funcionalidades em relação ao domínio. A partir de uma lista de classes dos sistemas, deve-se classificá-las como: similares, parcialmente similares ou distintas.
2. *Tratar as Classes Similares:* aquelas que estão presentes nos três sistemas escolhidos e que formam os elementos básicos do domínio. Nesse passo inicia-se a elaboração do modelo de classes do domínio.
3. *Tratar Classes Parcialmente Similares:* aquelas que estão presentes em dois dos três sistemas escolhidos e devem ser incorporadas ao modelo se acrescentarem funcionalidades importantes para o domínio.
4. *Tratar Classes Distintas:* aquelas que estão presentes em apenas um dos sistemas escolhidos. Devem ser acrescentadas ao modelo se suas funcionalidades acrescentam melhorias ao modelo de domínio
5. *Refinar o Modelo de Classes do Domínio:* para melhorar o modelo e corrigir possíveis erros.

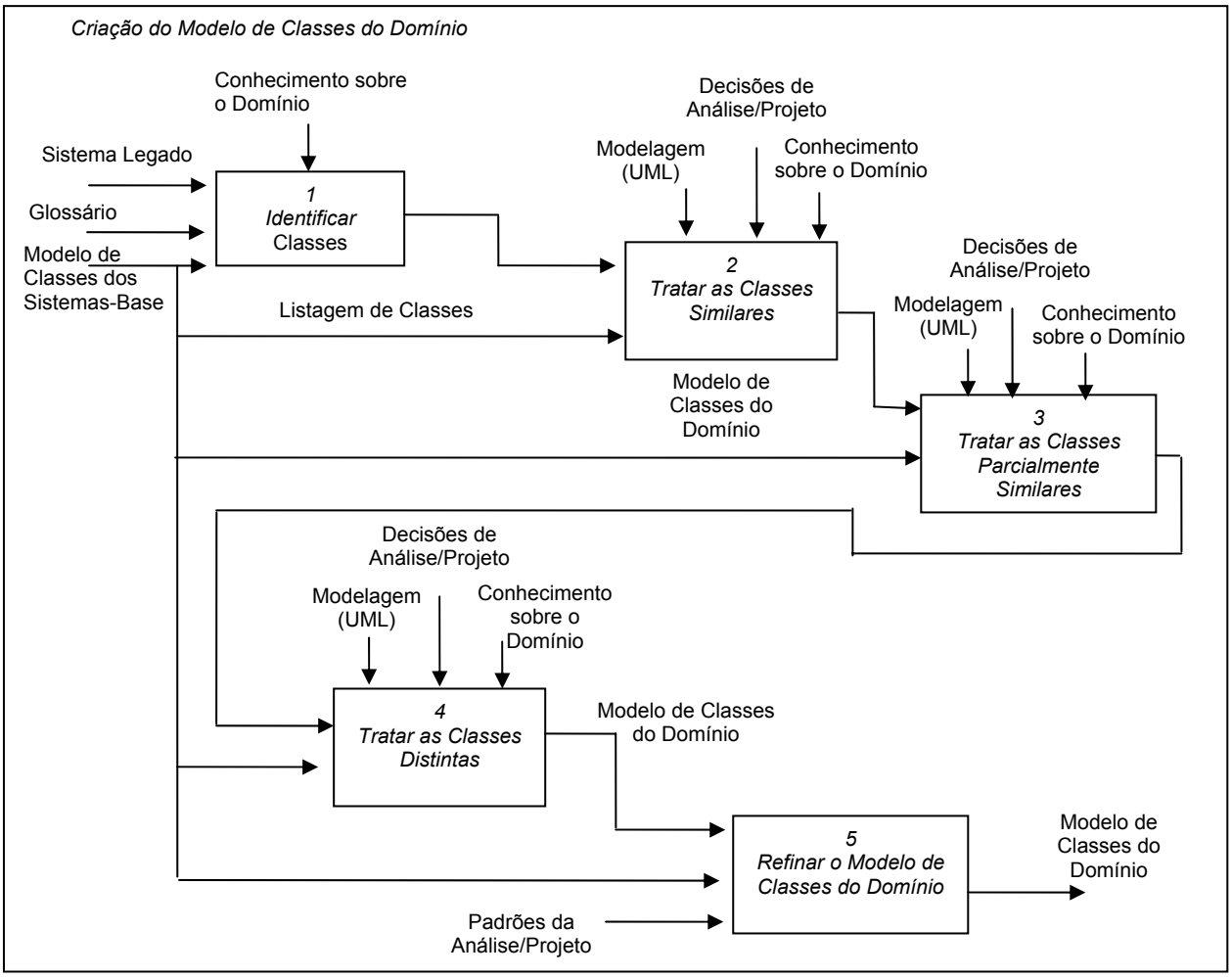


Figura 3.8 – Criação do modelo de classes do domínio(Ré,2002)

A Figura 3.9 mostra o modelo de classes do domínio de Clínica de Reabilitação, elaborado com base no processo apresentado, que busca a similaridade existente entre as classes de cada aplicação. As siglas S, PS e E inseridas no canto inferior de cada classe da Figura 3.9 representam, respectivamente: Similaridades, aquelas classes existentes nos três sistemas; Parcialmente Similares, aquelas classes existentes em dois sistemas; ou as Exclusivas, aquelas classes que existem em uma única aplicação.

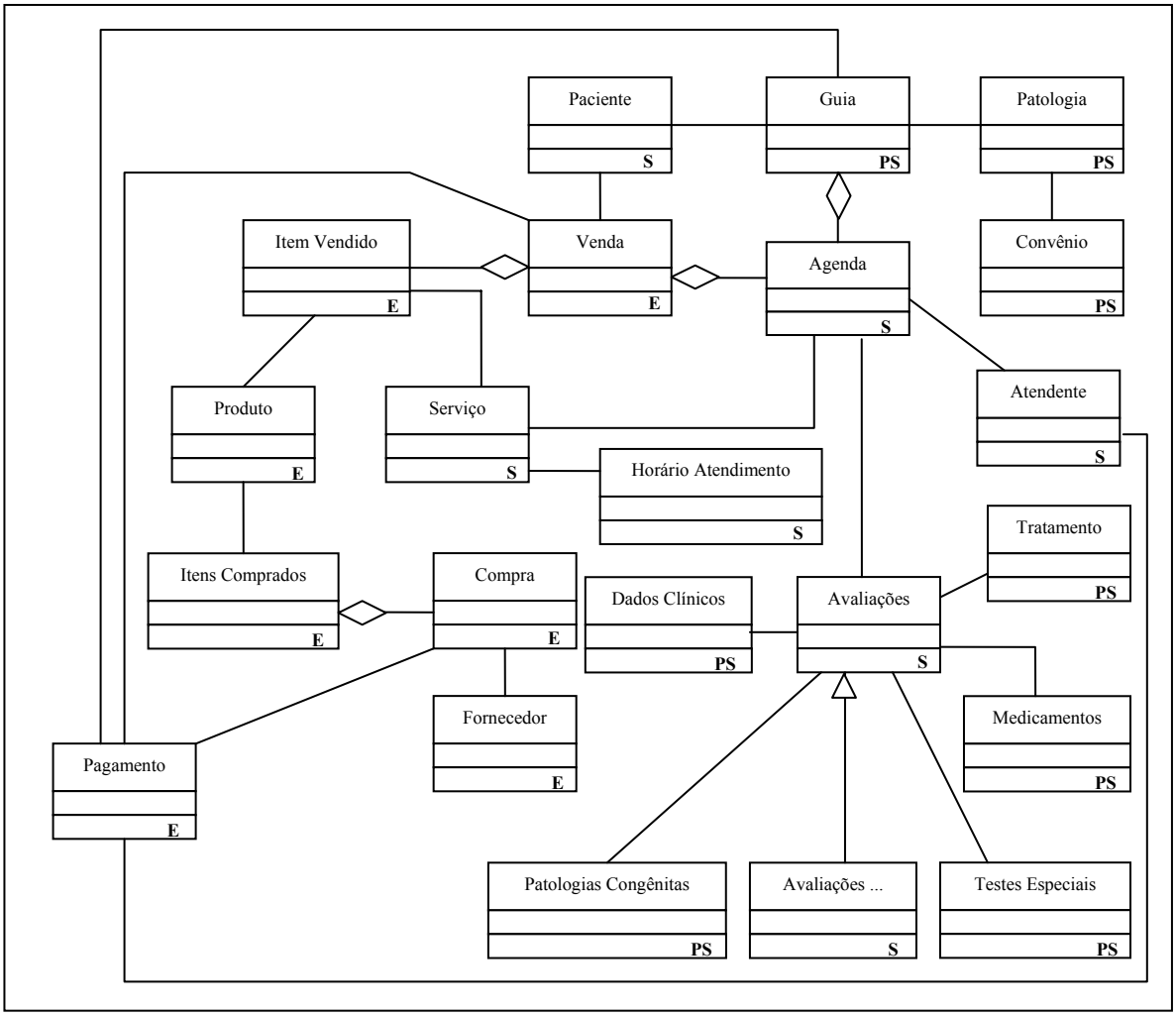


Figura 3.9 – Modelo de classes do domínio para Sistemas de Gestão de Clínicas de Reabilitação (SiGCli)

3.4. Processo de elaboração da linguagem de padrões SiGCli

Braga (2003) propõe um processo para a elaboração de uma linguagem de padrões, mostrado na Figura 3.10. As atividades a serem realizadas para a construção de uma linguagem de padrões são:

- *Análise de Domínio com Engenharia Reversa:* tem por objetivo definir o modelo de classes do domínio alvo, por meio de um processo de engenharia reversa de sistemas já existentes. Neste caso o processo de engenharia reversa produz modelos intermediários, representando cada um dos sistemas, para depois elaborar um modelo geral cobrindo o máximo de funcionalidades do sistema, o modelo de análise do domínio.

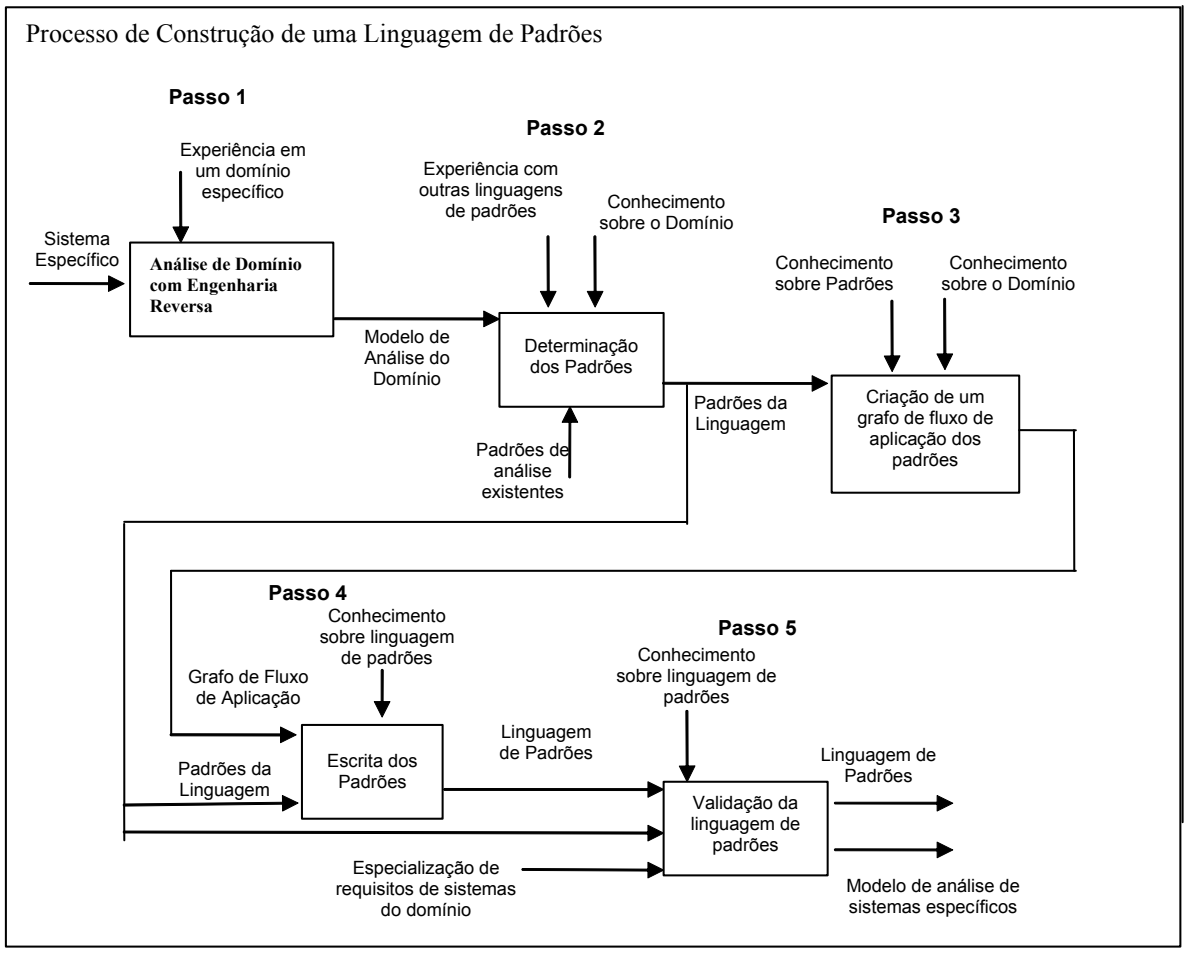


Figura 3.10 – Processo de construção de uma Linguagem de Padrões (Braga,2003)

- *Determinação dos Padrões:* com o modelo de domínio definido, deve-se agrupar a as classes de acordo com as funcionalidades que elas exercem no domínio, iniciando-se pelas funcionalidades mais básicas. Geralmente as classes que estão presentes em todos os sistemas desse domínio representam essas funcionalidades. Esse passo depende muito da experiência e do conhecimento do desenvolvedor sobre linguagem de padrões (Braga, 2003).
- *Criação de um Grafo de Fluxo de Aplicação dos Padrões:* com os padrões definidos no passo anterior, deve-se estudar a forma de interação entre eles, definido a ordem na qual os padrões devem ser aplicados, de forma a facilitar a modelagem dos sistemas usando a linguagem de padrões. Uma das formas de elaborar esse grafo é iniciando-se pelos padrões que representam as funcionalidades mais básicas do domínio e acrescentar, de forma gradativa, os padrões mais específicos.

- *Escrita dos Padrões:* é o passo mais importante, pois o padrão deve ser bem elaborado para que não haja dúvidas quanto à sua interpretação pelos analistas que o utilizarão. Pode-se optar por um dos formatos propostos na literatura, como por exemplo: "nome", "contexto", "problemas", "estrutura", "participantes", "padrões relacionados".
- *Validação da Linguagem de Padrões:* objetiva aplicar os padrões a diferentes sistemas do domínio estudado, comprovando sua eficiência.

Este processo foi usado para a elaboração da linguagem de padrões para **Si**stemas de **G**estão de **Cl**ínicas de Reabilitação (SiGCl). O passo 1, *Análise de Domínio com Engenharia Reversa,* foi realizado neste trabalho e ilustrado pela Figura 3.3, sendo detalhado na seção 3.3. O passo 2, *Determinação dos Padrões*, com base no modelo de domínio gerado no passo 1, as classes foram agrupadas conforme suas funcionalidades. Assim sendo, a Figura 3.11 apresenta o diagrama de classe do modelo de classes do domínio com uma numeração acima de cada classe que identifica a qual padrão essa classe pertence, conforme a legenda

O passo 3, *Criação de um Grafo de Fluxo de Aplicação dos Padrões*, foi elaborado com base nos padrões identificados na Figura 3.11. A definição da ordem de aplicação dos padrões baseou-se na análise das funcionalidades mais básicas do domínio. A Figura 3.12 apresenta o grafo de fluxo de aplicação dos padrões da SiGCli. Os padrões estão agrupados em três grupos, conforme as sua funcionalidades. O Grupo 1 trata das informações básicas relacionadas às clínicas, a identificação dos pacientes e dos serviços prestados. O Grupo 2 cuida do gerenciamento dos atendimentos, sendo possível acompanhar todas as fases do tratamento de um paciente. O Grupo 3 cuida do controle financeiro das clínicas.

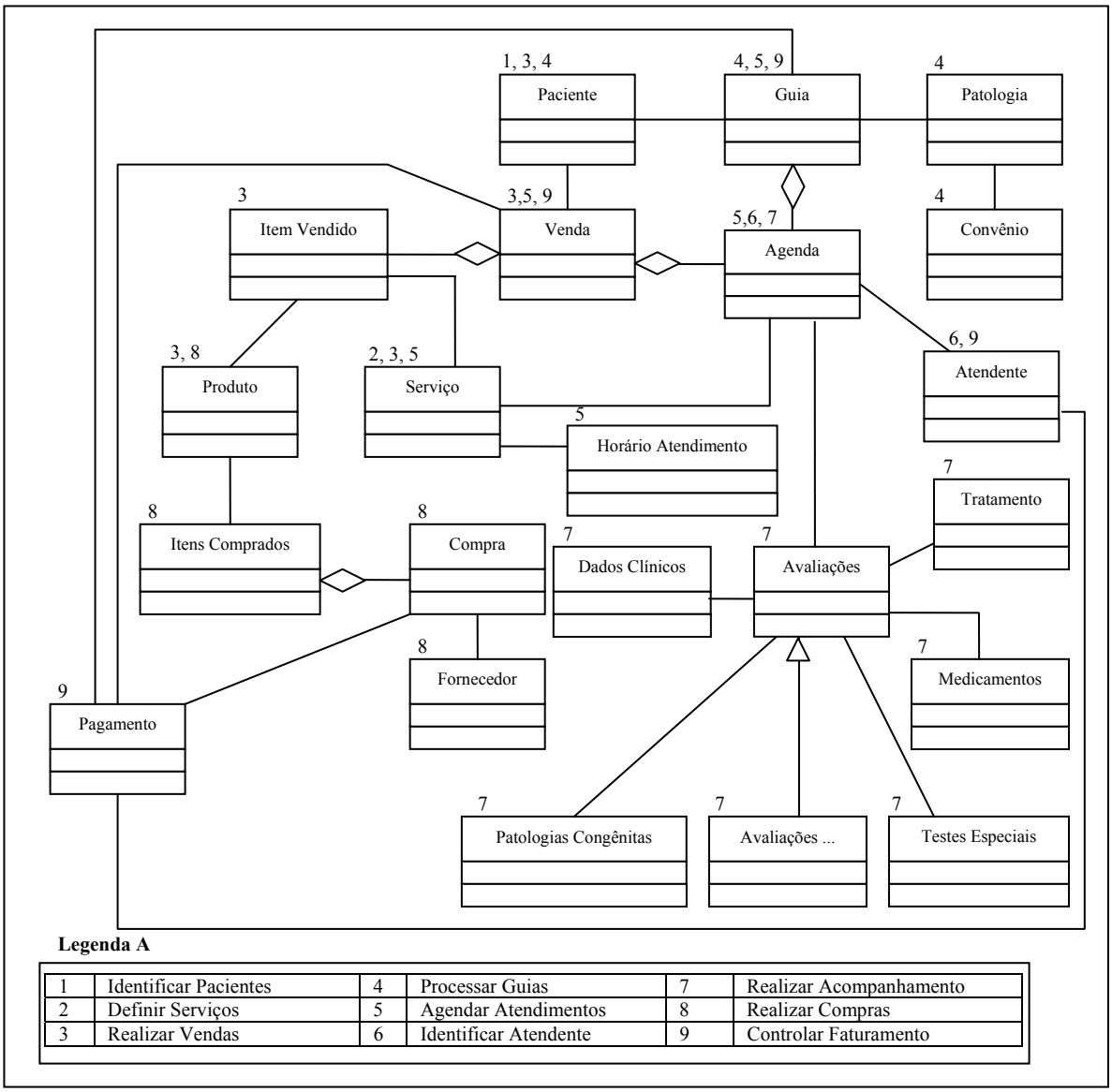


Figura 3.11 – Modelo de classes do domínio SiGCli com os respectivos padrões.

Durante o passo 4, *Escrita dos Padrões*, notou-se que a maioria dos padrões propostos para o domínio de clinicas de reabilitação era semelhante aos existentes na linguagem de padrões para Gestão de Recursos de Negócios (GRN) (Braga et al. 1999). Então, optou-se por aplicar a GRN ao domínio proposto, reusando os seus padrões ou fazendo adaptações, quando necessário. Dessa forma, algumas alterações no modelo de classes do domínio também foram necessárias.

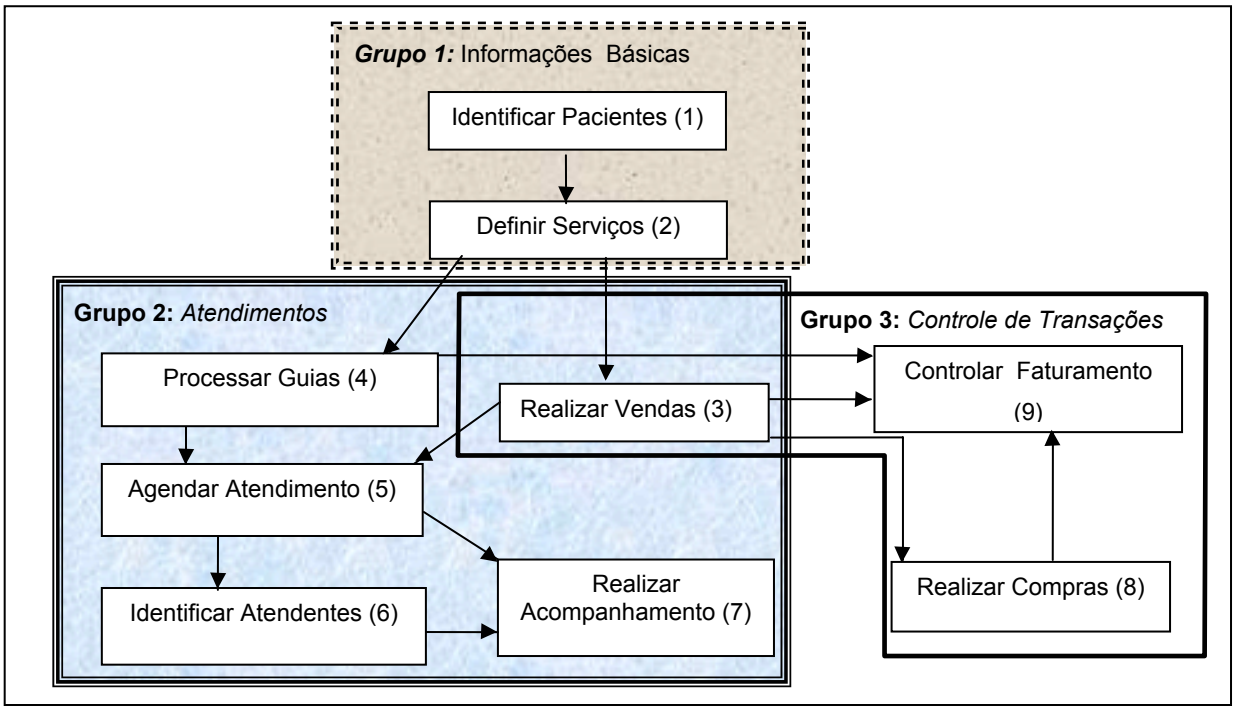


Figura 3.12 – Relacionamento entre os padrões da linguagem de padrões SiGCli

3.5. Aplicando a GRN para definir a SiGCli

Os padrões da SiGCli, como comentando anteriormente tiveram por base os padrões da GRN (Figura 2.2), sendo que em alguns foi mantida a estrutura original. Os diagramas de classes exibem a estrutura dos padrões sendo que os atributos e os métodos das classes, que estão em negrito e itálico, são específicos da SiGCli. Os padrões são apresentados com o seguinte formato: Objetivo; Padrões Usados da GRN; Outros Padrões Relacionados; Estrutura do Padrão; Equivalências Adotadas; Classes Adicionais (não mapeadas na GRN). A coluna **Variante** apresentada nos quadros exibidos em Padrões Usados da GRN refere-se às mutações usadas do padrão da GRN, listado na coluna **Padrão**, para elaborar a SiGCli. A linguagem de padrões SiGCli, completa pode ser encontrada no Apêndice I (Pazin et. al., 2004).

3.5.1. Padrão 1: Identificar Pacientes

Objetivo

Um sistema pertencente ao domínio de clínicas de reabilitação precisa armazenar informações sobre as de pessoas que buscam um certo tipo de tratamento . Essas pessoas

podem ser identificadas como pacientes de uma clínica. Esse padrão possibilita o armazenamento das informações relacionadas aos pacientes.

Padrões Usados da GRN

A Quadro 3.1 apresenta os padrões da GRN usados para a definição do padrão *Identificar Pacientes*.

Quadro 3.1 - Padrões da GRN usados para definir o padrão *Identificar Pacientes.*

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

A Figura 3.13 apresenta o diagrama de classes para o padrão *Identificar Pacientes.*

Equivalências Adotadas

A Quadro 3.2 apresenta as equivalências adotadas entre as classes da SiGCli com as classes da GRN para definir o padrão *Identificar Pacientes.*

Quadro 3.2 - Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão *Identificar Pacientes*.

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.

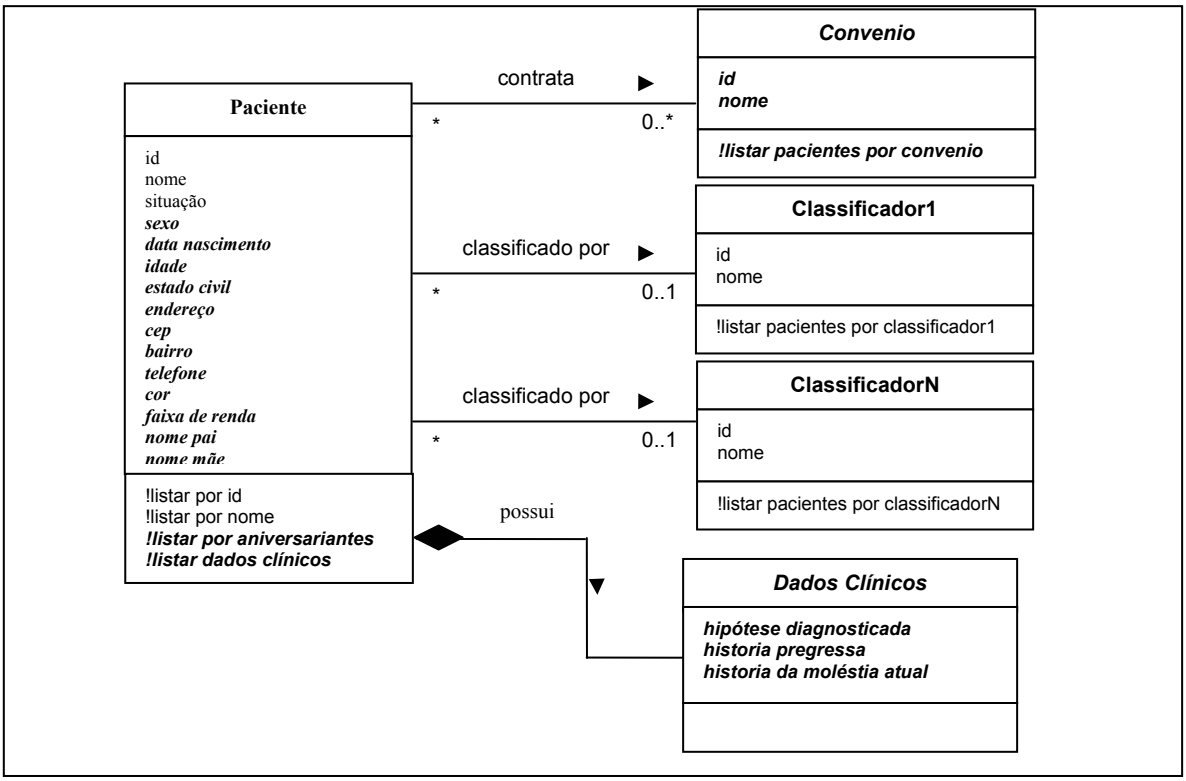


Figura 3.13 – Diagrama de classes para o padrão *Identificar Pacientes*

3.5.2. Padrão 2: Definir Serviços

Objetivo

Um paciente procura uma clínica para realizar um determinado tipo de tratamento. Esse tratamento, geralmente, é um serviço prestado pela clínica e para isso deve ser identificado. Esse padrão permite armazenar as informações sobre os tipos de tratamentos realizados pela clínica.

Padrões Usados da GRN

A Quadro 3.3 apresenta os padrões da GRN usados para a definição do padrão *Definir Serviços*.

Quadro 3.3 - Padrões da GRN usados para definir o padrão *Definir Serviços.*

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

A Figura 3.14 apresenta o diagrama de classes para o padrão *Definir Serviços*

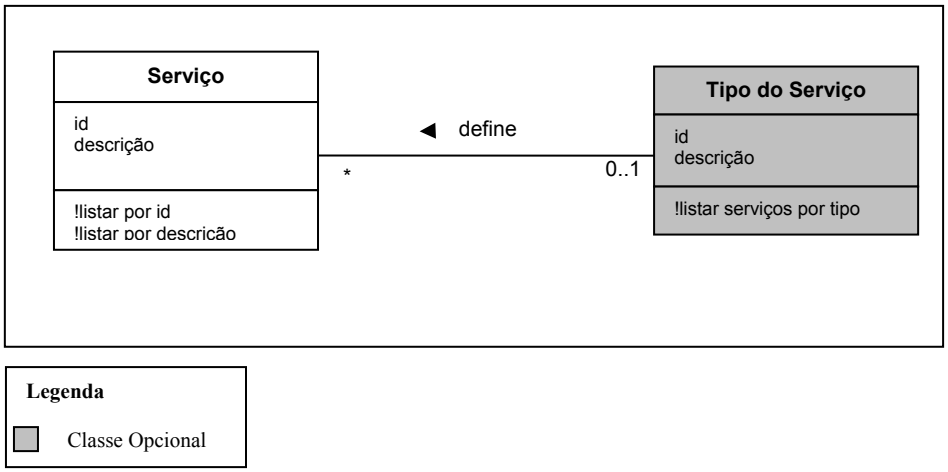


Figura 3.14 – Diagrama de classes para o padrão *Definir Serviços*.

Equivalências Adotadas

A Quadro 3.4 apresenta as equivalências adotadas entre as classes da SiGCli com as classes da GRN para definir o padrão *Definir Serviços.*

Quadro 3.4 - Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão *Definir Serviços*.

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.

3.5.3. Padrão 3: Realizar Vendas

Objetivo

Uma clínica pode comercializar serviços e produtos. Os serviços são atividades realizadas com os pacientes através de um agendamento prévio. Um produto é qualquer bem que sofre uma troca de propriedade, ou seja, pertencia à clínica e passa a pertencer ao paciente que o adquiriu. Tais produtos devem ser identificados e previamente cadastrados no sistema, para que seja possível fazer seu controle. Esse padrão permite que os dados referentes às transações comerciais sejam armazenados por uma clínica possibilitando o seu gerenciamento.

Padrões Usados da GRN

A Quadro 3.5 apresenta os padrões da GRN usados para a definição do padrão *Realizar Venda*.

Quadro 3.5 - Padrões da GRN usados para definir o padrão *Realizar Venda.*

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

A Figura 3.15 apresenta o diagrama de classes para o padrão *Realizar Vendas.*

Equivalências Adotadas

A Quadro 3.6 apresenta as equivalências adotadas entre as classes da SiGCli com as classes da GRN para definir o padrão *Realizar Vendas.*

Quadro 3.6 - 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.

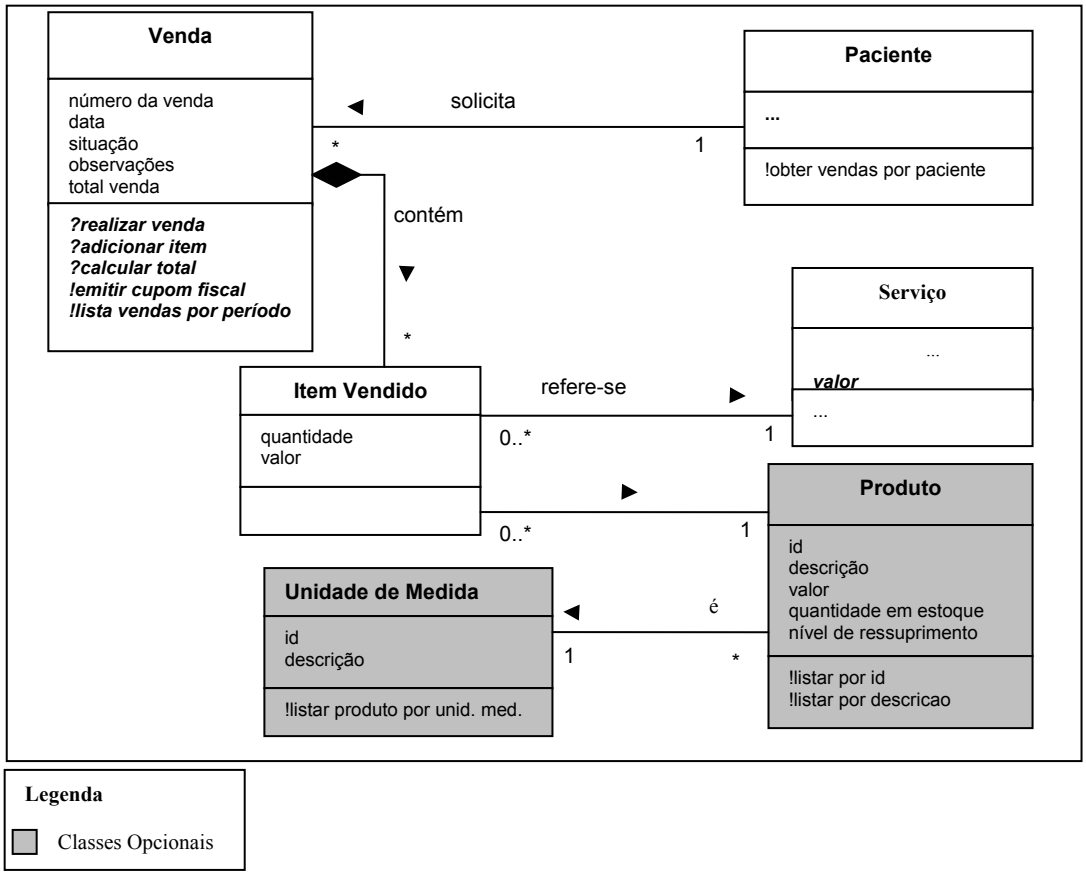


Figura 3.15 – Diagrama de classes para o padrão *Realizar Vendas*.

3.5.4. Padrão 4: Processar Guias

Objetivo

Uma clínica pode realizar atendimentos aos pacientes por meio de guias de convênios encaminhadas por médicos. Essas guias possibilitam o atendimento e agendamento dos pacientes na clínica. Esse padrão possibilita o gerenciamento de guias de empresas conveniadas à clínica.

Padrões Usados da GRN

A Quadro 3.7 apresenta o padrão da GRN usados para a definição do padrão *Processar Guias*.

