UNIVERSIDADE DE CAXIAS DO SUL

CENTRO DE CIÊNCIAS EXATAS E DA TECNOLOGIA

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

LEONARDO PELLIZZONI

DESENVOLVIMENTO DE ARQUITETURA DE SOFTWARE REUTILIZÁVEL PARA IMPLEMENTAÇÃO DE APLICAÇÕES EM .NET

Caxias do Sul – RS

2016

LEONARDO PELLIZZONI

DESENVOLVIMENTO DE ARQUITETURA DE SOFTWARE REUTILIZÁVEL PARA IMPLEMENTAÇÃO DE APLICAÇÕES EM .NET

Monografia apresentada como requisito para a obtenção do grau de Bacharel em Sistemas de Informação da Universidade de Caxias do Sul.

Orientador Prof. Me. Alexandre Erasmo Krohn Nascimento

Caxias do Sul – RS

2016

RESUMO

Desenvolve-se neste trabalho uma arquitetura de software reutilizável, para ser aplicada em projetos do Laboratório de Criação e Aplicação de Software, da Universidade de Caxias do Sul, com foco em possibilitar a intercambialidade de camadas e preservar a integridade das regras de negócio das aplicações, mesmo com múltiplas interfaces de usuário e locais de armazenamento. Duas aplicações são implementadas, como provas de conceito, para testar a arquitetura de software, de modo que, possibilita-se abstrair em frameworks algumas das funcionalidades, potencializando a reutilização em outras aplicações.

Palavras chave: Regras de negócio. Arquitetura de software. Reutilização. Frameworks. Design de software**.**

LISTA DE FIGURAS

Figura 1 – Visão geral do processo ICONIX....24
Figura 2 - Representação de arquitetura de software como uma ponte....26
Figura 3 - Modelo de camadas para divisão lógica em uma aplicação Java....28
Figura 4 - Exemplo de uma não arquitetura em camadas....29
Figura 5 - Exemplo de diagrama de camadas....30
Figura 6 - Divisão da interação com o usuário em três papéis distintos....32
Figura 7 - Ilustração do papel Modelo e Visão do padrão MVC....32
Figura 8 - Modelo parcial de domínio do Banco Imobiliário....33
Figura 9 – Exemplo em Java de repositórios com duas fontes de dados....35
Figura 10 - Camada de serviço em uma aplicação....36
Figura 11 - Código fonte de exemplo para utilização de injeção de dependência....37
Figura 12 - Cenários de aplicações e a relação com framework....38
Figura 13 - Hipótese de esforço gasto em design de software....39
Figura 14 - Relação entre frameworks e aplicações....40
Figura 15 - Visão futura do .NET....41
Figura 16 - Código compartilhado entre aplicações Android, iOS e Windows....42
Figura 17 - ASP.NET e ASP.NET Core....43
Figura 18 - Exemplo de página utilizando HTML e a sintaxe Razor....44
Figura 19 - Gerenciamento de pacotes pelo NuGet....45
Figura 20 - Criação de pacote no NuGet Package Explorer....46
Figura 21 - Exemplo de mapeamento de classe no NHibernate....47
Figura 22 - Exemplo de mapeamento de classe no Entity Framework....48
Figura 23 - Árvore de projetos da aplicação mojoPortal....51
Figura 24 - Arquitetura tecnológica do DotNetNuke....52
Figura 25 - Arquitetura de software de um módulo do DotNetNuke....53
Figura 26 - Arquitetura de software do servidor para o projeto OrderPad....54
Figura 27 - Controllers na arquitetura de software do OrderPad....55
Figura 28 - Funcionamento da arquitetura de software....58
Figura 29 - Exemplo com múltiplas tecnologias na camada de acesso aos dados....59
Figura 30 - Exemplo da camada de serviço....61
Figura 31 - Exemplo de implementação na camada de modelo....64
Figura 32 - Exemplo não recomendado de implementação na camada de modelo....64
Figura 33 - Exemplo de implementação da globalização....65
Figura 34 – Diagrama de classes de globalização....66
Figura 35 - Tela do cadastro de usuário....67
Figura 36 – Cadastro de perfil....68
Figura 37 - Exemplo das possíveis permissões de telas e funcionalidades....69
Figura 38 - Diagrama de classes básico para a camada de segurança....70
Figura 39 - Exemplo de injeção de dependência....71
Figura 40 - Exemplo de injeção de dependência através de fábricas....71
Figura 41 - Diagrama de Caso de Uso da Aplicação CRUD....72
Figura 42 - Diagrama de classes da Aplicação CRUD....72
Figura 43 - Funcionamento geral da Aplicação CRUD....73
Figura 44 - Casos de uso de Acompanhamento de Indicadores Clínicos....74
Figura 45 - Funcionamento geral da aplicação de Indicadores Clínicos....75

Figura 46 - Ilustração dos projetos desenvolvidos 77
Figura 47 - Configurações de pacote do Visual Studio 78
Figura 48 - Gerenciador de pacotes do Visual Studio para Aplicação CRUD 78
Figura 49 - Pacotes com as funcionalidades abstraídas 79
Figura 50 - Fluxo de implementação de funcionalidades 80
Figura 51 - Fluxo de criação da estrutura de aplicações 81
Figura 52 - Fluxo de utilização dos recursos de globalização 82
Figura 53 - Fluxo de utilização dos recursos de segurança 82

LISTA DE QUADROS

Quadro 1 - Atividades genéricas em metodologias de engenharia de software.....23
Quadro 2 - Benefícios de negócio e TI das arquiteturas corporativas.....26
Quadro 3 - Três Camadas Principais 28
Quadro 4 - Tipos de frameworks ao desenvolver em ASP.NET.....44

LISTA DE ABREVIATURAS E SIGLAS

.NET	.NET Framework
ASP	Active Server Pages
API	Application Programming Interface
CLI	Common Language Infrastructure
DI	Dependency Injection
DNN	DotNetNuke
HTML	Hyper Text Markup Language
IDE	Integrated Development Environment
Laboratório	Laboratório de Criação e Aplicação de Software
MSDN	Microsoft Developer Network
MVC	Model-View-Controller
ORM	Object-Relational Mapping
OSS	Open source software
PDF	Portable Document Format
RTM	Release to Manufacturing
SPA	Single Page Application
SQL	Structured Query Language
TI	Tecnologia da Informação
UCS	Universidade de Caxias do Sul
WPF	Windows Presentation Foundation
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO 19

1.1 QUESTÃO DE PESQUISA 20
1.2 OBJETIVO 21
1.3 ESTRUTURA DO TRABALHO 21

2 FUNDAMENTOS 23

2.1 ICONIX 24
2.2 ARQUITETURA DE SOFTWARE 25
2.2.1 Arquitetura em camadas 27
2.2.2 Padrões de projeto 30
2.2.2.1 MVC 31
2.2.2.2 Modelo de domínio 32
2.2.2.3 Repositórios 33
2.2.2.4 Serviços 35
2.2.2.5 Camada supertipo 36
2.2.2.6 Injeção de dependência 37
2.2.3 Frameworks 38
2.3 FERRAMENTAS E ESTRUTURAS 40
2.3.1 .NET Framework 40
2.3.2 Ambientes de desenvolvimento 41
2.3.3 ASP .NET 42
2.3.4 Gerenciadores de pacotes 45
2.3.5 Frameworks de persistência 47

3 TRABALHOS CORRELATOS 49

3.1 MAVERICK.NET 50
3.2 SPRING.NET 50
3.3 MOJO PORTAL 51
3.4 DOT NET NUKE 52
3.5 ORDER PAD 54

4 PROJETO 57

4.1 ARQUITETURA DE SOFTWARE 58
4.1.1 Camada de acesso aos dados 58
4.1.2 Camada de serviço 60
4.1.3 Camadas auxiliares por tipo de apresentação 62
4.1.4 Camada de apresentação 63
4.1.5 Camadas de suporte 63
4.1.5.1 Camada de modelo 63
4.1.5.2 Camada de globalização 65
4.1.5.3 Camada de segurança 67
4.1.5.4 Camada de Injeção de Dependência (IOC) 70

4.2 PROVAS DE CONCEITO 71
4.2.1 Aplicação CRUD 72
4.2.2 Acompanhamento de Indicadores Clínicos 73
5 DESENVOLVIMENTO 77
6 CONSIDERAÇÕES FINAIS 85
6.1 TRABALHOS FUTUROS 86
REFERÊNCIAS 87
INTRODUÇÃO AOS APÊNDICES 93
APÊNDICE A - CAMADA DE SEGURANÇA 95
APÊNDICE B - MODELAGEM DA APLICAÇÃO CRUD 99
APÊNDICE C - ACOMPANHAMENTO DE INDICADORES CLÍNICOS 101
1 MODELO DE DOMÍNIO 101
2 CASOS DE USO 103
3 DIAGRAMAS DE ROBUSTEZ 123
APÊNDICE D – GUIA DE DESENVOLVIMENTO 129
1 ESTRUTURA DE TRABALHO 129
2 ESTRUTURA DE TRABALHO: GLOBALIZAÇÃO 135
3 ESTRUTURA DE TRABALHO: SEGURANÇA 137
4 IMPLEMENTAÇÃO DE FUNCIONALIDADE 139
APÊNDICE E – ACOMPANHAMENTO DE INDICADORES CLÍNICOS DEPLOY 151

1 INTRODUÇÃO

Ferramentas possibilitam que tarefas sejam realizadas de forma otimizada. No âmbito tecnológico, as ferramentas agem como um facilitador para quem as utiliza. A engenharia de *software* progrediu e ocupa um lugar de relevância, com as aplicações projetadas e utilizadas na sociedade (SOMMERVILLE, 2011).

O desenvolvimento de aplicações faz parte do contexto do Laboratório de Criação e Aplicação de Software (Laboratório), localizado no bloco 71 da Universidade de Caxias do Sul (UCS), e que mantém como objetivo promover projetos reais de trabalho no ambiente acadêmico, proporcionando aos alunos do campo de computação aplicar a teoria à prática, aprender novas tecnologias e solidificar a formação acadêmica.

O Laboratório possui projetos sob demanda de diferentes áreas e clientes, e ainda que as aplicações tenham foco em áreas distintas algumas de suas funcionalidades são de comum utilização, como por exemplo persistir as informações no banco de dados, emitir relatórios, gerar gráficos, etc. Embora as aplicações desenvolvidas no Laboratório possuam funcionalidades semelhantes isto não implica na utilização total destas funcionalidades comuns. No entanto considerando o seu contexto torna-se interessante reutilizar as funcionalidades, possibilitando que outros projetos façam uso, concentrando a manutenção e direcionando a maior parte do tempo de desenvolvimento no negócio da aplicação.

Entre a UCS e Microsoft foi criada uma parceria, através do programa DreamSpark, no qual disponibilizam-se softwares para utilização, principalmente por alunos da área de computação. O Visual Studio, utilizado no desenvolvimento de software, comporta a criação de aplicações utilizando-se do .NET Framework (.NET), que é uma coleção de tipos reutilizáveis e orientado a objetos (MSDN, [s. d.]b).

Este trabalho fornece um meio prático de promover a reutilização e guiar o desenvolvimento e evolução das aplicações, através da arquitetura de *software* que preserva suas regras de negócio, e de *frameworks* criados a partir da abstração de funcionalidades comuns. O foco principal da arquitetura de *software* criada neste trabalho é a reutilização da arquitetura criada e a preservação do patrimônio das aplicações, que são suas regras de negócio, de modo a possibilitar que estas estejam

íntegras ao longo da utilização de novas tecnologias e formas de interação com o usuário. Para a validação da arquitetura de *software* foi desenvolvido duas aplicações como provas de conceito, sendo as funcionalidades comuns entre elas ou com potencial de reutilização abstraídas em *frameworks*. Esta arquitetura permite aos projetos do Laboratório continuar contribuindo com frameworks, para serem reutilizados entre outros projetos.

Este trabalho contribuiu com uma arquitetura de *software* que preserva as regras de negócio das aplicações, permitindo alternar entre camada de interação com usuário e de persistência de dados, além de fornecer um conjunto de frameworks para auxiliar no desenvolvimento de *software.* A arquitetura e os frameworks se complementam de forma que é viável utilizá-los de maneira separada ou conjunta, de acordo com a necessidade de cada projeto.

A metodologia ICONIX[1] auxiliou o desenvolvimento das provas de conceito, sendo uma a Aplicação CRUD, que implementa operações básicas em duas interfaces com o usuário (*web* e *desktop*), e a outra o primeiro incremento do sistema de Acompanhamento de Indicadores Clínicos, sendo este um projeto com usuários reais. O desenvolvimento da arquitetura de *software* aconteceu por meio de diagramas, exemplos de implementação para validar a viabilidade, e principalmente, por meio de trabalhos relacionados ao tema, em especial os que foram aplicados, sejam acadêmicos ou corporativos.

1.1 QUESTÃO DE PESQUISA

É possível criar e demostrar uma arquitetura de software reutilizável, para suportar o desenvolvimento de aplicações no Laboratório, que preserve a integridade das regras de negócio das aplicações mesmo com múltiplas formas de acesso e locais de armazenamento?

[1] Se optou pela utilização do ICONIX por ser um processo de desenvolvimento enxuto e ainda assim suficiente, sendo possível partir dos casos de uso e chegar ao código fonte.

1.2 OBJETIVO

Projetar e aplicar uma arquitetura de *software* reutilizável, para o desenvolvimento de *software* em .NET Framework, que viabilize “N” camadas, preservando a integridade das regras de negócio; de modo a auxiliar o Laboratório de Criação e Aplicação de Software da UCS a construir alguns dos projetos sob demanda.

O trabalho tem como objetivos específicos:

a) Projetar e implementar a arquitetura de *software* reutilizável;
b) Testar a arquitetura de software através de provas de conceito;
c) Provar a reutilização;
d) Demostrar a intercambialidade entre camadas;
e) Elaborar documentação para a utilização e desenvolvimento.

1.3 ESTRUTURA DO TRABALHO

Neste capítulo foi apresentado uma introdução ao contexto do trabalho. No capítulo 2 é abordado o referencial teórico, de forma simples e dedicando maior ênfase nas seções de arquitetura de *software*, do que em ferramentas e tecnologias. Os trabalhos correlatos, sejam acadêmicos ou corporativos, são abordados no capítulo 3, sendo detalhado apenas os que apresentam forte relação com os objetivos deste trabalho. O capítulo 4 detalha a proposta de solução, junto das provas de conceito, utilizadas para testar a arquitetura de *software*. O capítulo 5 apresenta as considerações práticas referente ao desenvolvimento deste trabalho, concluindo com as considerações finais apresentadas no capítulo 6.

2 FUNDAMENTOS

O *software* é criado a partir de técnicas, métodos, modelos e ferramentas. Os modelos funcionam como alicerces para a engenharia de software, indiferente do porte da aplicação que se deseja construir. Como atividades das metodologias, de maneira genérica, compreende-se a comunicação, planejamento, modelagem, construção e entrega, descritos no Quadro 1 (PRESSMAN; MAXIN, 2016).

Atividade	Descrição
Comunicação	Entender os objetivos dos envolvidos para o projeto e reunir requisitos que ajudem a definir os recursos e as funções do *software*.
Planejamento	Funciona como um mapa que ajuda a guiar a equipe. Define o trabalho de engenharia de *software*, descrevendo tarefas técnicas a serem conduzidas, riscos prováveis recursos necessários, produtos resultantes e um cronograma.
Modelagem	Cria-se um "esboço" para que se possa ter uma ideia do todo, qual será seu aspecto em termos de arquitetura, como as partes constituintes se encaixarão e várias outras características.
Construção	O que se projeta deve ser construído. Gera-se códigos e testes.
Entrega	O software, como entidade completa ou incremento, é entregue ao cliente, que avalia o produto entregue e fornece *feedback*.

Quadro 1 - Atividades genéricas em metodologias de engenharia de *software*

Fonte: Adaptado de Pressman; Maxin (2016, p. 17)

Os modelos de processos de *software*, que tendem a gerar e manter aplicações, possuem variações, adição ou exclusão de atividades, já que um processo não é rígido e pode ser adaptado, conforme as necessidades (PRESSMAN; MAXIN, 2016; SOMMERVILLE, 2011).

2.1 ICONIX

O ICONIX é um processo utilizado no desenvolvimento de *software*, originado a partir do mínimo de técnicas, ainda assim suficientes, do UML para que se chegue dos casos de uso ao código fonte (ROSENBERG; STEPHENS, 2007), considerando que, é viável modelar grande parte dos problemas com apenas algumas técnicas do UML (BOOCH; RUMBAUGH; JACOBSON, 1998). A Figura 1 exemplifica a visão geral do fluxo de processo do ICONIX.

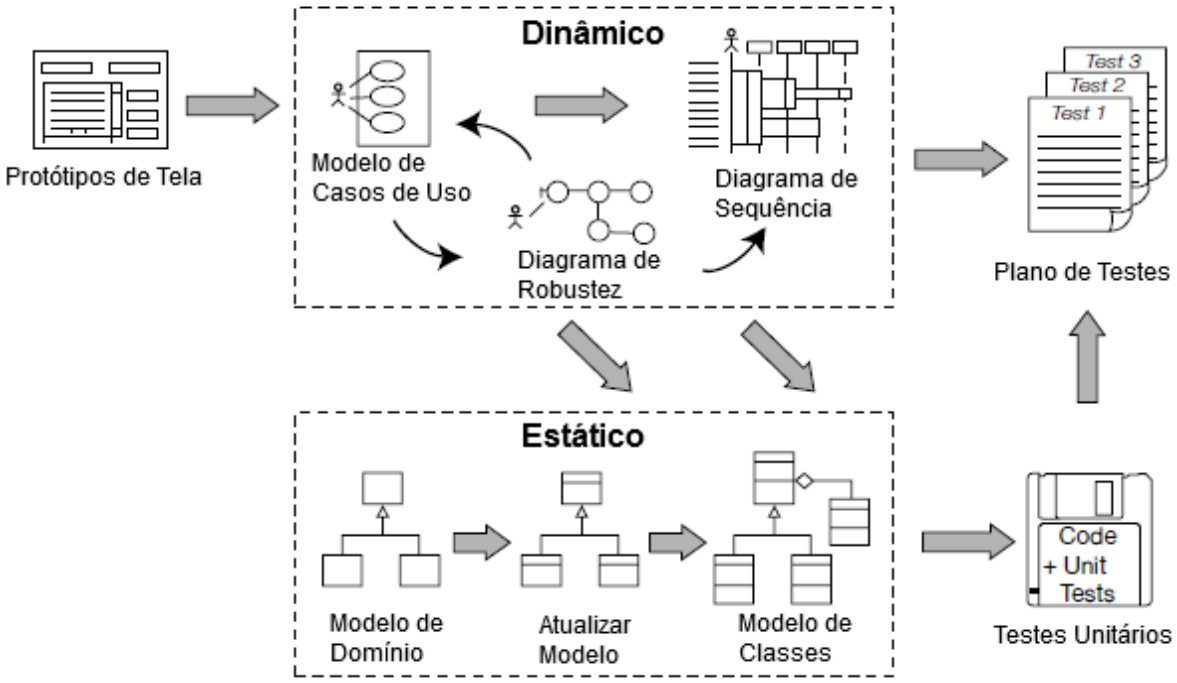


Figura 1 – Visão geral do processo ICONIX

Fonte: Adaptado de ROSENBERG; STEPHENS (2007, p. 1)

O processo, visualizado na Figura 1, consiste nos passos usuais relacionados a requisitos, análise (*design* preliminar), *design* detalhado e implementação. De acordo com Rosenberg e Stephens (2007) os passos possuem as características e responsabilidades a seguir:

1. Requisitos:
    - Identificar e definir o que o sistema será capaz de fazer, através dos requisitos funcionais;

- Modelagem do domínio, entendendo o contexto do problema;
- Definir como o usuário e o sistema deverão interagir;
- Validar se o conteúdo gerado atende as expectativas do cliente.

2. Análise (*design* preliminar):
    - Desenvolvimento do diagrama de robustez, uma representação dos objetos de um caso de uso, reescrevendo-os se necessário.
    - Atualização do modelo de domínio, de modo a descobrir novas classes, correções de ambiguidades ou alterações nos atributos.
3. *Design* detalhado:
    - Criação dos digramas de sequência, mostrando os detalhes da implementação do mesmo;
    - Atualização do modelo de domínio.
4. Implementação:
    - Desenvolvimento e criação dos testes unitários, não necessariamente nesta ordem;
    - Realizar uma revisão de código e de modelo para os próximos incrementos.

2.2 ARQUITETURA DE SOFTWARE

Arquitetura de *software* é largamente discutida e definida (CLEMENTS; SHAW, 2006, 2009), de forma objetiva, considera-se arquitetura, de acordo com a ISO/IEC 42010 e IEEE 1471-2000, a organização fundamental de um sistema inserido em seus componentes, a relação entre eles, e ao seu contexto, e os princípios que guiarão seu *design* e evolução.

O propósito das arquiteturas corporativas é fornecer a estrutura básica de tecnologias e processos, permitindo que, atinja-se um nível balanceado entre a eficiência na Tecnologia da Informação (TI) e na inovação do negócio. Confere-se os principais benefícios resultantes de uma arquitetura corporativa no Quadro 2. (TOGAF, 2011).

Área	Pontos positivos
Negócio	Diminui custos
	Torna a organização mais ágil
	Funcionalidades de negócio compartilhadas entre a organização
	Minimiza o custo de gestão
	Otimiza produtividade
TI	Baixo custo de desenvolvimento, suporte e manutenção do *software*
	Aumenta a portabilidade das aplicações
	Facilita a interoperabilidade entre as aplicações
	Melhora questões críticas gerais da organização, como segurança
	Facilita trocar e atualizar componentes de sistema

Quadro 2 - Benefícios de negócio e TI das arquiteturas corporativas

Fonte: Adaptado de TOGAF (2011)

A arquitetura é irrelevante ao usuário final do *software*, entretanto essencial para construí-lo de forma a ser entendível, estendido, reorganizado, mantido e testado (BOOCH et al, 2007). O propósito das estruturas arquitetônicas são, segundo Evans (2009, p. 70), “construir uma implementação que expresse o modelo de domínio e o utilize para resolver problemas importantes”.

Quando se percebeu que as arquiteturas são fatores críticos de sucesso para o *design* e desenvolvimento de aplicações, esta ganhou maior ênfase. Dentro da indústria foram destacadas duas tendências para o maior enfoque em arquitetura de *software*: (1) reconhecimento de um meio que compartilhasse métodos, técnicas, padrões e linguagens para aplicações complexas, e (2) explorar o conceito de requisitos comuns para promover frameworks reutilizáveis entre as aplicações. A arquitetura de software interage como uma ponte, ligando os requisitos e as implementações, como visualiza-se na Figura 2 (GARLAN, 2000).

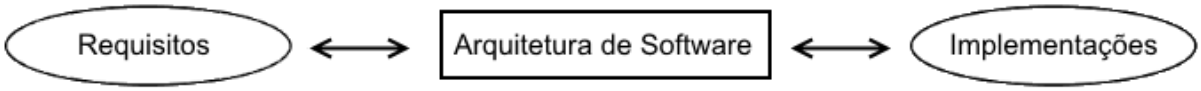


Figura 2 - Representação de arquitetura de software como uma ponte

Fonte: Adaptado de GARLAN (2000)

Os pontos chave entre os impactos gerados no desenvolvimento de aplicações por uma arquitetura de *software são*, (GARLAN, 2000; GARLAN; PERRY, 1994):

1) Entendimento: através da arquitetura simplifica-se a habilidade de compreensão de sistemas complexos;
2) Reutilização: suporte em múltiplos níveis, como bibliotecas, componentes e *frameworks*;
3) Construção: a arquitetura de *software* fornece uma planta para o desenvolvimento, indicando componentes e a dependência entre eles;
4) Evolução: entende-se mais adequadamente custos e ramificações de alterações, já que a arquitetura do *software* torna explicito as “paredes estruturais” da aplicação;
5) Análise: possibilita-se conferir a consistência e qualidade do sistema, através das delimitações da arquitetura;
6) Gestão: ao analisar a arquitetura, tipicamente conduz-se a um melhor entendimento de requisitos, estratégias de implementação e potenciais riscos.

Arquitetura em camadas e padrões de projeto, são abstrações de ideias, que permitem a implementação e utilização em diversas linguagens de programação, tecnologias e ferramentas.

2.2.1 Arquitetura em camadas

Técnica comumente utilizada entre projetistas de software, principalmente para decompor complexidade e responsabilidade, é a criação de camadas. Dentre a organização e utilização de camadas, é possível elencar, como sendo as três principais a de apresentação, domínio e fonte de dados, descritas no Quadro 3 (FOWLER, 2006).