Quadro 3.7 - Padrão da GRN usados para definir o padrão *Processar Guias.*

Nº	***Padrão***	***Variante***
9	Manter Recurso	Default

Estrutura do Padrão

A Figura 3.16 apresenta o diagrama de classes para o padrão *Processar Guias.*

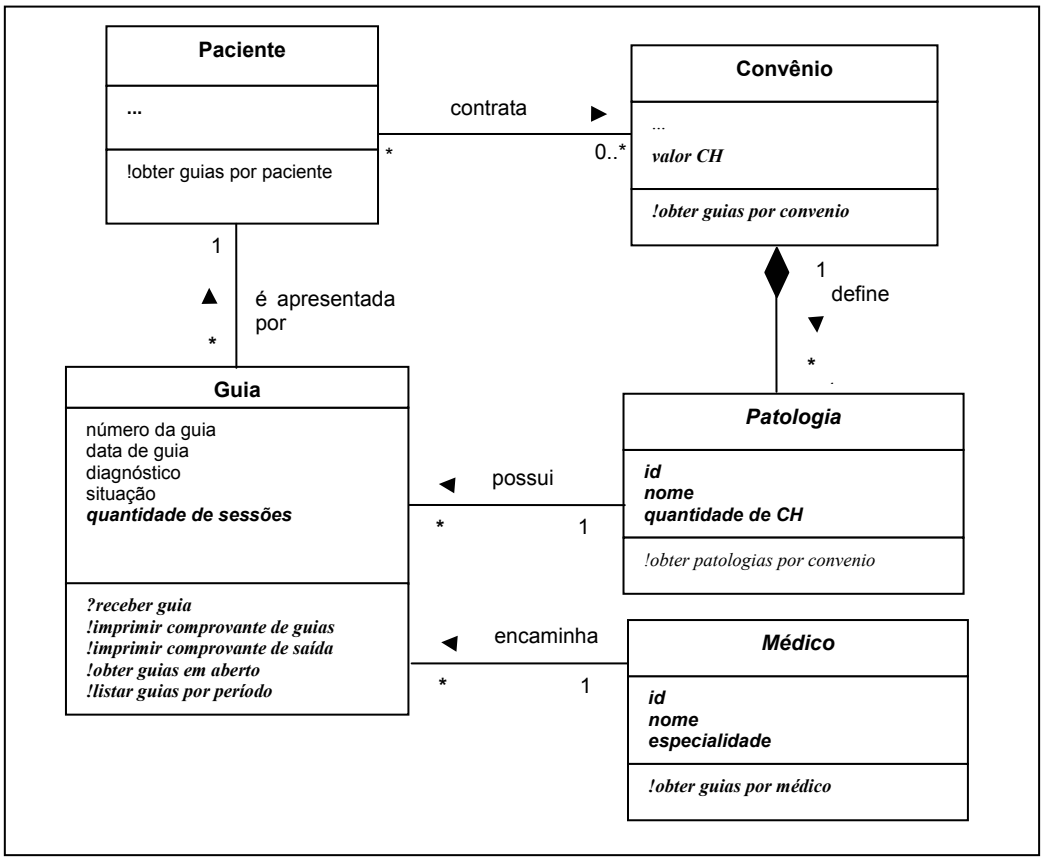


Figura 3.16 – Diagrama de classes para o padrão *Processar Guias*

Equivalências Adotadas

A Quadro 3.8 apresenta as equivalências adotadas entre as classes da SiGCli com as classes da GRN para definir o padrão *Definir Serviços.*

Quadro 3.8 - Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão *Processar Guias*.

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

Classes adicionais (não mapeadas na GRN)

Patologia, Médico

3.5.5. Padrão 5: Agendar Atendimentos

Objetivo

O agendamento de sessões para o tratamento de pacientes é uma atividade imprescindível para uma clínica, pois através dele é possível controlar os horários de atendimentos. Um agendamento pode ser resultado de uma venda de serviço ou a apresentação de uma guia pelo paciente. Um paciente só pode ser atendido pela clínica quando tiver as sessões agendadas. Esse padrão permite gerenciar e acompanhar o agendamento de sessões de um paciente.

Padrões Usados da GRN

A Quadro 3.9 apresenta o padrão da GRN usados para a definição do padrão *Agendar Atendimentos*.

Quadro 3.9 - Padrão da GRN usados para definir o padrão *Agendar Atendimento.*.

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

Estrutura do Padrão

A Figura 3.17 apresenta o diagrama de classes para o padrão *Agendar Atendimento.*

Equivalências Adotadas

A Quadro 3.10 apresenta as equivalências adotadas entre as classes da SiGCli com as classes da GRN para definir o padrão *Agendar Atendimento.*.

Quadro 3.10 - Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão *Agendar Atendimento*.

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.

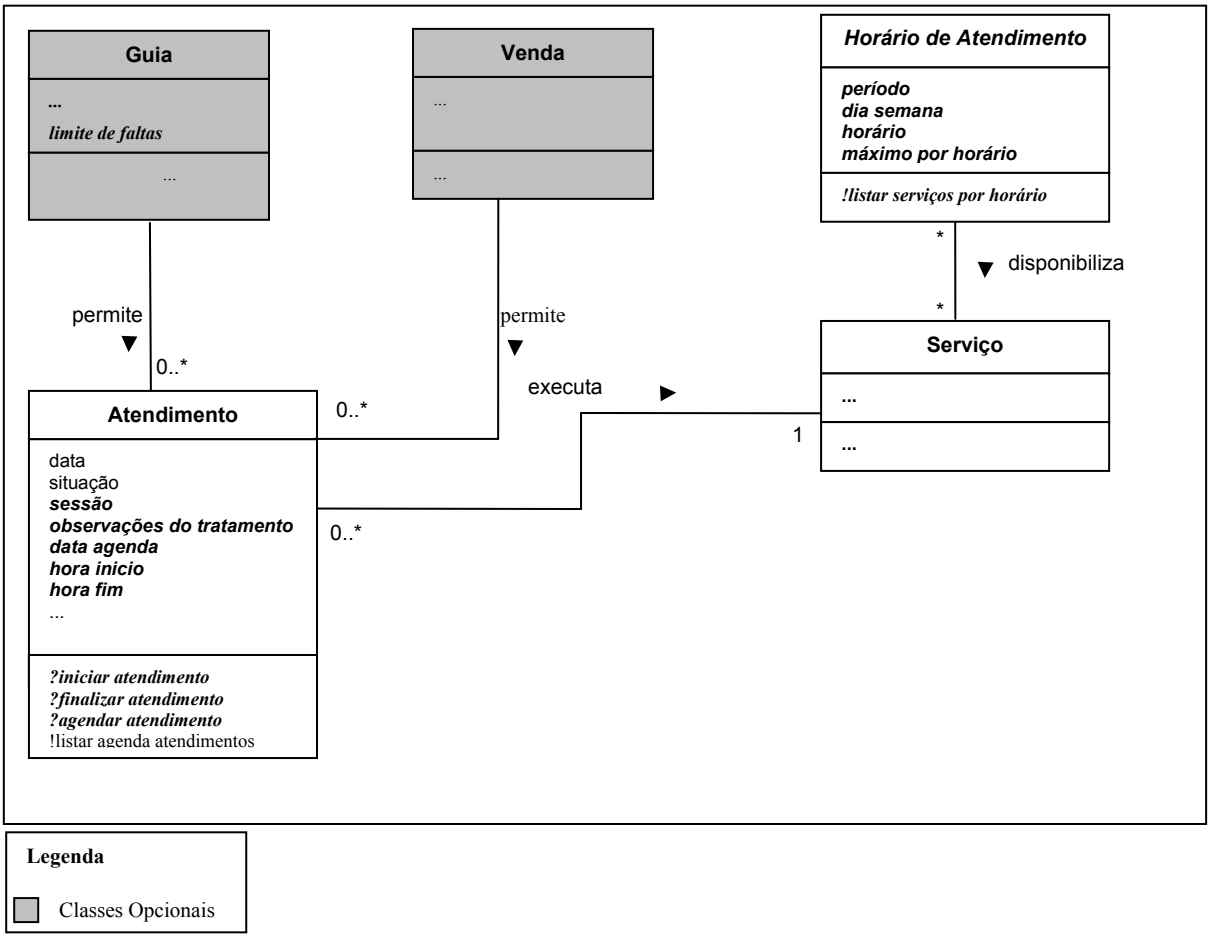


Figura 3.17 – Diagrama de classes para o padrão *Agendar Atendimentos*

3.5.6. Padrão 6: Identificar Atendentes

Objetivo

Uma clínica deve possuir profissionais capacitados para realizar os atendimentos aos pacientes. Tais profissionais, definidos como atendentes, podem ser: fisioterapeutas, terapeutas ocupacionais, professores de educação física, estagiários entre outros. Esse padrão permite identificar quem são esses atendentes.

Padrões Usados da GRN

A Quadro 3.11 apresenta o padrão da GRN usados para a definição do padrão *Identificar Atendentes*.

Quadro 3.11 - Padrões da GRN usados para definir o padrão *Identificar Atendentes.*

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

Estrutura do Padrão

A Figura 3.18 apresenta o diagrama de classes para o padrão *Identificar Atendente.*

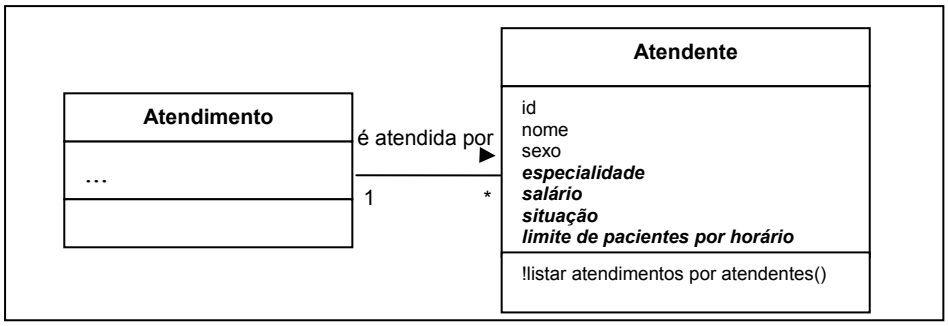


Figura 3.18 – Diagrama de classes para o padrão *Identificar Atendentes*.

Equivalências Adotadas

A Quadro 3.12 apresenta as equivalências adotadas entre as classes da SiGCli com as classes da GRN para definir o padrão *Identificar Atendentes.*

Quadro 3.12 - Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão *Identificar Atendentes*.

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.

3.5.7. Padrão 7: Realizar Acompanhamento

Objetivo

Para que se possa obter um histórico da evolução dos pacientes deve-se armazenar dados sobre os tratamentos realizados com os pacientes, bem como descrever as suas evoluções obtidas. Com esses dados os atendentes podem avaliar qual o melhor tratamento para uma determinada lesão. Esse padrão permite armazenar e gerenciar tais informações.

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 inúmeras avaliações, tais como: neurológica, cardiológica, ortopédica, nutricional, anamnese, dentre outras. Essas avaliações específicas devem ser classes filhas (especializações) da classe avaliação. A Figura 3.19 apresenta o diagrama de classes para o padrão *Realizar Acompanhamento.*

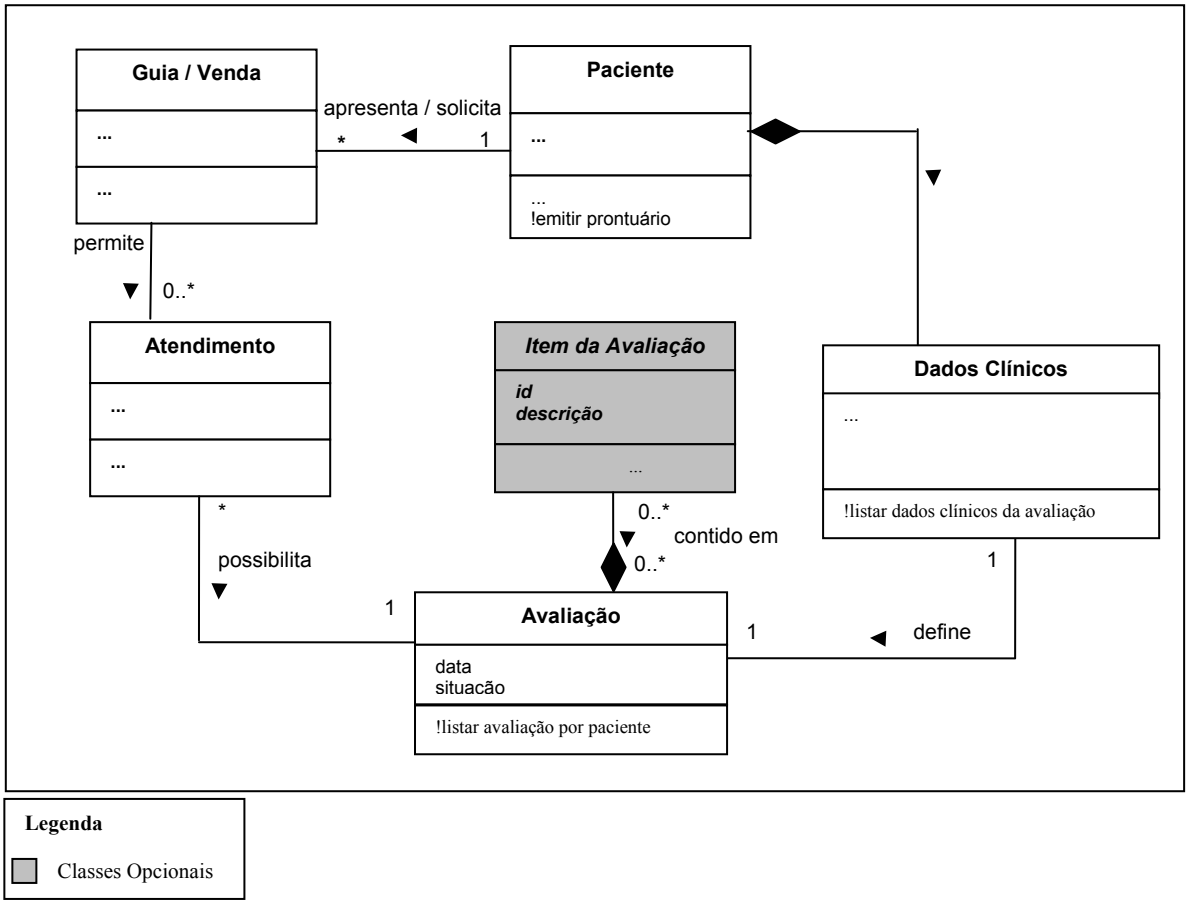


Figura 3.19 – Diagrama de classes para o padrão *Realizar Acompanhamento*

Equivalências Adotadas

Não há equivalências entre as classes.

Classes adicionais (não mapeadas na GRN)

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

3.5.8. Padrão 8: Realizar Compras

Objetivo

Uma clínica pode comercializar produtos e, nesse caso, torna-se necessário controlar a compra desses por meio de um pedido a um fornecedor responsável pela sua entrega. Esse padrão permite armazenar dados referentes a compra de produtos que serão comercializados pela clínica.

Padrões Usados da GRN

A Quadro 3.13 apresenta os padrões da GRN usados para a definição do padrão *Realizar Compras*.

Quadro 3.13 - Padrões da GRN usados para definir o padrão *Realizar Compras.*

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

Estrutura do Padrão

A Figura 3.20 apresenta o diagrama de classes para o padrão *Realizar Compras.*

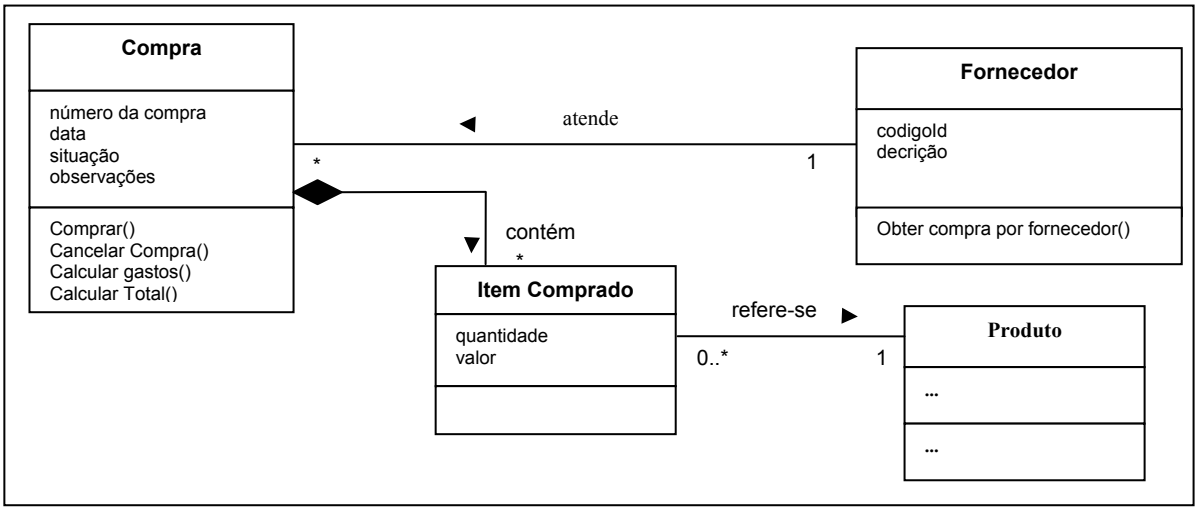


Figura 3.20 – Diagrama de classes para o padrão *Realizar Compras*.

Equivalências Adotadas

A Quadro 3.14 apresenta as equivalências adotadas entre as classes da SiGCli com as classes da GRN para definir o padrão *Realizar Compras.*

Quadro 3.14 - Equivalências adotadas entre as classes da SiGCli e da GRN para definir o padrão *Realizar Compras*.

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.

3.5.9. Padrão 9: Controlar Faturamento

Objetivo

Uma clínica gerencia os serviços e produtos comercializados por ela. Quando essa comercialização destina-se a um paciente, é gerado um pagamento do tipo "*fatura*". Quando a comercialização é referente à compra de produtos é gerado um pagamento do tipo "*despesa*". Esse padrão armazena dados referentes a pagamentos e recebimentos de uma clínica possibilitando analisar o controlar o seu faturamento.

Padrões Usados da GRN

A Quadro 3.15 apresenta o padrão da GRN usados para a definição do padrão *Controlar Faturamento*.

Quadro 3.15 - Padrão da GRN usados para definir o padrão *Controlar Faturamento.*

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

Estrutura do Padrão

A Figura 3.21 apresenta o diagrama de classes para o padrão *Controlar Faturamento.*

Equivalências Adotadas

A Quadro 3.16 apresenta as equivalências adotadas entre as classes da SiGCli com as classes da GRN para definir o padrão *Controlar Faturamento.*

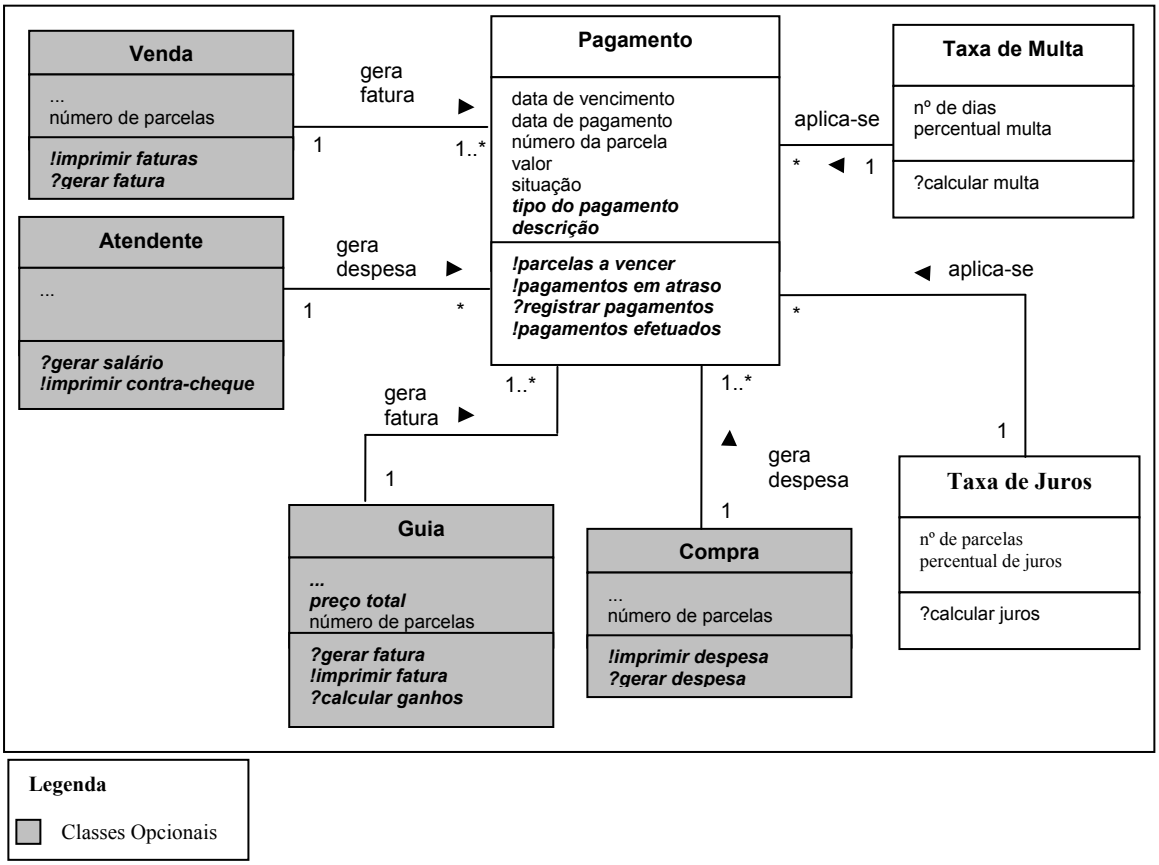


Figura 3.21 – Diagrama de classes para o padrão *Controlar Faturamento*

Quadro 3.16 - 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.

3.6. Considerações Finais

Este capítulo apresentou o processo de construção da linguagem de padrões, denominada SiGCli, elaborada por meio da análise de domínio, para identificar as principais

funcionalidades do domínio de gestão de clínicas de reabilitação, gerando-se assim um modelo de classes do domínio.

A definição do domínio é realizada por um processo de engenharia reversa de sistemas legados desenvolvidos em ZIM, utilizando-se dois apoios computacionais desenvolvidos, neste trabalho, especificamente para este fim, `Gera Java` e `Gera SQL`. Por meio deles são recuperadas as informações das aplicações ZIM possibilitando a reconstrução desses sistemas de uma maneira mais apropriada.

O processo de construção da linguagem de padrões proposto por Braga (2003), foi seguido para a definição da linguagem de padrões SiGCli, após a definição do modelo de classes do domínio. Durante utilização do processo observou-se que alguns dos padrões da SiGCli possuíam características similares às dos padrões da GRN (Braga et al., 1999). Dessa forma, procurou-se, sempre que possível, reusar esses padrões na elaboração dos padrões da SiGCli. Os padrões da GRN não cobrem totalmente algumas das principais funcionalidades do domínio das clínicas de reabilitação, como por exemplo, o acompanhamento detalhado do tratamento dos pacientes. Entretanto, eles auxiliaram na definição dos padrões cuja funcionalidade tratava especificamente da transação de negócios, como por exemplo, a realização de vendas de produtos e serviços.

Para suprir as funcionalidades específicas da SiGCli, foram realizadas algumas adaptações aos padrões da GRN, criando-se atributos, operações, associações entre classes, novas classes e até mesmo um novo padrão (padrão 7 - *Realizar Acompanhamento*) que modela o acompanhamento de um tratamento. Essas características permitem que a SiGCli represente de forma mais abrangente um sub-domínio restrito da GRN.

O *framework* GREN também foi analisado como uma possível solução para a geração de sistemas para o domínio desejado, entretanto, alguns fatores influenciaram para a sua não utilização (Pazin et al., 2004a):

a) Esse framework foi desenvolvido em linguagem *SmallTalk*, que embora seja bem difundida no meio cientifico, tem pouca representatividade no meio comercial;
b) As aplicações instanciadas pelo *framework* usam a arquitetura Cliente / Servidor e não são disponibilizadas na Web;
c) O *framework* faz uso da linguagem de padrões GRN para a instanciação de aplicações, sendo necessário definir quais papéis serão assumidos pelas classes da SiGCli. No caso específico do domínio de gestão de clínicas de reabilitação, algumas classes assumem papéis distintos durante a instanciação da aplicação

que não são cobertos pelo *framework* GREN. Isso dificulta o entendimento do desenvolvedor para instanciar a aplicação;

d) As funcionalidades específicas do domínio de gestão de clínicas de reabilitação não são previstas pelo *framework* exigindo do desenvolvedor conhecimento profundo de seu funcionamento e da linguagem *SmallTalk* para implementá-los.

Um **G**erador de **A**plicações baseadas na **W**eb para **C**línicas de Reabilitação (GAwCRe) com base na SiGCli para a sua definição será apresentado no capítulo 4.

CAPÍTULO 4

GAwCRe: Gerador de Aplicações baseadas na Web para Clínicas de Reabilitação

4.1. Considerações Iniciais

Geradores de aplicações são ferramentas capazes de automatizar parte do processo de desenvolvimento de software, convertendo especificações de alto nível em artefatos de software. Dessa forma, podem ser considerados como compiladores para uma linguagem de um domínio específico (Smaragdakis; Batory, 1998). Uma linguagem de domínio por ser representada por meio de uma linguagem de padrões ou usando uma Linguagem de Modelagem da Aplicação (LMA, originalmente AML – *Application Modeling Language*). As especificações escritas em LMA devem representar modelos, isto é, devem ser abstrações das aplicações que garantam certas propriedades importantes para o domínio (Weiss; Lai, 1999). O gerador de aplicações criado dessa forma deve analisar as especificações escritas nessa LMA para, a partir delas, gerar o produto final.

Para a construção de um gerador de aplicação deve-se obter a estrutura comum e variável do domínio alvo. O modelo do domínio pode auxiliar na fase de análise das similaridades desses sistemas, mas geralmente o estudo de suas variabilidades não pode ser feito de forma sistemática. As variações podem ser entendidas por meio do conhecimento do

desenvolvedor sobre o domínio desejado. Torna-se necessário entender as similaridades e as variabilidades para que se possa projetar uma ferramenta de geração poderosa, que cubra o máximo possível do domínio. A necessidade de se modelar as similaridades e variabilidades do sistema surge naturalmente no decorrer da evolução do processo de desenvolvimento, e torna-se muito importante no reuso (Jarzabeck, 1995).

Este capítulo apresenta o **G**erador de **A**plicações baseadas na **W**eb para **C**línicas de **Re**abilitação, chamado GAwCRe, para o domínio de clínicas de reabilitação que foi desenvolvido com base na linguagem de padrões SiGCli (Pazin et. al., 2004). A seção 4.2 apresenta as diretrizes usadas para a criar a LMA. As definições das *tags*, em XML, usadas para especificar a LMA e a linguagem de padrões são apresentadas na seção 4.3. A seção 4.4 explica o processo de definição dos gabaritos de código usados na elaboração dos artefatos da aplicação. A arquitetura do gerador é apresentada na seção 4.5. A seção 4.6 apresenta a arquitetura das aplicações geradas pelo gerador e as considerações finais são feitas na seção 4.7.

4.2. Especificação da Linguagem de Modelagem da Aplicação

A linguagem de padrões SiGCli define um conjunto de padrões para o domínio de clínicas de reabilitação visando identificar as principais funcionalidades 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.

A SiGCli foi usada para remodelar os sistemas que serviram como base no processo de análise de domínio, com o intuito de validar a linguagem de padrões e de analisar as similaridades e variabilidades existentes entre eles. A partir dessa análise foi possível definir uma LMA que represente as possíveis variações de clínicas usando a linguagem de padrões SiGCli. Essas variações devem ser definidas na LMA para que uma aplicação possa ser gerada corretamente, conforme os seus requisitos. Para a elaboração da LMA foram usadas a seguintes diretrizes:

1) **padrões aplicados sem variação** (similaridade): para a LMA utiliza-se somente o nome do padrão. Por exemplo: um paciente é identificado na clínica de fisioterapia da mesma forma que nas clínicas de terapia ocupacional e educação física. Assim,

o padrão (1) da SiGCli, *Identificar Paciente*, é modelado nas três clínicas sem variação. A definição da LMA para esse padrão é somente *Identificar Pacientes.*

2) **padrões aplicados com variação** (variabilidade): Duas características devem ser analisadas:
    2.1) **A variação é controlada pelo desenvolvedor**: nesse caso identifica-se a variação atribuindo-lhe um nome que será usado na LMA. Cabe ao desenvolvedor optar ou não pelo uso de tal variação, podendo ela ser exclusiva (única variabilidade no padrão) ou múltipla (pode-se usar mais que uma variabilidade para o mesmo padrão). Por exemplo: o padrão (3), *Realizar Vendas*, pode tratar de dois tipos de vendas: apenas serviços ou serviços e produtos, cabendo ao desenvolvedor analisar em qual dessas duas situações a clínica se encontra. A definição da LMA para a esse padrão é *Realizar Vendas* (que realiza apenas a venda de serviços), com a variação opcional *Com Produto* (que inclui na transação a venda de produtos).
    2.2) **A variação é controlada pela instanciação**: nesse caso, ela é resultante da aplicação de outros padrões e os diferentes tipos de instanciações de aplicações devem ser gerenciados a partir dos padrões escolhidos anteriormente pelo desenvolvedor. Por exemplo: o padrão (5), *Agendar Atendimentos*, permite agendar sessões de atendimentos para um paciente resultante de uma venda de serviço ou da apresentação de uma guia de convênio médico. Essa variação depende dos padrões aplicados anteriormente.

O Quadro 4.1 apresenta os padrões da SiGCli e as variabilidades (Nome e Descrição) identificadas quando as diretrizes apresentadas anteriormente para a elaboração da LMA foram aplicadas. Quando não existir variabilidade a coluna **Nome** apresenta a sentença "*Não possui*" e a coluna **Descrição** tem um traço diagonal. Quando a variabilidade não for controlada pelo desenvolvedor a coluna **Nome** apresenta a sentença "*Não possui*" e a coluna **Descrição** apresenta a variabilidade que é controlada pelo gerador.

A Figura 4.1 exibe a gramática definida para a LMA de acordo com o apresentado no Quadro 4.1. Para instanciar uma aplicação basta optar pelas regras definidas na gramática, considerando as regras de produção determinadas.

```
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> ::= <Com_Avaliações_Fisio> | <Com_Avaliações_TO>
                 |<Com_Avaliações_EF>
<Com_Avaliações_Fisio> ::= <Tipo_Avaliação_Fisio> ;
                           <Com_Avaliações_Fisio>
<Tipo_Avaliação_Fisio> ::= Neurológica_Infantil | Neurológica_Adulto
                          |Ortopedica | Pneumologica | Cardiológica |
                          Diária |  λ
<Com_Avaliações_TO> ::= <Tipo_Avaliação_TO> ; <Com_Avaliações_TO>
<Tipo_Avaliação_TO> ::= Anamnese_Infantil | Anamnese_Adulto|
                        Neurológica_Infantil| Psicomotora | TDE |
                        Física_Infantil | Física_Adulto | Portage |
                        Diária  | λ
<Com_Avaliações_EF> ::= <Tipo_Avaliação_EF> ; <Com_Avaliações_EF>
<Tipo_Avaliação_EF> ::= Anamnese | Antropométrica | Física |
                        Controle_Diário_Atividades | Ficha_Musculação | λ
```

Figura 4.1 – Regras da gramática definida para a LMA baseada na SiGCli

Quadro 4.1 - Variabilidades identificadas entre os padrões da SiGCli

Nº	Padrão	Nome Variabilidade	
		Nome	**Descrição**
1	Identificar Pacientes	*Não possui*	
2	Definir Serviços	Com Tipo Serviço	1. Uma clínica não precisa especificar os serviços prestados de forma detalhada
3	Realizar Vendas	Com Produto	1. Uma clínica realiza a comercialização de serviços, mas nem sempre comercializa

			produtos
4	Processar Guias	*Não possui*	1. Caso o padrão Controlar Faturamento tenha sido aplicado, deve-se acrescentar alguns atributos e métodos relacionados ao valor da Guia e dos atendimentos
5	Agendar Atendimentos	*Não possui*	1. O agendamento de sessões pode ser efetuado por meio de uma guia ou de uma venda.
6	Identificar Atendente	Com Atributos Atendentes	1. Quando for necessário armazenar informações sobre o ganho e a especialidade do atendente.
7	Realizar Acompanhamento	Com Avaliações Educação Física Com Avaliações Fisioterapia Com Avaliações Terapia Ocupacional	1. Cada clínica possui avaliações específicas, conforme os tratamentos que realizam, com atributos específicos. Por exemplo, os atributos analisados em uma avaliação ortopédica para uma clínica de fisioterapia são diferentes dos atributos avaliados para uma clínica de educação física. 2. Cada avaliação pode possuir um conjunto de fatores que são variantes. Por exemplo, em uma avaliação o uso de medicamentos deve ser considerado; em outra alguns testes especiais devem ser realizados para a conclusão do tratamento. Esses fatores são considerados como Itens da Avaliação.
8	Realizar Compras	*Não possui*	
9	Controlar Faturamento	*Não possui*	1. Depende da aplicação dos padrões Realizar Vendas, Processar Guias, Identificar Atendentes e Realizar Compras para definir quais classes devem ser usadas para o sistema

4.3. Definição das *Tags* XML para o Gerador GAwCRe

Para armazenar as informações referentes à linguagem de padrões e à LMA um meta-modelo em XML (XML, 2004) foi elaborado definindo-se um conjunto de *tags* e de atributos de *tags*. Essas informações poderiam ser representadas em uma base de dados com tabelas, como é feito com o GREN-Wizard (Braga,2003). Entretanto, o GREN-Wizard baseia-se no *framework* GREN e as regras de negócios estão embutidas nele. Assim, quando uma aplicação é definida, o GREN-WIzard instancia a aplicação, com os requisitos desejados e herdando as regras de negócio do *framework* GREN.

Para o gerador proposto às regras de negócio (métodos) também devem ser armazenadas juntamente com a definição da linguagem de padrões e, nesse caso, o uso de uma base de dados dificultaria o armazenamento, pois alguns métodos para a definição das

regras de negócio são constituídos de algumas dezenas de linhas de código. A XML possui um recurso denominado de *Seção Marcada* (`<![CDATA[...]]>`) que permite que os métodos das classes sejam escritos e representados de forma legível ao desenvolvedor. Esta seção descreve o papel que cada *tag* e atributo de *tag* exercem na definição das aplicações, possibilitando assim o mapeamento de outras linguagens de padrão para o gerador proposto.

4.3.1. *Tag*: `<linguagemPadrao> ... </linguagemPadrao>`

Descrição: Usada para definir qual o nome da linguagem de padrões usado pelo gerador.

Valor da *Tag*: Nome da linguagem de padrões.

Atributos de *Tag*: *Não possui.*

4.3.2. *Tag:* `<padrao> ... </padrao>`

Descrição: Usada para definir os padrões da linguagem.

Valor da *Tag*: *sem valor.*

Atributos de *Tag*:

- `numero:` Identifica o padrão unicamente;
- `nome:` Define o nome do padrão que será exibido pelo instanciador da LMA;
- `variante:` Identifica se o padrão possui ou não uma variante, (`S|N`) respectivamente. A omissão desse atributo garante que o padrão não possui variante;
- `opcional:` Identifica se o padrão é opcional ou não, (`S|N`) respectivamente. Sua omissão define que o padrão não é opcional. Um padrão é opcional quando a sua aplicação pode ser omitida.

4.3.3. *Tag:* `<classe> ... </classe>`

Descrição: Usada para definir as classes de um padrão, nela devem ser especificados os atributos, métodos e associações entre classes.

Valor da *Tag*: *sem valor.*

Atributos de *Tag*:

- `nome:` Define o nome das classes que serão usados para a geração das classes e interfaces das aplicações;
- `menu:` Define o menu (`cadastro`, `gerenciamento`, `avaliação` ou `relatório`) onde a classe deverá ser encontrada na aplicação`;`
- `tipoInterface:` Define qual o tipo de interface que a classe terá na aplicação. Podem ser classificadas como: a) uma *interface simples* em que as informações de todos os objetos cadastrados são exibidas, nesse caso recebe o valor `1;` b) uma *interface dependente da escolha de um*

objeto para exibir os objetos cadastrados associados ao escolhido, nesse caso recebe o valor `2`; c) uma interface usada para a avaliação de pacientes, nesse caso recebe o valor `avaliacao` ;d) uma interface com funcionalidade específica, que deve ser previamente definida no gerador de interfaces, nesse caso receber o valor `especifica;`

- `opcional:` Identifica se a classe é opcional ou não, (`S|N`) respectivamente. Sua omissão define que a classe não é opcional. Uma classe é opcional quando ela depende da instanciação de um outro padrão ou de uma variante.

4.3.4. *Tag:* `<atributo> ... </atributo>`

Descrição: Usada para definir os atributos de uma classe.

Valor da *Tag*: Nome do atributo na classe e/ou campo da tabela.

Atributos de *Tag*:

- `tipo:` Define o tipo de dado (`Date,int, double, String`) de um atributo. Esse tipo de dado é usado tanto para a criação das tabelas como para a criação das classes Java;
- `tamanho:` Especifica o tamanho do atributo. É usado na criação da tabela e na definição das interfaces com o usuário;
- `tipoCampo:` Define o tipo do campo das interfaces com o usuário. Valores possíveis: `select (Combo), hidden` (campo oculto), `textArea` (campo múltiplas linhas), `text` (campo de linha única), `radio, check, label` (exibe valor) e `lista` (cria uma lista de valores pré cadastrados em uma tabela. É usado para quando há uma associação entre as classes);
- `nome:` Define o nome a ser exibido nas interfaces com o usuário;
- `primaryKey:` Define se o atributo é ou não uma chave primária, (`S|N.`) respectivamente. Sua omissão define o atributo como não sendo chave primária. Também é usado para definir o valor a ser armazenado em uma tabela quando há associações entre classes e há um atributo do tipo `lista` definido;
- `requerido:` Define se o atributo é requerido ou não para o preenchimento na tabela e/ou nas interfaces com o usuário,(`S|N) respectivamente.` Sua omissão define o atributo como não requerido;
- `opcional:` Define se o atributo é opcional ou não em relação à classe que ele pertence, (`S|N`) respectivamente. Um atributo é opcional quando a sua existência depende da aplicação de outros padrões da linguagem. Sua

omissão define o atributo como não sendo opcional;

- `padraoId`: Define qual padrão deve ser aplicado, no caso do atributo ser opcional. O valor deve ser igual a algum `numero` de padrão existente;
- `varianteId`: Define qual variante do padrão deve ser aplicado, no caso do atributo ser opcional. O valor deve ser igual a um número de variante existente no padrão definido pelo atributo de *tag* `padrãoId`. Caso o padrão não possua variante o valor usado deve ser `default`.

4.3.5. *Tag*: `<valor> ... </valor>`

Descrição: 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.

Valor da *Tag*: *sem valor.*

Atributos de *Tag*:

- `exibe`: Define o dado que será exibido nas interfaces com o usuário;
- `grava`: Define o dado que será armazenado na tabela quando o usuário selecionar um valor da lista.

4.3.6. *Tag*: `<associacao> ... </associacao>`

Descrição: Usada para definir as associações entre as classes dos padrões.

Valor da *Tag*: Nome da associação entre as classes.

Atributos de *Tag*:

- `ocorrenciaMax`: Define a ocorrência máxima de uma associação existente entre as classes. Valores possíveis `1|N;`
- `classe`: Define a classe a qual se possui uma associação. Valores possíveis: qualquer nome de classe que tenha sido definido na linguagem de padrões;
- `opcional`: Define se a associação é opcional ou não (`S|N` respectivamente), isto é, se a associação depende da aplicação de algum outro padrão da linguagem. Sua omissão define a associação como não opcional.
- `padraoId`: Define qual padrão deve ser aplicado, no caso da associação ser opcional. O valor deve ser igual a algum `numero` de padrão existente.
- `varianteId`: Definir qual variante do padrão deve ser aplicado, no caso da associação ser opcional. O valor deve ser igual a um número de variante existente no padrão definido pelo atributo de *tag* `padrãoId`. Caso o padrão não possua variante o valor usado deve ser `default`.

4.3.7. *Tag*: `<metodo> ... </metodo>`

Descrição: Usada para definir os métodos de cada classe.

Valor da *Tag*: Nome do método a ser definido na classe Java

Atributos de *Tag*:

- `tipo`: Define o tipo do método (`public`, `protected`, `private`) para a classe Java.;
- `tipoRetorno`: Define o tipo de dado de retorno do método (`void`, `Date`, `int`, `double` ou `String`);
- `menu`: Define em qual menu (`cadastro`, `gerenciamento`, `avaliação` ou `relatório`) o método deve ser chamado. Sua omissão determina que o método será executado pela interação entre classes e não por meio de uma ação do usuário;
- `nome`: Define o nome do método a ser exibido nas interfaces com os usuários. A interface só será criada se houver um `menu` definido;
- `opcional`: Define se o método é opcional ou não (`S|N` respectivamente), isto é, se depende da aplicação de algum outro padrão da linguagem. Sua omissão define o método como não opcional;
- `padraoId`: Define qual padrão deve ser aplicado, no caso do método ser opcional. O valor deve ser igual a algum `numero` de padrão existente;
- `varianteId`: Define qual variante do padrão deve ser aplicado, no caso do método ser opcional. O valor deve ser igual a um número de variante existente no padrão definido pelo atributo de *tag* `padrãoId`. Caso o padrão não possua variante o valor usado deve ser `default`.

4.3.8. *Tag*: `<parametro> ... </parametro>`

Descrição: Usada para definir os parâmetros usados pelos métodos.

Valor da *Tag*: Nome do parâmetro a ser usado na chamada do método

Atributos de *Tag*:

- `tipo`: Define o tipo de dado (`Date`, `int`, `double`, `String`) do parâmetro do método;
- `tamanho`: Determina o tamanho do campo usado nas interfaces com os usuários;
- `tipoCampo`: Define o tipo do campo das interfaces com os usuários. Valores permitidos: `select` (Combo), `hidden` (campo oculto), `textArea` (campo múltiplas linhas), `text` (campo de linha única), `radio`, `check`, `label` (exibe valor);
- `origem`: Informa qual classe originará os valores exibidos em um campo do tipo `select`, quando esse necessitar exibir valores cadastrados em uma tabela do

banco de dados. Valores permitidos qualquer classe definida em algum padrão existente na linguagem.
- ♦ `nome`: Define o nome do parâmetro a ser exibido nas interfaces com os usuários.

4.3.9. *Tag*: `<valor> ... </valor>`

Descrição: Usada quando a *tag* `<parametro>` possui o atributo de *tag* `tipoCampo` definido como `select` ou `radio`. Nesse caso cria-se uma lista valores pré definidos.

Valor da *Tag*: *sem valor.*

Atributos de *Tag*:
- ♦ `exibe`: Define o dado que será exibido nas interfaces com os usuários.
- ♦ `usa`: Define o valor que será usado na passagem do parâmetro.

4.3.10. *Tag*: `<corpoMetodo> ... </corpoMetodo>`

Descrição: Usada para definir o corpo do método.

Valor da *Tag*: Método escrito em linguagem Java. Deve vir escrito dentro de uma seção `<![CDATA[ METODO ]]>` pois assim seu valor não será interpretado pelo parser XML.

Atributos de *Tag*: *Não possui*

4.3.11. *Tag*: `<variante> ... </variante>`

Descrição: Usada para definir a variante de um padrão.

Valor da *Tag*: *sem valor.*

Atributos de *Tag*:
- ♦ `numero`: Define o número de identificação de uma variante para um padrão. Esse número deve ser único para um padrão, podendo ser repetido entre os padrões
- ♦ `nome`: Define o nome atribuído a variante, o que permite diferenciá-la em cada padrão.

Cada padrão da linguagem de padrões é constituído de uma ou muitas 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. A Figura 4.2 apresenta as regras de criação de um documento XML seguindo a ordem de uso das *tags* para que o gerador funcione corretamente.

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

Figura 4.2 – Ordem de uso das *tags* XML para o gerador

A Figura 4.3 apresenta a definição do padrão (2), *Definir Serviços*, da linguagem de padrões SiGCli, para exemplificar o uso das *tags* quando esse padrão for utilizado pelo gerador. Pela definição desse padrão, pode-se observar que ele possui duas classes: a `Serviço` e a `Tipo de Serviço`, representadas pela *tag* `<classe>`. A classe `Serviço` possui três atributos, sendo um deles (`valor`) opcional, isto significa que ele depende da aplicação do padrão 3 (`padraoId="3"`) juntamente com a sua variante implícita (`varianteId="default"`) para que essa informação seja incorporada à aplicação gerada. A associação entre as classes `Serviço` e `Tipo de Serviço` também é opcional, uma vez que ela depende da aplicação do padrão 2 (`padraoId="2"`) juntamente com sua variante 1 (`varianteId="1"`). A classe `Tipo de Serviço` depende da variante `Com tipo servico`. Seguindo esse exemplo todos os padrões da SiGCli foram escritos em XML e são utilizados para gerar as aplicações do gerador GAwCRe.

```
<linguagemPadrao> SiGCli
...
<padrao numero="2" nome="Definir Serviços" variante="S">
   <classe nome="Serviço" menu="cadastro" tipoInterface="1">
     <atributo tipo="int" tamanho="4" tipoCampo="text" nome="Código" primaryKey="S"
               requerido="S">codigoId</atributo>
     <atributo tipo="String" tamanho="25" tipoCampo="lista" nome="Serviço"
               requerido="S">descricao</atributo>
      <atributo tipo="double" tamanho="10,2" tipoCampo="text" opcional="S"
               nome="Valor a ser pago"  padraoId="3"
                varianteId="default">valor</atributo>
      <associacao ocorrenciaMax="1" classe="Tipo Serviço" opcional="S" padraoId="2"
                   varianteId="1">tpservico</associacao>
       <metodo tipo="public" tipoRetorno="String" menu="relatorio"
                 nome="Relatório de Serviços">  listaServicos
           <corpoMetodo>
           <![CDATA[  ...  ]]>
          </corpoMetodo>
       </metodo>
    </classe>
    <variante numero="1" nome="Com tipo serviço">
       <classe nome="Tipo Serviço" opcional="S" menu="cadastro" tipoInterface="1">
         <atributo tipo="int" tamanho="3" tipoCampo="text" nome="Código"
                   requerido="S"  primaryKey="S">codigoId</atributo>
         <atributo tipo="String" tamanho="25" tipoCampo="lista"
                    nome="Tipo do Serviço"  requerido="S">descricao</atributo>
        <método ...  > ... </metodo>
       </classe>
     </variante>
</padrao>
...
</linguagemPadrao>
```

Figura 4.3 – Exemplo do uso das *Tags* para o padrão *Definir Serviços* da linguagem de padrões SiGCli

4.4.Os Gabaritos de Criação dos Artefatos da Aplicação.

Um artefato é qualquer item criado como parte da definição, manutenção, ou utilização de um processo de software, podendo ou não ser entregue a um cliente ou usuário final (Franca; Staa, 2001). Por essa definição todos os produtos gerados por um gerador, desde os *scripts* de criação do banco de dados até a aplicação final são considerados artefatos.

Gabaritos (*templates*) são estruturas pré-existentes de código usadas para definir os produtos que se deseja gerar por intermédio de um gerador de aplicações. Eles possuem partes fixas, presentes em todos os artefatos gerados, e partes variáveis que possibilitam a geração de diferentes artefatos usando o mesmo gabarito. Neste trabalho, foram elaborados gabaritos para gerar os artefatos produzidos pelo GAwCRe.

A seguir é descrito o processo utilizado para a construção de um gabarito para criação de *scripts* escritos em linguagem SQL (*Structured Query Language*), seguindo o padrão do banco de dados Oracle(2002), para gerar tabelas no GAwCRe. Analisando-se alguns *scripts* existentes para tal finalidade, partes fixas e variáveis foram identificadas. A Figura 4.4 apresenta um conjunto de *scripts* de criação de tabelas usando SQL, sendo que os itens circulados representam as partes fixas encontradas nos *scripts* analisados. A Figura 4.5 apresenta o gabarito definido esses *scripts, a*s partes fixas estão definidas de acordo com a identificação na Figura 4.4 e as partes variáveis são colocadas entre *tags.*

```
CREATE TABLE CIDADE (
       CODIGOID NUMBER(4)          NULL,
       DESCRICAO VARCHAR2(40)      NULL,
       ESTADO VARCHAR2(2)          NULL,
       PAIS VARCHAR2(20)           NULL);

CREATE TABLE PROFISSAO (
       CODIGOID NUMBER(4)          NULL,
       DESCRICAO VARCHAR2(20)      NULL);

CREATE TABLE PATOLOGIA (
       CODIGOID VARCHAR2(20)       NULL,
       NOME VARCHAR2(40)           NULL,
       QTDECH NUMBER(4)            NULL,
       FK_CONVENIO NUMBER(3)       NULL);
```

Figura 4.4 – Conjunto de *scripts* de criação de tabelas usando SQL

```
CREATE TABLE <NOME_DA_TABELA> (
       <ATRIBUTOS> <TIPOS>(<TAMANHOS>) NULL);
```

Figura 4.5 –Gabarito definido para os *scripts* de criação das tabelas

Para que o artefato possa ser gerado pelo GAwCRe, as partes fixas dos gabaritos são previamente definidas no código fonte do gerador. As partes variáveis são substituídas pelas informações existentes no documento XML em que foi definida a linguagem de padrões SiGCli. Sendo o gerador desenvolvido em linguagem Java (Java,2003), para que ele possa ter

acesso e manipular as informações existentes em um documento XML, utiliza-se o padrão da W3C (*World Wide Web Consortium*) denominado DOM (*Document Object Model*) (DOM, 2004), que possibilita a navegação entre as *tags* do documento XML (o DOM utilizado neste trabalho é proprietário Oracle (2003)). A Figura 4.6 exemplifica a construção do artefato: *script* SQL de criação de tabelas. Os caracteres em negrito da Figura 4.5 são as partes fixas do gabarito, embutidos no código Java como mostra a Figura 4.6 (a). Dessa forma, o gerador deve completar as informações como `NOME_DA_TABELA`, `ATRIBUTOS`, `TIPOS` e `TAMANHOS` a partir das definições XML do padrão (2) da SiGCli, exibido na Figura 4.3. O artefato gerado é então exibido na Figura 4.6(b).

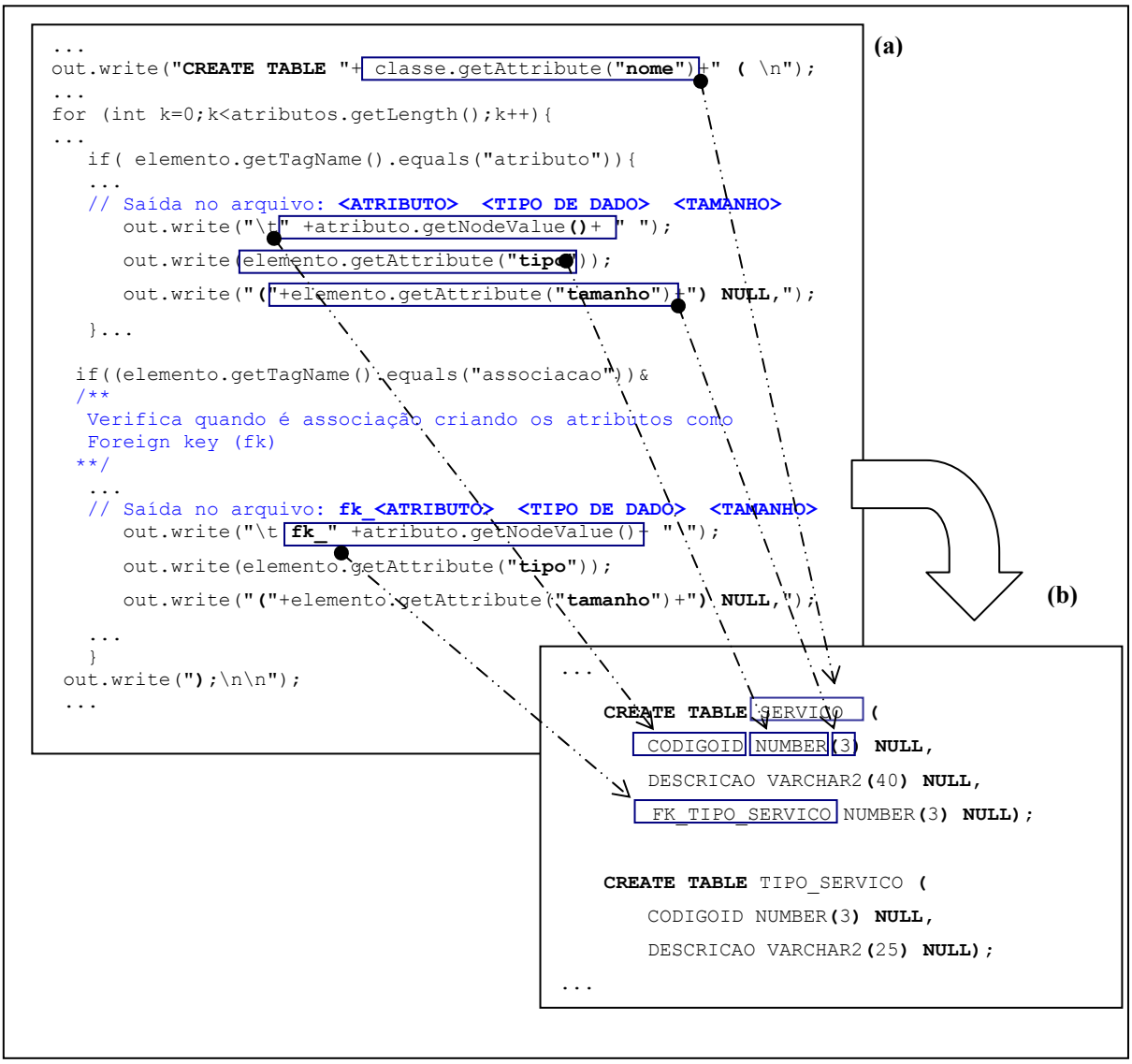


Figura 4.6 – Esquema de construção do artefato *script* SQL de criação de tabelas

Todos os artefatos produzidos pelo gerador (classes Java, scripts de criação do banco de dados e diversos tipos de interfaces) passaram pela mesma forma de identificação das partes variáveis e fixas, definindo-se os gabaritos para cada artefato desejado. Após a definição dos gabaritos, eles foram embutidos no código da classe Java responsável pela geração do artefato no gerador

4.5. Arquitetura do Gerador

O gerador GAwCRe possui em sua arquitetura um conjunto de classes responsáveis por gerenciar e gerar as diferentes instanciações das aplicações, de acordo com os requisitos as especificações da LMA. A Figura 4.7 exibe o diagrama de classes (em alto nível) desse gerador.

A classe `FrameGerador` é responsável por exibir a interface de instanciação das aplicações com o desenvolvedor por meio de um objeto do tipo frame, construído dinamicamente, com base nas informações existentes no documento XML referentes a especificação da LMA. A classe `XMLManager` obtém os dados a serem exibidos pela classe `FrameGerador`, utilizando o *parser* DOM.

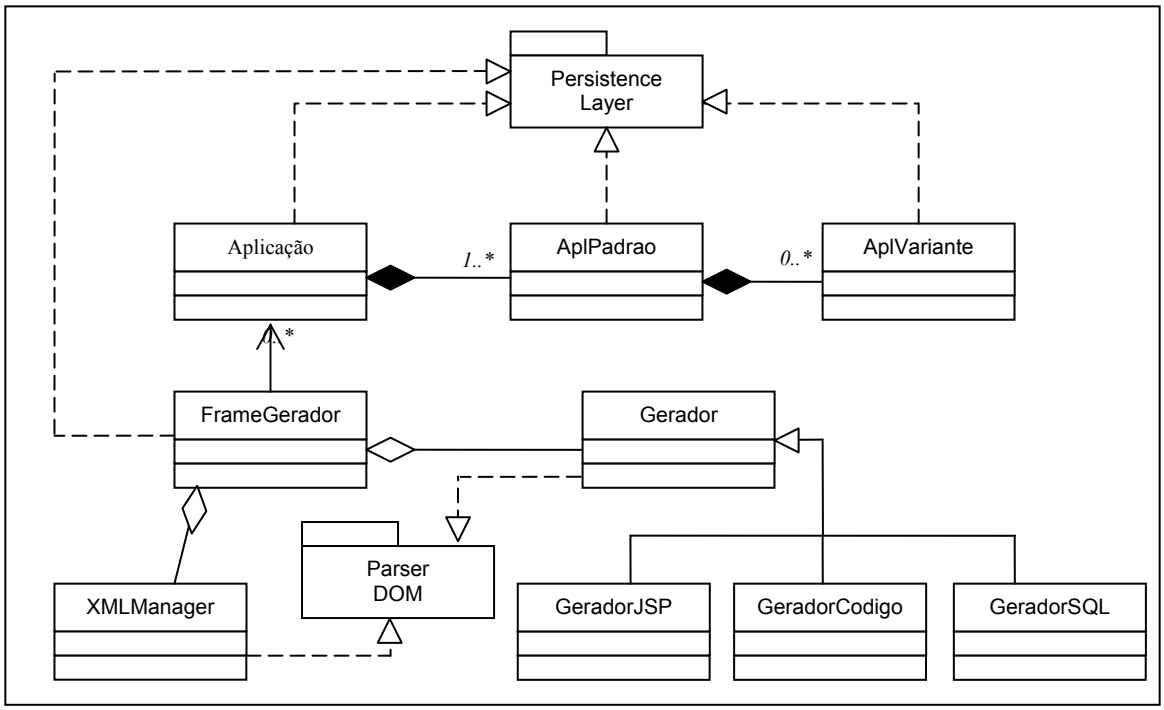


Figura 4.7 - Diagrama de classes do gerador de GAwCRe

As classes `GeradorCodigo`, `GeradorSQL` e `GeradorJSP` herdam da classe `Gerador` os métodos responsáveis por manipular os dados do documento XML. A classe `GeradorCodigo` é responsável por gerar o código Java das aplicações; a classe `GeradorSQL` gera os *scripts* de criação das tabelas e a classe `GeradorJSP` cria a interface da aplicação com os usuários. As classes `Aplicacao`, `AplPadrao` e `AplVariante` representam, respectivamente as aplicações, os padrões e as variantes de cada aplicação instanciada pelo gerador. As informações das aplicações instanciadas são armazenadas em um banco de dados relacional utilizando o padrão *Persistence Layer* (Yoder et. al., 1998). Esse padrão também é usado para persistir dos dados das aplicações geradas pelo gerador. As seções seguintes descrevem os cinco elementos básicos responsáveis pelo processo de geração do gerador: as *Bibliotecas para as Aplicações Geradas*, o *Instanciador da LMA*, o *Gerador de Scripts SQL*, o *Gerador das Classes Java (Beans)* da aplicação e o *Gerador das Interfaces JSP*.

4.5.1. Bibliotecas para as Aplicações Geradas

O gerador possui algumas bibliotecas que são usadas em todas as aplicações por ele geradas. Elas estão hospedadas nos servidor responsável pela interpretação das interfaces JSP, para suprir algumas funcionalidades que não necessitam ser geradas para cada aplicação.

Essas bibliotecas são: `menudefs.js`, `menueventos.js` e `menustyle.js`, escritas em linguagem *Java Script*(Goodman, 2001) e são responsáveis pelo funcionamento do menu das aplicações. A biblioteca `menudefs.js` possui as definições gerais do menu, como por exemplo, em que local da interface da aplicação esse menu deve ser exibido. A `menueventos.js` define quais ações do usuário serão respondidas pelo menu, como por exemplo, o que deve acontecer quando o usuário escolher uma determinada opção do menu. A biblioteca `menustyle.js` define a folha de estilo do menu, ou seja, as cores e fontes em que serão exibidos os menus.

O padrão *Persistence Layer* (Yoder et. al., 1998), implementado em Linguagem Java, também faz parte de uma biblioteca, denominado `PL`, hospedado no servidor para que as aplicações possam persistir seus dados, em um banco de dados Oracle, usando as funcionalidades desse padrão.

4.5.2. O Instanciador da LMA

A interface de instanciação das aplicações é construída dinamicamente a partir das definições do documento XML criado para esse propósito. A classe `XMLManager`, do gerador, lê o documento XML, recuperando as informações das *tags* `<padrao>` e `<variante>` por meio do o atributo `nome`, que contém as informações da especificação LMA a serem exibidas na interface de instanciação, em forma de *checkbox's*. Para o desenvolvedor especificar quais itens da LMA são necessários para a aplicação, basta selecionar os *checkbox's* que satisfaçam os requisitos da aplicação. Dessa forma, sempre que um *checkbox* é selecionado, as informações são armazenadas na base de dados Oracle do gerador, o que possibilita, sempre que necessário, a recuperação e a alteração da especificação LMA para cada aplicação gerada.

Cada um dos padrões da SiGCli é apresentado, individualmente, na interface de instanciação do gerador, juntamente com o seu conjunto de variantes, se houver. Para os padrões que são definidos como obrigatórios para a LMA (aqueles cujo atributo `opcional` da *tag* `<padrão>` possui o valor igual a `N`), o gerador só permite continuar o processo de instanciação da LMA após sua aplicação. A Figura 4.8(b) apresenta a interface de instanciação da LMA do gerador GAwCRe para o padrão 7 da SiGCli, *Realizar Acompanhamento*, exibindo também o trecho do documento XML que o define (Figura 4.8(a)). Esse padrão possui variantes (como pode ser observado pelo atributo de *tag* `variante="S"`) e é obrigatório para as aplicações desse domínio (como pode ser observado pelo atributo de *tag* `opcional="N"`).

4.5.3. O Gerador de *Scripts* SQL

Com as informações das aplicações armazenadas na base de dados do gerador, o módulo *Gerador de Scripts SQL* obtém as informações sobre a aplicação a ser gerada comparando-as com as definições do documento XML. Assim, para cada padrão usado na aplicação, é feita análise de quais tabelas devem ser criadas com seus respectivos atributos e chaves. Cada `<classe>` é analisada individualmente sendo gerados três artefatos distintos (seguindo as regras SQL determinadas pela Oracle), porém relacionados: `ScriptTabelas.sql, ScriptPK.sql, ScriptFK.sql.`

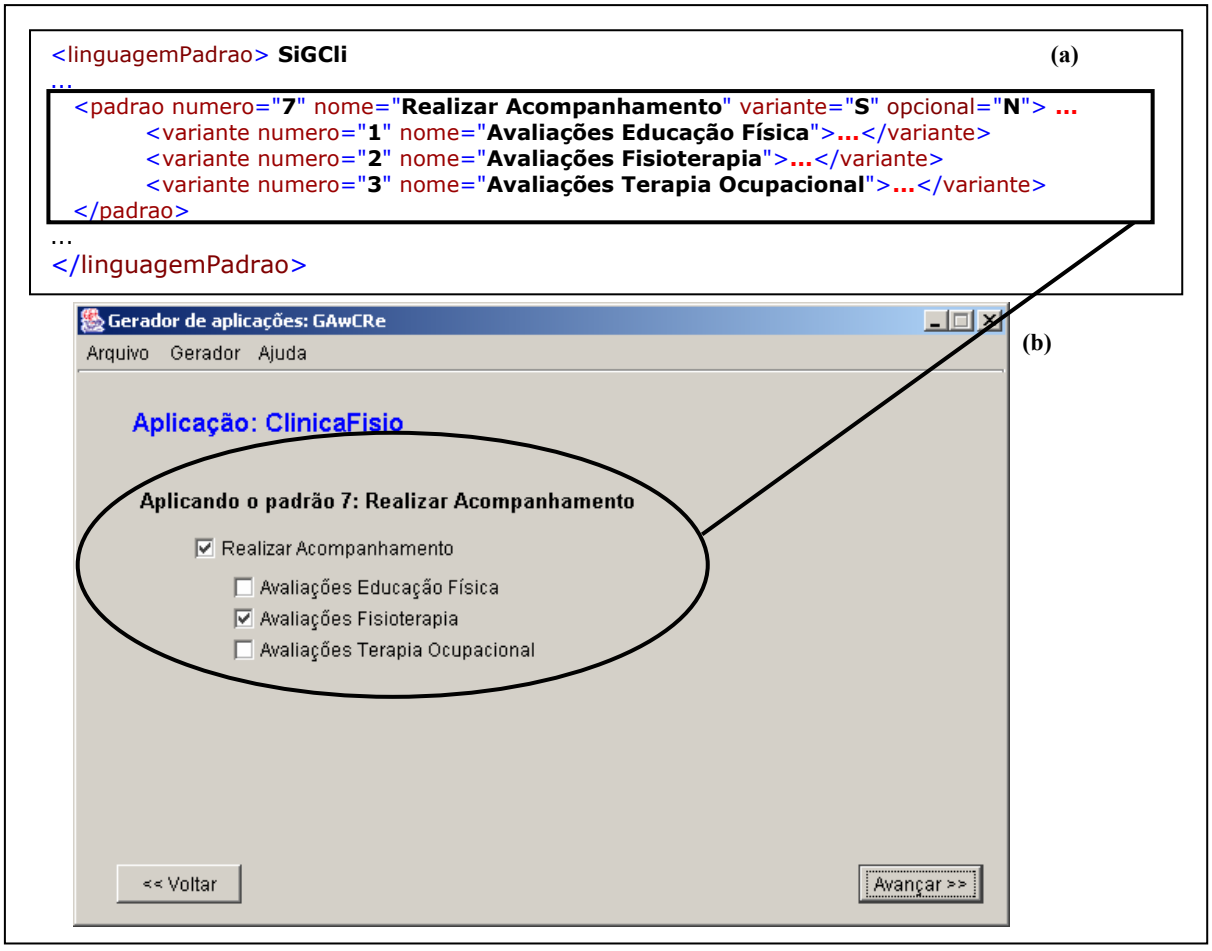


Figura 4.8 – A interface de instanciação da LMA do Gerador GAwCRe para o padrão 7, *Realizar Acompanhamento*, da SiGCli

O artefato `ScriptTabelas.sql` define a estrutura das tabelas da base de dados, sendo para a sua elaboração analisadas as *tags: a)* `<classe>` possui um atributo denominado `nome` que define o nome da tabela que deverá ser gerada para a aplicação; b) `<atributo>` possui os atributos `tipo` e `tamanho` que definem o tipo de dados e o tamanho dos campos da tabela, respectivamente. O nome do campo da tabela é definido pelo valor atribuído à *tag*. c) `<associacao>` possui o atributo `classe` que define qual a classe que se relaciona com a atual. Sempre que uma associação for definida, um atributo com um prefixo "`FK_`" e o nome da classe (identificada pelo atributo `classe` de tag `<associacao>`) são adicionados ao artefato. O tipo de dado e o tamanho do atributo "`FK_`" são definidos pelo `<atributo>` identificado como chave primária na classe associada. A Figura 4.9 apresenta o gabarito dos scripts de criação das tabelas com a lógica usada para a sua definição.

```
CREATE TABLE <NOME DA CLASSE> (
// para cada atributos ou associação no documento XML
// da classe atual
       <VALOR DA TAG ATRIBUTO> <TIPO>(<TAMANHO>) NULL,
   // se for uma associação
      FK_<CLASSE ASSOCIADA> <TIPO DA PK DA CLASSE ASSOCIADA>
                    (<TAMANHO DA PK DA CLASSE ASSOCIADA>) NULL,
   //fim se
);
```

Figura 4.9 –Gabarito dos *scripts* de criação das tabelas

O artefato `ScriptPK.sql` altera a estrutura das tabelas definidas no artefato `ScriptTabela.sql` criando as suas chaves primárias (*primary key*). Para a sua elaboração, a *tag* `<atributos>` é analisada e sempre que o atributo de *tag* `primaryKey` for igual a "`S`", ele será definido como chave primária. A estrutura do campo é alterada para não permitir valor nulo e uma chave primária como prefixo "`PK_`", seguido do nome da classe atual é criada. A Figura 4.10 apresenta o gabarito de definição das chaves primárias.

```
// para cada atributo definido como pk
ALTER TABLE <NOME DA CLASSE> (
   MODIFY     <VALOR DA TAG ATRIBUTO PK> <TIPO PK>(<TAMANHO PK>)
         NOT NULL);

ALTER TABLE <NOME DA CLASSE> (
   ADD (CONSTRAINT PK_<NOME DA CLASSE>
        PRIMARY KEY (<VALOR DA TAG ATRIBUTO PK>));
```

Figura 4.10 –Gabarito de definição das chaves primárias

O artefato `ScriptFK.sql` altera a estrutura das tabelas definidas no artefato `ScriptTabela.sql` criando as chaves estrangeiras (*foreign key*) definindo os relacionamentos entre as tabelas. Para a sua elaboração são analisadas as *tags* `<associacao>`. Sempre que existir uma associação entre classes, um relacionamento deve ser estabelecido entre as tabelas da base de dados. Para isso, é adicionada uma chave estrangeira com o prefixo "`FK_`", seguido do nome da classe que está sendo atualmente analisada pelo gerador e o nome da classe associada (definida pelo atributo de *tag* `classe` da *tag* `<associacao>`). A Figura 4.11 apresenta o gabarito de definição das chaves estrangeiras.

```
// para cada associação encontrada
ALTER TABLE <NOME DA CLASSE ATUAL> (
 ADD (CONSTRAINT FK_<NOME DA CLASSE ATUAL>_<NOME DA CLASSE ASSOCIADA>
 FOREIGN KEY (FK_<NOME DA CLASSE ASSOCIADA>)
 REFERENCES <NOME DA CLASSE ASSOCIADA>
            (<VALOR DA TAG ATRIBUTO PK DA CLASSE ATUAL>);
```

Figura 4.11 –Gabarito de definição das chaves estrangeiras

4.5.4. O Gerador das Classes Java (*Beans*) da Aplicação

Com as informações das aplicações armazenadas na base de dados do gerador, o módulo *Gerador das Classes Java (Beans) da Aplicação* obtém as informações sobre a aplicação a ser gerada comparando-as com as definições do documento XML. Assim, para cada padrão usado na aplicação, é feita análise de quais classes, atributos, associações e métodos devem ser criados para satisfazer os requisitos da aplicação. Esse módulo gera todas as classes referentes à aplicação, onde estão suas regras de negócio. Essas classes serão usadas como *Beans* para a aplicação Web. *Beans* são componentes de software escritos em Java. Assim, como muitos outros componentes de software, eles encapsulam tanto o estado quanto o comportamento. São classes Java que seguem um conjunto de convenções de projeto e de nomeação definidos de JavaBeans(Fields; Kolb ,2000).