Camada	Responsabilidades
Apresentação	Fornecimento de serviços, exibição de informações (p. ex., em Windows ou HTML, tratamento de solicitações do usuário (cliques com o mouse, pressionamento de teclas) requisições HTTP, chamadas em linha de comando, API em lotes)
Domínio	Lógica que é o real propósito do sistema

Fonte de Dados	Comunicação com o banco de dados, sistema de mensagens, gerenciadores de transações, outros pacotes

Quadro 3 - Três Camadas Principais

Fonte: FOWLER (2006, p. 41)

Não existe quantidade obrigatória de camadas em uma arquitetura para ser considerada correta, a variação é em função dos tipos de aplicações que farão uso da arquitetura, podendo uma destas ter mais camadas do que outra. As três camadas principais, apresentadas no Quadro 3, funcionam como uma espécie de ideia inicial e não regra absoluta, podendo ser alteradas em função das necessidades a serem resolvidas, a exemplo analisa-se na Figura 3 uma extensão das três camadas principais.

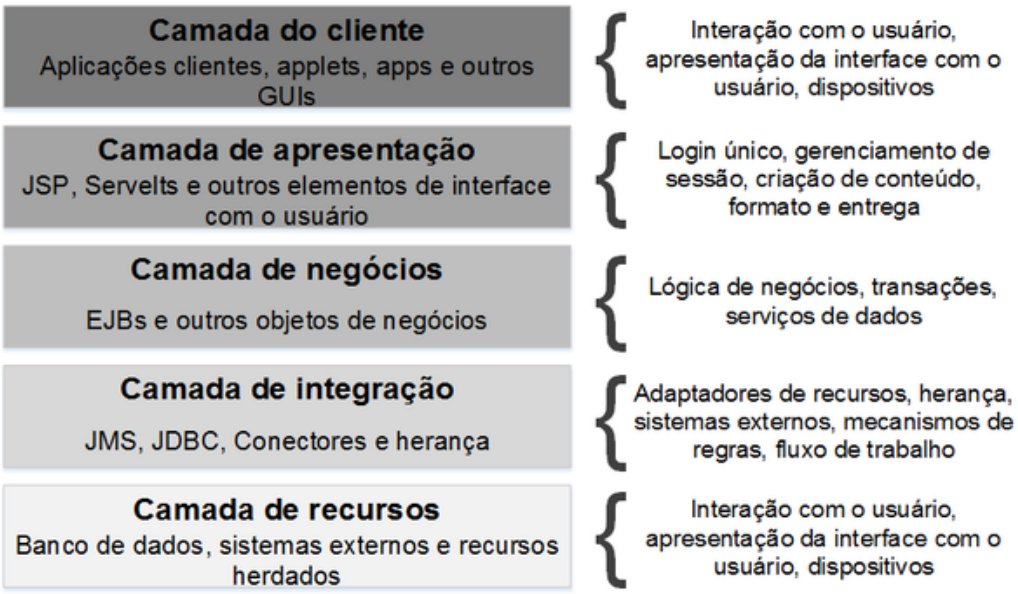


Figura 3 - Modelo de camadas para divisão lógica em uma aplicação Java

Fonte: Adaptado de ALUR, CRUPI e MALKS (2004, p. 102)

Menção importante referente a camadas, é a relação entre os termos "*layer*" e "*tier*". Ambas as palavras significam "camada" na língua inglesa, e segundo Fowler (2006), algumas vezes são utilizadas como sinônimos, entretanto, para o termo "*tier*" implica-se uma separação física, como por exemplo aplicações cliente-servidor, enquanto que, para o termo "*layer*" não é necessário que exista a separação física, apenas lógica, de forma que todas as camadas podem estar no mesmo local.

A arquitetura em camadas é uma das abordagens para resolver problemas relacionados a criação de aplicações, porém mesmo problema pode ser solucionado através de diversos tipos e formas de arquiteturas (GARLAN, SHAW; 1994; PERRY, WOLF; 1992). Utiliza-se camadas em um contexto para criar e evoluir independentemente partes de uma aplicação, de forma a suportar maior portabilidade, facilitar alterações e reuso destas (BASS, CLEMENTS, KAZMAN, 2012).

Em virtude das alterações frequentes no contexto das aplicações (através de novas interfaces de acesso, otimizações de hardware, integrações com sistemas, etc.) e que as regras de negócio destas aplicações tende a manter-se estável por maior tempo (BROOKS, 1986), a arquitetura em camadas é uma estratégia adequada. Segundo Evans (2009, p. 65), “o valor das camadas é que cada uma se especializa em um determinado aspecto de um programa de computador”, potencializando designs coesos e facilitando seu entendimento (EVANS, 2009).

A cada camada se atribui uma responsabilidade distinta, ou única para a aplicação. Existem restrições na relação de uso entre as camadas que compõem a arquitetura da aplicação, sendo necessário que a relação entre elas seja unidirecional; a Figura 4 ilustra o comportamento que descaracteriza uma arquitetura em camadas, porque todas as elas acessam-se entre si, não evidenciando a separação de responsabilidades para camada (ALUR, CRUPI e MALKS, 2004; BASS, CLEMENTS, KAZMAN, 2012).

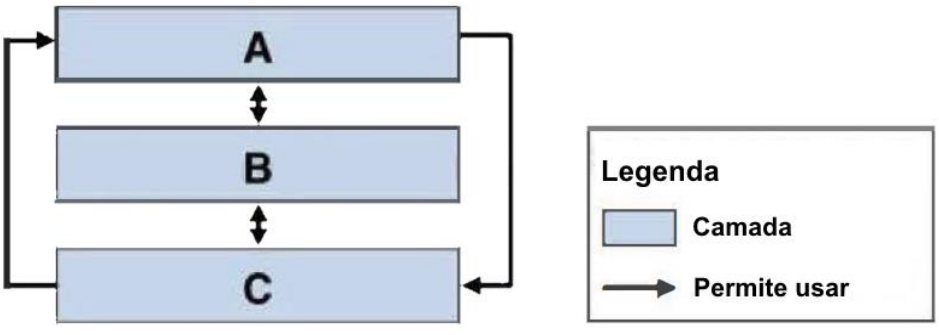


Figura 4 - Exemplo de uma não arquitetura em camadas

Fonte: Adaptado de BASS, CLEMENTS, KAZMAN (2012, p. 208)

Uma estrutura de camadas empilhada uma sobre a outra não constitui uma arquitetura em camadas, é fundamental observar a relação na utilização entre elas,

para que, a implementação em uma camada utilize o software da mesma camada ou a camada imediatamente abaixo, ou ainda a camada a direita (BASS, CLEMENTS, KAZMAN; 2012), como pode visualiza-se na Figura 5.

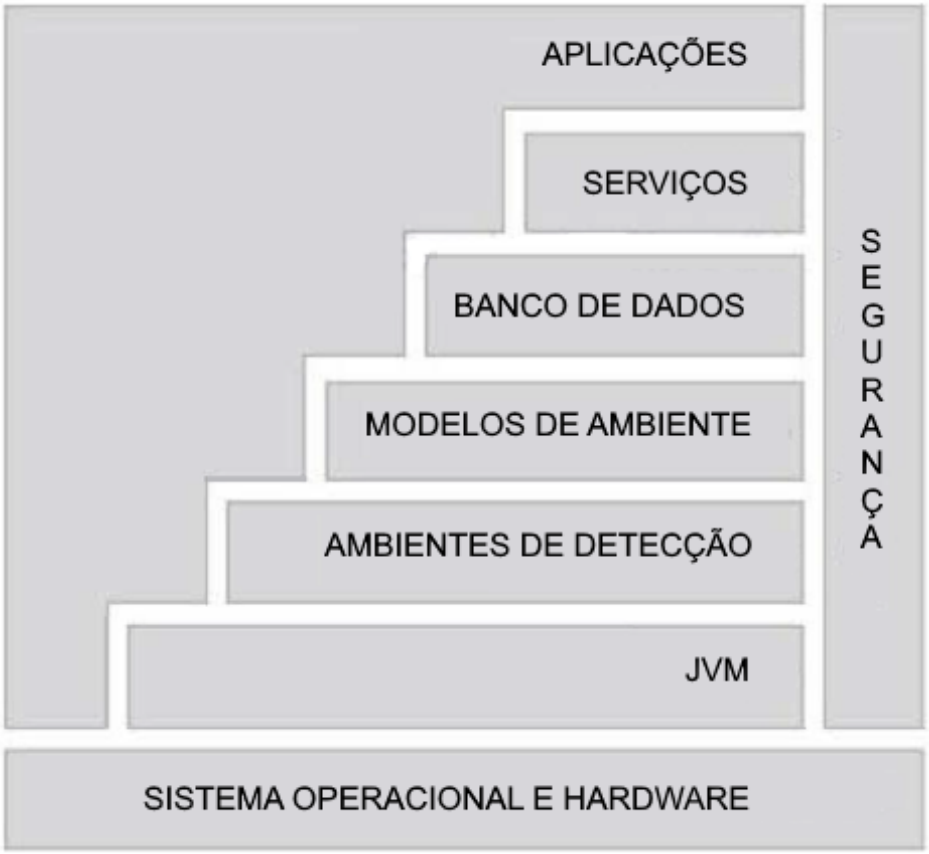


Figura 5 - Exemplo de diagrama de camadas

Fonte: Adaptado de BASS, CLEMENTS, KAZMAN (2012, p. 207)

2.2.2 Padrões de projeto

Padrões são uma entidade de linguagem que foram destiladas através de situações práticas ao longo do tempo, vinculado a esforços no planejamento e construção de edificações arquitetônicas, como casas, edifícios, escolas e escritórios. O conceito de padrões de projeto em computação, tem como origem a área de arquitetura e urbanismo, onde o padrão descreve um problema que ocorre repetidas vezes em um contexto, bem como a ideia chave de solução para o problema, de modo que, se utiliza esta solução diversas vezes, sem a obrigatoriedade de ser implementada da mesma maneira (ALEXANDER et al 1977). Segundo Gama et al (2000, p. 19) “nossas soluções são expressas em termos de objetos e interfaces em

vez de paredes e portas, mas no cerne de ambos os tipos de padrões está a solução para um problema num determinado contexto".

A separação de responsabilidades foi abordada na seção 2.2.1 Arquitetura em camadas, sendo que de alguma maneira estas devem conectar-se entre si, de modo a não perder os benefícios da separação, sendo esta a motivação de diversos padrões, mas não a única (EVANS, 2009), sendo essencial compreender mais do que padrões isolados, e focar em melhores práticas para alavancar a criação de grandes soluções (ALUR, CRUPI e MALKS, 2004). Objetivamente, Larman (2007, p. 295) define padrão como "[...] par problema/solução denominado e bem conhecido que pode ser aplicado em novos contextos [...]". Como e quando utilizá-los é descrito nas bibliografias do tema, assim como os benefícios, custos e consequências de sua utilização. Alguns dos padrões de projeto são utilizados de maneira conjunta, enquanto outros como alternativa[2].

O restante deste capítulo explica e detalha os padrões de projeto que possuem relação com este trabalho.

2.2.2.1 MVC

O padrão Modelo Vista Controlador (*Model View Controller*) (MVC) possui "influência na maioria dos frameworks para a interface com o usuário e no pensar sobre o projeto de interfaces com o usuário" (FOWLER, 2006, p. 315). Este padrão de apresentação web "divide a interação da interface com o usuário em três papéis distintos" (FOWLER, 2006, p. 315), como visualiza-se na Figura 6.

[2] O relacionamento entre alguns padrões de projeto visualiza-se em GAMMA et al (2000, pag. 27), e com divisão dos padrões por camada em ALUR, CRUPI e MALKS (2004, p. 113)

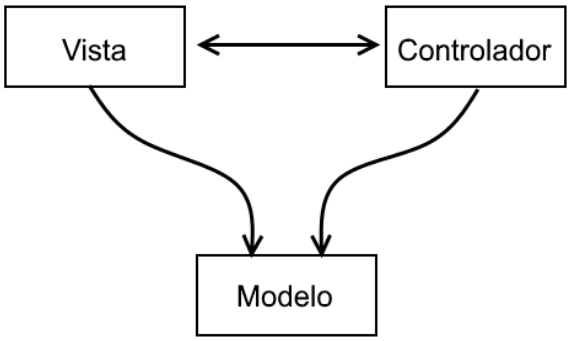


Figura 6 - Divisão da interação com o usuário em três papéis distintos

Fonte: FOWLER (2006, p. 315)

De forma objetiva, e sem o papel do controlador para simplificar, a Figura 7 apresenta uma ilustração do potencial do padrão, sendo possível ligar múltiplas visões a um mesmo modelo, fornecendo assim, diferentes apresentações (GAMMA et al, 200).