Para elaboração das classes Java são analisadas as *tags* `<classe>` e todas as outras definidas internamente a ela. Assim, o nome de uma classe é definido pelo atributo `nome` existente na *tag* `<classe>`. Os atributos da classe são identificados na *tag* `<atributo>` que define o tipo de dado (`tipo`) do atributo e o nome do atributo (valor da *tag*). Para as associações (`<associacao>`) existentes entre as classes são criados objetos do tipo da classe associada. Os métodos mais simples como os de atribuição (*set)* ou os de recuperação (*get*) são gerados dinamicamente. Já os métodos que tratam de funcionalidades específicas de cada classe, são identificados pela *tag* `<metodo>`, que pode possuir um conjunto de *tags* `<paramentro>`, caso o método tenha parâmetros que definam a sua chamada, e uma *tag* `<corpoMetodo>` que implementa a sua funcionalidade. A Figura 4.12 apresenta parte do gabarito definido para a geração das classes, dando ênfase à parte variável do gabarito.

```
package <nome da aplicacao>;
...
public class <nome da classe> implements PersistentObject {

// Objetos para a manipulação dos dados usando PersistenceLayer
       ...
// Para cada atributos da classe
       private <tipo do atributo> <valor da tag atributo>;

// para cada associação entre as classes
       private <classe associada> <classe associada>;

// Método construtor
       public <nome da classe>(){...    }

// Para cada atributo da classe
//********** Métodos SET da classe **********/
       public synchronized void set<valor da tag atributo>(<tipo do atributo>
valor){
              this.<valor da tag atributo> = valor;
       }
//********** Métodos GET da classe **********/
       public <tipo do atributo> get<valor da tag atributo>(){
              return this.<valor da tag atributo>;
       }
//*********** Métodos para Persistencia e manipulação da base da dados *******
       public boolean save() {...}
       public boolean delete (){...}
       public ResultSet findall (){...}
       public ResultSet findlike (){...}
       public int getLast(){...}
       public int getCount() {...}

// Para cada método específico da classe
//*********** Métodos específicos da classe *******
      <tipo metodo> <tipo retorno metodo> <valor da tag metodo>(
                    <tipo parametro> <valor da tag parametro>) {
             <valor da tag corpoMetodo>
       }

}// fim da classe
```

Figura 4.12 – Parte do gabarito definido para a geração das classe Java

4.5.5. O Gerador das Interfaces JSP

Com as informações das aplicações armazenadas na base de dados do gerador, o módulo *Gerador das Interfaces JSP* obtém as informações sobre a aplicação a ser gerada comparando-as com as definições do documento XML. Assim, para cada padrão usado na aplicação, é feita análise de quais interfaces de interação com o usuário devem ser geradas, bem como quais itens do menu da aplicação serão disponibilizados aos usuários. Esse módulo gera um conjunto de artefatos. O artefato `menudados.js` define o menu de opções para a chamada das interfaces JSP da aplicação. O atributo de *tag* que define o menu é denominado `menu` e pode ser encontrado nas *tags* `<classe>` e `<metodo>`, do documento XML. Esse

atributo especifica em qual item do menu a interface deve chamada, existindo quatro opções: cadastro, gerenciamento, avaliações e relatórios. Sempre que o módulo *Gerador das Interfaces JSP* encontra uma classe ou um método específico da classe, ele cria uma interface correspondente. Essas interfaces geradas são adicionadas ao `menudados.js`. Esse menu é escrito em linguagem *Java Script* (Goodman, 2001).

O artefato `menu.html` define a formatação do menu da aplicação e para que seu funcionamento seja correto, o artefato `menudado.js` e outras bibliotecas do gerador, descritas na seção 4.5.1, são utilizadas. Os demais artefatos possuem extensão `.jsp` e definem as interfaces da aplicação com o usuário, sendo geradas com base no tipo de interface necessária para satisfazer os requisitos do sistema. Existem cinco tipos básicos de interfaces definidas para o gerador:

- As interfaces de métodos: são geradas apenas se a *tag* `<metodo>` possuir um valor no atributo `menu`, criando então, um item no menu da aplicação para a execução do `<metodo>`, o que possibilita a interação do usuário com as funcionalidades da aplicação. Por exemplo, os métodos que geram os relatórios;
- As interfaces simples: são geradas para as classes que possuem o atributo `tipoInterface = "1"`. Elas possibilitam o cadastro de informações sem que seja necessário informar quais os valores dos objetos que estão associados a ela;
- As interfaces dependentes da escolha de um objeto: são geradas para as classes que possuem o atributo `tipoInterface = "2"`. Elas permitem o cadastro de informações somente após o usuário ter informado os valores de um ou mais objetos associados a ela, ou seja, quando a informação, a ser inserida, depende obrigatoriamente da existência dos objetos a ela associados;
- As interfaces de avaliação: são geradas para as classes que possuem o atributo `tipoInterface = "avaliacao"`. Elas possuem algumas funcionalidades diferentes dos tipos anteriormente comentados e, por isso, devem ter um gabarito específico para a sua criação;
- As interfaces específicas: são as que possuem funcionalidades distintas das anteriormente comentadas, devendo ser definidas internamente ao gerador Sempre que uma interface for específica o atributo `tipoInterface` deve ser definido como "`especifca`".

Para a elaboração das interfaces a maioria das *tags* e atributos de *tags* definidos no documento XML são utilizados, com exceção da *tag* `<linguagemPadrao>` e `<padrao>`. A Figura 4.13 apresenta parte do gabarito definido para a geração das interfaces simples.

```
<%@page import="<nome da aplicacao>.*"%>
<%@page import="java.sql.*"%>
<jsp:useBean id="<nome da classe>" class="<nome da aplicacao>.<nome da
classe>"/>
<jsp:setProperty name="<nome da classe>" property="*" />

<HTML> <HEAD> <TITLE><nome da classe></TITLE>
<%@ include file="menu.html" %>

<SCRIPT LANGUAGE='JavaScript'>
// Métodos escritos em JavaScript para definir o comportamento da interface
</SCRIPT>
</HEAD>

<BODY onLoad='carrega()'>
Cadastro de <nome da classe>
<form name= 'formulario' method='POST' action='<nome da classe>_resp.jsp'>
<input type='button' value='Novo' name='btnNovo' onClick='novo()'>
<input type='submit' value='Salvar' name='btn'>
<input type='button' value='Limpar' name='btnLimpar'
                      onClick="location='<nome da classe>.jsp'">
<input type='submit' value='Excluir' name='btn'>
...
<select size='8' name='combo' onChange='seleciona_lista()' tabindex='50'>
<% try {
   rs = <nome da classe>.findall();
   while (rs.next()) {  %>
  // Carrega valores da classe no campo select da interfce
   ...
  <%}
  } catch (Exception e){} %>
</select>
...
// Para cada atributo da classe
<nome do atributo> <input type='<tipoCampo do atributo>'
                   name='<valor da tag atributo>'
                   size='<tamanho atributo>'>

// Para cada associação entre as classes
<select size='1' name='fk_<nome da classe associada>'>
<%
 <nome da classe associada> obj= new  <nome da classe associada> ();
 try {
 rs = obj.findall();
 while (rs.next()) {  %>
  // Carrega valores da classe no campo select da interfce
 ...
 <%}
 }catch (Exception e){}%>
</select>
...
</BODY></HTML>
```

Figura 4.13 – Parte do gabarito definido para a geração das interfaces simples

Os demais gabaritos são semelhantes ao apresentado, sendo que as principais diferenças estão na parte de definição dos métodos escritos em *Java Script* e na apresentação de alguns objetos HTML. Esses gabaritos podem ser encontrados em (Pazin; Penteado, 2004); A Figura 4.14 apresenta esquema de funcionamento do gerador GAwCRe e os artefatos por ele gerado.

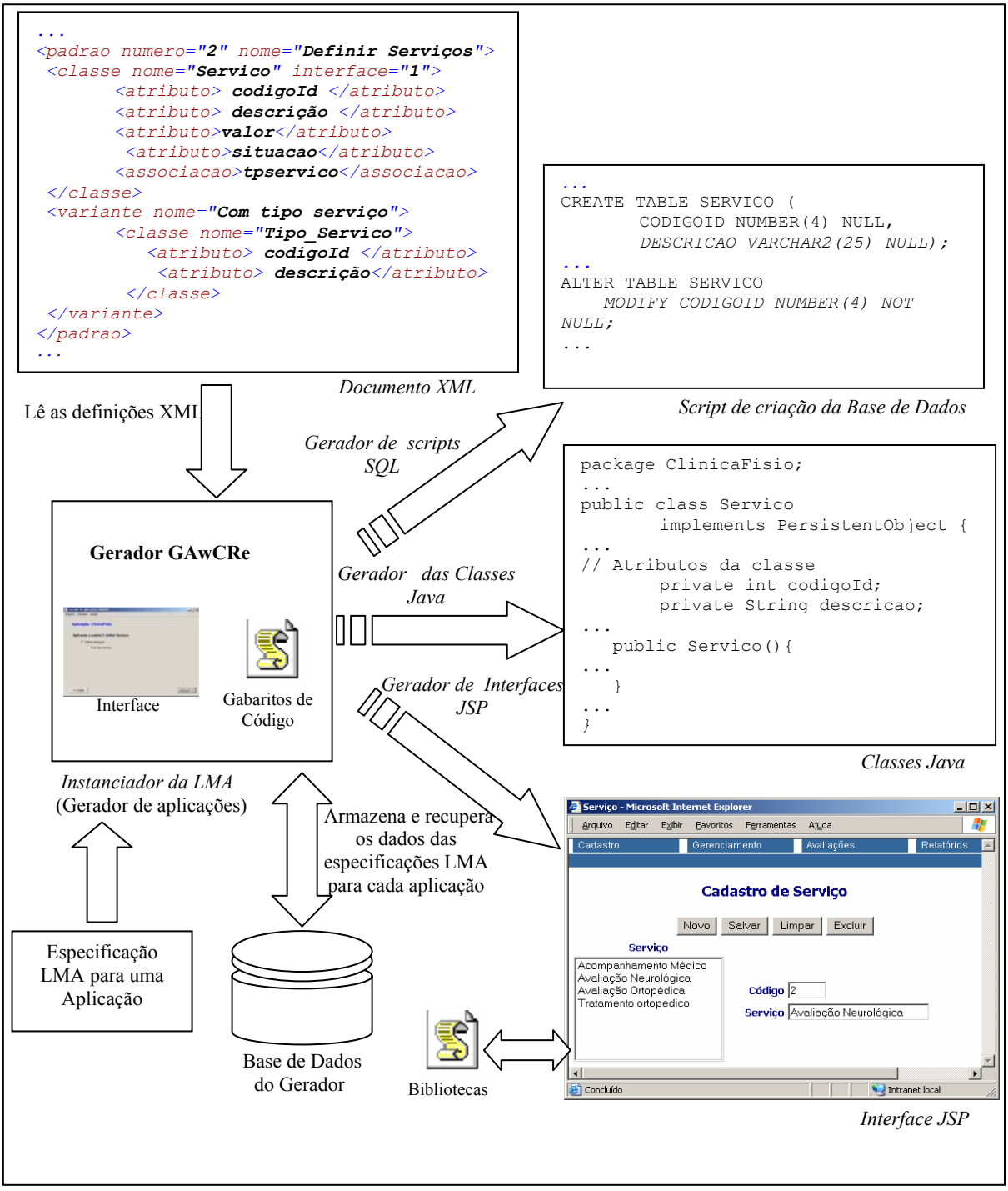


Figura 4.14 – Esquema de funcionamento do gerador GAwCRe e os artefatos por ele gerado

4.6. Arquitetura das Aplicações Geradas

As aplicações geradas pelo gerador GAwCRe são sistemas baseados na Web e desenvolvidos usando a arquitetura três camadas. A cada solicitação do usuário as páginas são geradas pela Camada de Aplicação e lhes são apresentadas por meio de um *browser*. O armazenamento e a recuperação das informações no/do sistema é realizada com a solicitação à Camada de Apresentação. Essa por sua vez, comunica-se com a Camada de Aplicação que se comunica com a Camada de Persistência. A Figura 4.15 apresenta a arquitetura das aplicações.

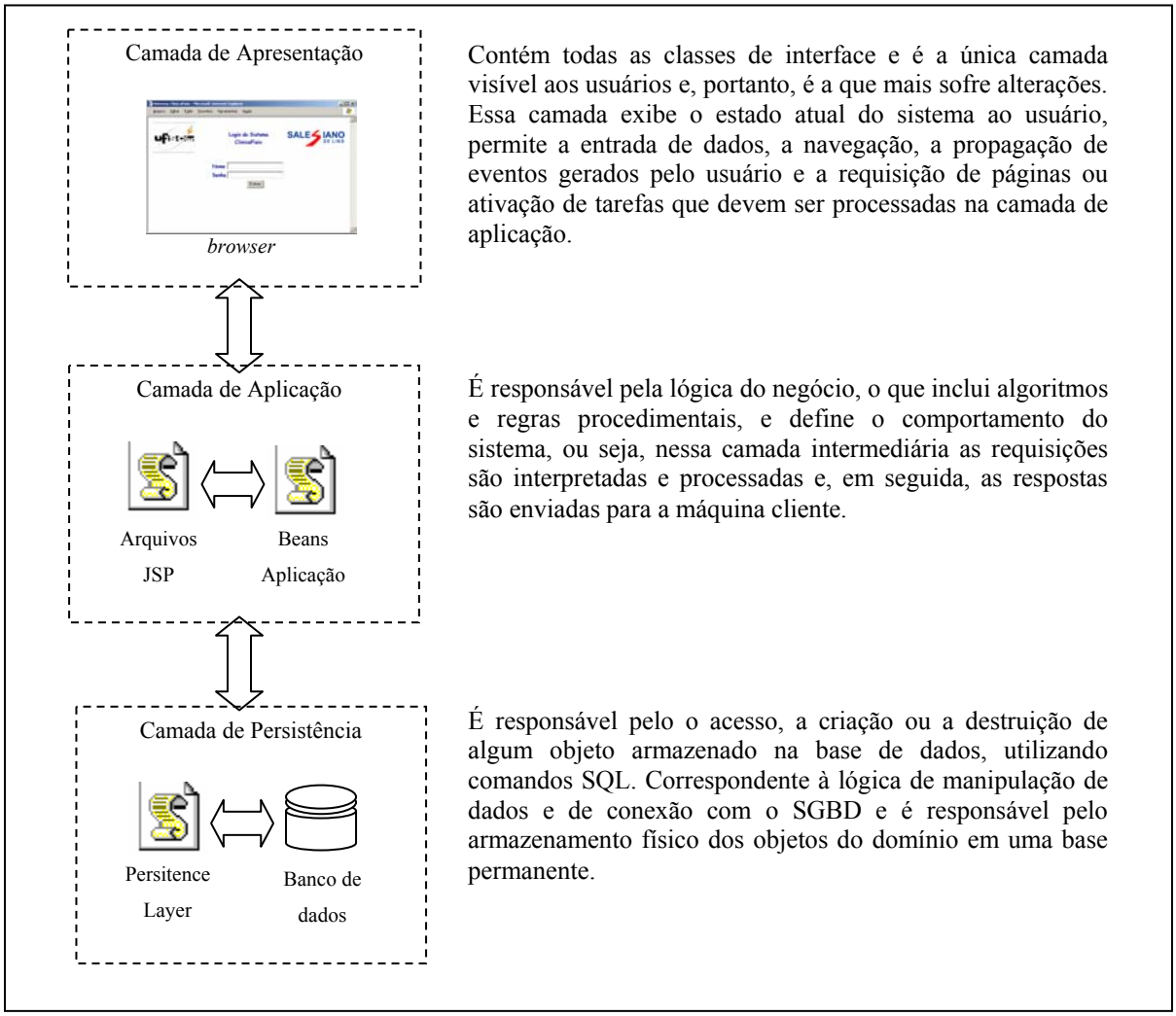


Figura 4.15 – Arquitetura as aplicações geradas pelo gerador GAwCRe

4.7.Considerações Finais

O gerador de aplicações apresentado neste Capítulo foi elaborado para facilitar as instanciações de aplicações no domínio coberto pela linguagem de padrões SiGCli (Pazin, et al., 2004), apresentada em maiores detalhes no Capítulo 3.

Uma Linguagem de Modelagem da Aplicação (LMA) foi elaborada, com base na SiGCli, possibilitando definir as aplicações a partir de especificações em alto nível no gerador. A definição da LMA ocorreu após a aplicação da SiGCli para remodelar os sistemas que serviram como base no processo de engenharia reversa. Assim, foi possível identificar as mutações sofridas durante a aplicação de cada padrão da SiGCli. A cada aplicação e/ou mutação de um padrão um nome foi atribuído de forma a identificá-la unicamente na especificação.

Um meta-modelo em XML foi definido, com base em *tags* e atributos de *tags*, para representar tanto a LMA quanto a linguagem de padrões SiGCli. As definições feitas nesse meta-modelo são utilizadas pelo gerador, para gerar os artefatos da aplicação. Para isso, gabaritos de código foram definidos no código fonte do gerador e sempre que um dos módulos geradores de artefatos é acionado, o gerador analisa as especificações LMA para a aplicação e a compara com os valores definidos no meta-modelo XML, gerando um artefato totalmente correspondente às especificações da LMA da aplicação.

O gerador GAwCRe define uma família de produtos de software que é a de Sistemas de Gestão de Clínicas de Reabilitação. Considerando as variabilidades presentes nos padrões e representadas na LMA, é possível gerar um conjunto distinto de aplicações, considerando as diferentes combinações permitidas para a aplicação dos padrões e de suas variantes.

Um diferencial para este gerador é o fato dele possuir uma linguagem de padrões usada para representar o domínio e uma LMA que apóia o processo de geração. Um outro fator importante é a possibilidade de reusar a estrutura do gerador para outros domínios representados por linguagens de padrão, bastando para isso definir uma LMA, com base nas diretrizes apresentadas na seção 4.2, e mapeando essas linguagens para o documento XML.O uso da XML para armazenar a especificação dos padrões e da LMA facilita o reuso, bem como eventual interoperabilidade com outras ferramentas, como por exemplo, a Rose. Basicamente a estrutura do gerador permanecerá a mesma, mudando-se a LMA para a especificação das aplicações e os gabaritos de código das interfaces que possuem funcionalidades específicas para o domínio desejado. O Capítulo seguinte apresenta um exemplo de uso para o gerador de aplicações GAwCRe.

CAPÍTULO 5

Exemplo de Uso do Gerador GAwCRe

5.1. Considerações Iniciais

Este capítulo apresenta um exemplo de uso para o gerador de aplicações GAwCRe, exibindo o todo o processo para a elaboração de uma aplicação para gerir uma clínica de fisioterapia. A clínica escolhida é a mesma que serviu como base para o processo de análise de domínio. Os requisitos da clínica foram apresentados no Capítulo 3, na seção 3.2.1.

A seção 5.2 apresenta como a linguagem de padrões SiGCli foi aplicada para modelar o sistema, conforme os requisitos definidos. A especificação LMA e os artefatos gerados para a aplicação são apresentados na seção 5.3. Na seção 5.4 é exemplificado o funcionamento do sistema gerado. As considerações finais sobre esse capítulo são apresentadas na seção 5.5.

5.2. Modelagem do Sistema Usando a Linguagem de Padrões SiGCli

Para facilitar o entendimento do gerador GAwCRe o grafo de fluxo de aplicação dos padrões apresentado na Figura 5 é o mesmo do Capítulo 3, Figura 3.12.

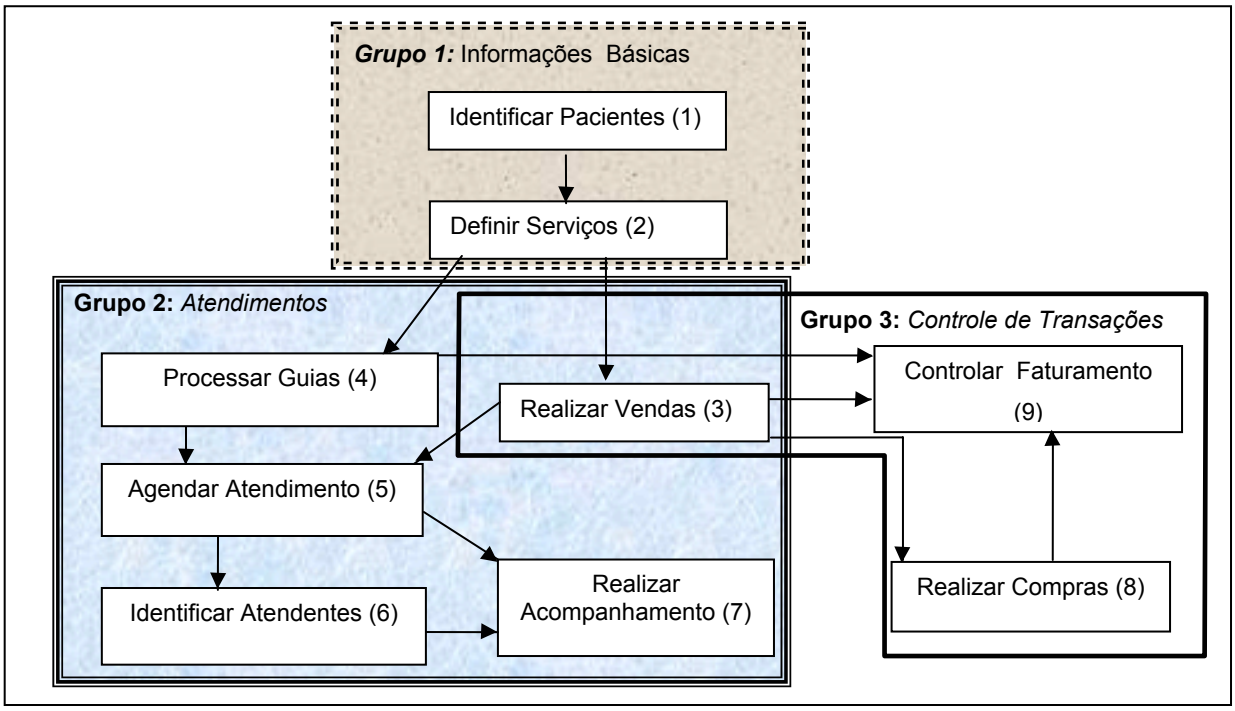


Figura 5.1 – Grafo de fluxo de aplicação dos padrões da SiGCli

A linguagem de padrões SiGCli foi usada para elaborar o diagrama de classes da clínica de Fisioterapia, com base nos requisitos do sistema e com auxílio do grafo de fluxo de aplicação dos padrões da SiGCli. Para que essa modelagem ocorra de forma natural, é necessário que o desenvolvedor conheça a finalidade de cada padrão da linguagem SiGCli, podendo assim usufruir suas vantagens. Assim, a identificação de quais padrões são os mais indicados para solucionar determinados problemas da aplicação pode ser realizada de forma mais rápida.

Para exemplificar a modelagem do sistema de gestão da clínica de fisioterapia alguns fragmentos do documento de requisitos são apresentados juntamente com a justificativa de como os padrões da SiGCli foram utilizados.

> ”... q*uando um paciente inicia o tratamento na clínica, a recepcionista realiza o seu cadastro, classificando-o segundo alguns atributos: sexo, cor da pele, grau de escolaridade, faixa de renda, faixa etária, etc..”.*

Nesse trecho do documento de requisitos fica clara a necessidade de obter as informações pertinentes aos pacientes que procuram a clínica para se recuperar de determinadas lesões. O padrão da SiGCli que permite a identificação e a classificação do

paciente é o *Identificar Paciente* (1). Esse padrão é usado sendo criadas as classes que apresentam (1) junto aos seus nomes na Figura 5.2.

Seguindo o grafo de fluxo de aplicação dos padrões, o padrão (2), *Definir Serviços*, também deve ser aplicado. Embora nos requisitos não constem informações específicas sobre os serviços, esse padrão é obrigatório para a linguagem de padrões. Como no documento de requisitos também não há informações sobre como classificar tais serviços, eles foram definidos sem detalhamento dos seus tipos específicos, ou seja, o padrão foi aplicado sem que a classe opcional fosse utilizada. A classe participante desse padrão é a que apresenta (2) junto ao seu nome na Figura 5.2

"...Com o paciente cadastrado, é aberta uma guia de consulta para que ele possa iniciar o tratamento. Na guia de consulta constam informações sobre o convênio do paciente, número de sessões a ser realizadas, que varia de acordo com o convênio; o nome do médico responsável pelo encaminhamento do paciente e a patologia por ele diagnosticada. Para cada convênio existe um código de classificação da patologia com a quantidade de coeficiente de honorário (chs) correspondente a cada uma, usados para efetuar o pagamento dos tratamentos..."

Pelo grafo de fluxo de aplicação dos padrões, após definir os serviços deve-se aplicar ou o padrão (3) *Realizar Vendas* ou o padrão (4) *Processar Guias*. De acordo com o documento de requisitos da clínica, os pacientes são atendidos somente mediante a apresentação de guias de convênios médicos, identificando-se o convênio, a patologia e o médico responsável pelo encaminhamento do paciente. Essas características são tratadas pelo padrão (4), *Processar Guias*. Não há referências quanto à comercialização de serviços no documento de requisitos, portanto, o padrão (3) não será aplicado para esse sistema. No diagrama da Figura 5.2, as classes que representam a utilização desse padrão são as que apresentam (4) junto aos seus nomes.

"...Após a guia ser aberta, é realizado o agendamento das sessões. As sessões são agendadas conforme a disponibilidade de horário dos estagiários e do paciente. O mesmo estagiário acompanha o paciente durante todas as sessões de uma mesma guia. Dessa forma, o paciente fica vinculado a um determinado estagiário, mas caso ocorra algum problema com ele, um outro pode atender o paciente. Após ter o seu horário agendado, o paciente é informado do inicio do tratamento. A cada sessão a recepcionista confirma a presença do paciente no sistema....

O agendamento das sessões de tratamento do paciente depende da disponibilidade de horários dos estagiários (atendentes) que realizam o tratamento. Na ordem definida para a aplicação dos padrões, primeiro deve-se aplicar o padrão para gerenciar os agendamentos da clínica, *Agendar Atendimento,* padrão (5). Como os atendimentos são vinculados a estagiários, que são as pessoas responsáveis por atender os pacientes, torna-se necessário aplicar também *Identificar Atendentes,* padrão (6). No diagrama de classes da Figura 5.2 as classes referentes aos padrões (5) e (6) são as que esses números junto aos seus nomes.

"... na primeira sessão do paciente, são armazenados os seus dados clínicos e é realizada a avaliação do seu estado pelo estagiário responsável (...). A avaliação do estado do paciente é realizada segundo alguns critérios pré-definidos pelos fisioterapeutas responsáveis pela clínica. Essas avaliações estão divididas em áreas conforme o tratamento a ser realizado, por: ortopedia, hidroterapia, pneumologia, cardiologia, neurologia (...). A cada sessão são armazenadas as evoluções do paciente e a avaliação da sessão (...). Assim o tratamento pode ser avaliado e pode-se identificar se devem ocorrer mudanças no tratamento. Somente após a avaliação da sessão do paciente, é confirmada a sua saída da agenda diária do estagiário".

As evoluções dos pacientes são armazenadas a cada sessão o que possibilita que os estagiários obtenham informações dos tratamentos anteriormente realizados na clínica. O padrão (7), *Realizar Acompanhamento*, é responsável por esse tipo de controle, é utilizado sendo criadas as classes que apresentam (7) junto aos seus nomes na Figura 5.2.

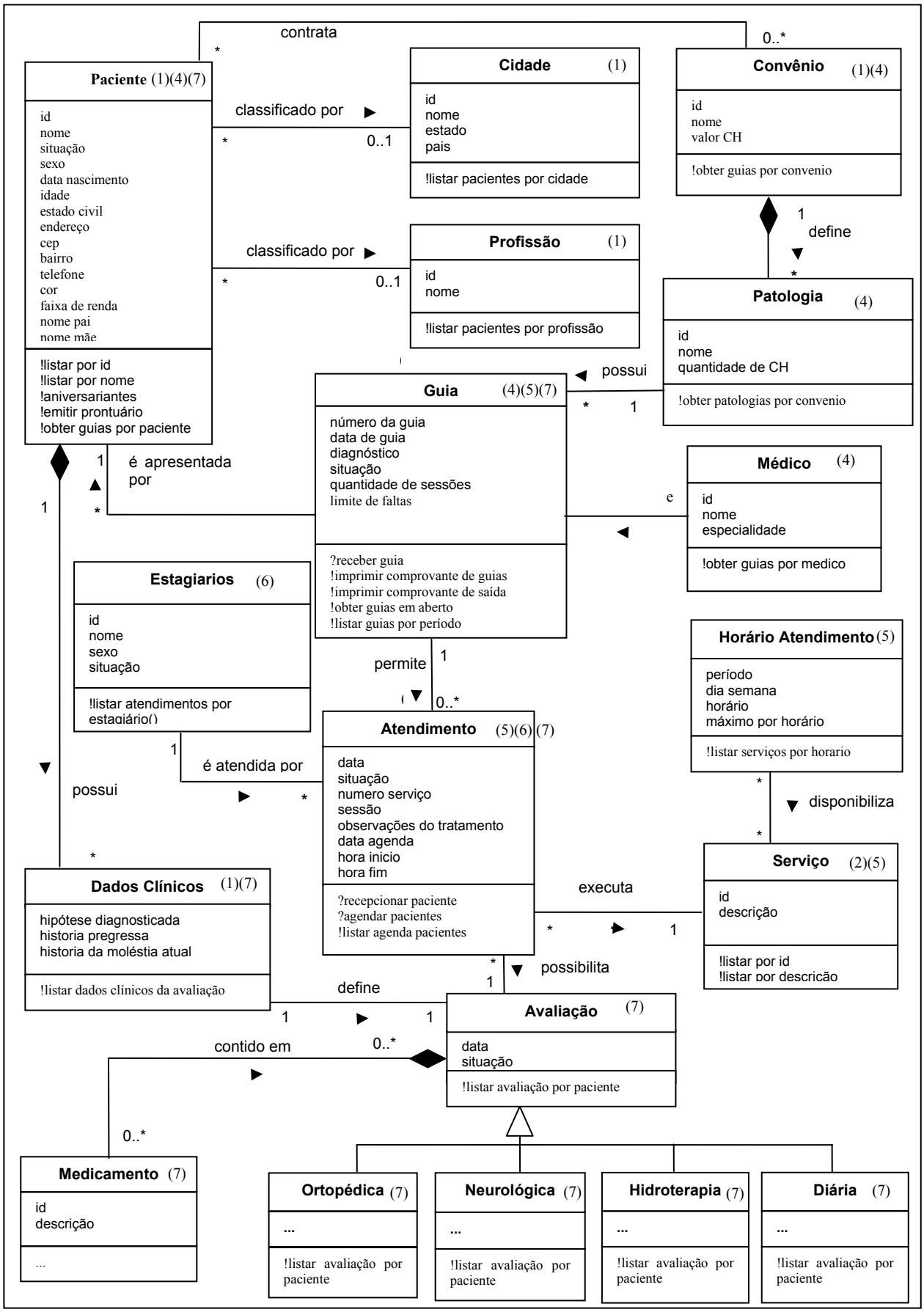


Figura 5.2 – Diagrama de classes da clínica de Fisioterapia

5.3. Gerando o Sistema Usando o GAwCRe

O desenvolvedor, por meio do diagrama de classes modelado com os padrões da SiGCli, pode começar a instanciação desse sistema usando o gerador GAwCRe, a partir da LMA definida com base na linguagem de padrões SiGCli. Assim, cabe ao desenvolvedor selecionar quais padrões são usados na aplicação, utilizando ou não o conjunto de variantes. A especificação da clínica de Fisioterapia usando a LMA é a exibida na Figura 5.3.

```
Identificar Pacientes;
Definir Serviços;
Processar Guias;
Agendar Atendimentos;
Identificar Pacientes;
Realizar Acompanhamentos
    com Avaliações Fisioterapia;
```

Figura 5.3 - Especificação da LMA para a Clínica de Fisioterapia

5.3.1. Instanciando a Aplicação

A primeira atividade a ser realizada para instanciar uma aplicação, usando o GAwCRe, é definir o seu nome e um diretório no sistema operacional onde os artefatos deverão ser armazenados. Esse nome é específico para cada aplicação identificando-a dentre as aplicações geradas pelo gerador. Dessa forma, é possível recuperar e alterar as especificações LMA para uma determinada aplicação, sempre que for necessário. O prosseguimento do processo só é possível após a definição da aplicação a ser gerada. A Figura 5.4 apresenta a interface de definição da aplicação da clínica de Fisioterapia (ClinicaFisio). no gerador.

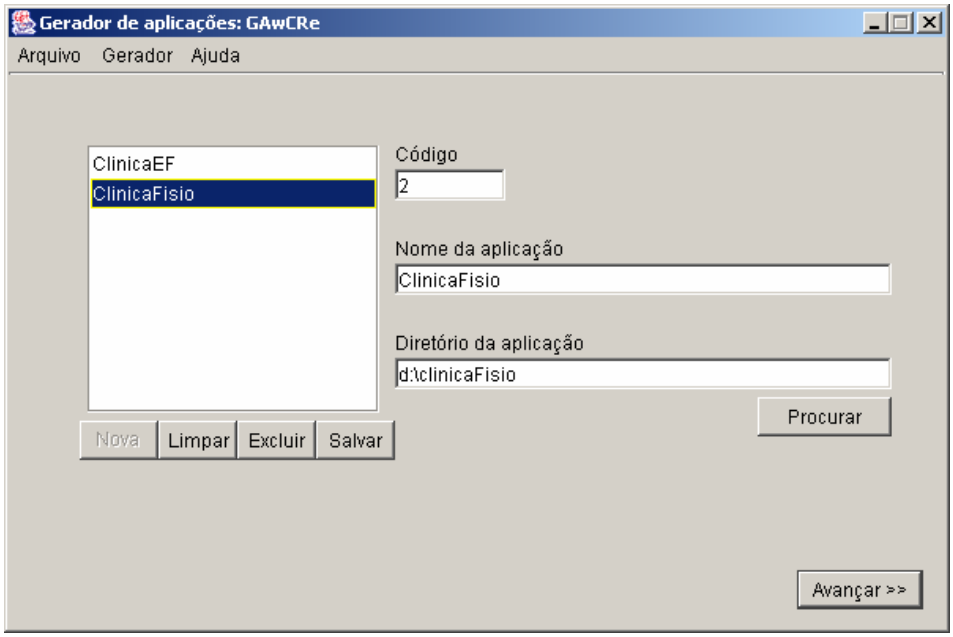


Figura 5.4 – Interface de definição de uma aplicação

A seguir, cada um dos padrões da SiGCli é apresentado no formato definido na LMA e o desenvolvedor seleciona quais padrões devem fazer parte do sistema. A Figura 5.5 apresenta a interface de instanciação para o padrão (2) da SiGCli, *Definir Serviços*. Conforme visto na seção anterior, apenas o padrão é aplicado, sem a sua variante.

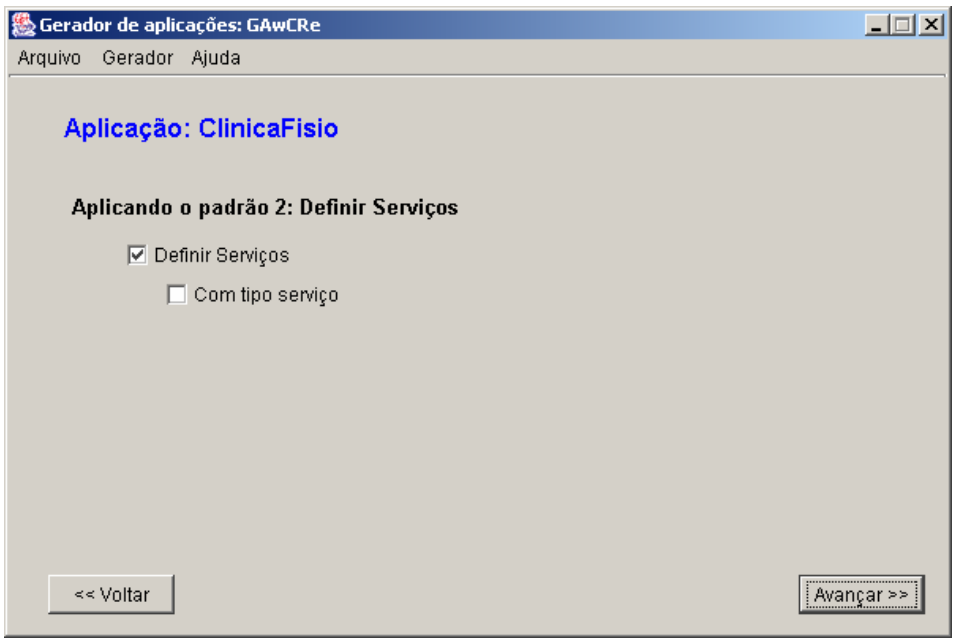


Figura 5.5 – Interface de especificação para o padrão (2) *Definir Serviços*

Com base na Figura 5.3, continua-se o processo de instanciação da aplicação de modo análogo ao da Figura 5.5. O GAwCRe armazena as informações referentes a LMA da aplicação em sua base de dados, podendo exibi-las a qualquer momento que o desenvolvedor

desejar. A Figura 5.6 exibe a interface que apresenta a relação dos padrões aplicados para a ClínicaFisio.