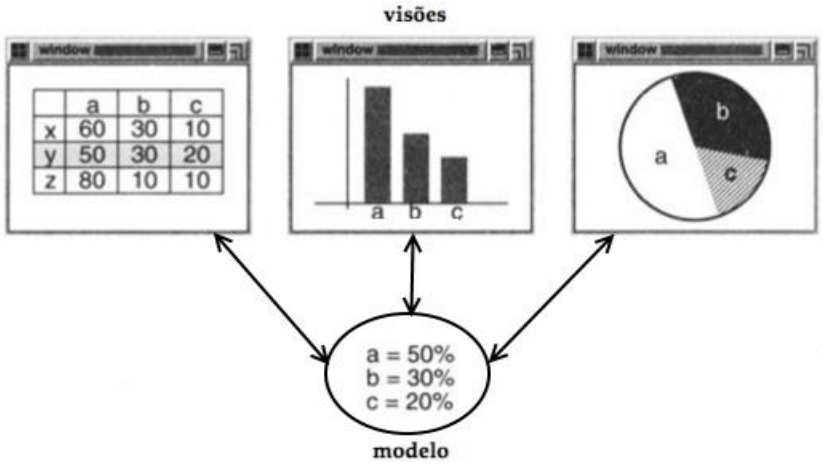


Figura 7 - Ilustração do papel Modelo e Visão do padrão MVC

Fonte: GAMMA et al (2000, pag. 21)

2.2.2.2 Modelo de domínio

Incorpora tanto o comportamento quanto os dados, através dos objetos que compõem o modelo. Este padrão regularmente é representado por uma camada e se relaciona com objetos e regras de negócio, sendo que, estas regras possuem

diferentes casos e formas, variando de aplicação para aplicação. Apesar de poder existirem diferenças, este modelo frequentemente reflete o modelo de banco de dados (FOWLER, 2006; LARMAN, 2007).

O modelo de domínio “cria uma rede de objetos interconectados em que cada um representa algum conceito significativo” (FOWLER, 2006, p. 126), de tal forma que agrega valor à aplicação. Segundo Evans (2009), esta camada funciona como o coração do software, e quanto maior o isolamento e baixo acoplamento com tecnologias melhor, pois é esta camada que abriga o verdadeiro patrimônio do software.

A Figura 8 ilustra um modelo de domínio em construção, sem as possíveis validações de negócio.

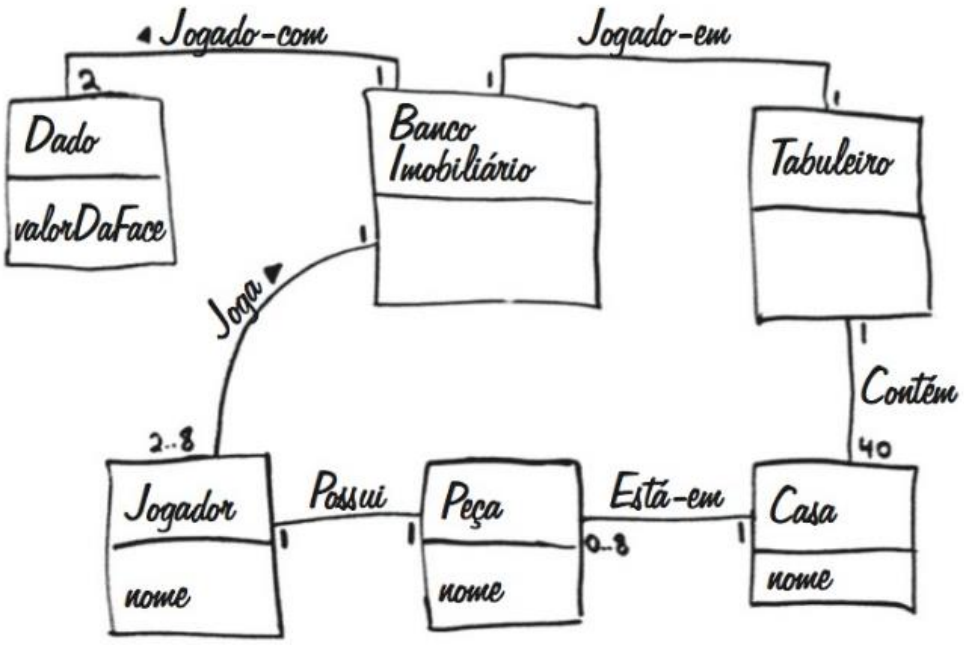


Figura 8 - Modelo parcial de domínio do Banco Imobiliário

Fonte: LARMAN (2007, p. 192)

2.2.2.3 Repositórios

Repositórios representam objetos do domínio que são persistidos e as operações realizadas sobre eles, através de uma visão orientada a objetos da camada de acesso aos dados. É possível alcançar uma separação limpa entre o domínio e o

local de armazenamento das informações[3], através do desacoplamento da tecnologia de persistência, de várias estratégias do banco de dados e até das várias fontes de dados (EVANS, 2009; FOWLER, 2006).

"Um repositório retira um grande fardo das costas do cliente, que agora pode se comunicar com uma interface simples, reveladora de intenções, e pedir o que precisa em termos do modelo" (EVANS, 2009, p. 144), e por consequência, "os clientes nunca precisam pensar em SQL e podem escrever código puramente em termos de objetos" (FOWLER, 2006, p. 311).

Este padrão possibilita a implementação de diversas fontes de dados, o que segundo Fowler (2006), é outra situação que evidência o seu valor. Na Figura 9 é visualizado um exemplo de implementação em Java com dois repositórios de dados, um faz uso de uma estratégia relacional (como um banco de dados), enquanto o outro implementa uma estratégia de objetos em memória. Referente ao exemplo da Figura 9, a utilização da interface EstrategiaDeRepositorio na aplicação, permitiria trocar o local de armazenamento sem a necessidade de refatorar a aplicação, bastando alterar o local onde vincula-se a classe concreta na interface. Este padrão pode ser associado a outros padrões, como injeção de dependência e fábricas, com o intuito de facilitar a troca de locais de armazenamento.

[3] Tipicamente um banco de dados, entretanto, poderiam ser objetos serializados em arquivos ou mesmo mantidos em memória.

```
abstract class Repositorio{
    private EstrategiaDeRepositorio estrategia;

    protected List satisfazendo(Criterios algunsCriterios){
        return estrategia.satisfazendo(algunsCriterios);
    }
}

public class EstrategiaRelacional implements EstrategiaDeRepositorio{
    protected List satisfazendo(Criterios criterios){
        Pesquisa pesquisa = new Pesquisa (classeDoMeuObjetoDeDominio);
        pesquisa.adicionarCriterios(criterios);
        return pesquisa.executar();
    }
}

public class EstrategiaEmMemoria implements EstrategiaDeRepositorio{
    private Set objetosDoDominio;

    protected List satisfazendo(Criterios criterios){
        List resultados = new ArrayList();
        Iterator it = objetosDoDominio.iterator();

        while(it.hasNext()){
            ObjetoDeDominio cada = (ObjetoDeDominio) it.Next();

            if(criterios.saoSatisfeitosPor(cada)){
                resultados.add(cada);
            }
        }

        return resultados;
    }
}
```

Figura 9 – Exemplo em Java de repositórios com duas fontes de dados

Fonte: Adaptado de FOWLER (2006, p. 313)

2.2.2.4 Serviços

Serviços são tipicamente organizados como uma camada, que define uma fronteira da aplicação, bem como um conjunto de operações disponíveis, encapsulando a lógica de negócio da aplicação criando um design limpo e pragmático, minimizando o risco de um objeto de domínio perder sua clareza conceitual, além de, potencializar a pureza e reutilização destes objetos dentro da aplicação (EVANS, 2009; FOWLER, 2006). Um benefício, pelo qual é indicado a utilização deste padrão, é por definir “um conjunto comum de operações da aplicação disponível para muitos tipos de clientes e coordenar uma resposta da aplicação em cada operação”

(FOWLER, 2006, p. 144). A Figura 10 representa a implementação de uma camada de serviço, e é perceptível que estas são acessadas por diferentes clientes como carregadores de dados interfaces com os clientes e portas de integração.

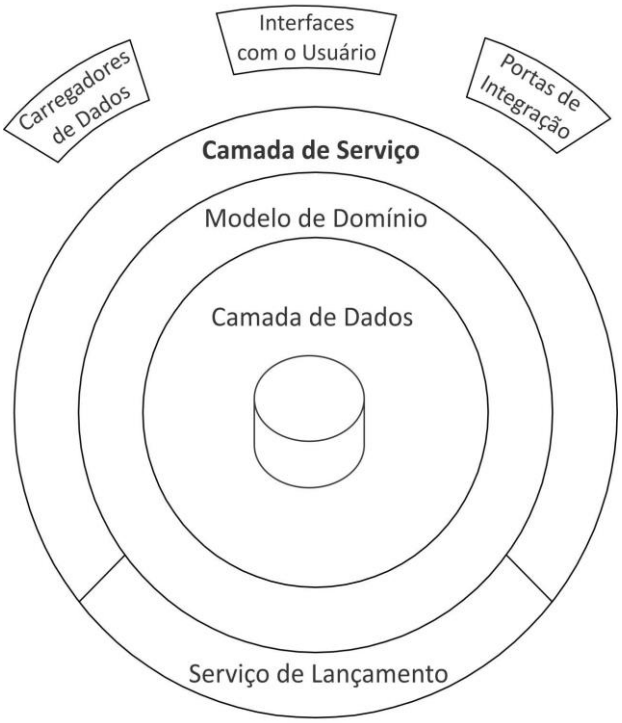


Figura 10 - Camada de serviço em uma aplicação

Fonte: FOWLER (2006, p. 141)

A divisão de serviços entre camadas é sugerida por Evans (2009), de forma a cada um manter para si a responsabilidade da camada no qual está inserido, como por exemplo serviços nas camadas de apresentação (digere entradas em JSON), domínio (interage com a regra de negócio da aplicação) e infraestrutura (notificações por e-mail).

2.2.2.5 Camada supertipo

A Camada Supertipo abstrai atributos, métodos e comportamentos semelhantes a um conjunto de objetos, sendo possível criar “um tipo que atua como supertipo para todos os objetos de sua camada” (FOWLER, 2006, p. 444).

Segundo Fowler (2006) é uma ideia simples que conduz a um padrão curto, porém traz benefícios ao se utilizar de características comuns a diversos objetos.

Um exemplo recorrente seriam objetos de domínio com o atributo identificador (ID), sem uma classe supertipo, faz-se necessário defini-lo em cada um dos objetos que utilizariam o atributo, ao invés de abstrai-lo em uma classe reutilizável entre os objetos de domínio.

2.2.2.6 Injeção de dependência

A injeção de dependência é uma das formas de implementação do padrão de inversão de controle (*Inversion of Control*, IOC) (FOWLER, 2005). Este padrão potencializa o meio para que arquiteturas de software e *frameworks* ajam como esqueletos extensíveis, delegando as aplicações que as utilizam a decisão dos detalhes e formas da implementação, de modo a promover o design reutilizável e o baixo acoplamento (JOHNSON; FOOTE, 1988).

```
public interface IBancoDeDados
{
    /* Métodos */
}

public class BancoDeDadosComEntityFramework : IBancoDeDados
{
    /* Implementação utilizando a Entity Framework */
}

public class BancoDeArquivosXML : IBancoDeDados
{
    /* Implementação utilizando arquivos serializados em XML */
}
```

Figura 11 - Código fonte de exemplo para utilização de injeção de dependência

Fonte: Elaborado pelo autor.

A Figura 11 mostra uma implementação na qual seria possível utilizar a injeção de dependência e não acoplar a tecnologia de bando de dados na aplicação, pois a utilização acontece por meio da interface, que posteriormente é associada a uma classe concreta que a implementa. A associação entre classe e interface é tratado

bibliotecas dedicadas apenas a este padrão, como por exemplo StructureMap e Ninject mas não unicamente.

2.2.3 Frameworks

Uma estrutura reutilizável que sustenta, de modo geral, funcionalidades de diversas aplicações, é definida como um *framework*, segundo a ISO/IEC/IEEE 24765. *Frameworks* devem agrupar funcionalidades comuns entre aplicações, e permitir que estas e outras aplicações os utilizem como parte de sua estrutura, e caso necessário, estender os recursos conforme a demanda (BOOCH et al, 2007; SAUVÉ, [s.d.]a). A Figura 12, ilustra aplicações em dois possíveis cenários, um em que as aplicações possuem baixa relação de funcionalidades semelhantes entre si, e outro onde fica evidenciado, pela intersecção das aplicações, as funcionalidades comuns.