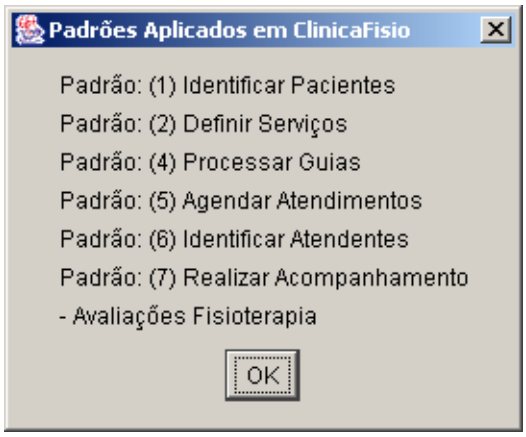


Figura 5.6 – Lista de padrões aplicados para a geração da ClínicaFisio

Após a especificação total ou parcial de uma aplicação, o GAwCRe permite a instanciação dos seus artefatos. Para isso, o menu **Gerador** deve ser ativado e uma das suas opções deve ser selecionada, Figura 5.7.

Figura 5.7 – Menu **Gerador**

A Figura 5.8 apresenta parte do documento XML, exibindo a especificação completa para o padrão (2) *Definir Serviços* com atributos, métodos, associações e variantes. O gerador lê essas informações, substituindo as *tags* dos gabaritos de criação (apresentados no Capítulo 4) pelos valores definidos no documento XML. O conteúdo da Figura 5.8 será utilizado para explicar como a aplicação é gerada com base nas informações definidas pelo desenvolvedor. Pode-se observar que existem mais informações que as necessárias para instanciar o padrão (2) na ClinicaFisio. Por exemplo, na Figura 5.8 aparecem referencias às variantes desse padrão que não são mostradas nos artefatos gerados.

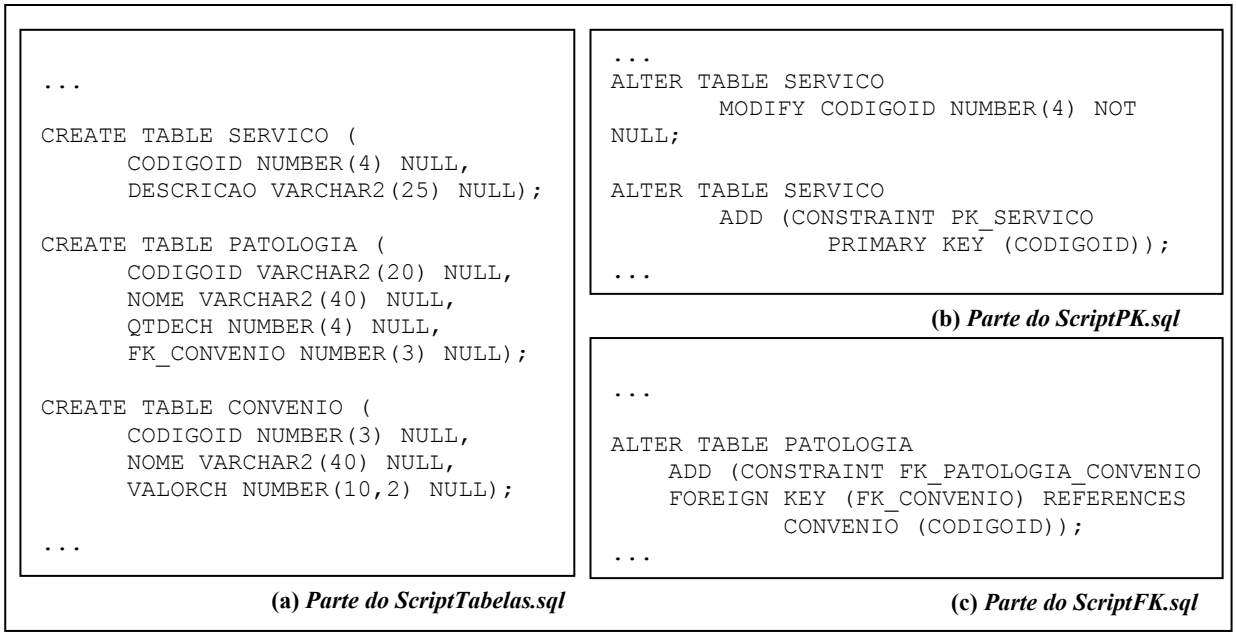
O menu **Gerar SQL**, Figura 5.7, aciona o módulo gerador de *scripts* SQL que gera a estrutura do banco de dados Oracle para a aplicação. Dessa forma, o gerador realiza as substituições das *tags* identificadas nos gabaritos de criação (seção 4.5.2) gerando os três artefatos apresentados na Figura 5.9. Ressalta-se que os tipos de dados apresentados nos *scripts* são diferentes dos definidos tipos definidos para os atributos no documento XML. Isso

acontece porque a definição usada no documento XML é baseada nos tipos de dados definidos para a linguagem Java. O módulo gerador de scripts SQL possui um método que faz a conversão dos tipos de dados Java para os tipos de dados usados pelo banco de dados Oracle.

```
...
<padrao numero="2" nome="Definir Serviços" variante="S">
   <classe nome="Serviço" menu="cadastro" tipoInterface="1">
     <atributo tipo="int" tamanho="4" tipoCampo="text" nome="Código" primaryKey="S"
               requerido="S">codigoId</atributo>
     <atributo tipo="String" tamanho="25" tipoCampo="lista" nome="Serviço"
               requerido="S">descricao</atributo>
      <atributo tipo="double" tamanho="10,2" tipoCampo="text" nome="Valor a ser pago"
               opcional="S" padraoId="3" varianteId="default">valor</atributo>
      <associacao ocorrenciaMax="1" classe="Tipo Serviço" opcional="S" padraoId="2"
                  varianteId="1">tpservico</associacao>
       <metodo tipo="public" tipoRetorno="String" menu="relatorio"
                nome="Relatório de Serviços">  listaServicos
           <corpoMetodo>...</corpoMetodo>
       </metodo>
    </classe>
    <variante numero="1" nome="Com tipo serviço">
       <classe nome="Tipo Serviço" opcional="S" menu="cadastro" tipoInterface="1">
         <atributo tipo="int" tamanho="3" tipoCampo="text" nome="Código" requerido="S"
                   primaryKey="S">codigoId</atributo>
         <atributo tipo="String" tamanho="25" tipoCampo="lista" nome="Tipo do Serviço"
                   requerido="S">descricao</atributo>
        <método ... > ... </metodo>
       </classe>
     </variante>
</padrao>
...
```

Figura 5.8 – Especificação em XML do Padrão (2) *Definir Serviços*

No *script* de criação da tabela `Serviço`, Figura 5.9 **(a),** não há o atributo de associação com a tabela `Tipo Servico`, previsto no documento XML pela *tag* `<associacao>` (Figura 5.8), pois a variante *Com tipo serviço* não é aplicada na especificação LMA da ClínicaFisio. Caso a variante, *Com Tipo Serviço,* tivesse sido usada os artefatos gerados seriam como os apresentados na Figura 5.10.



```
...

CREATE TABLE SERVICO (
      CODIGOID NUMBER(4) NULL,
      DESCRICAO VARCHAR2(25) NULL);

CREATE TABLE PATOLOGIA (
      CODIGOID VARCHAR2(20) NULL,
      NOME VARCHAR2(40) NULL,
      QTDECH NUMBER(4) NULL,
      FK_CONVENIO NUMBER(3) NULL);

CREATE TABLE CONVENIO (
      CODIGOID NUMBER(3) NULL,
      NOME VARCHAR2(40) NULL,
      VALORCH NUMBER(10,2) NULL);

...
```

(a) ***Parte do ScriptTabelas.sql***

```
...
ALTER TABLE SERVICO
       MODIFY CODIGOID NUMBER(4) NOT
NULL;

ALTER TABLE SERVICO
       ADD (CONSTRAINT PK_SERVICO
              PRIMARY KEY (CODIGOID));
...
```

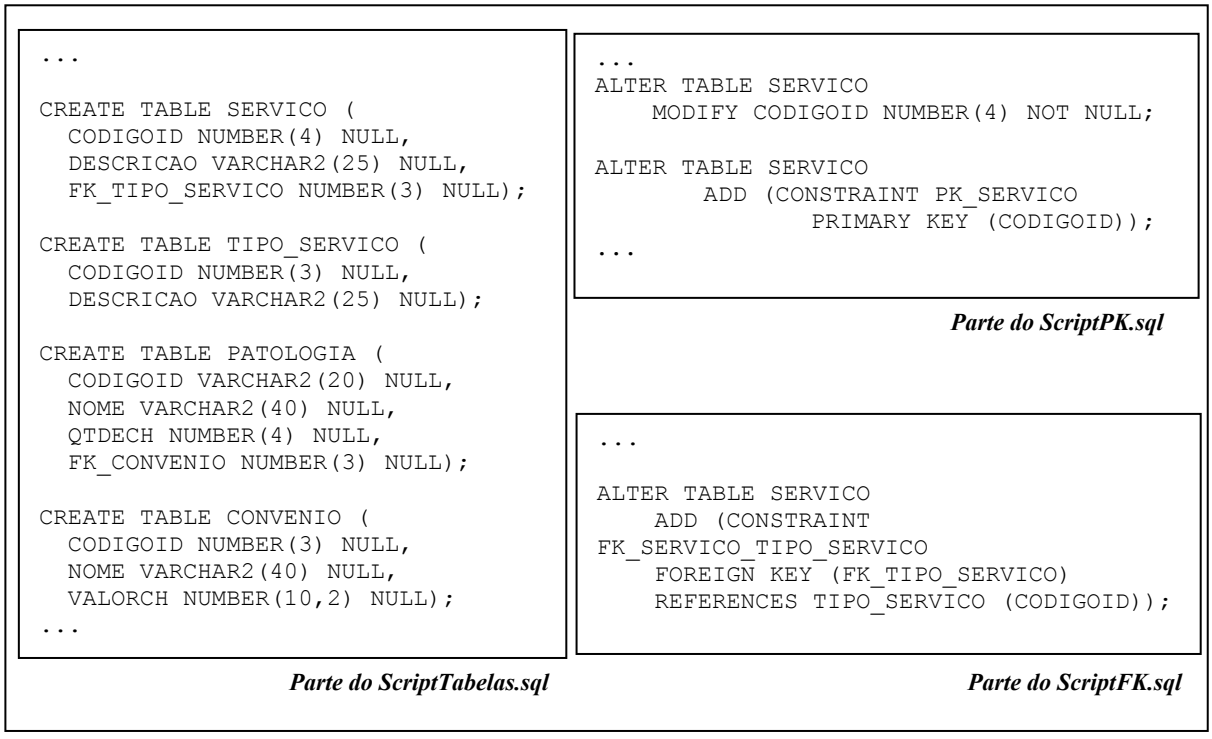
(b) ***Parte do ScriptPK.sql***

```
...

ALTER TABLE PATOLOGIA
    ADD (CONSTRAINT FK_PATOLOGIA_CONVENIO
    FOREIGN KEY (FK_CONVENIO) REFERENCES
            CONVENIO (CODIGOID));
...
```

(c) ***Parte do ScriptFK.sql***

Figura 5.9 –Parte dos artefatos gerados pelo Módulo Gerador de Scripts SQL



```
...

CREATE TABLE SERVICO (
  CODIGOID NUMBER(4) NULL,
  DESCRICAO VARCHAR2(25) NULL,
  FK_TIPO_SERVICO NUMBER(3) NULL);

CREATE TABLE TIPO_SERVICO (
  CODIGOID NUMBER(3) NULL,
  DESCRICAO VARCHAR2(25) NULL);

CREATE TABLE PATOLOGIA (
  CODIGOID VARCHAR2(20) NULL,
  NOME VARCHAR2(40) NULL,
  QTDECH NUMBER(4) NULL,
  FK_CONVENIO NUMBER(3) NULL);

CREATE TABLE CONVENIO (
  CODIGOID NUMBER(3) NULL,
  NOME VARCHAR2(40) NULL,
  VALORCH NUMBER(10,2) NULL);
...
```

Parte do ScriptTabelas.sql

```
...
ALTER TABLE SERVICO
    MODIFY CODIGOID NUMBER(4) NOT NULL;

ALTER TABLE SERVICO
       ADD (CONSTRAINT PK_SERVICO
              PRIMARY KEY (CODIGOID));
...
```

Parte do ScriptPK.sql

```
...

ALTER TABLE SERVICO
    ADD (CONSTRAINT
FK_SERVICO_TIPO_SERVICO
    FOREIGN KEY (FK_TIPO_SERVICO)
    REFERENCES TIPO_SERVICO (CODIGOID));
```

Parte do ScriptFK.sql

Figura 5.10 –Parte dos artefatos gerados pelo Módulo Gerador de *Scripts* SQL usando a variante *Com Tipo Serviço* tivesse sido selecionada

O menu **Gerar Código**, Figura 5.7, aciona o módulo gerador das classes Java (*Beans*) que têm as regras de negócios da aplicação. Dessa forma, o gerador realiza as substituições das *tags* identificadas nos gabaritos de criação (seção 4.5.3), gerando como

resultado somente as classes necessárias para satisfazer os requisitos do sistema que foram especificados na LMA. A Figura 5.11 apresenta parte da classe `Servico` gerada seguindo os requisitos especificados na LMA da ClínicaFisio.

```
package ClinicaFisio;
...
public class Servico implements PersistentObject {

// Objetos para a manipulação dos dados usando PersistenceLayer
      ...
// Atributos da classe
      private int codigoId;
      private String descricao;

//Associacoes entre as classe

// Método construtor
      public Servico(){
      ...
      }
//********** Métodos SET da classe **********/
      public synchronized void setcodigoId(int valor){
            this.codigoId = valor;
      }//**********

      public synchronized void setdescricao(String valor){
            this.descricao = valor;
      }//**********
//********** Métodos GET da classe **********/
      public int getcodigoId(){
            return this.codigoId;
      }//**********

      public String getdescricao(){
            return this.descricao;
      }//**********
//*********** Métodos para Persistencia e manipulação da base da dados *******
      public boolean save() {...}
      public boolean delete (){...}
      public ResultSet findall (){...}
      public ResultSet findlike (){...}
      public int getLast(){...}
      public int getCount() {...}
//*********** Métodos específicos da classe *******
      public String listaServicos() { ... }
}
```

Figura 5.11 – Parte da classe Java `Servico` gerada com base nas especificações da LMA para a clínica de Fisioterapia

Após as classes serem criadas elas devem ser compiladas e disponibilizadas no servidor Web onde a aplicação será hospedada. Esse servidor deve ser capaz de interpretar o código Java. TomCat(TomCat,2004), Oracle Application Server(OAS, 2004)(usado neste trabalho), WebSphere Application Server (IBM,2004) são exemplos de servidores com essas características.

O menu **Gerar Interface**, Figura 5.7, aciona o módulo gerador de interfaces JSP que cria as interfaces Web em que o usuário final interage com a aplicação. Dessa forma, o gerador realiza as substituições das *tags* identificadas nos gabaritos de criação (seção 4.5.4), gerando como resultado o menu da aplicação e os diferentes tipos de interfaces, dependendo da especificação LMA definida nesse caso. A Figura 5.12 apresenta o menu gerado para a ClínicaFisio, quando o item Serviço é selecionado.

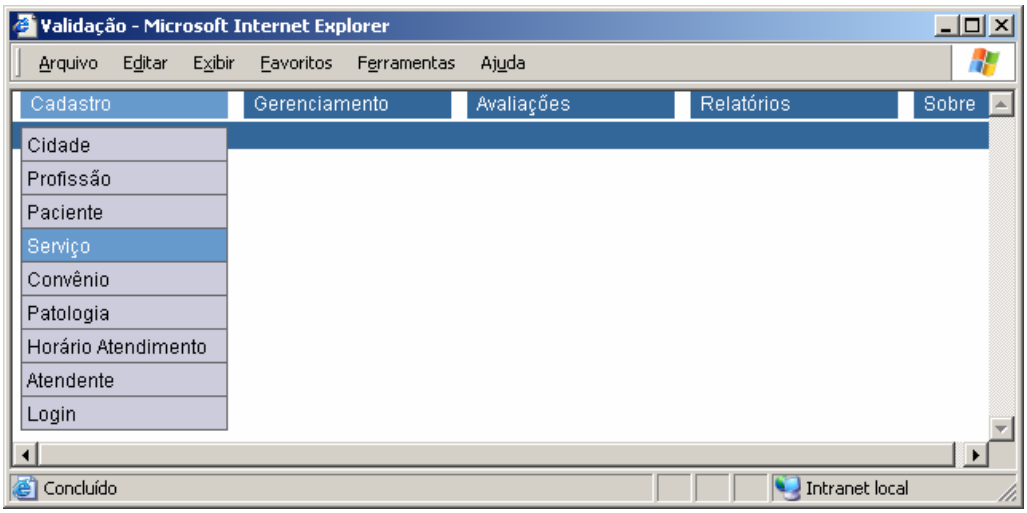


Figura 5.12 – Menu de gerado para a Clínica de Fisioterapia

A Figura 5.13 apresenta a interface para o Cadastro de Serviços oferecidos pela clínica, sendo classificada como interface simples pois, exibe uma lista de todos os serviços (objetos) cadastrados sem a necessidade de que algum dado adicional seja informado. A Figura 5.14 apresenta o Cadastro de Patologias de um convênio médico, sendo classificada como interface dependente da escolha de um objeto, pois a lista patologias (objetos) cadastradas só será exibida quando um convênio (objeto relacionado) for selecionado.

A maioria das interfaces de cadastro do sistema gerado utiliza um dos dois tipos apresentados nas Figuras 5.13 e 5.14. Entretanto, há situações em que as funcionalidades disponibilizadas por essas interfaces não são suficientes para satisfazer as necessidades da aplicação. Nesses casos, devem ser geradas interfaces com funcionalidades específicas, como a apresentada na Figura 5.15 para o Cadastro de Guias de convênios médicos que são apresentadas pelos pacientes. Essa interface é definida como sendo do tipo específica e o seu gabarito só é utilizado para gerar a interface para a classe `Guia`.

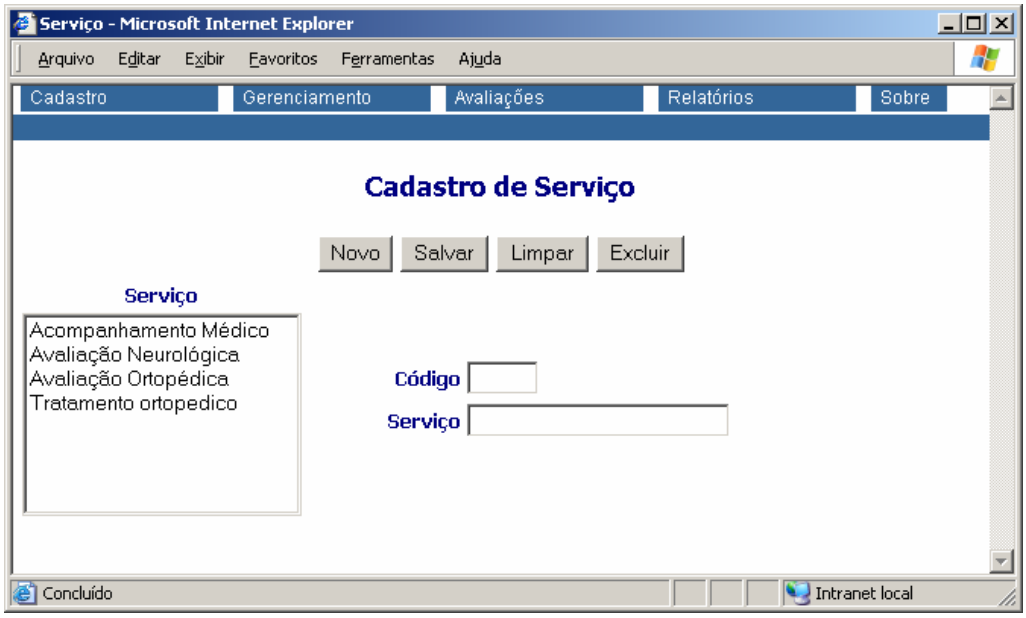


Figura 5.13 – Interface para o cadastro de serviços oferecidos pela clínica

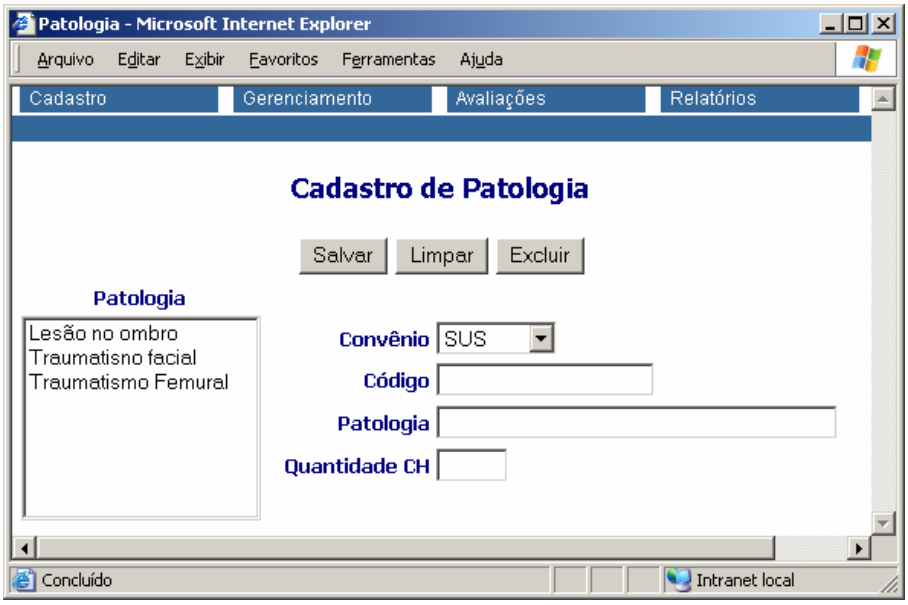


Figura 5.14 – Interface para o cadastro de patologias de um convênio médico

Alguns métodos existentes nas classes da aplicação são usados para que o usuário possa executar tarefas específicas no gerenciamento das clínicas. Para esses métodos são definidas interfaces com o usuário, por exemplo, a da Figura 5.16 que se refere à interface gerada para o método Recepcionar Pacientes da classe `Atendimentos`.

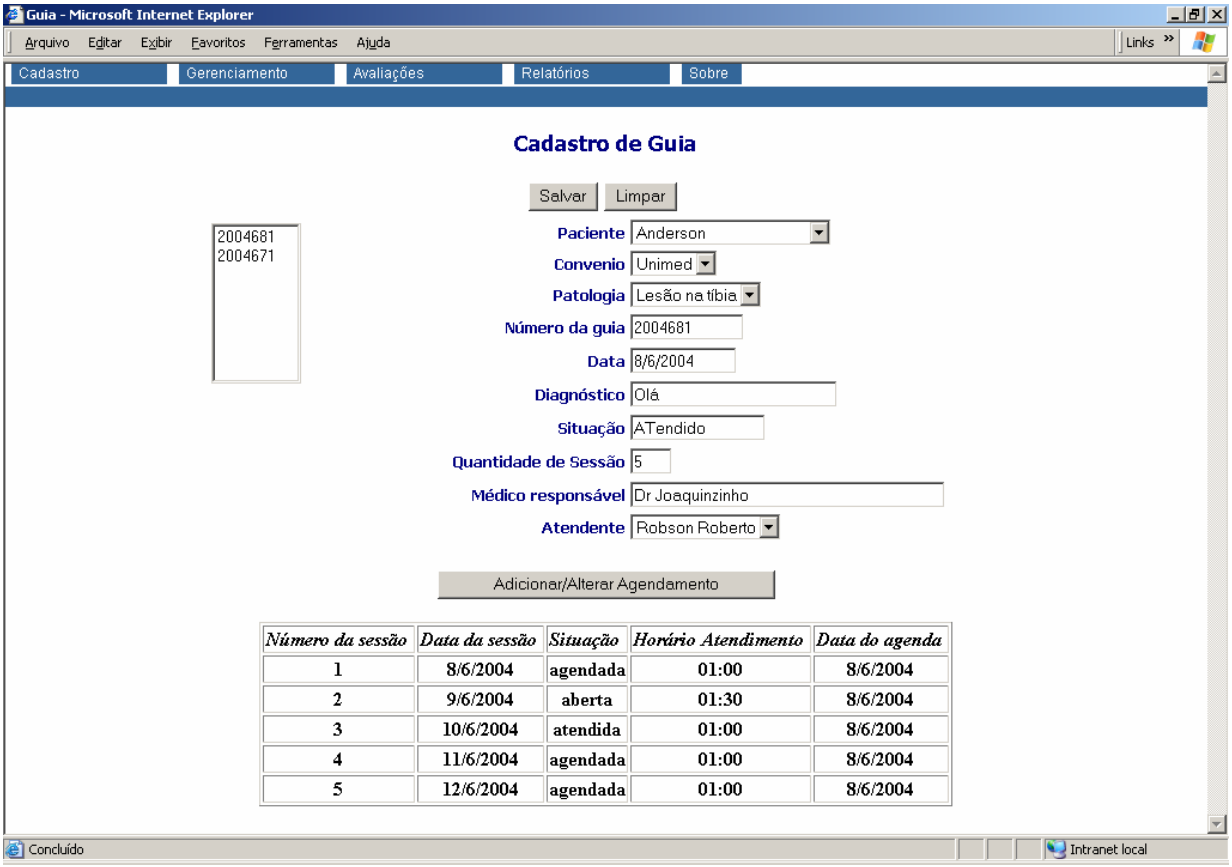


Figura 5.15– Interface para o cadastro de guias de convênios médicos apresentadas por um paciente

Figura 5.16 – Interface gerada para o método Recepcionar Paciente

As avaliações realizadas com os pacientes também possuem interfaces com funcionalidades específicas, entretanto essas são comuns entre as avaliações possibilitando a

definição de um gabarito específico para a criação das mesmas. A Figura 5.17 apresenta a interface correspondente à realização da Avaliação Diária do paciente.

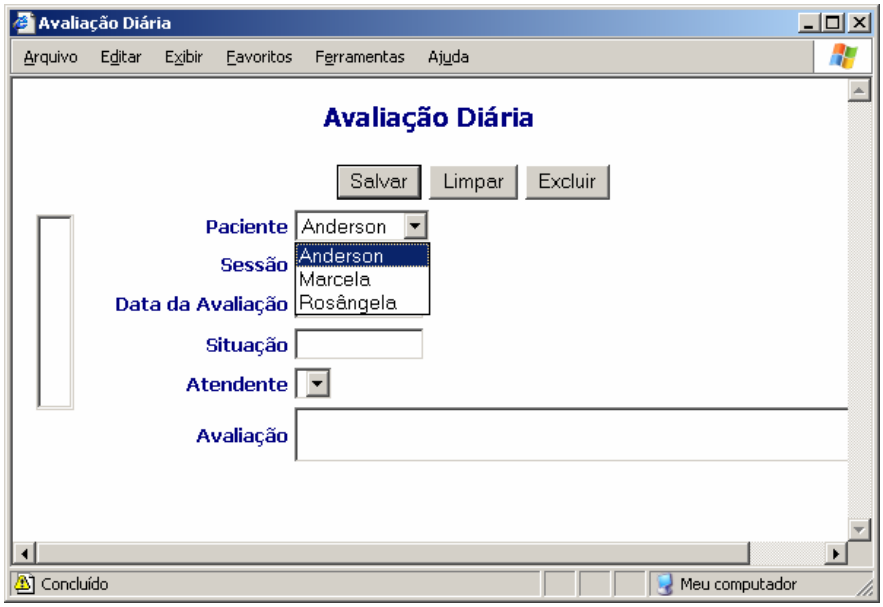


Figura 5.17 – Interface gerada para realizar a avaliação diária do paciente

5.4. Usando a Aplicação Gerada pelo Gerador

Após a elaboração desses artefatos são necessárias três atividades para que a aplicação possa funcionar corretamente:

1. Criar do banco de dados da aplicação, para isso os scripts SQL gerados devem ser executados na ferramenta *SQL Plus* da Oracle (2002);
2. Compilar as classes Java. Elas devem ser compiladas e os arquivos gerados, como resultado, devem ser hospedados no servidor de aplicações;
3. Copiar os arquivos JSP gerados para o servidor de aplicação.

Com essas atividades realizadas pode-se executar a aplicação, sendo a primeira interface exibida a de *login* da aplicação, como mostra a Figura 5.18. Após a validação do usuário é disponibilizado o menu da aplicação permitindo ao usuário realizar as atividades referentes ao gerenciamento da clínica.

Figura 5.18 – Interface de *login* para ao sistema ClínicaFisio

5.5. Considerações Finais

Este capítulo apresentou um exemplo, de forma detalhada, do funcionamento e do uso do gerador de aplicações GAwCRe para instanciar uma aplicação para o sistema de uma clínica de fisioterapia. A modelagem desse sistema, que foi realizada utilizando a linguagem de padrões SiGCli, apresenta diferenças em relação ao modelo obtido pelo processo de engenharia reversa (apresentado no Capítulo 3). A SiGCli permite uma modelagem totalmente orientada a objetos em nível de análise. Já o modelo obtido pelo processo de engenharia reversa produz uma modelagem em nível de implementação de acordo com o que existe no ambiente ZIM. Por exemplo, as classes Escolaridade, Faixa de Renda, Cor, etc do diagrama de classes da Figura 3.7, são tabelas com valores fixos sendo colocados como atributos multivalorados no diagrama de classes da Figura 5.2, e são gerados como uma lista de valores nas interfaces da aplicação quando o gerador GAwCRe é usado. Outras classes existentes na Figura 3.7 sofreram alteração em seus nomes, como por exemplo: EvoluçãoPaciente para Diária (Herdeira de Avalição), HistóricaPregressa foi incorporada a Dados Clínicos, etc. Alguns relacionamentos foram alterados por estarem em desacordo com a arquitetura OO, como por exemplo, os relacionamentos existentes entre as classes Guias, Patologias, Convênios da Figura 3.7.

Após a especificação, total ou parcial, da LMA para a aplicação no gerador, o desenvolvedor pode escolher, por meio de um menu, quais artefatos da aplicação que devem

ser gerados Essa característica possibilita a geração de protótipos que podem ser testados individualmente, sem necessariamente se criar toda a aplicação.

Uma outra característica importante do gerador é que ele permite que várias aplicações sejam especificadas usando a mesma LMA. As informações referentes a cada aplicação são armazenadas em uma base de dados, podendo ser recuperadas e alteradas sempre que for do desejo do desenvolvedor. Com isso, se algum requisito da aplicação for alterado ou se por algum motivo for adicionado um novo item para a LMA, não é necessário elaborar uma nova especificação LMA para a aplicação, bastando apenas recuperar a já existente fazendo as alterações necessárias.

Os artefatos gerados devem ser utilizados conforme a necessidade da aplicação. O gerador ainda não possui módulos capazes de automatizar o processo: de criação da base dados ou de compilação das classes Java, o que deve ser feito manualmente pelo desenvolvedor. As classes compiladas, bem como os arquivos gerados para a interface com o usuário devem ser hospedados em um servidor capaz de interpretar código Java.

Considerando-se apenas os padrões usados para a aplicação apresentada neste Capítulo (padrões (1), (2), (4), (5), (6), (7)), as diferentes combinações entre as variantes desses padrões possibilitam a instanciação de 48 aplicações usando os mesmos padrões. O Quadro 5.1 apresenta as variantes permitidas pelos padrões usados nessa aplicação e a Figura 5.19 apresenta as possíveis instanciações de aplicações com base nos padrões usados para gerar a aplicação ClinicaFisio, sendo os valores P(x) e V(x) são os que aparecem do Quadro 5.1.

Quadro 5.1 – Padrões usados na aplicação e suas variantes

Padrões		**Variações**	
P(1)	Identificar Paciente		-
P(2)	Definir Serviços	V(1)	Com tipo serviço
P(4)	Processar Guias		-
P(5)	Agendar Atendimento		-
P(6)	Identificar Atendentes *(opcional)*	V(1)	Com atributos atendentes
P(7)	Realizar Avaliação	V(1)	Avaliação Educação Física
		V(2)	Avaliação Fisioterapia
		V(3)	Avaliação Terapia Ocupacional

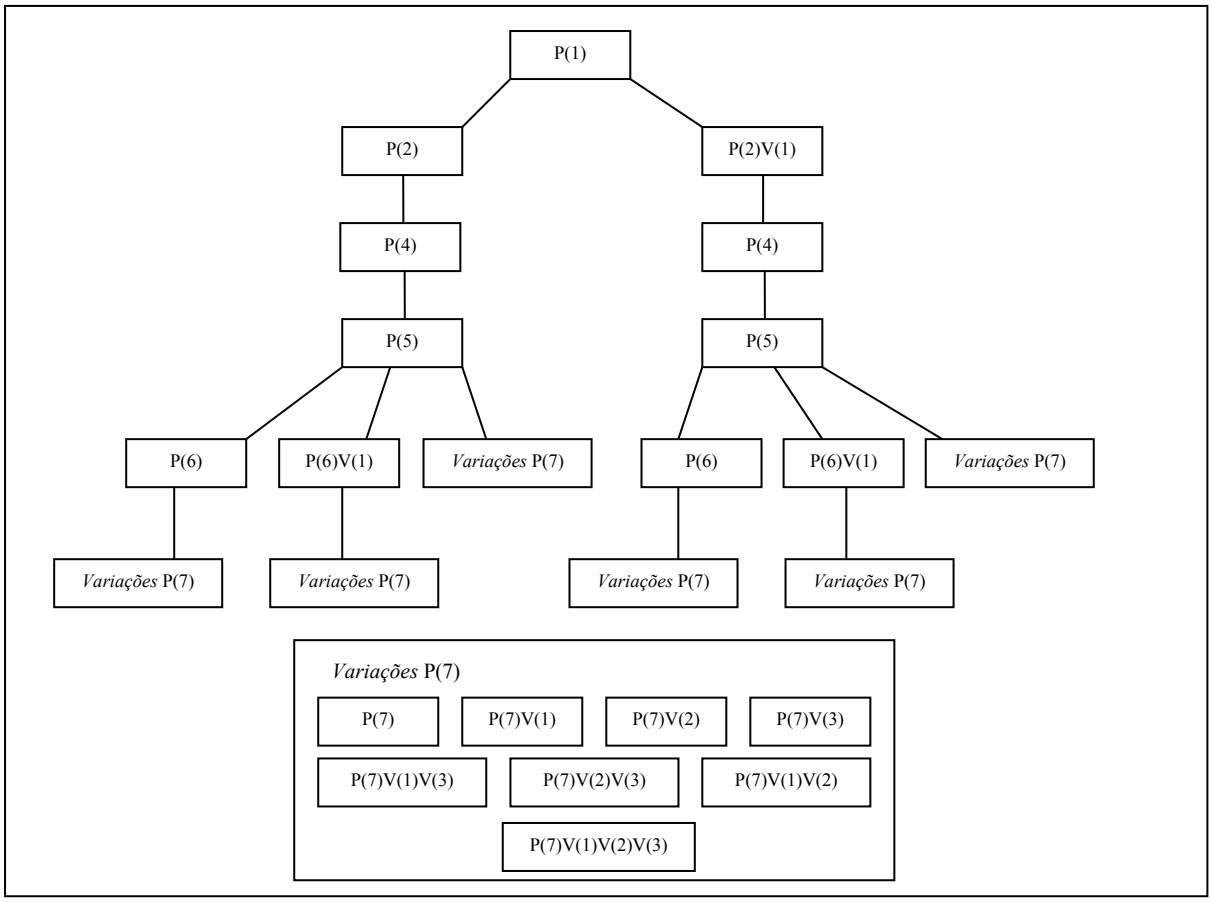


Figura 5.19 – Possíveis instanciações de aplicações com base nos padrões usados para gerar a aplicação ClinicaFisio

CAPÍTULO 6

Considerações Finais

6.1.Considerações Iniciais

A automatização do processo de desenvolvimento de software pode ser facilitada com o uso de geradores de aplicações, que são ferramentas capazes de obter especificações em alto nível e gerar produtos de software (artefatos) (Smaragdaski; Batory, 1998). Assim sendo, não somente a aplicação, mas todos os produtos intermediários gerados (documentação, *scripts* de criação de tabelas, código fonte) são considerados artefatos, independentemente de serem entregues ao usuário final.

O Gerador de Aplicações baseadas na Web para Clínicas de Reabilitação (GAwCRe), proposto neste trabalho, gera três artefatos distintos para a aplicação: os *scripts* de criação do banco de dados; o código fonte da aplicação escrito em linguagem Java; e as interfaces Web, definidas em JSP, para a interação com o usuário. Esses artefatos são gerados com base em uma especificação da aplicação, feita em alto nível, por meio de uma Linguagem de Modelagem da Aplicação (LMA), também aqui definida após um processo detalhado, que consistiu das seguintes etapas:

- Processo de engenharia reversa dos sistemas legados desenvolvidos em ZIM: auxiliou a fase da análise de domínio, pois por meio dele revitalizou-se as

informações pertinentes às aplicações já existentes para esse domínio. Dois apoios computacionais, `GeraJava` e `GeraSQL`, foram elaborados e diretrizes foram definidas para conduzir o processo de engenharia reversa de sistemas legados ZIM;

- Definição do modelo de domínio para clínicas de reabilitação: usando o processo proposto por Ré (2002), elaborou-se o modelo de domínio, analisando-se as similaridades e variabilidades entre os modelos obtidos com o processo de engenharia reversa. O modelo de domínio contém as principais funcionalidades referentes às aplicações do domínio de sistemas de gerenciamento de clínica de reabilitação (SiGCli);
- Elaboração da linguagem de padrões SiGCli: realizada com base no processo proposto por Braga (2003). Dessa forma, as classes do modelo de domínio foram agrupadas, segundo as suas funcionalidades, formando-se pseudo-padrões que foram analisados e comparados a padrões existentes na literatura, e que resultaram na linguagem de padrões SiGCli (Pazin et al., 2004). Essa linguagem de padrões sofreu aprimoramento em seus padrões após sua submissão a uma conferência PLoP*.
- Definição das diretrizes para a especificação de uma LMA com base em uma linguagem de padrões: permite mapear as variações existentes, durante a modelagem de diferentes aplicações, usando a linguagem de padrões. Um nome deve ser atribuído para cada variação identificada durante essa modelagem.

A LMA definida para o domínio SiGCli é apresentada no gerador, pela interface de instanciação das aplicações. Sendo a LMA baseada na linguagem de padrões SiGCli, é necessário o conhecimento do desenvolvedor para que utilize a melhor estrutura para gerar a aplicação satisfazendo os requisitos desejados pelo usuário. A especificação da LMA, apresentada pelo gerador, está contida em um documento XML definido por *tags* e atributos de *tags* que permitiram a sua documentação em alto nível bem como a da linguagem de padrões. Essas *tags* e atributos de *tags* são lidos pelo gerador, que possui gabaritos de códigos, para cada artefato da aplicação a ser gerado, com pontos de substituição para os

* *Pattern Language of Program* – Conferência sobre linguagens de padrões que ocorrem anualmente em diversos locais no mundo. A mais tradicional é a PLoP realizada nos Estados Unidos desde 1994, mas existem outras como a EuroPLoP (a partir de 1995), a ChiliPLoP (a partir de 1998) e a KoalaPLoP (a partir de 2000). No Brasil, já foram realizadas quatro edições da SugarLoafPLoP –"Conferência Latino-Americana em Linguagens de Padrões para Programação".

valores lidos do documento XML, definindo-se assim como os artefatos devem ser produzidos.

6.2. Resultados Obtidos

Antes de decidir pelo desenvolvimento de um gerador de aplicações para o domínio de gestão de clínicas de reabilitação foi realizado um estudo sobre a viabilidade de se utilizar o *framework* GREN para tal objetivo (Pazin et al., 2004a). Analisando a linguagem de padrões GRN, que é a base do *framework* GREN, para o domínio proposto, algumas funcionalidades importantes para a gestão de clínicas de reabilitação não são facilmente nele modeladas, por exemplo, o acompanhamento detalhado de um tratamento do paciente. Dessa forma, uma linguagem de padrões específica, SiGCli, foi elaborada utilizando alguns padrões da GRN sem alterações e adaptando outros para suprir as funcionalidades não cobertas.

Após essa elaboração foram realizados três estudos de caso com o objetivo de minimizar os esforços na instanciação das aplicações, utilizando o GREN-Wizard. Para isso, os três sistemas usados no processo de análise de domínio foram remodelados, seguindo os padrões propostos pela SiGCli e aplicados no GREN-Wizard. Dois fatores importantes que dificultaram a instanciação de aplicações foram observados:

a) A definição dos papéis que cada classe exerce nos padrões mapeados. Por exemplo, a classe *Paciente* assume dois papéis distintos durante a instanciação da aplicação, ora sendo o recurso ora sendo o destino da comercialização de um produto, outras classes também sofrem esse tipo de mutação. Tais mudanças de papéis não são cobertas pelo *framework* GREN, o que dificulta o entendimento do desenvolvedor para instanciar a aplicação;
b) As funcionalidades específicas do domínio da SiGCli que não são previstas pelo *framework* exigindo do desenvolvedor conhecimento profundo do *framework* GREN e da linguagem *SmallTalk* para implementá-los.

Considerando que os sistemas implementados em linguagem Java, com interface Web usando a arquitetura três camadas são os de interesse neste projeto, e *Smalltalk* uma linguagem bem difundida no meio científico, mas sem grande representatividade no mercado, a alteração do *framework* GREN exigiria muito esforço e os resultados obtidos não iriam satisfazer, por exemplo, as necessidades reais das aplicações existentes nas Faculdades

Salesianas de Lins. Uma solução seria a incorporação de um novo padrão à linguagem de padrões GRN, o padrão 7 da SiGCli, adicionado-o à estrutura do *framework* de modo que essa arquitetura não fosse violada. Dessa forma, alterações seriam necessárias nas camadas: *GREN-Negócio*, *GREN-Interface Gráfica com o Usuário* e *GREN-Wizard* do *framework* GREN.

Devido a não viabilidade da utilização do *framework* GREN com a linguagem de padrões SiGCli optou-se pelo desenvolvimento de um gerador de aplicações GAwCRe, elaborado após a análise de domínio de aplicações existentes utilizadas para o gerenciamento das clínicas do Centro de Reabilitação Física Dom Bosco, das Faculdades Salesianas de Lins-SP.

Na fase análise de domínio utilizou-se um processo de engenharia reversa desses sistemas. Embora o objetivo principal deste trabalho não seja esse, pelo processo de engenharia reversa foram definidos alguns passos importantes para revitalização de informações de sistemas legados desenvolvidos no ambiente ZIM. Dois apoios computacionais elaborados, `Gera Java` e `Gera SQL`, auxiliaram esse processo. O ambiente ZIM foi muito utilizado no início dos anos 90, mas devido a sua não evolução tecnológica, atualmente é uma ferramenta obsoleta e pouco encontrada em empresas. Mesmo assim, o processo de engenharia reversa torna-se importante para empresas que ainda fazem uso desse ambiente e pretendem migrar seus sistemas para uma tecnologia mais atual, por exemplo Oracle, como é o caso da Missão Salesiana de Mato Grosso, pois ele automatiza parte do processo de engenharia reversa desses sistemas (Pazin; Penteado, 2003).

O gerador possui em sua interface de instanciação uma LMA que representa todas as possíveis combinações para a especificação, em alto nível, de uma aplicação com base na linguagem de padrões SiGCli. Essas informações são armazenadas em um banco de dados sendo recuperadas a qualquer momento pelo gerador permitindo que se conheçam todos os padrões aplicados. Um protótipo pode ser gerado, sem que necessariamente todos os padrões referentes à aplicação tenham sido aplicados. Caso algum padrão ou variante tenha sido aplicado de forma incorreta é possível gerar uma nova aplicação, sem necessariamente gerar uma nova especificação, mas simplesmente alterando uma já existente (*backtracking*).

O gerador gera suas aplicações com base em gabaritos de códigos de artefatos definidos internamente a ele (em seu código fonte). Esses gabaritos possuem pontos variáveis que são substituídos, no momento da geração, pelos valores definidos no documento XML. Para que essa substituição aconteça de forma eficiente, o documento XML foi elaborado

visando detalhar ao máximo as informações úteis para gerar: as tabelas, as classes e as telas de interface com o usuário.

Conhecendo-se a estrutura do documento XML e as possíveis combinações de suas *tags* qualquer informação pode ser facilmente incorporada ao gerador, desde simples atributos de classe até padrões inteiros. No caso de incorporação de novas classes ou padrões a maior dificuldade encontrada é definir os gabaritos de código para gerar as interfaces específicas da aplicação, uma vez que esses gabaritos devem ser pré-escritos no código fonte do gerador. Outra dificuldade encontrada com relação à incorporação de novas classes e padrões ao documento XML do gerador diz respeito aos métodos específicos das classes, que antes de serem inseridos no documento XML devem ser previamente testados e estar livres de erros para que as compilações das classes Java ocorram sem maiores problemas.

Os dois problemas apresentados são passíveis de solução. No caso dos métodos, o desenvolvimento de uma aplicação convencional exigiria o mesmo esforço, usando o gerador esse esforço é otimizado, uma vez que o método poderá ser reutilizado para tantas outras aplicações do domínio. Já as interfaces específicas podem ser definidas externamente ao gerador sendo geradas ou adicionadas a uma aplicação quando as suas funcionalidades forem de interesse da aplicação.

A grande vantagem do uso da XML para armazenar as informações referentes a LMA e a linguagem de padrões é a sua flexibilidade, o que possibilita que outras linguagens de padrões sejam mapeadas para a estrutura do documento XML definida para o gerador, possibilitando assim o reuso do gerador em outros domínios.

O GAwCRe permite a instanciação de diferentes aplicações dentro de um mesmo domínio, considerando aplicações diferentes aquelas que possuem desde atributos até classes e/ou padrões distintos entre si.

A Figura 6.1 apresenta um esquema para representar as possíveis aplicações que podem ser geradas com esse gerador, considerando-se o grafo de fluxo de aplicação dos padrões da SiGCli (Figura 5.1) e a LMA. Essa figura é composta de duas partes: Principal e Detalhes. A parte Principal exibe a hierarquia de possíveis variações para a instanciação das aplicações. O total de aplicações que pode ser gerado pelo GAwCRe é definido pela equação:

Aplicações Geradas = ((8* ***Variações* B**) + (4* ***Variações* A**))

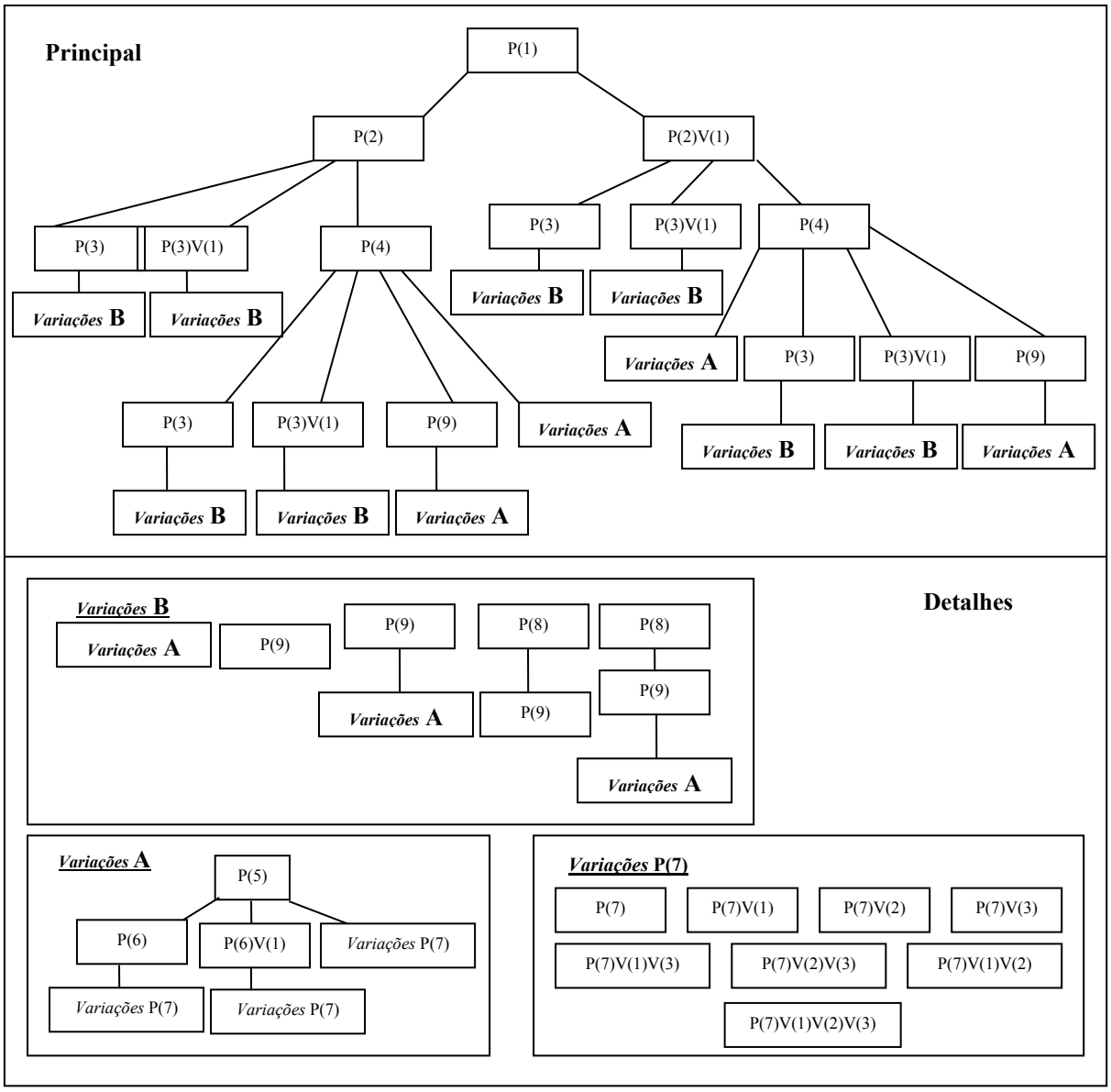


Figura 6.1 – Possibilidades de aplicações geradas pelo gerador GAwCRe

A parte Detalhes exibe os elementos ***Variações* A** , ***Variações* B** e ***Variações* P(7)** que são encontrados na parte Principal da Figura 6.1. Assim, ***Variações* A** pode ser representada pela equação:

$$\textbf{\textit{Variações}\ A} = (3 * \textbf{\textit{Variações}\ P(7)})$$

Sempre que o elemento ***Variações* P(7)** aparecer no esquema significa que oito (8) aplicações diferentes podem ser geradas, pois as variantes do padrão (7) (V(1) Avaliação Educação Física; V(2) Avaliação Fisioterapia; V(3) Avaliação Terapia Ocupacional) não são exclusivas, ou seja, podem ser aplicadas de oito formas diferentes.

O elemento ***Variações* B**, da parte Detalhes, pode ser representada pela equação:

Variações* B** = ((3Variações* A**)+2)

Assim sendo, o gerador é capaz de gerar 688 aplicações distintas como mostrado abaixo:

Aplicações Geradas = ((8* ***Variações* B**) + (4* ***Variações* A**))
Aplicações Geradas =((8* ((3***Variações* A**)+2)) + (4*(3***Variações* P(7)**)))
Aplicações Geradas =((8* ((3*(3***Variações* P(7)**))+2)) + (4*(3*8)))
Aplicações Geradas = ((8* ((3*(3*8))+2)) + 96)
Aplicações Geradas = ((8* 74) + 96) = 688

O Quadro 6.1 apresenta os padrões e as variantes disponíveis para gerador GAwCRe, que pode ser usado como legenda para a Figura 6.1.

Quadro 6.1 – Padrões e variantes disponíveis para o gerador GAwCRe

Padrões		**Variantes**	
P(1)	Identificar Pacientes	-	-
P(2)	Definir Serviços *(opcional)*	V(1)	Com tipo serviços
P(3)	Realizar Vendas *(opcional)*	V(1)	Com produtos
P(4)	Processar Guias	-	-
P(5)	Agendar Acompanhamentos	-	-
P(6)	Identificar Atendentes *(opcional)*	V(1)	Com atributo atendentes
P(7)	Realizar Acompanhamento	V(1)	Avaliação Educação Física
		V(2)	Avaliação Fisioterapia
		V(3)	Avaliação Terapia Ocupacional
P(8)	Realizar Compras *(opcional)*	-	-
P(9)	Controlar Faturamento *(opcional)*	-	-

6.3. Contribuições

As contribuições deste trabalho foram quanto à elaboração de um processo de engenharia reversa para sistemas legados ZIM. Esses apoios foram utilizados para a realização do processo de engenharia reversa e análise de domínio.

Com os resultados obtidos na análise de domínio procedeu-se a criação de uma linguagem de padrões para o domínio de clinicas de reabilitação (Pazin et al., 2004) denominada SiGCli. Essa linguagem de padrões possibilitou a definição de diretrizes para a criação de uma LMA. Essa LMA representa, por meio de regras gramaticais, as especificações, em alto nível, possíveis para definir aplicações do domínio de clínicas de reabilitação usando a SiGCli.

Um meta-modelo em XML para documentar linguagens de padrões e LMA's foi elaborado o que permite que outras linguagens de padrões possam ser utilizadas e não somente a SiGCli.

Finalmente, a contribuição para criação de um gerador de aplicações baseadas na Web para clínicas de reabilitação usando tecnologia Oracle.

6.4. Trabalhos Futuros

A seguir são comentados alguns trabalhos que podem dar continuidade a este.

- Considerando que as interfaces humano computador são vitais para a utilização de softwares em geral, técnicas propostas na literatura podem ser estudadas visando a melhoria das interfaces, tanto as de instanciação quanto às das aplicações geradas
- As aplicações geradas não possuem documentação específica, somente a do gerador é que existe. Nesse sentido, pode-se desenvolver um módulo capaz de documentar o gerador e as aplicações geradas. Esse módulo poderá fornecer informações quanto a constituição dos padrões utilizados.
- A escolha de XML para armazenar as informações referentes à SiGCli e à LMA ocorreu, com intuito de que outras linguagens de padrões, diferentes da SiGCli, possam ser utilizadas. Dessa forma, um estudo utilizando outras linguagens de padrões, usando as diretrizes para especificar a LMA, pode ser realizado para testar a reusabilidade do gerador, bem como o seu aperfeiçoamento;

- Atualmente, os gabaritos das interfaces estão armazenados no código fonte do gerador. Um estudo sobre a possibilidade de aprimorar as formas de armazenamento das interfaces da aplicação pode ser realizado, evitando, assim, que o desenvolvedor altere o código fonte do gerador.
- Os atributos existentes nas classes definidas pela linguagem de padrões pode não ser suficiente em determinadas aplicações. Dessa forma, o processo de geração das aplicações pode ser melhorado, permitindo a adição ou a remoção de atributos de classes para cada aplicação, sempre que necessário;
- Podem ser aplicadas técnicas para controle de qualidade dos produtos gerados (controle de versão,...);
- O processo de criação das tabelas e compilação das classes da aplicação é feito de forma manual pelo desenvolvedor. Pode-se estudar a possibilidade de automatizar esse processo ;
- Como esse gerador de aplicações será utilizado na prática e a tecnologia utilizada é da Oracle (2002), ele foi desenvolvido utilizando *parser* e banco de dados dessa plataforma. A utilização do GAwCRe com componentes gratuitos é uma evolução desejada para que o custo das aplicações geradas sejam compatível com a realidade da maioria das clínicas. Assim, tanto um *parser* XML quanto um gerenciador de banco de dados livres devem substituir os atuais.

Referências

APACHE, ***XML.APACHE.ORG***. Disponíveis em:<http://xml.apache.org>. Acesso em 23 nov. 2003.

BRAGA, R. T. V.; GERMANO, F. S. R.; MASIERO, P. C. ***A family of patterns for business resource management.*** In: *5th Annual Conference on Pattern Languages of Programs (PLOP'98)*, Washington University in St. Louis – Missouri, USA, on-Line. Disponível em http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions. Consultado em 31/01/2003., 1998.

BRAGA, R. T. V.; GERMANO, F. S. R.; MASIERO, P. C. ***A pattern language for business resource management***. In: *6th Pattern Languages of Programs Conference (PLoP'99)*, Monticello – IL, USA, 1999.

BRAGA, R. T. V.; MASIERO, P. C. ***A process for framework construction based on a pattern language.*** In: *26th Annual International Computer Software and Applications Conference (COMPSAC 2002)*, Oxford–England, 2002.

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.

BROWN, A. F. **Como desenvolver aplicações com o SGBD ZIM.** Rio de Janeiro: LTC – Livros Técnicos e Científicos Editora, 1990.

CAMARGO, V. V. **Reengenharia Orientada a Objetos de Sistemas COBOL com a Utilização de Padrões de Projeto e Servlets**. Dissertação de Mestrado – Programa de Pós Graduação em Ciência da Computação, Universidade Federal de São Carlos, São Carlos - SP, 2001.

CAGNIN M. I. **Avaliação das vantagens quanto à facilidade de manutenção e expansão de sistemas legados sujeito a engenharia reversa e a segmentação.** Dissertação de Mestrado – Programa de Pós Graduação em Ciência da Computação, Universidade Federal de São Carlos, São Carlos – SP. Trabalho premiado no XIII Concurso de Teses e Dissertações promovido pela SBC-2000, 1999.

CHIKOFSKY, J. E.; CROSS, J. H. - ***Reverse Engineering and Design Recovery: A Taxonomy*****,** IEEE Software, v. 7, n. 1, p. 13-17, Jan. 1990.

CLEAVELAND, J. C. – ***Program Generators with XML and JAVA***. Ed. Prentice-Hall PTR, 2001.

CZARNECKI, K. ; EISENECKER, U. W. – ***Generative Programming: Methods, Tools, and Applications*****.** Ed. Addison –Wesley, 2000.

CMM - ***Capability Maturity Model for Software*** – Disponível em: <http://www.sei.cmu.edu/cmm/cmm.html>. Acesso em: 10 fev. 2003.

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, S.; Ducasse, S.; Nierstrasz, O. - ***A Pattern Language for Reverse Engineering***, Proceedings, of the 5th European Conference on Pattern Languages of Programming and Computing, (EuroPLOP'2000), Andreas Ruping(Ed.), 2000.

DOM - ***W3C Document Object Model*** – Disponível em:< http://www.w3.org/DOM/>. Acesso em: 15 jan. 2004.

FAYAD, M. E.; E.JOHNSON, R.; SCHMDIT, D. C. ***Building application frameworks: Objectoriented foundations of framework design***. John Wiley & Sons, 1999.

FIELDS D. K.; KOLB M. A. **Desenvolvendo na Web com JavaServer Pages,** Ed. Ciência Moderna Ltda. Rio de Janeiro, 2000.

FOWLER, M. ***Analysis patterns - Reusable object models***, Addison- Wesley, 1997.

FRANCA, L. P. A. **Um Processo para a Construção de Geradores de Artefatos**. Tese de Doutorado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro – RJ, 2000.

FRANCA, 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, E., HELM, R., JOHNSON, R., VLISSIDES J. ***Design Patterns: Elements of Reusable Object-Oriented software.*** Addison –Wesley, 1995.

GIMENES, I. M. S.; TRAVASSOS, G. H. **O enfoque de linha de produto para desenvolvimento de software.** In: Anais do XXII Congresso da Sociedade Brasileira de Computação - Tutoriais, 2002.

GOODMAN, D. **Java Script – A Bíblia.** Ed. Campus, 2001.

HOTDRAW, ***HOTDRAW Home Page.*** Disponível em: <http://st-www.cs.uiuc.edu/users/brant/HotDraw/HotDraw.html>. Acesso em: 23 jan. 2003.

IBM, ***IBM AlphaWorks***. Disponível em:<http://www.alphaworks.ibm.com>. Acesso em: 9 nov. 2003.

IBM, ***WebSphere Application Server***. Disponível em: <http://www-306.ibm.com/software/info1/websphere/index.jsp>. Acesso em: 10 jan. 2004.

JAVA, ***JAVA Technology***. Disponível em: <http://wwws.sun.com/software/learnabout/java>. Acesso em: 23 nov. 2003.

JACOBSON, I.; LINDSTOM, F. – ***Re-engineering of Old Systems to na Object-Oriented Architecture*****,** In: OOPSLA'91. Proceedings. ACM, p. 340-350. 1991.

JARZABEK, S. - ***From reuse library experiences to application generation architectures***. Symposium on Software Reusability (SSR'95). ACM-SIGSOFT, 1995

JOHNSON, Ralph E.; FOOTE B. ***Designing Reusable Classes***. Journal of Object Oriented Programming – JOOP, 1(2):22-35, Junho/Julho 1998.

JOHNSON, R., WOOLF, B.. ***Type Object***. In "Martin, Robert C. (ed.); Riehle, Dirk (ed.) and Buschmann, Frank (ed.) Pattern Languages of Program Design 3", Addison-Wesley, pp. 47-65, 1998.

LEMOS, G. S. PRE/OO – **Um Processo de Reengenharia Orientado a Objetos com Ênfase na Garantia de Qualidade**. Dissertação de Mestrado – Programa de Pós Graduação em Ciência da Computação – Universidade Federal de São Carlos, São Carlos – SP, 2002.

MVC - ***Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)***. Disponível em:< http//st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html >. Acesso em: 10 set. 2002.

MVC – ***Understand MVC***. Disponível em: <http://otn.oracle.com/oramag/webcolumns/2003/techarticles/mills_mvc.html>. Acesso em: 10 mai. 2004.

MVC – **Arquitetura MVC**. Disponível em:<http://hercules.nce.ufrj.br/arq-mvc.html>. Acesso em: 10 mai.2004a.

OMNISPHERE, I. S. C. ***Omni-Sphere – open application generator.*** Disponível em: <http://www.omni-sphere.com/overview/overview.htm>. Acesso em: 12 fev. 2003 .

OAS – ***Oracle Application Server***. Disponível em:<http://www.oracle.com/appserver>. Acesso em: 10 jan. 2004.

ORACLE, ***Oracle Tecnology Network.*** Disponível em: < http://otn.oracle.com >. Acesso em: 10 dez. 2002.

ORACLE, ***XML Tecnology Center***. Disponível em: <http://otn.oracle.com/tech/xml/index.html>. Acesso em: 10 nov. 2003.

PAZIN, A; PENTEADO, R. A. D. **Proposta de Apoio Computacional Para Auxiliar o Processo de Engenharia Reversa em Sistemas Legados ZIM usando a FaPRE/OO.** Documento de Trabalho – Departamento de Computação - UFSCar, 2003.

PAZIN, A; PENTEADO, R. A. D. **Instanciação da Aplicação da Clínica de Fisioterapia usando o GAwCRe.** Documento de Trabalho – Departamento de Computação - UFSCar, 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 (*SugarLoafPlop*), Porto das Dunas – CE, Brasil, 2004.

PAZIN, A., RAMOS, R. A.; PENTEADO, R. A. D. **Estudo da Viabilidade de Utilização o Framework GREN para Instanciar Aplicações no Domínio de Clínicas de Reabilitação.**: 30ª Conferência Latino-Americana de Informática (*CLEI*), Arequipa – Peru, 2004a.

PENTEADO, R. A. D. **Um Método para Engenharia Reversa Orientada a Objetos.** Tese de Doutorado, Instituto de Física de São Carlos, Universidade de São Paulo - São Carlos. 1996.

PITTS-MOULTIS N.; KIRK,C. **XML BLACK BOX – Solução e Poder**, p. 627, Makrow Books do Brasil, 2000.

PL - ***Product Lines – Software Productivity Consortium.*** Disponível em: <http://www.software.org/pub/products/productlines.asp>. Acesso em: 23 nov. 2002.

RAMOS, R. A.; PAZIN, A.; PENTEADO, R. A. D. **Reengenharia de Sistemas Orientados a Objetos para Sistemas Orientados a Aspectos.**: 30ª Conferência Latino-Americana de Informática (*CLEI*), Arequipa – Peru, 2004.

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.

RÉ, R**. Um processo para construção de frameworks a partir da engenharia reversa de sistemas de informação baseados na Web**: Aplicação ao domínio dos leilões virtuais. Dissertação de Mestrado, ICMC/USP, São Carlos – SP, 2002.

RECCHIA, E. L. **Engenharia Reversa e Reengenharia Baseada em Padrões**. Dissertação de Mestrado – Programa de Pós Graduação em Ciência da Computação – Universidade Federal de São Carlos, São Carlos – SP, 2002.

RECCHIA, E. L.; PENTEADO, R. A. D. **FaPRE/OO: Uma Família de Padrões para Reengenharia Orientada a Objetos de Sistemas Legados Procedimentais**, IN: The Second Latin American Conference on Pattern Languages of Programming - SugarLoafPLoP, Itaipava-RJ v1 p 110-132, 2002.

ROBERTS, D.; JOHNSON, R. ***Evolving frameworks: A pattern language for developing object oriented frameworks*** in Martin, R.C., Riehle, D. , Buschmann, F. Pattern Languages of Program Design 3, Addison-Wesley, p. 471–486, 1998.

ROSE - ***Visual Modeling with Rational Rose Home***. Disponível em: <http://www.rational.com/products/rose/index.jsp>. Acesso em: 16 abr. 2003.

ROSS, D. ***Structured analysis: A language for communicating ideas.*** *IEEE Trans. Soft. Eng.*, v. 3, n. 1, 1977.

SMARAGDAKIS, Y.; BATORY, D. ***Application Generators***. Department of Computer Sciences. The University of Texas at Austin. Artigo retirado da página pessoal Disponível em:< http://www.cc.gatech.edu/~yannis>. Acesso em: 27 mar. 2003.

SPL - ***A Framework for Software Product Line***. Disponível em: <http://www.sei.cmu.edu/plp/framework.html>. Acesso em: 24 nov. 2002.

SUN, **XML**. Disponível em: <http://java.sun.com/xml>. Acesso em: 24 nov. 2003.

TOMCAT – ***The Jakarta Site - Apache TomCat***. Disponível em: <http://jakarta.apache.org/tomcat>. Acesso em 10 jan. 2004.

UML – ***Unified Modeling Language***. Disponível em: < http://www.uml.org>. Acesso em: 15 dez. 2002.

W3C – ***World Wide Web Consortium***. Disponível em: <http://www.w3c.org>. Acesso em: 19 mai. 2004.

WEISS, D., LAI, C. T. R. ***Software Product-Line Engineering: a family-based software development process***. Ed. Addison Wesley, 1999.

XML - ***eXtensible Markup Language.*** Disponível em: <http://www.w3.org/XML>. Acesso em: 19 mai. 2004.

YODER, J.W.; JOHNSON, R.E.; WILSON, Q.D**.** ***Connecting Business Objects to Relational Databases.*** In: Conference on the Pattern Languages of Programs, 5, Monticello-IL, EUA. Proceedings, 1998.

APÊNDICE I

A Linguagem de Padrões SiGCli

Uma Linguagem de Padrões para Gestão de Clínicas de Reabilitação

A linguagem de padrões SiGCli, cujo grafo é mostrado na Figura 1, é específica para o domínio de sistemas de gestão de clínicas de reabilitação, nas quais se incluem clínicas de fisioterapia, terapia ocupacional e educação física. No seu contexto, uma clínica preocupa-se em obter as informações sobre o paciente, acompanhando o seu tratamento por meio de avaliações contínuas para analisar a sua evolução.

A SiGCli é constituída de 9 padrões, mas em uma aplicação nem todos precisam ser aplicados. Os padrões *Identificar Pacientes*(1), e *Definir Serviços*(2), da Figura 1, devem ser aplicados obrigatoriamente em todos os sistemas desse domínio. Em seguida pode-se optar por *Processar Guias*(4) ou *Realizar Vendas*(3), que levarão ao padrão *Agendar Atendimentos*(5). Esse padrão, por sua vez, leva a outros dois padrões, *Identificar Atendente*(6) e/ou *Realizar Acompanhamento*(7). Tanto o padrão 4 quanto o padrão 3 podem levar ao padrão *Controlar Faturamento*(9), sendo que o padrão 3 ainda pode ter um padrão intermediário ao 9, que é o padrão *Realizar Compras*(8). Embora o grafo de fluxo de aplicação dos padrões 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.

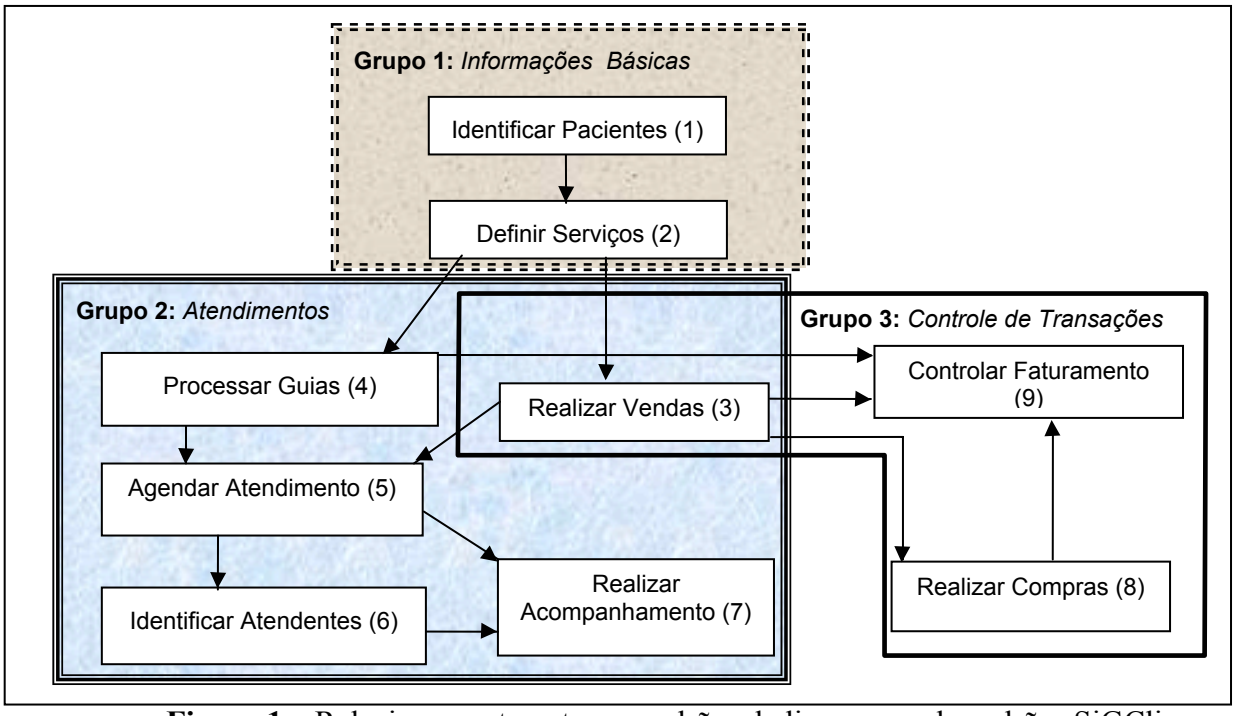


Figura 1 – Relacionamento entre os padrões da linguagem de padrões SiGCli.