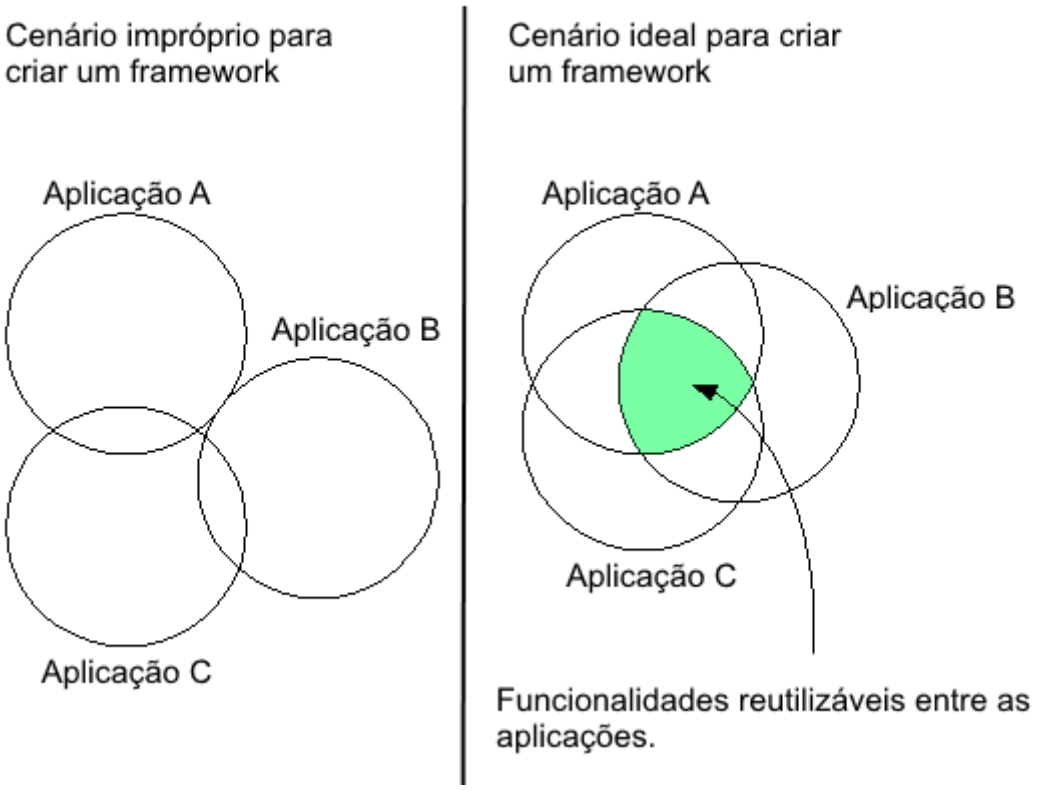


Figura 12 - Cenários de aplicações e a relação com *framework*

Fonte: Adaptado de SAUVÉ ([s. d.]a)

Ainda que os cenários descritos na Figura 12 envolvam várias aplicações, é possível criar *frameworks* a partir de uma aplicação, ao considerar que, as aplicações tendem a possuir alguns requisitos não-funcionais compartilhados entre si, sendo uma boa prática de *design* conseguir reutilizá-los, sendo estes *frameworks* incrementados ao longo do tempo, na medida que outras funcionalidades comuns forem identificadas entre as aplicações que o utilizam.

O desenvolvimento de *frameworks* sacrifica a implementação de funcionalidades em projetos específicos no curto prazo, pois o esforço está voltado para a criação e organização de uma estrutura reutilizável, mas em longo prazo tende a trazer retornos (BOOCH et al, 2007; COCKBURN, 1998; SAUVÉ, [s. d.]b). Somente a priorização de funcionalidades sem *design* para economizar tempo, tende a potencializar um débito técnico (CUNNINGHAM, 1992; FOWLER, 2003b; 2007).

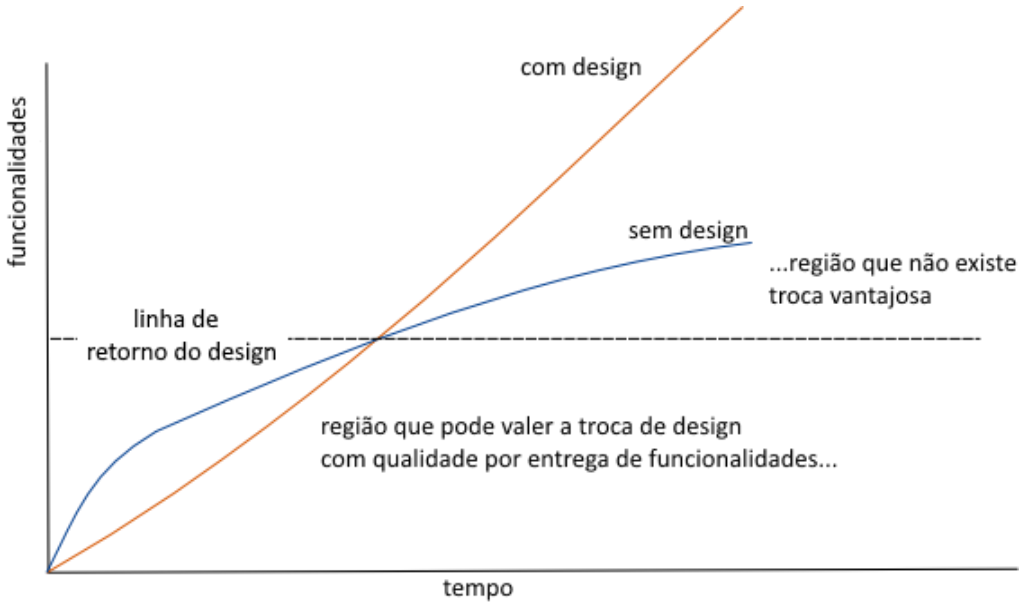


Figura 13 - Hipótese de esforço gasto em design de software

Fonte: Adaptado de FOWLER (2007)

De acordo com o pseudográfico, representado na Figura 13, que expressa uma hipótese de energia utilizada em *design*, se visualiza incrementos de funcionalidades ao longo do tempo, em *software* com e sem *design*, além de que a partir da linha de retorno do *design*, este se torna uma vantagem ao desenvolvimento.

O *framework* não é necessariamente uma peça única, que a tudo suporte, e seja o pilar estrutural de todas as aplicações construídas na empresa, a Figura 14 mostra um relacionamento entre vários *frameworks* e aplicações, de modo que, cada aplicação utiliza o necessário (SAUVÉ, [s. d.]c).

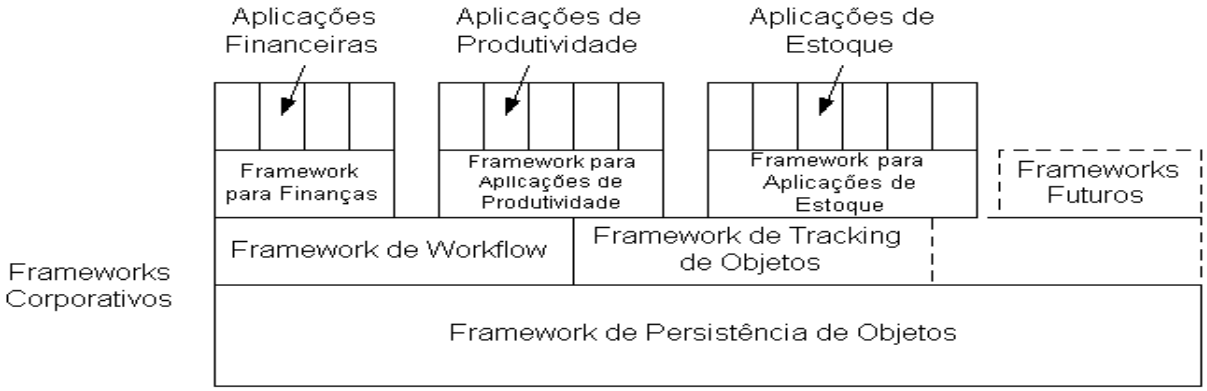


Figura 14 - Relação entre frameworks e aplicações

Fonte: SAUVÉ ([s. d.]c)

2.3 FERRAMENTAS E ESTRUTURAS

Na criação e manutenção de aplicações, ferramentas de desenvolvimento e estruturas são utilizadas, como *frameworks* e bibliotecas.

2.3.1 .NET Framework

É um *framework* desenvolvido pela Microsoft, para suportar a criação e execução de aplicações, nas linguagens de programação Visual Basic, C#, F# e C++. As linguagens sob o .NET respeitam a especificação[4] *Common Language Infrastructure* (CLI). O framework tem como objetivos prover um ambiente consistente para programação orientada a objetos, minimizar conflitos em *deploy*[5] e versões de *software*, criar uma experiência consistente no desenvolvimento de aplicações *web* e

[4] Informações detalhadas: ECMA C# and Common Language Infrastructure Standards (https://www.visualstudio.com/en-us/mt639507)
[5] O *deploy* trata da entrega ou implantação do conjunto desenvolvido ao cliente final do software.

desktop, promover a execução segura dos códigos fontes e integrar as aplicações em .NET com quaisquer outras aplicações. (MSDN, [s. d.]b ,[s. d.]c).

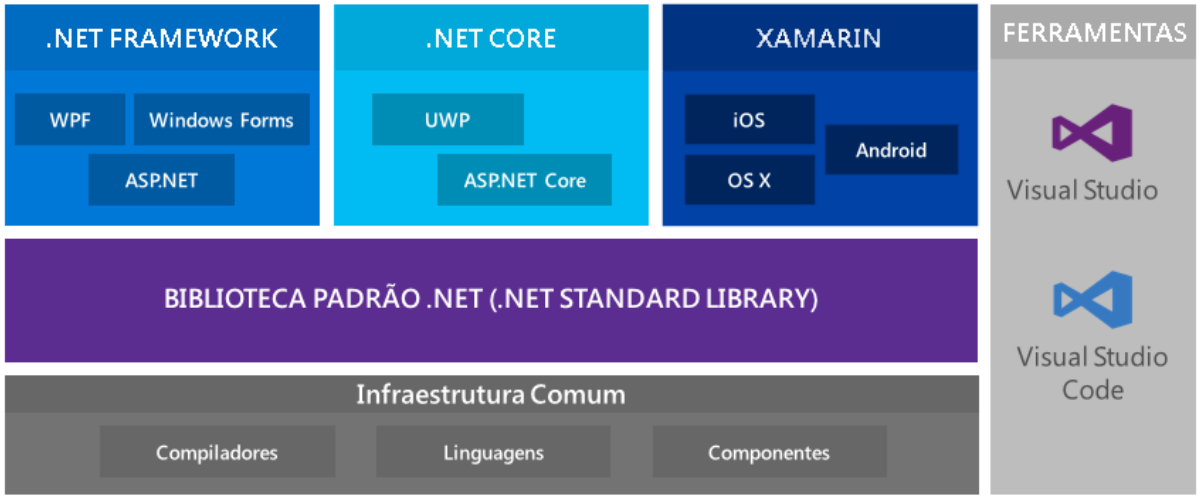


Figura 15 - Visão futura do .NET

Fonte: Adaptado de BUILD (2016b)

O .NET foi reestruturado para ser *open source*[6] e *cross platform*, permitindo que aplicações executem em ambientes Linux e iOS. Duas ressalvas a respeito do código aberto, são: (1) contribuições de terceiros ao *framework* passam pelo processo de desenvolvimento da Microsoft, ou seja, uma eventual contribuição pode não ser aceita, e (2) não foi disponibilizado o código fonte total do *framework*. O ambiente *cross platform* está em testes, mas já pode ser utilizado. Ao final da reestruturação o .NET terá uma biblioteca comum, chamada de .NET Standard Library, como pode ser visualizado na Figura 15 (BUILD, 2016b).

2.3.2 Ambientes de desenvolvimento

O Visual Studio atualmente na versão 2015 e possui cinco variações[7], é um ambiente integrado de desenvolvimento (*Integrated Development Environment*) (IDE), que suporta a criação de *software* (desde a fase de planejamento até o design da interface com usuário), testes, *debug* e analise de qualidade e performance de

[6] Repositório dos códigos fontes é disponibilizado pela .NET Foundation em https://github.com/dotnet
[7] Comparação entre as versões do Visual Studio 2015 em https://www.visualstudio.com/en-us/products/compare-visual-studio-2015-products-vs.aspx

códigos. (MSDN, [s. d.]d). Entre as variações a Community se destaca por ser gratuita, e a Code por ser *cross platform*. Através da IDE é possível desenvolver aplicações para nuvem, *web*, *desktop*, *mobile*, além de outras (MSDN, [s. d.]a). As aplicações *mobile* podem ser executadas em Android, iOS e Windows Phone, sendo possível utilizar o compartilhamento do núcleo da aplicação entre as interfaces de usuário dos aplicativos, como pode ser visualizado na Figura 16.

Figura 16 - Código compartilhado entre aplicações Android, iOS e Windows

Fonte: MSDN ([s. d.]e)

O OmniSharp, um projeto independente e *open source*, que reúne um conjunto de ferramentas, bibliotecas e editores para permitir o desenvolvimento de aplicações em .NET, indiferentemente do sistema operacional utilizado (OMNISHARP, [s. d.]). Entre os editores disponíveis estão: ATOM, Brackets, Yeoman, Emacs, Sublime, Vim e Visual Studio Code.