Os padrões da Figura 1 estão agrupados em três grupos, conforme as sua funcionalidades. O Grupo 1 trata das informações básicas relacionadas às clínicas, a identificação dos pacientes e dos serviços prestados. O Grupo 2 cuida do gerenciamento dos atendimentos, sendo possível acompanhar todas as fases do tratamento de um paciente. O Grupo 3 cuida do controle financeiro das clínicas. O Quadro 1 apresenta os padrões da SiGCli descrevendo suas finalidades.

Quadro 1. Descrição dos padrões da SiGCli

Nº	Padrão	Descrição
01	Identificar Paciente	Define as informações relacionadas aos pacientes que procuram uma clínica com o objetivo de realizar o tratamento de alguma lesão.
02	Definir Serviços	Define as informações sobre os serviços prestados pela clínica aos pacientes.
03	Realizar Vendas	Controla a venda de produtos e serviços da clínica aos pacientes, quando existir.
04	Processar Guias	Permite agendar atendimentos por meio de convênios médicos, que são responsáveis pelo pagamento dos tratamentos.
05	Agendar Atendimentos	Controla os horários das sessões dos pacientes e define uma agenda de atendimentos. Essa agenda pode ser gerada por uma venda de serviços ou pelo recebimento de guias de convênios médicos
06	Identificar Atendentes	Identifica quais atendentes são responsáveis pelo tratamento de um paciente.
07	Realizar Acompanhamento	Permite acompanhar o tratamento dos pacientes por meio de avaliações realizadas a cada sessão.
08	Realizar Compras	Gerencia a compra de produtos que são comercializados por uma clínica
09	Controlar Faturamento	Gerencia os valores recebidos pela da clínica, pelos serviços prestados

Os padrões da SiGCli, aqui apresentados, são padrões de análise e seguem o seguinte formato: Intuito, Contexto, Problema, Influências, Estrutura, Participantes, Exemplo, Dinâmica, Variantes, Usos Conhecidos Padrões Relacionados e Próximos Padrões. A estrutura desses padrões foi parcialmente baseada nos modelos GoF[12] e POSA[13], com alguns itens adicionais como por exemplo: Participantes, Exemplo, Variantes e Próximos Padrões. Esses itens possibilitam descrever a interação entre os padrões, caracterizando a linguagem de padrões.

Cada padrão corresponde a um ou mais casos de uso do domínio. A Figura 2 apresenta alguns casos de uso mais relevantes, indicando a qual padrão ele está associado. Casos de uso simples, como inserção e manutenção de objetos, não são apresentados.

[12] Gamma, E., Helm, R., Johnson, R., Vlissides J. ***Design Patterns: Elements of Reusable Object-Oriented software***. Addison –Wesley, 1995

[13] Buschmann, F.; Meunier R.; Rohnert, H.; Sommerlad, P.; Stal, M. **Pattern - Oriented Software Architecture - A System of Patterns**, John Wiley & Sons, 1996

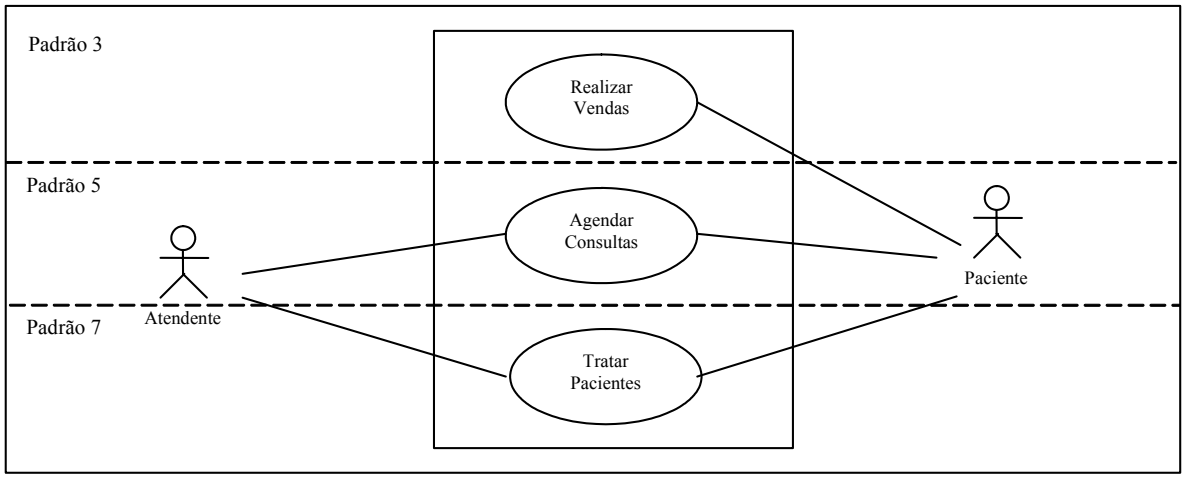


Figura 2 – Diagrama de casos de uso das principais atividades de uma clínica de reabilitação

Padrão 1: Identificar Pacientes

Intuito

Obter as informações referentes aos pacientes que procuram a clínica para poder identificá-los.

Contexto

Cada clínica tem que identificar os pacientes que vêm obter tratamentos específicos.

Problema

Como representar as informações que devem ser armazenadas sobre um paciente?

Influências

- Pacientes possuem características gerais comuns. Armazenar informações pessoais sobre cada paciente é de suma importância para o gerenciamento de uma clínica.
- A definição de possíveis valores para alguns atributos pode auxiliar a geração de relatórios estatísticos. Por exemplo, determinadas patologias ocorrem com mais freqüência em pessoas que exercem determinadas profissões ou em certas idades.
- Alguns tipos de tratamentos podem ser influenciados pela faixa etária ou cor da pele do paciente. Por exemplo, em uma clínica de fisioterapia um tratamento com laser não é recomendado a pessoas negras. Armazenar tais informações pode auxiliar em futuros diagnósticos e tratamentos.
- Algumas informações adicionais sobre o paciente podem ser necessárias, para permitir um melhor gerenciamento do sistema, como cidade onde mora, profissão que exerce etc.

Estrutura

A Figura 3 mostra o diagrama de classes para o padrão *Identificar Pacientes*.

Participantes

Paciente: representa a pessoa que freqüenta a clínica, podendo fazer uso dos serviços e produtos oferecidos por ela. O atributo id é utilizado para identificar um paciente de forma única pelo sistema, os demais atributos servem para definir as principais

características de um paciente. Pode-se adicionar outros atributos conforme a necessidade de uma aplicação particular.

Classificador: corresponde ao conjunto de dados que classificarão um paciente. Por exemplo, a cidade onde um paciente mora; sua profissão, etc.

Convênio: refere-se à empresa que realiza o pagamento dos serviços prestados pelas clínicas aos pacientes.

Dados Clínicos: são as informações sobre a lesão atual do paciente.

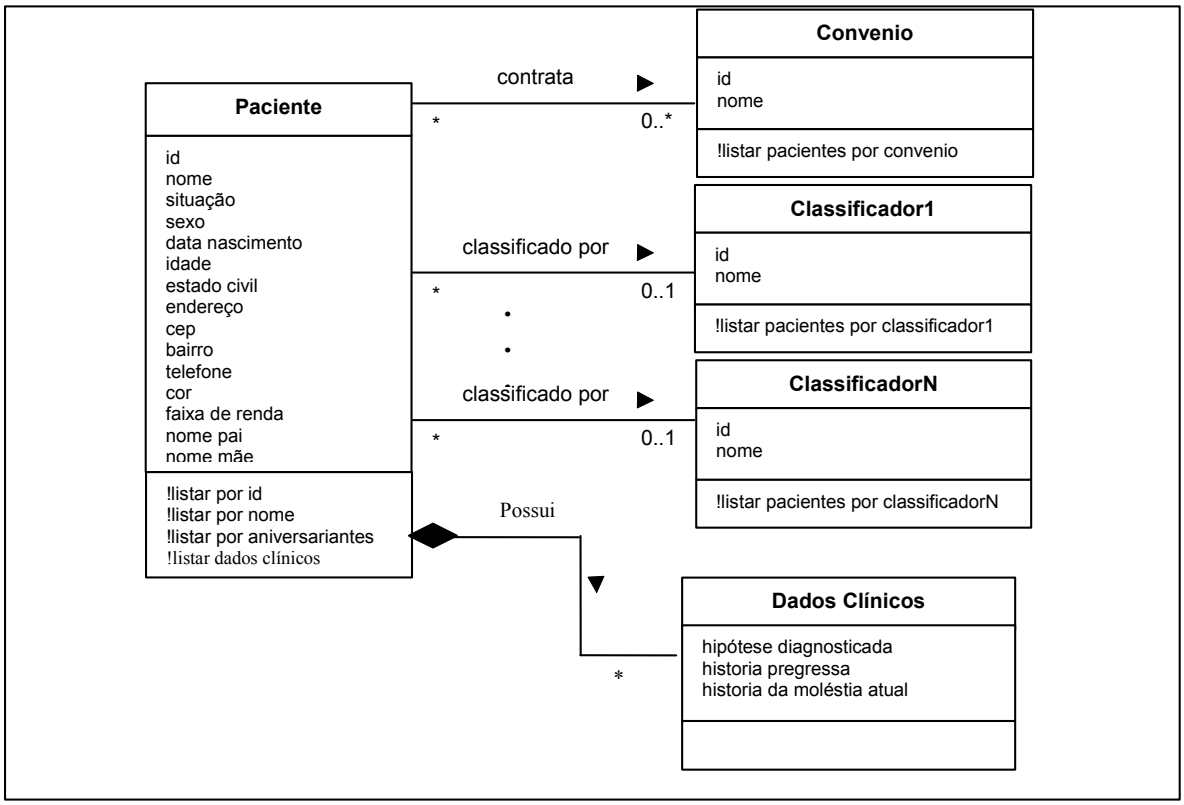


Figura 3 – Diagrama de classes para o padrão *Identificar Pacientes*.

Exemplos

A Figura 4 representa a instanciação do padrão *Identificar Pacientes* para uma clínica de Fisioterapia.

Variantes

Alguns atributos podem assumir valores pré-existentes em um Classificador. Por exemplo, pode-se definir um conjunto de valores para o atributo cidades. Deve-se fazer uma análise desses atributos para definir quais deles podem ser usados como um Classificador. Nessas condições deve-se fazer uso de um tipo de agrupamento definido por Jonhson[14] como objetos com multiplos tipos (*multiple type objects*). Assim, para cada atributo com múltiplos valores cria-se uma nova classe.

[14] Johnson, R. and Woolf, B. ***Type Object***. In "Martin, Robert C. (ed.); Riehle, Dirk (ed.) and Buschmann, Frank (ed.) Pattern Languages of Program Design 3", Addison-Wesley, pp. 47-65, 1998.

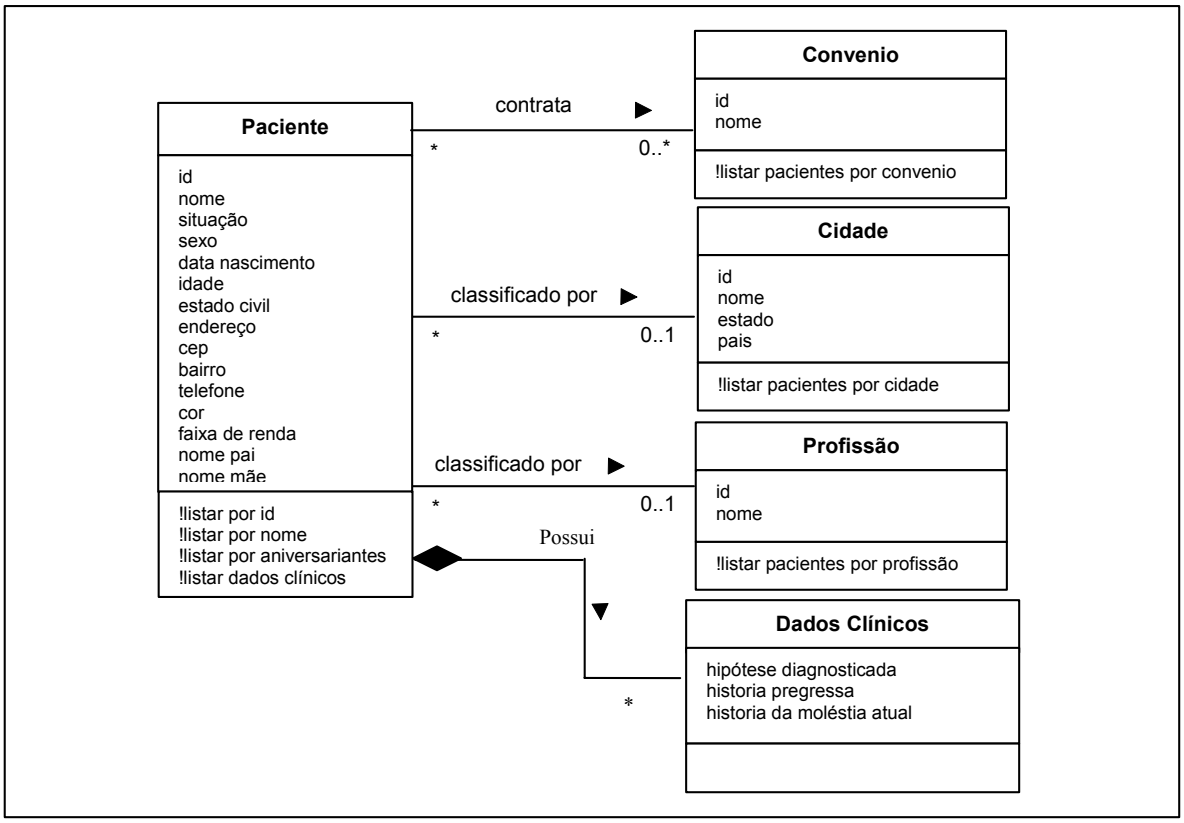


Figura 4 – Instanciação do padrão *Identificar Pacientes* para uma clínica de Fisioterapia.

Usos conhecidos

Qualquer clínica que necessite obter e armazenar informações sobre os pacientes a serem tratados, como é o caso das clínicas de fisioterapia e terapia ocupacional do Centro de Reabilitação Física Dom Bosco, Lins - SP.

Padrões Relacionados

Identificar o Recurso (P1), Quantificar o Recurso (P2) da GRN[15], Type Object[3] e Observation[16].

Próximos padrões

Depois de *Identificar Pacientes(1)* deve-se definir quais são os serviços que devem ser realizados pela clínica, para isso o padrão *Definir Serviços(2)* deve ser aplicado.

Padrão 2: Definir Serviços

Intuito

Obter informações sobre os serviços que serão prestados pela clínica.

Contexto

Clínicas que oferecem serviços para a realização de um tipo de tratamento específico necessário para a reabilitação do paciente.

---

[15] Braga, R.T.V.; Germano, F.S.R. e Masiero, P.C. ***A Pattern Language for Business Resource Management***, Proceedings of the 6th Annual Conference on Pattern Languages of Programs (PLOP'99), Monticello, Illinois, EUA, v7, p. 1-33, agosto de 1999

[16] Fowler, M. ***Analysis patterns -- Reusable object models***, Addison- Wesley, 1997

Problema

Como definir os serviços que serão prestados aos pacientes de uma clínica?

Influência

- O paciente deve saber quais os serviços (tratamentos) oferecidos pela clínica.
- Os serviços podem ser classificados por um determinado tipo: uma avaliação, um tratamento específico, etc.

Estrutura

A Figura 5 mostra o diagrama de classes do padrão *Definir Serviços*.

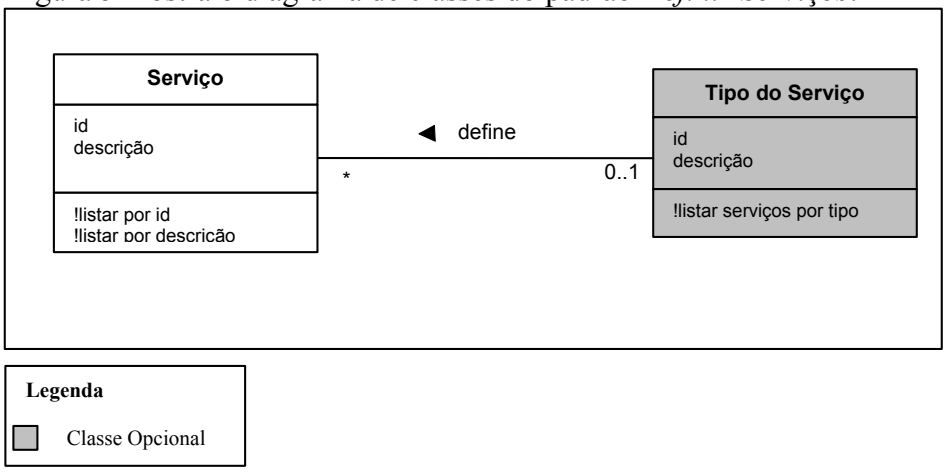


Figura 5 – Diagrama de classes para o padrão *Definir Serviços*.

Participantes

Serviço: é qualquer atividade que possa ser realizada com um paciente para acompanhar a sua evolução durante o tratamento

Tipo do Serviço: é a classificação de um serviço. Por exemplo: Avaliação, Atividade Física, Atividade na Piscina, etc.

Exemplo

A Figura 5 já representa a instanciação para o padrão *Definir Serviços*, para uma clínica de fisioterapia, pois as classes da aplicação recebem os mesmos nomes definidos para o padrão.

Variantes

Nem sempre é necessário definir o tipo de serviço para a clínica. Assim sendo, a utilização da classe Tipo do Serviço torna-se opcional para uma aplicação.

Usos Conhecidos

Qualquer empresa que realize prestação de serviços e precisa identificar quais serviços são oferecidos, como é o caso das clínicas de fisioterapia e terapia ocupacional do Centro de Reabilitação Física Dom Bosco, Lins - SP.

Padrões Relacionados

Identificar o Recurso (P1), Quantificar o Recurso (P2) da GRN[4] e Type object[3].

Próximos padrões

Se a clínica permite atendimento por convênios médico, aplique o padrão *Processar Guias (4)*. Se não, aplique o padrão *Realizar Vendas (3)*. Vale salientar que uma clínica pode realizar o atendimento tanto por convênios médicos quanto por venda de serviços, e nesse caso deve-se aplicar os dois padrões.

Padrão 3: Realizar Vendas

Intuito

Registrar e controlar as vendas de produtos e serviços da clínica para seus pacientes.

Contexto

Clínicas que comercializam produtos e serviços. Os serviços são atividades realizadas durante os tratamentos dos pacientes. Os produtos são quaisquer bens que sofrem uma troca de propriedade, ou seja, pertenciam à clínica e passam a pertencer aos pacientes que os adquiriram.

Problema

Como gerenciar a venda de produtos e serviços oferecidos pela clínica aos seus pacientes?

Influências

- A venda de produtos e serviços é uma atividade que armazena informações sobre a comercialização. Por meio dela pode-se chegar a relatórios importantes sobre o funcionamento da clínica.
- Quando a venda trata de serviços, deve-se verificar se a clínica oferece um determinado serviço para um futuro agendamento. Quando a venda trata de produtos deve-se verificar a existência de estoque.
- Nem sempre a clínica faz comércio de produtos.

Estrutura

A Figura 6 mostra o diagrama de classes para o padrão *Realizar Vendas.*

Participantes

Venda: apresenta os detalhes envolvidos na comercialização de produtos e/ou serviços. Cada comercialização recebe um número de identificação (número da venda).

Item Vendido: armazena os itens comercializados, com informações que permitirão recuperar o histórico das comercializações.

Serviço: conforme já definido no padrão 2, acrescentando-se o atributo valor do serviço.

Produto: armazena as informações dos produtos comercializados pela clínica. Deve-se armazenar a quantidade existente em estoque e o nível de ressuprimento, para que o sistema decida quando um produto deve ser comprado.

Unidade de Medida: representa as possíveis unidades de medida para um produto a ser armazenado. Por exemplo: quilograma, metro, pacotes etc.

Paciente: conforme já definido no padrão 1, acrescentando-se a operação para obter as vendas por paciente.

Exemplo

A Figura 6 representa a instanciação para o padrão *Realizar Vendas*, pois as classes da aplicação recebem os mesmos nomes definidos para o padrão. Nesse caso a clínica trata tanto da comercialização de produtos quanto da de serviços. Nos casos em que uma clínica só trata de prestação de serviços, ou seja, comercializa apenas serviços, as Classes Opcionais não devem ser utilizadas.

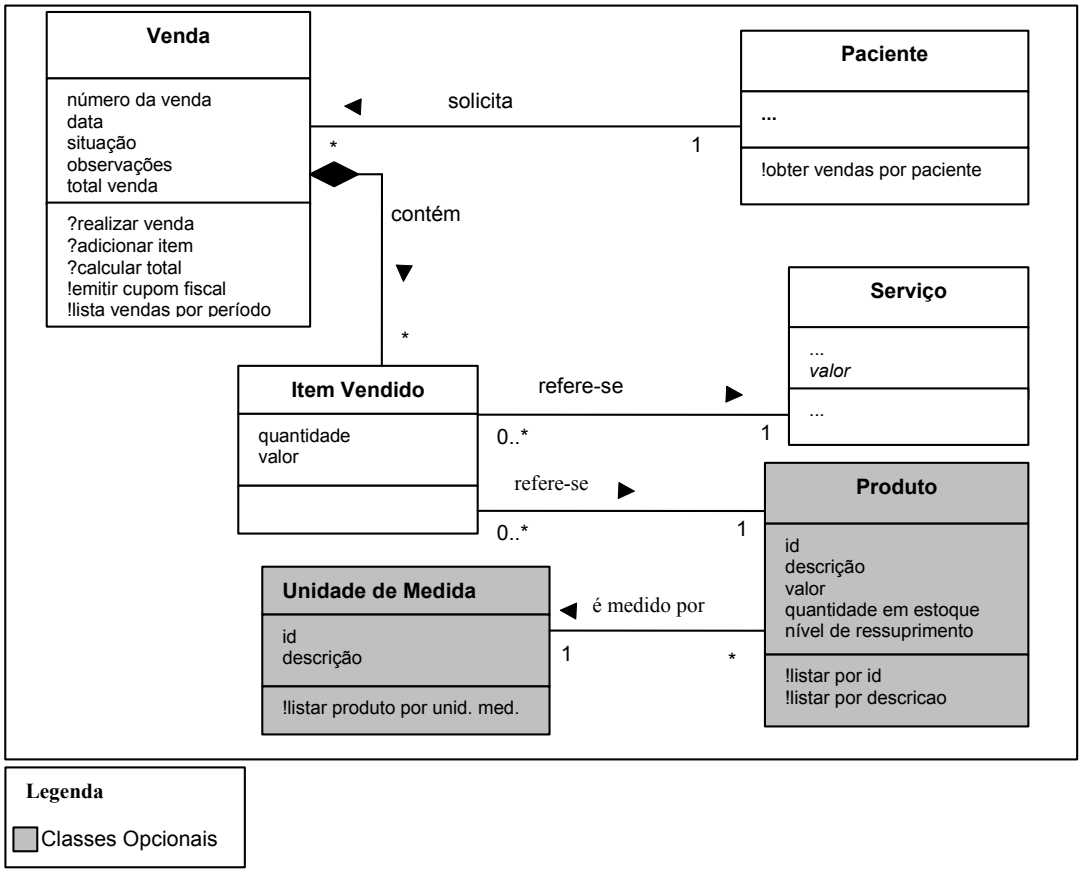


Figura 6 – Diagrama de classes para o Padrão *Realizar Vendas.*

Dinâmica

A Figura 7 apresenta o diagrama de seqüência para o caso de uso Realizar Vendas, em que são comercializados apenas produtos e serviços. Os diagramas de seqüência correspondentes aos cursos alternativos não são exibidos.

Variantes

Uma clínica pode lidar com a comercialização de serviços, de produtos ou com as duas. A prestação de serviço é uma atividade obrigatória para a clínica, já a comercialização de produtos é uma atividade opcional. Por esse motivo as classes Produto e Unidade de Medida são opcionais para o padrão.

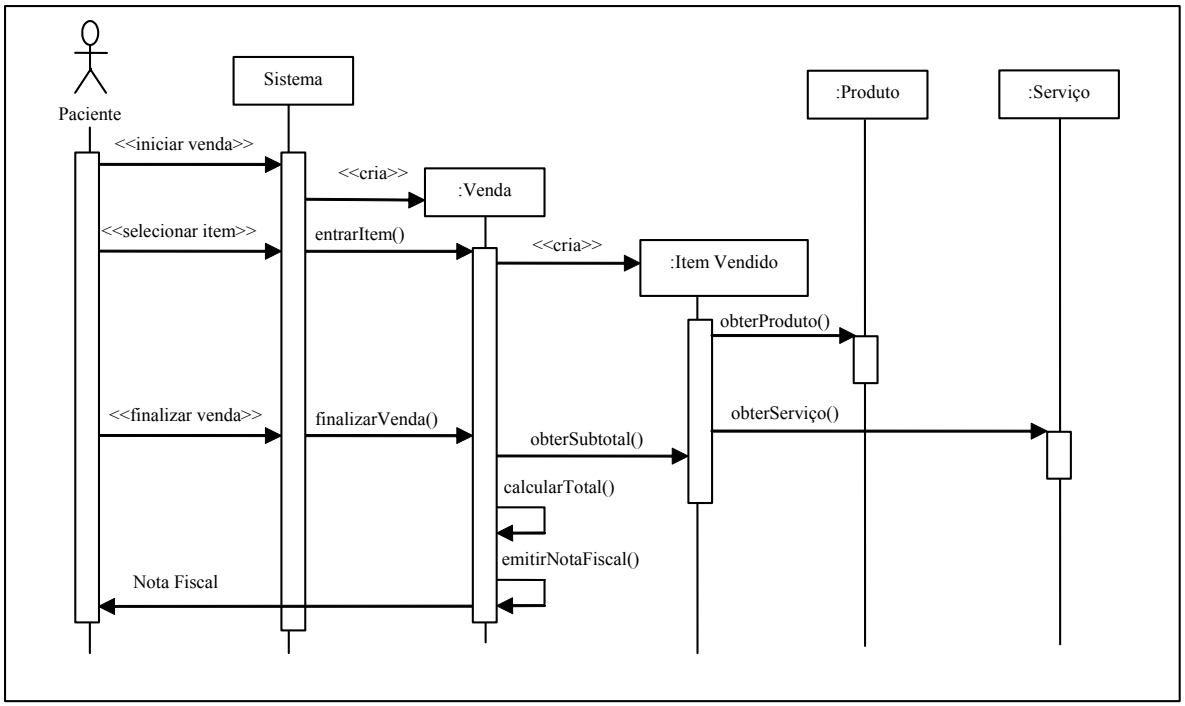


Figura 7 - Diagrama de seqüências para o caso de uso Realizar Vendas.

Usos Conhecidos

Qualquer empresa que realiza algum tipo de comercialização. Exemplo: lojas de calçados, confecções, etc.

Padrões Relacionados

Identificar o Recurso (P1), Quantificar o Recurso(P2), Comercializar o Recurso (P6) e Itemizar Transação do Recurso (P11) da GRN[4].

Próximos padrões

Após a venda de um serviço deve-se agendar as sessões para possibilitar o atendimento dos pacientes, aplicando-se o padrão *Agendar Atendimentos(5)*. Se a clínica realiza também a comercialização de produtos, então o padrão *Realizar Compras(8)* deve ser aplicado. Se a clínica não realiza a comercialização de produtos, mas se preocupa em fazer o controle de faturamento, o padrão *Controlar Faturamento(9)* deve ser aplicado.

Padrão 4: Processar Guias

Intuito

Obter informações sobre as guias de encaminhamento médico que possibilitam o atendimento de um paciente pela clínica.

Contexto

Clínicas que possibilitam o agendamento e o atendimento de pacientes por meio de guias de convênios médicos.

Problema

Como controlar o atendimento de pacientes com convênios em uma clínica?

Influências

- Existem diversos detalhes sobre as guias que devem ser armazenados. Por exemplo, o nome do médico que encaminhou o paciente, a patologia por ele diagnosticada e o convênio a que o paciente pertence.
- Cada convênio possui códigos próprios para as patologias.
- Cada patologia possui uma quantidade de chs (coeficiente de honorário) que deve ser paga pelos convênios à clínica. Cada convênio paga um valor específico de chs.

Estrutura

A Figura 8 mostra o diagrama de classes para o padrão *Processar Guias*.

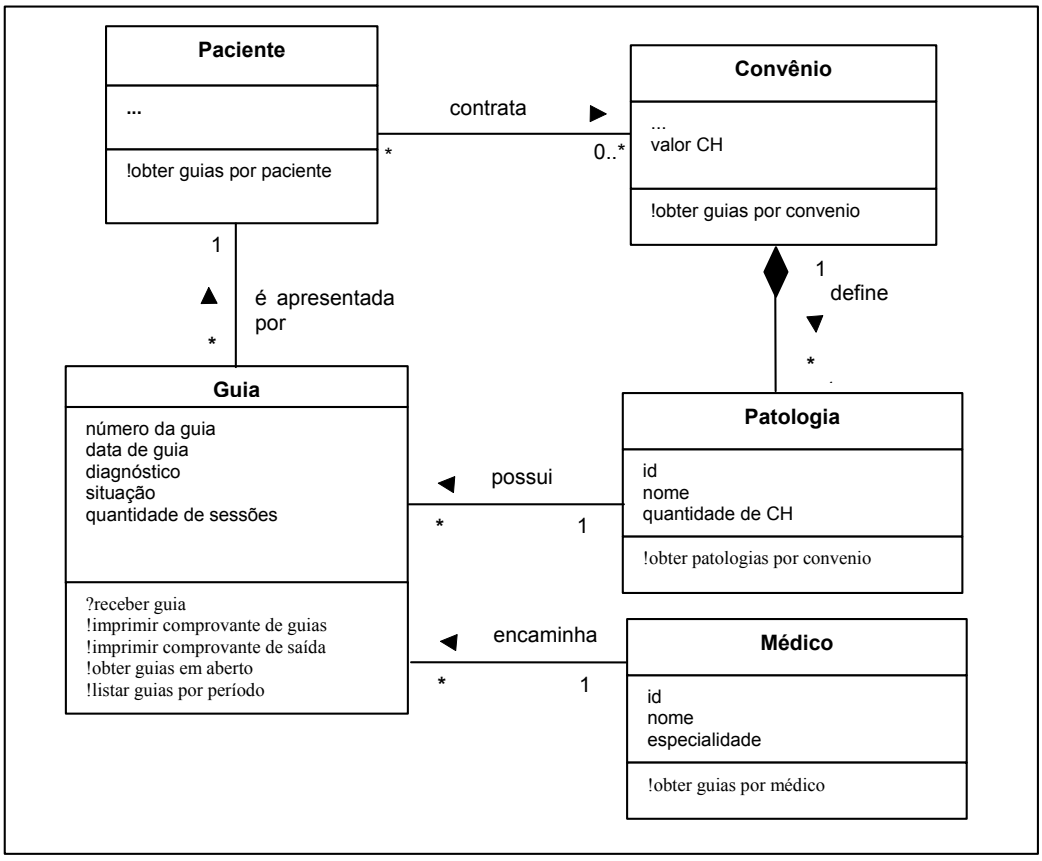


Figura 8 – Diagrama de classes para o padrão *Processar Guias.*

Participantes

Guia: é usada para autorizar o tratamento do paciente na clínica. Ela determina o número de sessões que o paciente pode realizar. Esse número de sessões depende do convênio. Nela constam informações relatadas pelo médico responsável pelo encaminhamento do paciente à clínica. O atributo situação define se a guia já foi avaliada pelo padrão *Realizar Acompanhamento*(7).

Convênio: conforme já definido no padrão 1, acrescentando-se o atributo valor CH e a operação para obter guias por convenio.

Patologia: é a doença, diagnosticada pelo médico do convênio, que motiva o tratamento de um paciente na clínica. Cada convênio classifica a patologia seguindo normas próprias.

Médico: identifica a pessoa responsável pelo encaminhamento de um paciente para realizar a reabilitação na clínica.

Paciente: conforme já definido no padrão 1, acrescentando-se a operação para obter as guias por paciente.

Exemplo

A Figura 8 já representa a instanciação do padrão *Processar Guias*, pois as classes da aplicação recebem os mesmos nomes definidos para o padrão.

Usos conhecidos

Clínicas que realizam atendimentos mediante apresentação de guias de convênios médicos, sendo esses os responsáveis pelo pagamento do tratamento do paciente, como é o caso das clínicas de fisioterapia e terapia ocupacional do Centro de Reabilitação Física Dom Bosco, Lins-SP.

Padrões Relacionados

Manter Recurso (9) da GRN[4].

Próximos padrões

Após receber as guias dos pacientes, deve-se agendar as sessões de atendimento, aplicando-se o padrão *Agendar Atendimentos(5)*. Se o controle de faturamento for de interesse da clínica, o padrão *Controlar Faturamento*(9) também deve ser aplicado. Esse padrão pertence ao grupo 3 responsável pelo controle de transações.

Padrão 5: Agendar Atendimentos

Intuito

Obter informações que possibilitem o agendamento de sessões para os atendimentos aos pacientes.

Contexto

Clínicas que realizam atendimentos após agendamentos prévios. Esses agendamentos são resultados de vendas de serviços ou de guias apresentadas pelos pacientes (convênios médicos).

Problema

Como controlar o agendamento de sessões para realizar o atendimento aos pacientes em uma clínica?

Influência

- Um agendamento sempre ocorrerá após a venda de algum serviço ou após a apresentação de uma guia de convênio pelo paciente.
- Após a recepção do paciente, a clínica deve permitir o agendamento das sessões para o seu tratamento. A quantidade de sessões é definida pela guia apresentada pelo paciente ou pela quantidade de serviços comprados pelo paciente.
- Para agendar uma sessão é necessário verificar se há um horário disponível para a atividade desejada e se o total de atendimentos permitidos para aquele horário ainda não foi ultrapassado.
- A agenda de um paciente pode ser cancelada após uma quantidade de faltas pré-determinada.

- Sempre que um paciente chega à clínica, deve-se verificar a existência de sessões agendadas para o dia. Caso haja, deve-se recepcionar o paciente informando ao atendente sobre sua chegada e disponibilizando o formulário de avaliação da sessão do paciente ao atendente.

Estrutura

A Figura 9 ilustra o diagrama de classe para o padrão *Agendar Atendimentos*.

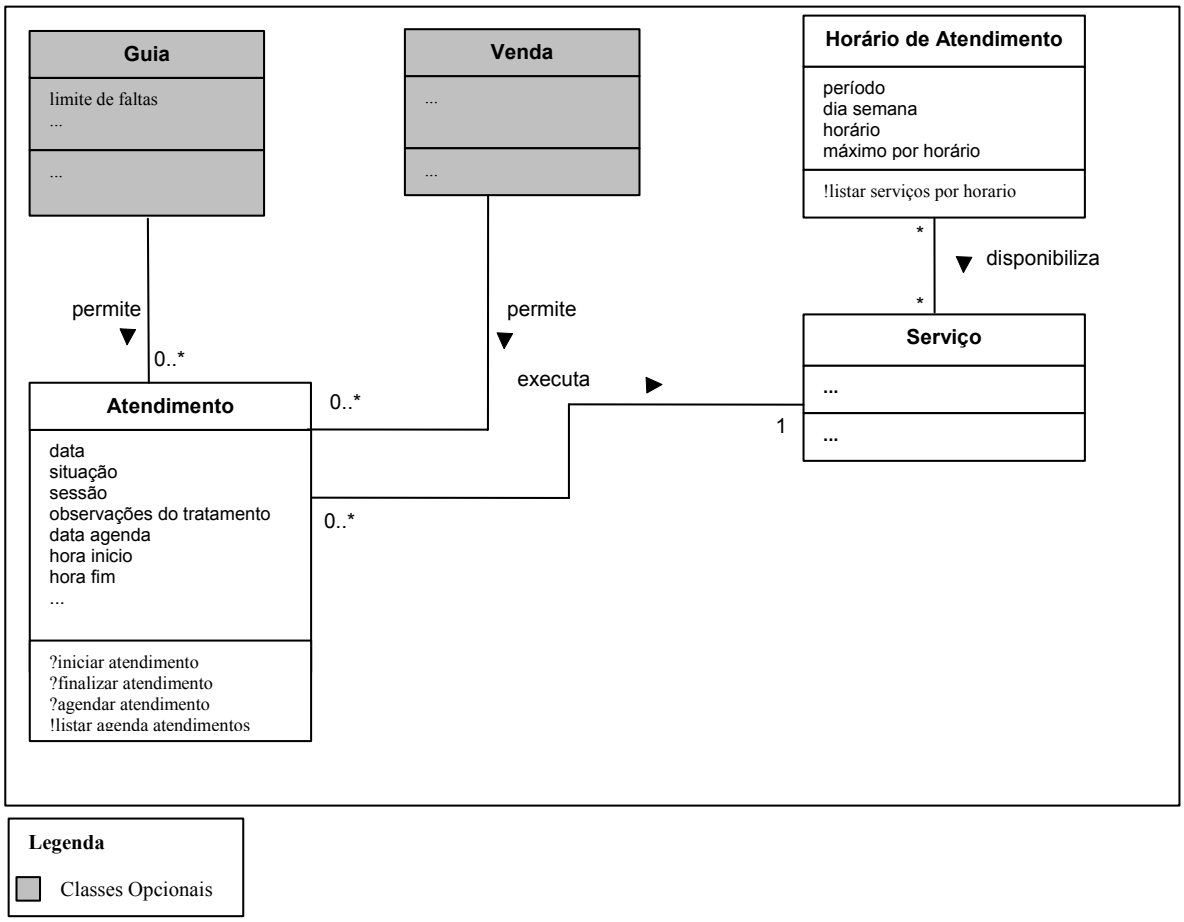


Figura 9 – Diagrama de classes para o padrão *Agendar Atendimentos.*

Participantes

Atendimento: contém todos os detalhes referentes ao agendamento de sessões de um paciente para a execução de um serviço: data e horário da agenda, horário de atendimento, horário de chegada do paciente, além da situação da agenda que pode ser atendida, aberta, cancelada ou agendada. Existem operações para agendar, iniciar e finalizar o atendimento. O operação iniciar atendimento altera a situação do atendimento de agendada para aberta, possibilitando a avaliação do paciente no padrão *Realizar Acompanhamento*(7). Após a avaliação a situação é alterada para atendida.

Horário de Atendimento: contém informações referentes aos horários de serviços disponíveis para agendar uma sessão para um paciente. O atributo período define em qual período o serviço será executado (matutino, vespertino ou noturno), o dia corresponde ao dia da semana em que o serviço pode ser prestado, o máximo por horário define a quantidade de atendimento que pode ser realizada naquele horário.

Guia: conforme já definido no padrão 4, acrescentando-se o atributo limite de faltas permitido pela guia.
Venda: conforme já definido no padrão 3.
Serviço: conforme já definido no padrão 2.

Exemplo

A Figura 9 já representa uma instanciação do padrão *Agendar Atendimentos*, pois as classes da aplicação possuem os mesmos nomes definidos para o padrão.

Dinâmica

A Figura 10 apresenta o diagrama de seqüência para o caso de uso Agendar Consulta quando uma guia de convênio médico é apresentada.

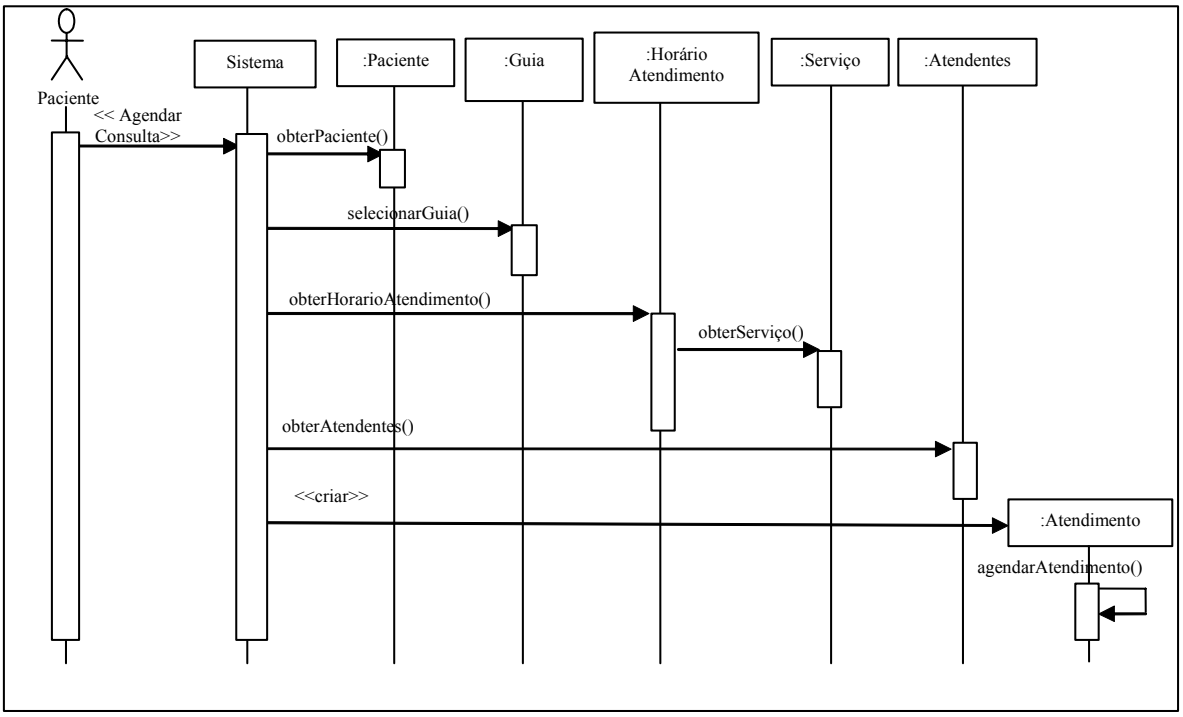


Figura 10 - Diagrama de seqüências para o caso de uso Agendar Consulta.

Variantes

Uma agenda pode ser gerada somente através da venda de um serviço ou da apresentação de uma guia pelo paciente, dependendo dos padrões aplicados anteriormente. Por esse motivo as classes Venda e Guia são opcionais, sendo pelo menos uma delas obrigatórias em uma aplicação.

Usos conhecidos

Clínicas que realizam atendimentos por meio de agendamento de consultas, como é o caso das clínicas de fisioterapia e terapia ocupacional do Centro de Reabilitação Física Dom Bosco, Lins-SP.

Padrões Relacionados

Identificar as Tarefas da Manutenção (14) da GRN[4].

Próximos padrões

Após agendar os atendimento, uma clínica pode se preocupar em identificar os responsáveis pelo atendimento aos pacientes, aplicando-se o padrão *Identificar Atendente*(6). Se a clínica não se preocupa em identificar os atendentes o padrão *Realizar Acompanhamento*(7) deve ser aplicado.

Padrão 6: Identificar Atendentes

Intuito

Obter informações referentes às pessoas responsáveis por realizar atendimentos aos pacientes.

Contexto

Clínicas que identificam as pessoas responsáveis pelos atendimentos aos pacientes .

Problema

Como identificar os atendentes que atuam na clínica?

Influência

- Identificar um atendente pode ser importante quando, por exemplo, deseja-se analisar as avaliações realizadas por determinados atendentes. Outro importante uso dessa identificação seria para efetuar pagamento de um atendente, levando-se em consideração que um atendente pode receber por horas trabalhadas.

Estrutura

A Figura 11 ilustra o diagrama de classes para o padrão *Identificar Atendentes*.

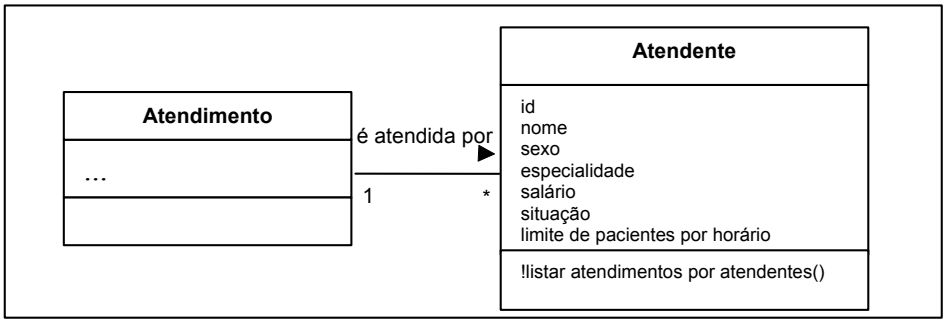


Figura 11 – Diagrama de classes para o padrão *Identificar Atendentes.*

Participantes

Atendente: contém as informações referentes aos responsáveis pelo atendimento a pacientes na clínica. Os principais atributos são: id, nome, sexo, situação, limite de pacientes por horário. Em alguns casos pode ser necessário armazenar informações tais como especialidade e salário do atendente.
Atendimento: conforme já definido no padrão 5.

Exemplo

A Figura 12 representa a instanciação do padrão *Identificar Atendente* para uma clínica de Fisioterapia.

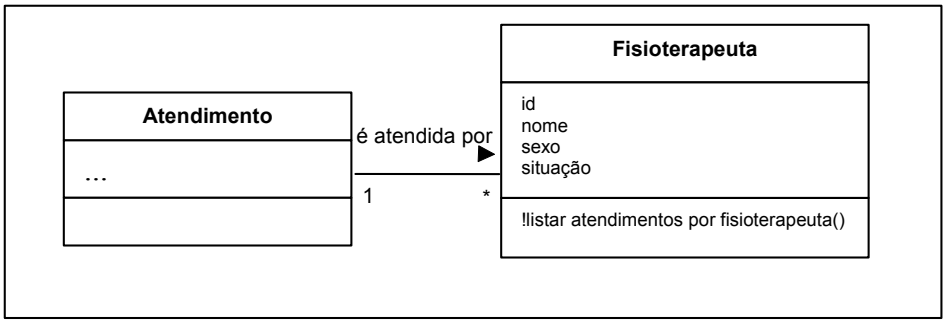


Figura 12 – Instanciação do padrão *Identificar Atendentes.*

Variantes

Em alguns casos torna-se necessário definir a especialidade do atendente, bem como o seu salário. Por exemplo, um paciente com lesões ortopédicas deve ser encaminhado a um atendente especialista em ortopedia. Esses atributos são opcionais para a classe.

Usos conhecidos

Clínicas que necessitam identificar os responsáveis pelo atendimento de um paciente, o que ocorre praticamente em todo tipo de clínica, como é o caso das clínicas de fisioterapia e terapia ocupacional do Centro de Reabilitação Física Dom Bosco, Lins-SP.

Padrões Relacionados

Identificar Executor da Transação (P13) da GRN[4].

Próximos padrões

Após identificar o atendente responsável por um atendimento agendado, deve-se aplicar o padrão *Realizar Acompanhamento*(7).

Padrão 7: Realizar Acompanhamento

Intuito

Obter informações que possibilitem relatar a evolução e acompanhar o tratamento dos pacientes, visando futuras avaliações dos acompanhamentos realizados.

Contexto

Clínicas que necessitam armazenar dados sobre os tratamentos realizados com os pacientes, bem como acompanhar a evolução do tratamento.

Problema

Como armazenar e recuperar os dados referentes às evoluções dos pacientes durante o tratamento em uma clínica?

Influência

- Guardar informações sobre o estado do paciente é relevante para uma clínica. Dessa forma, uma avaliação inicial (anamnese) sobre o estado atual do paciente deve ser realizada. Quando o mesmo paciente retorna à clínica para dar continuidade ao tratamento, não é mais necessário realizar tal avaliação.

- Em algumas clínicas, como por exemplo as de fisioterapia, essas avaliações podem ocorrer a cada nova guia apresentada pelo paciente, uma vez que o motivo pelo qual ele retornou à clínica pode ser diferente daquele que o levou a realizar um tratamento anterior.
- As avaliações são específicas para cada área (ortopedia, pneumologia, neurologia...), assim torna-se difícil definir atributos comuns para cada avaliação.
- A cada sessão é necessário fazer um acompanhamento da evolução do paciente, atribuindo notas que correspondem à evolução do paciente e descrevendo o seu comportamento.
- Identificar o responsável pela avaliação também é um fator importante.

Estrutura

A Figura 13 mostra o diagrama de classes para o padrão *Realizar Acompanhamento*.

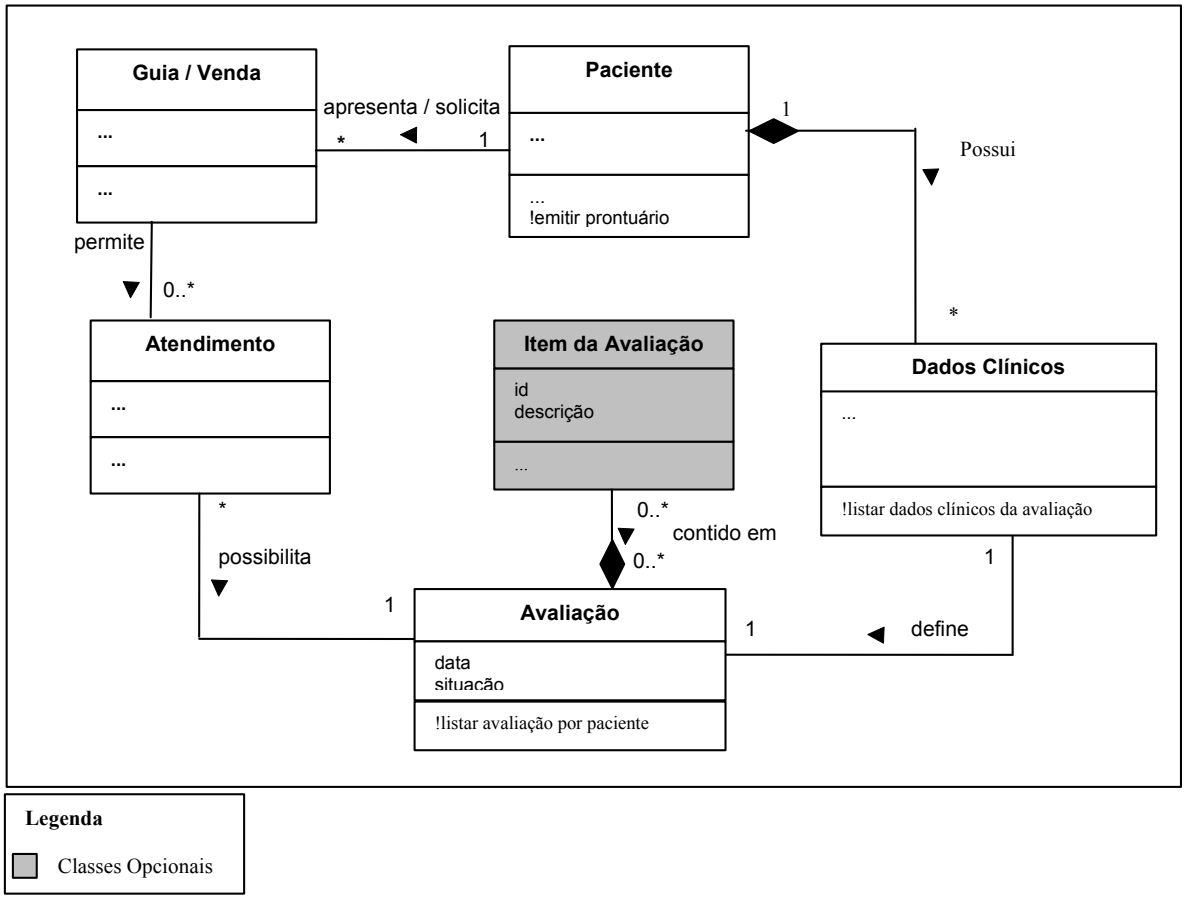


Figura 13 - Diagrama de classes para o padrão *Realizar Acompanhamento*.

Participantes

Atendimento: conforme já definido no padrão 5.

Item da Avaliação: é qualquer recurso ou requisito necessário para conduzir uma avaliação. Esses itens podem ser medicamentos usados por pacientes, tipos de testes realizados durante a avaliação, etc.

Dados Clínicos: conforme já definido no padrão 1, acrescentando-se a operação para listar dados clínicos da avaliação.

Avaliação: é o acompanhamento realizado com o paciente, dependendo da patologia diagnosticada. As informações sobre a avaliação são usadas como referência a

tratamentos posteriores. Cada avaliação possui um conjunto de atributos bem específico, o que torna difícil definir um padrão de avaliação para um conjunto de clínicas distintas. Por exemplo, o conjunto de atributos analisado em uma avaliação ortopédica é diferente entre as clínicas de fisioterapia, terapia ocupacional e educação física.

Guia / Venda: conforme já definido nos padrões 4 e 3 respectivamente.

Paciente: conforme já definido no padrão 1, acrescentando-se a operação emitir prontuário

Exemplo

A Figura 14 representa a instanciação para o padrão *Realizar Acompanhamento* para uma clínica de Terapia Ocupacional. Nela pode-se observar que as sub-classes da classe Avaliação também são opcionais, uma vez que cada clínica possui avaliações próprias. As classes Medicamento e Teste Especial referem-se à classe Item da Avaliação.

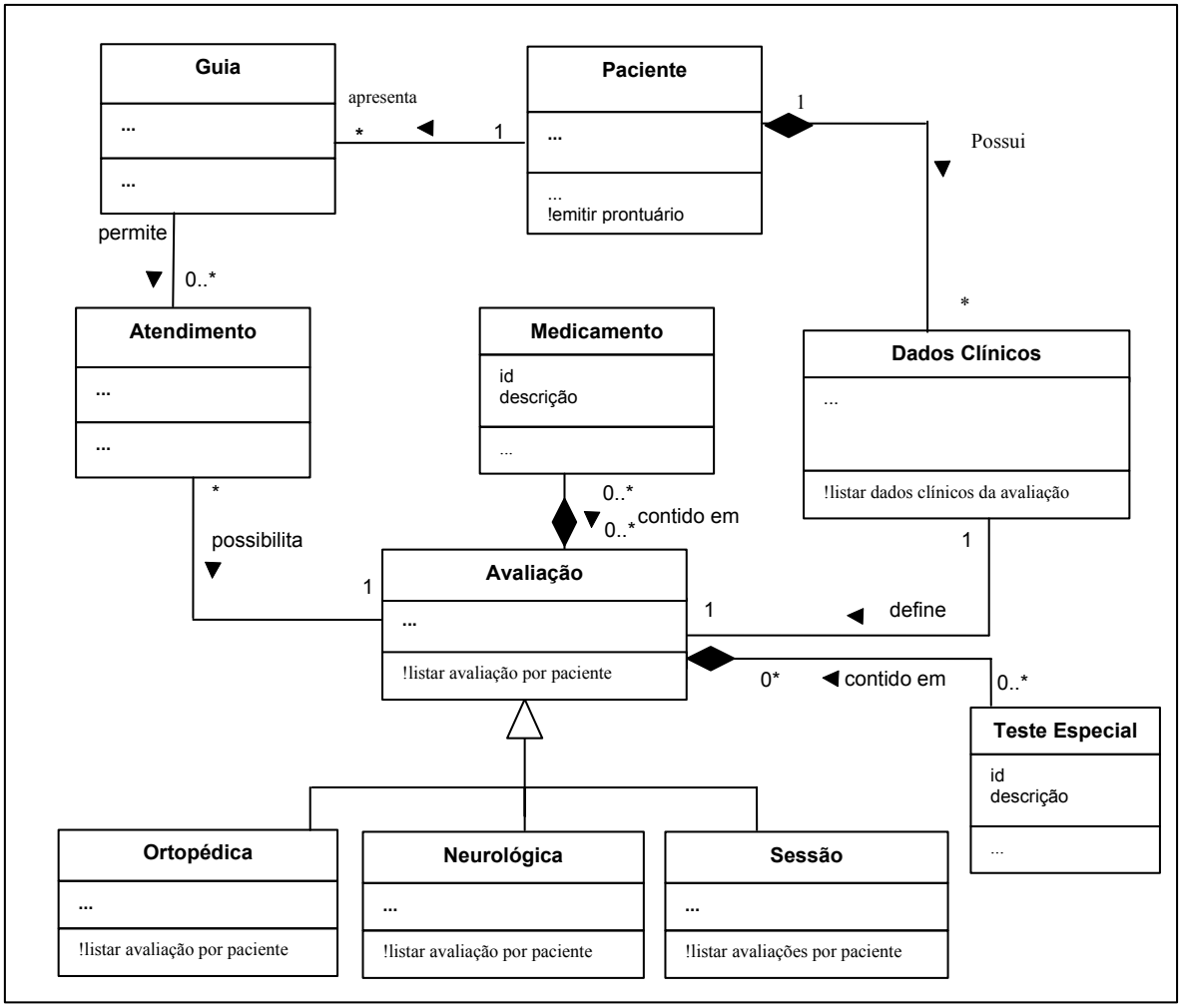


Figura 14 - Instanciação do padrão *Realizar Acompanhamento*.

Dinâmica

A Figura 15 apresenta o diagrama de seqüência para o caso de uso Tratar Pacientes.

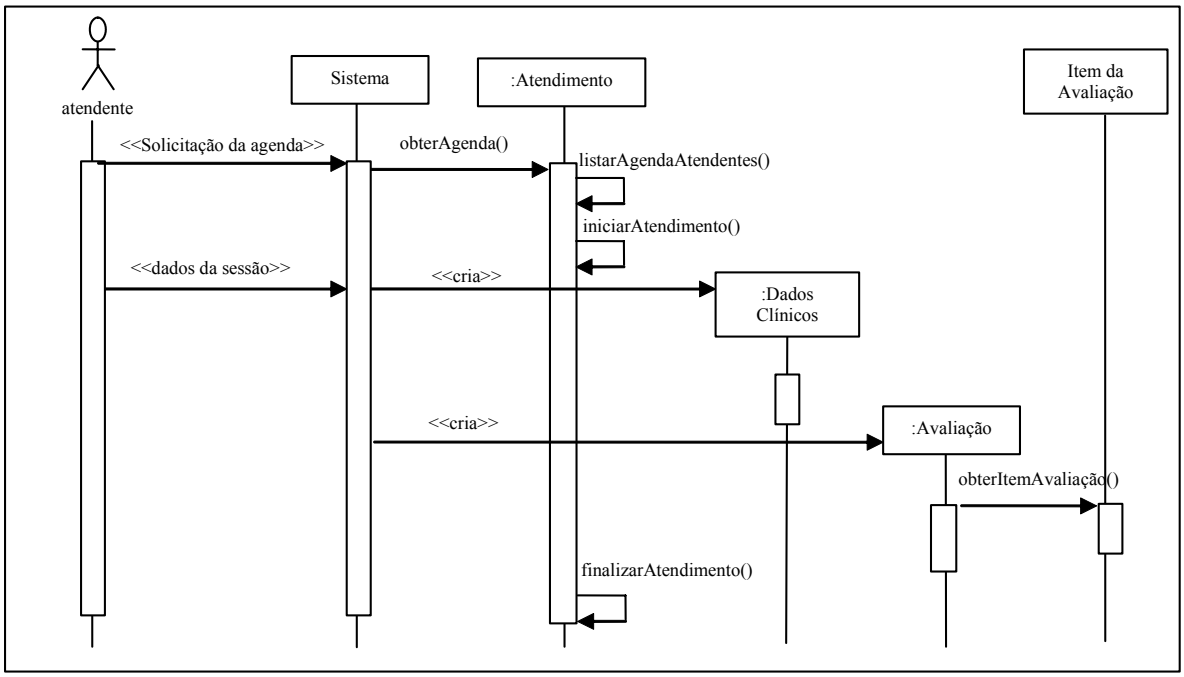


Figura 15 - Diagrama de seqüênias para o caso de uso Tratar Pacientes

Variantes

Uma clínica pode possuir um conjunto de avaliações distinto, dependendo dos serviços que ela presta e da sua finalidade. Cada avaliação pode ou não possuir itens de avaliação, dependendo do propósito da avaliação, por esse motivo a classe Item da Avaliação é opcional.

Usos conhecidos

Clínicas que realizam avaliações constantes para analisar a evolução e as reações dos pacientes que são submetidos a determinados tratamentos, como é o caso das clínicas de fisioterapia e terapia ocupacional do Centro de Reabilitação Física Dom Bosco, Lins-SP.

Próximos padrões

Após aplicar o padrão *Realizar Acompanhamento(7)*, deve-se verificar se a clínica necessita fazer venda de serviços e/ou produtos. Em caso positivo deve-se aplicar o padrão *Realizar Vendas(3)*. Caso esse padrão já tenha sido aplicado, deve-se analisar a necessidade de fazer o controle de compras de produtos e faturamento. Se for necessário controlar a compra de produtos aplica-se *Realizar Compras(8)*. Caso não seja necessário esse controle e sim controlar o faturamento, aplica-se o padrão *Controlar Faturamento(9)*. Se o padrão *Realizar Vendas(3)* não tiver sido aplicado, mas o *Processar Guias(4)* foi e é necessário controlar o faturamento, aplica-se *Controlar Faturamento(9)*.

Padrão 8: Realizar Compras

Intuito

Obter informações sobre os produtos comercializados pela clínica, possibilitando a gestão das compras realizadas para suprir o estoque desses produtos.

Contexto

Clínicas que realizam comércio de produtos e necessitam comprá-los para repor os estoques.

Problema

Como gerenciar a compra de produtos que serão comercializados por uma clínica?

Influência

- A compra de produtos é uma atividade que armazena informações para gerar relatórios importantes para o funcionamento da clínica.

Estrutura

A Figura 16 ilustra o diagrama de classe para o padrão *Realizar Compras*.

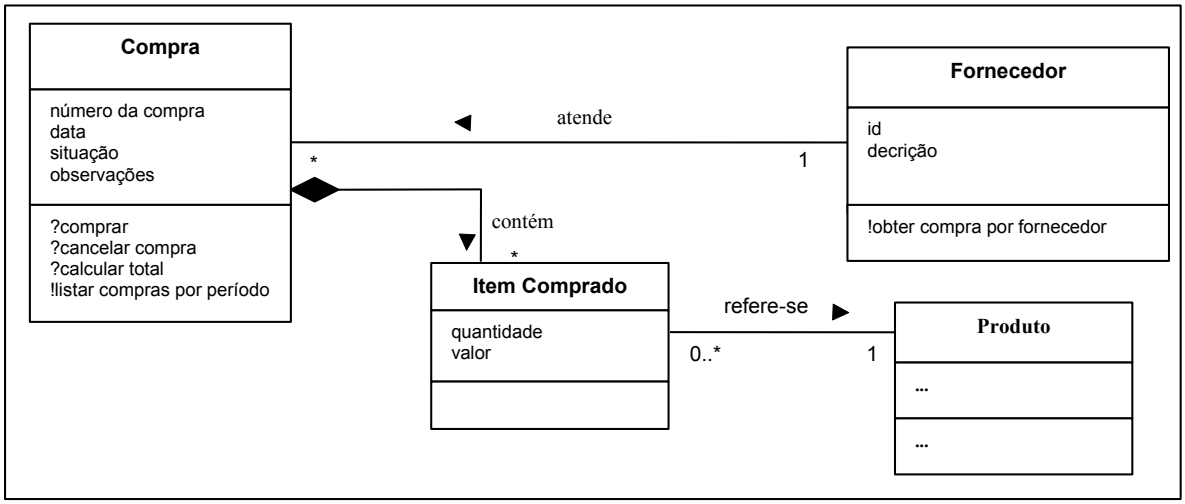


Figura 16 – Diagrama de classes para o padrão *Realizar Compras.*

Participantes

Compra: apresenta os detalhes referentes à compra do produto, recebendo um número que a identifica.

Item Comprado: armazena informações dos itens comprados que permitirão recuperar o histórico das compras realizadas.

Produto: conforme já definido no padrão 3

Fornecedor: proprietário original do produto.

Exemplo

A Figura 16 representa a instanciação do padrão *Realizar Compras*, pois as classes da aplicação recebem os mesmos nomes definidos para o padrão.

Usos conhecidos

Empresas que realizam comércio de produtos e necessitam repor seus estoques, permitindo um melhor controle financeiro da empresa.

Padrões Relacionados

Comercializar Recursos (P6) e Itemizar Transação do Recurso (P11) da GRN[4].

Próximos padrões

Após aplicar o padrão *Realizar Compras*(8), deve-se aplicar o padrão *Controlar Faturamento*(9).

Padrão 9: Controlar Faturamento

Intuito

Obter informações sobre receitas e despesas que ocorrem na clínica, possibilitando o controle de faturamento da clínica.

Contexto

Clínicas que gerenciam os serviços e produtos comercializados e necessitam controlar o faturamento mensal e/ou anual.

Problema

Como controlar as receitas e despesas de uma clínica?

Influência

- Possuir informações históricas sobre os pagamentos e recebimentos permite analisar, de forma mais eficiente, o faturamento mensal e anual de uma clínica.
- Tratar cada parcela individualmente aumenta a complexidade do sistema.
- É possível identificar parcelas não pagas e parcelas a vencer.

Estrutura

A Figura 17 ilustra o diagrama de classes para o padrão *Controlar Faturamento.*

Participantes

Pagamento: representa os pagamentos realizados. Eles são classificados pelo atributo tipo do pagamento. Se o tipo do pagamento for uma fatura ele foi gerado por uma Guia ou uma Venda. Se o tipo de pagamento for uma despesa ele foi gerado por uma Compra ou por um Atendente.

Taxa de Juros: define as regras de negócio que guiam os cálculos de taxas adicionais quando um pagamento é realizado após o seu vencimento

Taxa de Multa: define as regras de negócio que guiam os cálculos de taxas adicionais quando o pagamento é feito em mais de uma parcela.

Venda: conforme já definido no padrão 3, acrescentando-se o atributo numero de parcelas gerada para o pagamento e as operações para gerar fatura e imprimir fatura.

Guia: conforme já definido no padrão 4, acrescentando-se os atributos numero de parcelas e preço total gerada para o pagamento e as operações para gerar fatura, imprimir fatura, calcular ganhos.

Compra: conforme já definido no padrão 5, acrescentando-se o atributo numero de parcelas gerada para o pagamento e as operações para gerar despesas e imprimir despesa.

Atendente: conforme já definido no padrão 6 acrescentando-se as operações para gerar salário e imprimir contra-cheque.

Exemplo

A Figura 17 representa a instanciação do padrão *Controlar Faturamento,* pois as classes da aplicação possuem os mesmos nomes definidos para o padrão.

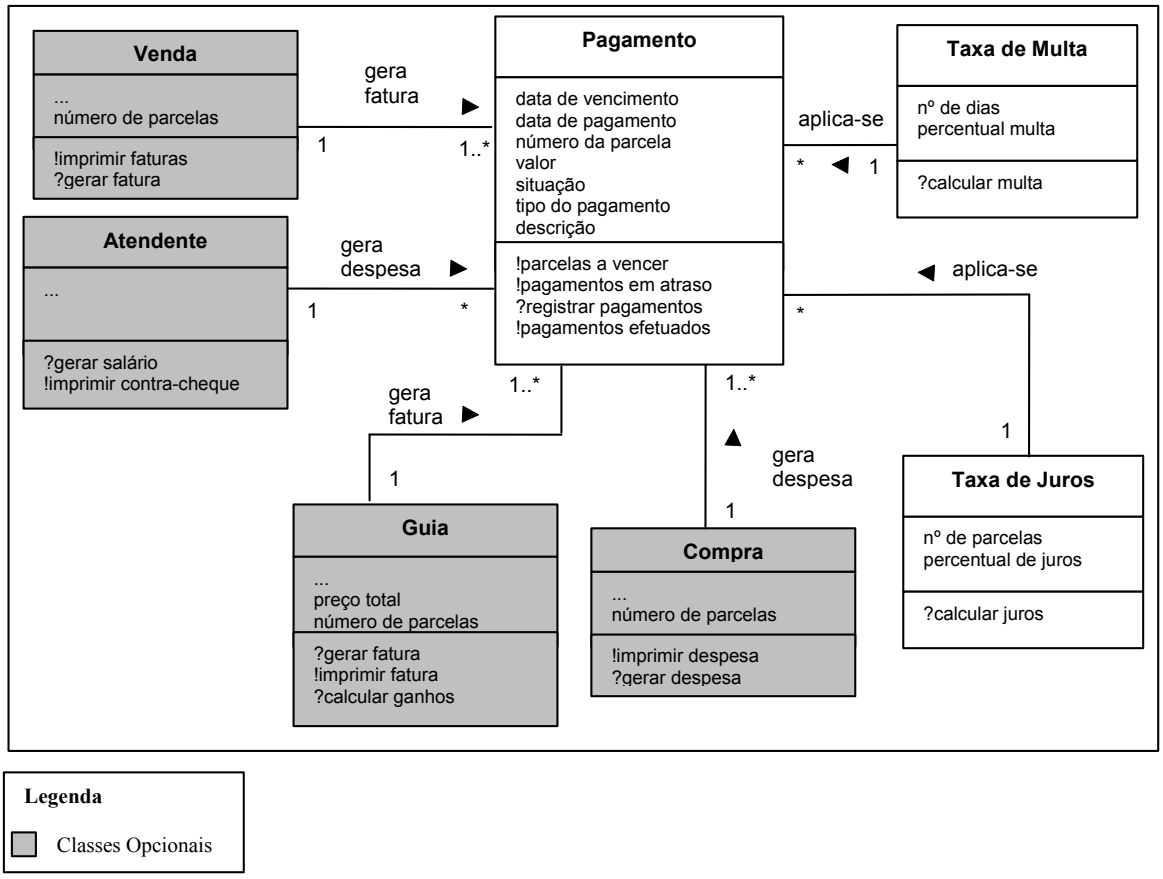


Figura 17 – Diagrama de classes para o padrão *Controlar Faturamento*.

Variantes

As classes opcionais da Figura 17 são resultantes dos padrões aplicados anteriormente. Por exemplo, a classe Atendente, definida no padrão 6, será usada neste padrão somente se o padrão 6 foi aplicado com a sua variante.

Usos conhecidos

Empresas que precisam controlar o faturamento.

Padrões Relacionados

Pagar pela Transação do Recurso (P12) da GRN[4].

Próximos padrões

Verifique se a clínica permite que pacientes sejam atendidos mediante a apresentação de guias de convênios médicos e caso isso seja possível aplique o padrão *Processar Guias(4)*. Se a clínica realiza atendimentos comercializando serviços, nesse caso aplique o padrão *Realizar Vendas*(3). Caso os dois padrões já tenham sido aplicados, a clínica já está instanciada com as suas principais funcionalidades.