2.3.3 ASP .NET

Framework que possibilita criar aplicações Web, é *open-source* e recentemente foi reestruturado para ser *cross-platform*, assim é possível executa-lo nos ambientes de computação em nuvem da Amazon, Azure e Docker, por exemplo (BUILD, 2016a), através da estratégia ASP.NET Core, que pode ser visualizada na Figura 17.

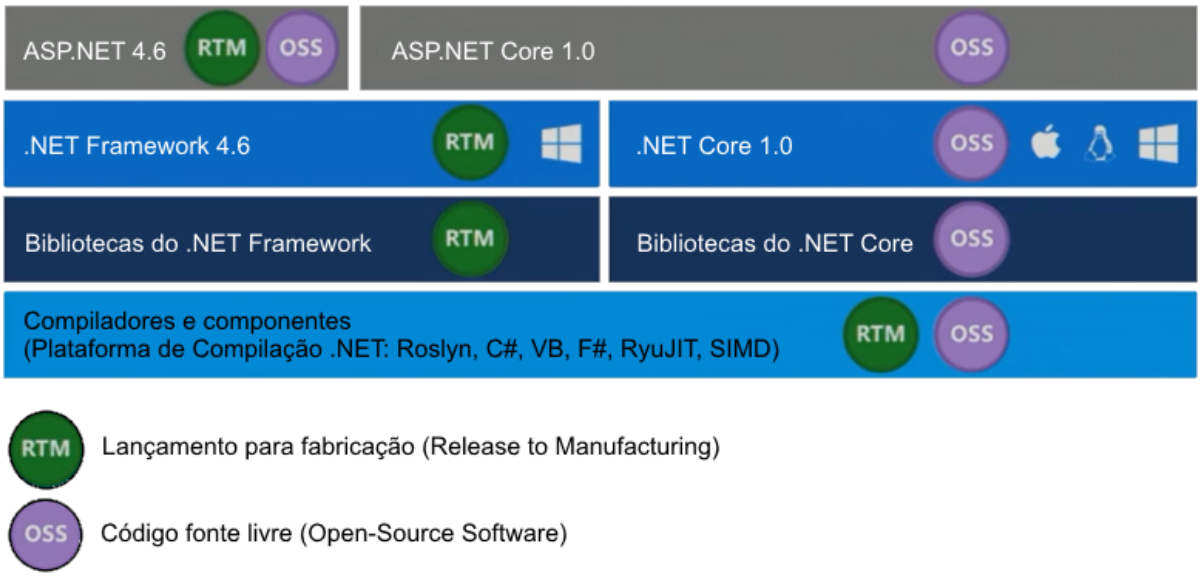


Figura 17 - ASP.NET e ASP.NET Core

Fonte: Adaptado de BUILD (2016a)

As principais funcionalidades do ASP.NET são:

1. Website padrão: aplicando HTML5, CSS3 e JavaScript, por exemplo (ASP.NET [s. d.]a);
2. Mobile: por meio de aplicações nativas aliado e uma Web API de back end (ASP.NET [s. d.]b);
3. Web API: através de um framework que permite construir serviços HTTP. Considera-se a plataforma ideal para construção de aplicações RESTfull sobre o .NET (ASP.NET [s. d.]c).
4. SignalR: recurso que permite adicionar a aplicações Web a funcionalidade de tempo real, entre cliente e o servidor, de maneira mais fácil e otimizada (ASP.NET 2014a).

O ASP.NET suporta o desenvolvimento web pelos *frameworks* Web Pages, Web Forms e MVC no ASP.NET, que variam entre estilos de desenvolvimento e nível de experiência, conforme visualizados no Quadro 4. Apesar de distintos entre os estilos, os três *frameworks* foram desenvolvidos sob o .NET, e compartilham funcionalidades comuns como controles de *login* e segurança, gerenciamento de chamadas e manipulação de sessões (ASP.NET [s. d.]a).

Framework	Estilo de desenvolvimento	Familiaridade
Web Pages	Sintaxe HTML padrão, linguagem .NET junto com o arquivo HTML.	ASP clássico e PHP
Web Forms	Permite utilizar bibliotecas com controles que encapsulam a sintaxe HTML	Win Forms, WPF e .NET
MVC	Controle total sobre a sintaxe HTML. Permite separar a linguagem .NET do arquivo HTML e a criação de testes. Aconselhável para aplicações *mobile* e *single-page applications* (SPA)	Ruby on Rails e .NET

Quadro 4 - Tipos de frameworks ao desenvolver em ASP.NET

Fonte: Adaptado de ASP.NET ([s. d.]a)

A sintaxe Razor possibilita o vínculo do .NET a uma página web, e executa comandos de linguagem de programação C# ou VB.NET dentro da própria página que será exibida no navegador. Ao utilizar os recursos de programação .NET dentro de uma página web, o servidor executa este código fonte antes de envia-lo ao cliente (ASP.NET, 2014b). A Figura 18 ilustra um exemplo de código fonte em uma página *web*, utilizando HTML e a linguagem C#.

```
1   @{ DateTime hoje = DateTime.Now; }
2
3   <h3> Hoje é: @hoje.ToShortDateString() - @hoje.DayOfWeek </h3>
4
5   <p><b>Os dias da semana de @hoje.Day/@hoje.Month para os 10 anos seguintes são:</b></p>
6
7   <ul>
8       @{
9
10          for (int i = 1; i <= 10; i++)
11          {
12              <li>@hoje.AddYears(i).DayOfWeek.ToString()</li>
13          }
14      }
15  </ul>
```

Figura 18 - Exemplo de página utilizando HTML e a sintaxe Razor

Fonte: Elaborado pelo autor

2.3.4 Gerenciadores de pacotes

Existem alguns *frameworks* e bibliotecas disponíveis ao desenvolvimento de aplicações. Ferramentas que controlam o vínculo entre bibliotecas/*frameworks* e o uso nas aplicações são denominadas gerenciadores de pacotes, como por exemplo o npm[8], Bower e NuGet. O Visual Studio faz uso do Bower, para controle de bibliotecas e *frameworks* relacionados a JavaScript, e o NuGet para o restante das dependências. Através destes gerenciadores é possível pesquisar, instalar, atualizar e remover partes terceiras à aplicação (NUGET, [s. d.]a).

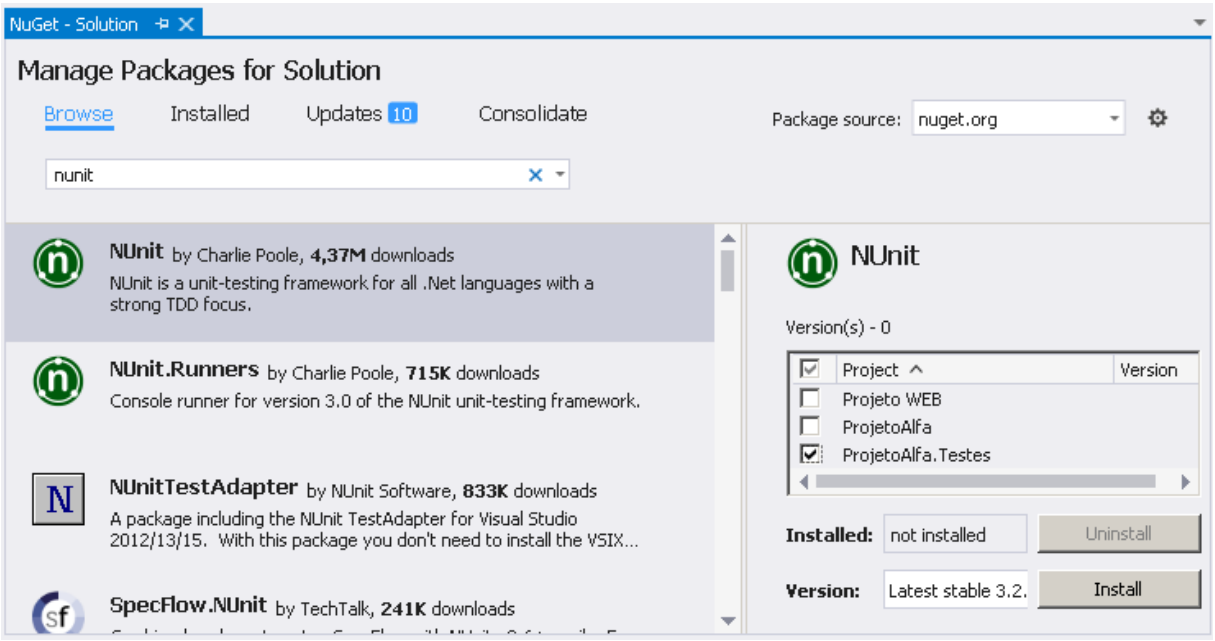


Figura 19 - Gerenciamento de pacotes pelo NuGet

Fonte: Elaborado pelo autor.

A Figura 19 exibe a interface do NuGet e os pacotes relacionados a busca pela palavra “nunit”, além dos projetos vinculados a *solution*[9] de trabalho, para manipular as dependências. A Figura 19, expressa o exemplo da instalação do pacote de testes NUnit para a aplicação ProjetoAlfa.Testes.

[8] Gerenciador de pacotes npm: https://www.npmjs.com/
[9] Solution é uma estrutura organizacional de projetos no Visual Studio, mais detalhes encontram-se em https://msdn.microsoft.com/en-us/library/b142f8e7.aspx

Este gerenciamento de pacotes pode ser aplicado internamente, o NuGet Package Explorer[10] permite criar pacotes customizados e utilizá-los no gerenciador, assim, ao desenvolver um recurso reutilizável sua instalação e atualização pode ser realizada de forma automatizada, entre os diversos projetos de uma organização por exemplo. O compartilhamento dos pacotes criados não é obrigatório a nível global, sendo viável utilizar a estrutura do NuGet em ambientes internos, possibilitando que, manter a mesma organização de gerenciamento de pacotes paralelo ao servidor oficial (MSDN, [s. d.]f, [s. d.]g; NUGET, [s. d.]a, [s. d.]b).

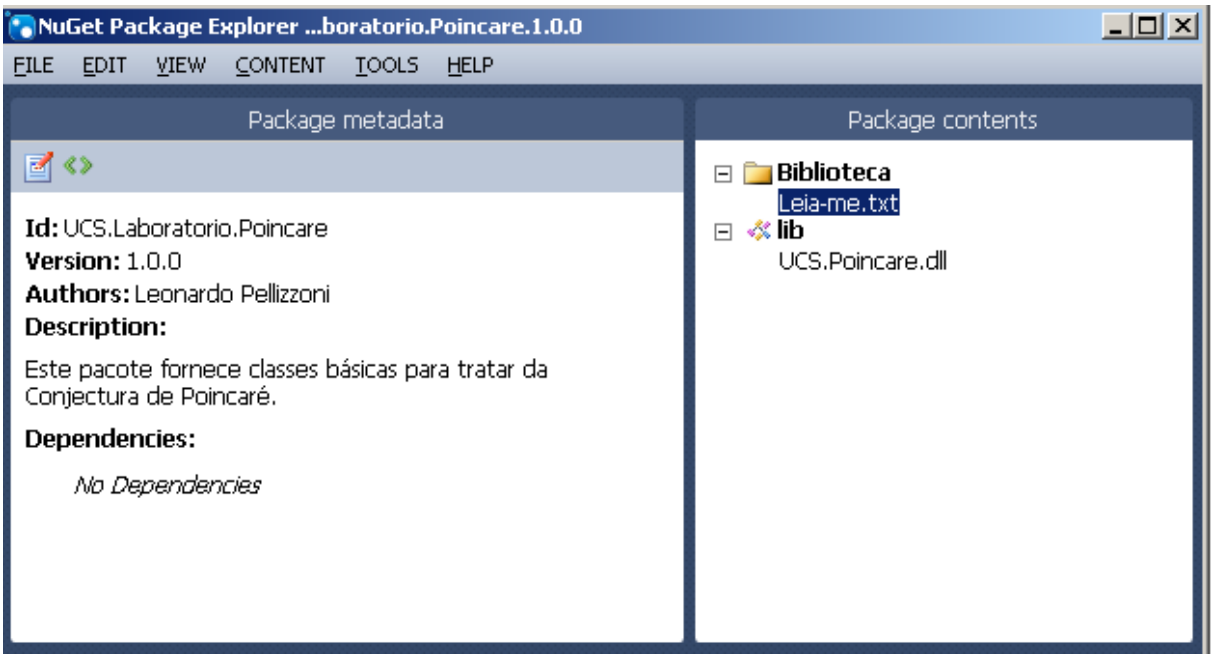


Figura 20 - Criação de pacote no NuGet Package Explorer

Fonte: Elaborado pelo autor.

A interface do software NuGet Package Explorer e a estrutura de criação de um pacote fictício é ilustrada na Figura 20.

Após criados os pacotes das funcionalidades reutilizáveis, estes devem ser hospedados em servidores para permitir o acesso compartilhado entre outras aplicações. A disponibilização pode acontecer de três formas principais, no servidor oficial[11] do NuGet, em um site mantido e controlado internamente na organização (por

[10] Site oficial do projeto em https://github.com/NuGetPackageExplorer/NuGetPackageExplorer
[11] Repositório oficial dos pacotes NuGet em: https://www.nuget.org/packages

meio do NuGet Server, solução gratuita e *open source*), ou através do myget que funciona como uma intersecção das formas anteriores, e possibilita hospedar um repositório de pacotes gratuitamente, com algumas restrições (MYGET, [s. d.]).

2.3.5 Frameworks de persistência

Um *framework* de mapeamento objeto relacional (*Object-Relational Mapping*) (ORM) cria um vínculo bidirecional entre objetos da aplicação e o banco de dados, mapeando classes e propriedades a tabelas e campos. Abstrai funcionalidades de comandos *Structured Query Language* (SQL), tomando para si a responsabilidade de manipular o conteúdo do banco de dados, de acordo com os objetos da aplicação (HIBERNATE, [s. d.]; ORACLE, 2013).

O Entity Framework (EF) e o NHibernate, adaptação para o .NET da versão Hibernate utilizada em Java, são exemplos de *frameworks* ORM, ambos *open source* (CODEPLEX, [s. d.]; NHIBERNATE, [s. d.]a). O EF disponibiliza também ferramentas para alterar a estrutura do banco de dados de forma independente dos objetos da aplicação, por meio de mecanismos de migrações; desta forma este recurso poderia ser utilizado para evoluir a estrutura e o conteúdo do banco de dados (DEVART, 2012).

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
                   assembly="FirstSolution"
                   namespace="FirstSolution.Domain">

  <class name="Product">
    <id name="Id">
      <generator class="guid" />
    </id>
    <property name="Name" />
    <property name="Category" />
    <property name="Discontinued" />
  </class>

</hibernate-mapping>
```

Figura 21 - Exemplo de mapeamento de classe no NHibernate

Fonte: NHibernate ([s. d.]b)

O mapeamento de classes da aplicação que realizam o vínculo entre uma tabela e classe, é exemplificado na Figura 21 para o NHibernate, através de arquivo XML, e na Figura 22 para o EF, utilizando a própria linguagem C# do .NET.

```
class MapDBCliente: EntityTypeConfiguration<Cliente>
{
    public MapDBCliente()
    {
        Property(p => p.Nome).HasColumnName("NOME");
        Property(p => p.RazaoSocial).HasColumnName("RAZAOSOCIAL");
        Property(p => p.TipoDaInscricao).HasColumnName("TIPOINSCRICAO");
        Property(p => p.Inscricao).HasColumnName("INSCRICAO");
        Property(p => p.Endereco).HasColumnName("ENDERECO");
        Property(p => p.Cidade).HasColumnName("CIDADE");
        Property(p => p.Telefone).HasColumnName("TELEFONE");
        Property(p => p.Email).HasColumnName("EMAIL");
        Property(p => p.PessoaDeContato).HasColumnName("PESSOACONTATO");
        Property(p => p.TelefoneDaPessoaDeContato).HasColumnName("PESSOACONTATOFONE");

        ToTable(Banco.Tabelas.Clientes);
    }
}
```

Figura 22 - Exemplo de mapeamento de classe no Entity Framework

Fonte: Elaborado pelo autor

Foram apresentados os conceitos e tecnologias que possuem algum vínculo direto com este trabalho, sendo apresentados no capítulo seguinte os trabalhos correlatos.

3 TRABALHOS CORRELATOS

Esta seção destaca os trabalhos correlacionados com arquiteturas de *software* e *frameworks*, ambos analisados com foco nos objetivos deste trabalho.

Em diversas áreas se aprende analisando o que já foi feito, de modo a servir como fonte de ideias e avisos de possíveis dificuldades. Contribui para este aprendizado também os projetos que não foram bem sucedidos, já que estes mostram um caminho a ser evitado (FOWLER, [s. d]a). Os trabalhos relacionados nesta seção podem não ser utilizados total e diretamente, porém algumas de suas ideias ou funcionalidades serviram como pilar de formação para este trabalho. Entre as aplicações, foram analisadas aquelas que apresentaram semelhança com este trabalho, através de múltiplos projetos ou pacotes, para uma possível separação de responsabilidades e promover a reutilização.

Analisou-se trabalhos acadêmicos, como Perreira e Silva (2000), que trata do desenvolvimento e utilização de *frameworks* e componentes, tendo como base o OCEAN Framework, direcionado à operações de óleo e gás, além de uma aplicação para automatizar escolhas de padrões de projeto, e a partir da modelagem gerar códigos fonte. Este tipo de abordagem automatizada para uso de padrões de projeto, também é o foco de Martins Azevedo (2014), que trata da seleção de padrões de arquitetura de *software*. Estes trabalhos, apesar de estarem sob o mesmo tema, são pouco relacionados com este.

Através de artigos pesquisados[12] no Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software (SBCARS), o trabalho de Lucrédio et al (2010) se destaca pelos padrões de projeto, desenvolvidos pelos próprios autores, para a criação de arquiteturas de *software* orientadas pelo domínio das aplicações.

Uma abordagem iterativa e incremental no desenvolvimento de *frameworks*, e sua a aplicação em produtos é abordado por Silva Carneiro (2003). Chaudhary (2014) fornece uma direção sobre escopo de *frameworks* corporativas em .NET e suas funcionalidades, ainda que o trabalho não tenha sido implementado.

[12] Pesquisou-se publicações de 2009 até 2015, último ano que aconteceu o simpósio.

Fernandes Monteiro (2015) apresenta uma forma para transformar requisitos em arquiteturas de software, através da técnica Four-Step Rule Set, que utiliza-se de diagramas de caso de uso para gerar diagramas de objetos.

A seguir serão explicados os trabalhos que possuem forte relação com o presente trabalho.

3.1 MAVERICK.NET

O Maverick.NET é um *framework* mínimo de desenvolvimento para Web, que utiliza e tem como principal foco o padrão MVC. Não foi utilizado diretamente neste trabalho, em função de que, os próprios frameworks Microsoft .NET oferecem recursos de nível semelhante a este, sendo sua última versão datada de 2006 (MAVERICK, [s. d.]a, [s. d.]b).

3.2 SPRING.NET

O Spring.NET é um *framework open source* para desenvolvimento de aplicações, tem como origem o projeto Spring desenvolvido em Java. Possui como pilar fundamental de sua estrutura o padrão de injeção de dependência (DI), além de promover outras boas práticas de desenvolvimento (de acordo com padrões de projeto). O Sprint.NET permite uma estratégia modularizada de uso, fundamentado na necessidade de modo a permitir que seja utiliza-lo parcialmente (POLLAK, 2008; SPRING.NET, 2011)

Este framework agregou ao trabalho conjuntos e exemplos de boas práticas a serem utilizadas em maior grau do que a implementação concreta do framework. Não foi útil perante os objetivos deste trabalho utiliza-lo integralmente, pois existem bibliotecas com foco exclusivo em determinados módulos deste framework, e o tratam de modo mais simplificado.

3.3 MOJO PORTAL

Aplicação desenvolvida em .NET, é *open source* e com foco em criar sites corporativos com integração de recursos de mídia social (MOJO PORTAL, [s. d.]a). A estruturada é dividida em três camadas, definidas por Mojo Portal ([s. d.]b) como, camada de apresentação, de negócio e de acesso a dados.

A organização dos projetos abstrai componentes para Web, de interface com o usuário e funcionalidades à aplicação, potencializando a reutilização. O projeto optou por não utilizar um *framework* ORM, sendo que, para cada banco de dados suportado pela estrutura, existe um projeto desenvolvido (com o auxílio de geradores de códigos) para acesso ao mesmo, além de outro relacionado as funcionalidades correspondentes ao tipo do banco de dados (MOJO PORTAL, [s. d.]c; [s. d.]d), conforme é observado na Figura 23 que representa a *solution* com os projetos que compõem a aplicação mojoPortal.

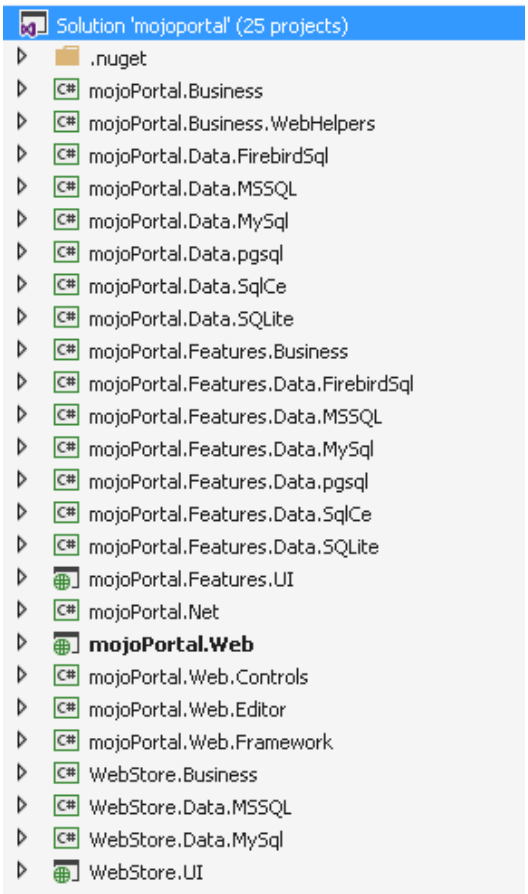


Figura 23 - Árvore de projetos da aplicação mojoPortal

Fonte: Elaborado pelo autor

Na camada de negócio foi observado através de referências entre os projetos e a implementação, que esta assume responsabilidade relacionado a persistência das informações no banco de dados. Esta é foi uma abordagem evitada no presente trabalho, pois afastaria o objetivo de isolamento da regra de negócio das aplicações, onde, a troca do local ou meio para armazenar as informações ocasionaria alterações na camada de negócio.

3.4 DOT NET NUKE

O DotNetNuke (DNN) fornece um conjunto de soluções para criar sites, é *open source* e desenvolvido em .NET, sendo a base de fundação de mais de 750.000 sites (DNN, [s. d.]a). A arquitetura de *software* da aplicação permite que sejam construídos múltiplos sites, cada um dos sites pode conter diversos módulos, sobre a mesma estrutura (DNN, [s. d.]b), o que caracteriza “N” camadas de apresentação. A representação da arquitetura tecnológica do DNN é visualizada na Figura 24.

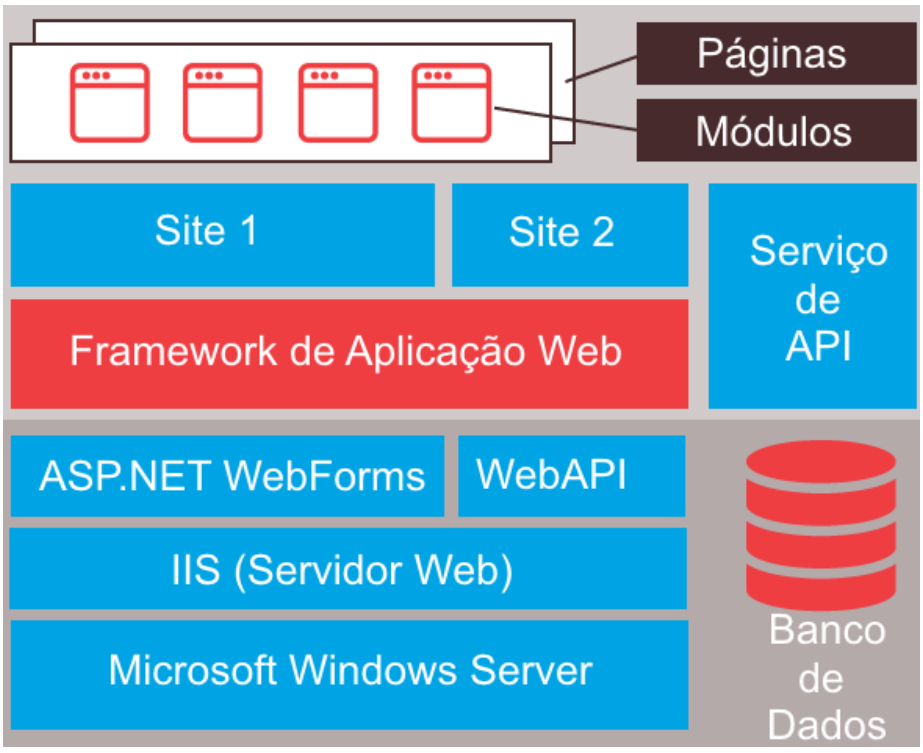


Figura 24 - Arquitetura tecnológica do DotNetNuke

Fonte: Adaptado de DNN ([s. d.]b)

É observado na Figura 24 que as aplicações desenvolvidas sobre o DNN executam em ambientes *Windows*, e que os diversos sites que podem ser criados utilizam um *framework* comum para aplicação web, que foi desenvolvido sobre o *framework* ASP.NET, além de possuir um serviço de API como porta de acesso para outras aplicações.

Cada um dos módulos criados para um site, indiferente das funcionalidades, segue a mesma arquitetura de software (DNN, [s. d.]c).

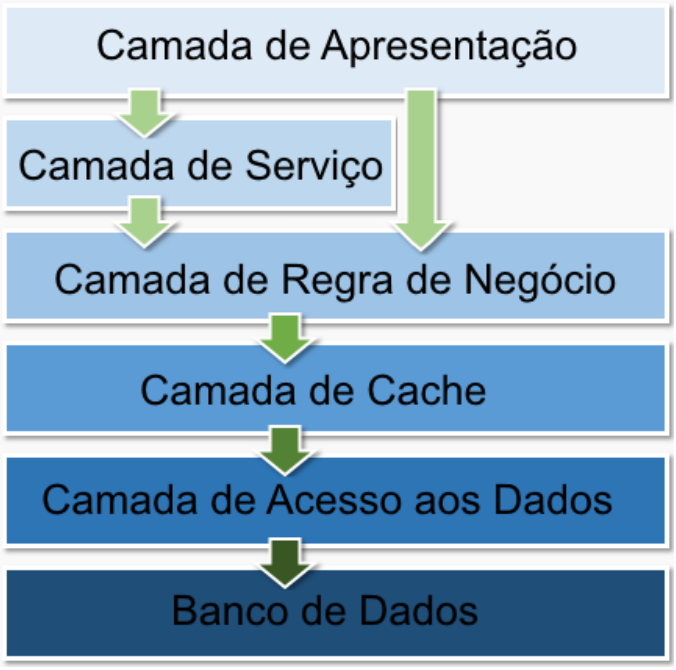


Figura 25 - Arquitetura de software de um módulo do DotNetNuke

Fonte: Adaptado de DNN ([s. d.]c)

A arquitetura dos módulos é apresenta na Figura 25, e as camadas são descritas a seguir, conforme DNN ([s. d.]c):

1. Camada de apresentação: comporta a interação com o usuário;
2. Camada de serviço: visa facilitar o acesso entre projetos comuns do DNN, através de um *framework* específico e *web services*;
3. Camada de regra de negócio: mantém as regras de negócio da aplicação, a responsabilidade desta camada é a lógica do módulo em questão.

4. Camada de *cache*: controla o *cache* entre o conteúdo no banco de dados e os exibidos nos sites, com o intuito de minimizar o acesso ao banco de dados para consultas complexas e custosas.
5. Camada de acesso aos dados: responsável por acessar o banco de dados, através de *frameworks* ORM ou com bibliotecas de comandos SQL.

3.5 ORDER PAD

O projeto OrderPad se relaciona com a modelagem e construção de uma aplicação *web*, para ser utilizado pelas equipes do supermercado Morrisons através de *tablets*, para fornecer informações e ordens de reabastecimento, em mais de 100 lojas, com um pico de utilização de 1000 usuários simultâneos, e milhões de ordens de reabastecimento por dia (MILES; FOWLER, 2014).

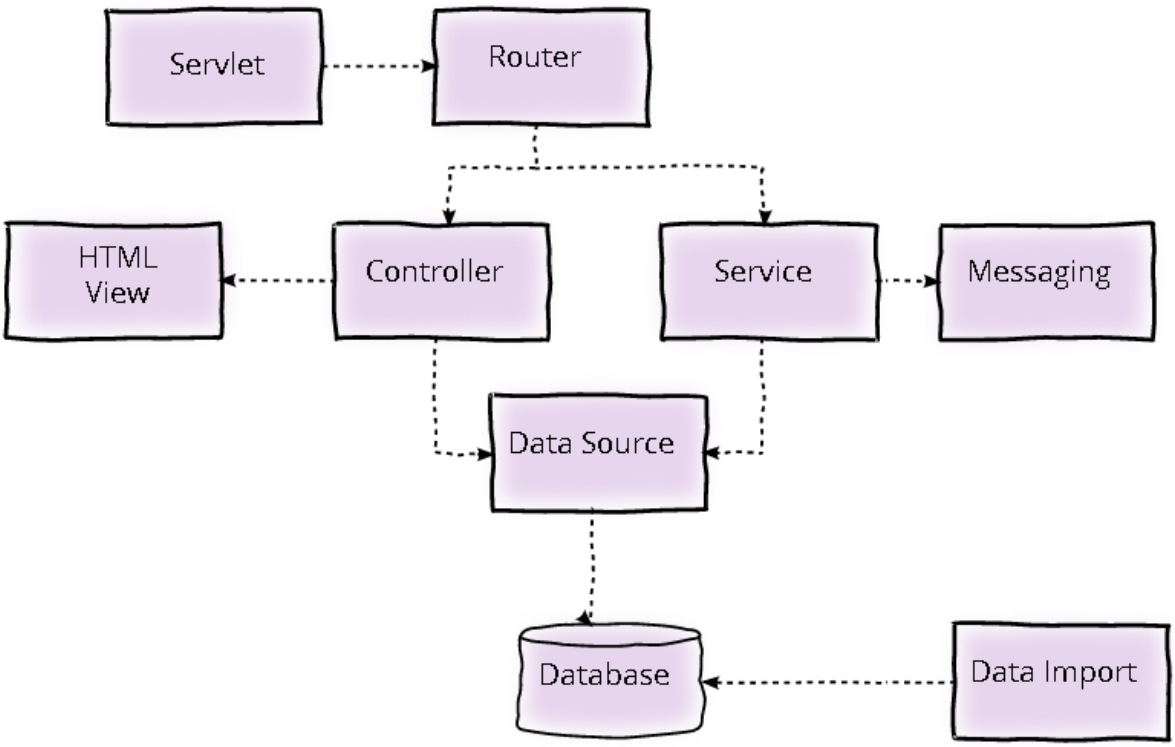


Figura 26 - Arquitetura de software do servidor para o projeto OrderPad

Fonte: MILES; FOWLER (2014)

A arquitetura de *software* da camada servidor da aplicação é visualizada na Figura 26 e detalhada da seguinte forma, de acordo com Miles e Fowler (2014):

1. *Servlet*: o servidor executa um *servlet*[13] embutido em um servidor HTTP;
2. *Router*: framework de rotas, construído durante o projeto, que faz uso da URL das requisições para o roteamento.
3. *Controller* e HTML *View*: os dados são obtidos pelos controllers, para posteriormente, utilizando *templates*, gerar uma resposta em HTML. Este processo visualiza-se na Figura 27;
4. *Service*: responsáveis por enviar algumas informações ao cliente, mas não em HTML, ou controlar requisições do tipo POST;
5. *Data Source* e *database*: acessam as informações no banco de dados, utilizando o framework de persistência Hibernate, são acessados por *controller* e *service*;
6. *Data Import*: a importação de dados é realizada diretamente pelo banco de dados, por meio de *stored procedures*.

Os *controllers* agem como uma ponte entre o banco de dados e as telas, sendo o acesso ao banco de dados por meio dos repositórios, para posteriormente popular um objeto de tela, que será utilizado pela *View* para produzir o HTML a ser enviado para o cliente, como pode ser visualizado na Figura 27.

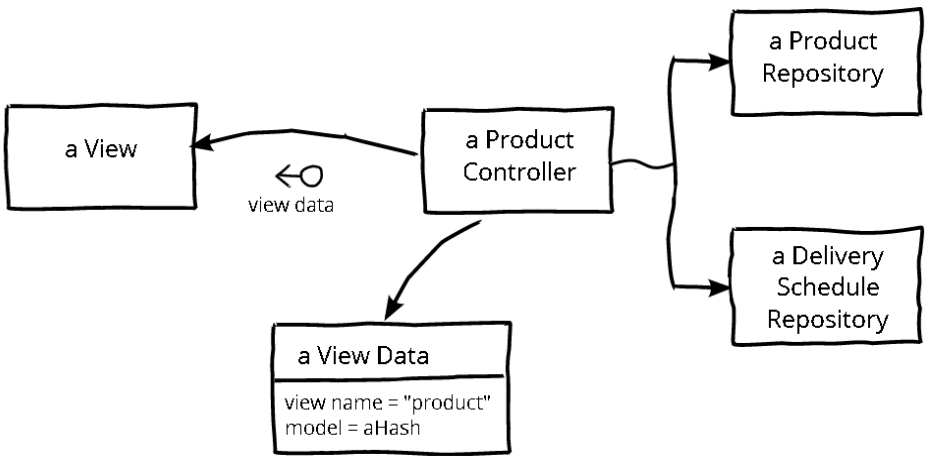


Figura 27 - *Controllers* na arquitetura de software do OrderPad

Fonte: MILES; FOWLER (2014)

[13] Interface Servlet (https://docs.oracle.com/javaee/7/api/javax/servlet/Servlet.html)

De acordo com Miles e Fowler (2014), esta arquitetura foi utilizada em mais de um projeto Java, desde 2008, e mantem como principais atributos o fácil entendimento, simplicidade para construir as aplicações e é altamente testável.

Um ponto interessante é a não utilização de *frameworks* JavaScript MVC, como Knockout, Backbone ou Angular, sendo a alternativa utilizar-se de um padrão de projeto chamado Segregated DOM[14], que cria um ponto de acesso para o *Controller*, e gerencia o acesso a biblioteca JavaScript jQuery, manipulando os objetos DOM. Alinhado ao mesmo raciocínio optou-se por não utilizar frameworks CSS, para controle de estilos, sendo esta uma decisão do qual arrependem-se (MILES; FOWLER, 2014).

Este capítulo apresentou os trabalhos correlatos, de porte acadêmico e corporativo que tenham relação com o presente trabalho, sendo detalhados apenas os com forte relação. O capítulo seguinte aborda o projeto da arquitetura de *software* que este trabalho desenvolveu e aplicou.

[14] Detalhes deste padrão de projeto em http://martinfowler.com/bliki/SegregatedDOM.html

4 PROJETO

Foi desenvolvido neste trabalho uma arquitetura de *software*, que visa auxiliar o Laboratório a construir aplicações com a estrutura em .NET, mas não necessariamente as camadas de interface com usuário. O auxílio ocorre pela elaboração da arquitetura de *software* para criação de aplicações, que foi desenvolvida com base em padrões de projeto e análise de trabalhos relacionados. Esta estrutura organiza o desenvolvimento, de modo a:

1. Não impor o uso específico de tecnologias e meios para armazenar o conteúdo da aplicação[15], ou de interface com o usuário, de forma que cada projeto desenvolvido realize a escolha com base em seus requisitos;
2. Preservar as regras de negócio das aplicações;
3. Potencializar a reutilização;
4. Possibilitar a inclusão ou remoção de camadas de acordo com a demanda de cada projeto.

Não foi escopo deste trabalho criar uma estrutura de desenvolvimento absoluta, com todas as funcionalidades, camadas e *frameworks* necessários para o desenvolvimento de aplicações. O trabalho trata de um ponto de partida quando a estratégia é preservar as regras de negócio, de modo a permitir e otimizar várias interfaces com o usuário (*Web, Mobile*, etc.), baixo acoplamento com tecnologias em constante evolução (Entity Framework, NHibernate, ASP.NET, AngularJS, etc.) e possibilitar a reutilização de funcionalidades e a própria arquitetura. Futuros projetos do Laboratório podem contribuir com novas funcionalidades, camadas e *frameworks*, porém é crucial que, estes agreguem valor ao desenvolvimento quando o escopo do projeto permitir, não obrigando que sejam implementadas em todos os projetos.

[15] Poderia ser um banco de dados (SQL Server, MySQL, PostgreSQL), um conjunto de arquivos serializados (XML, binário) ou somente a utilização dos dados em memória.

4.1 ARQUITETURA DE SOFTWARE

Para possibilitar a intercambialidade entre camadas de dados e apresentação, de modo a preservar a regra de negócio das aplicações, a arquitetura de software proposta por este trabalho tem sua representação na Figura 28, sendo cada camada detalhada em sua subseção. O nível de acesso, das camadas superiores para as inferiores, acontece por chamada de método; retorno de métodos e exceções configuram o nível de resposta das camadas inferiores para as superiores.

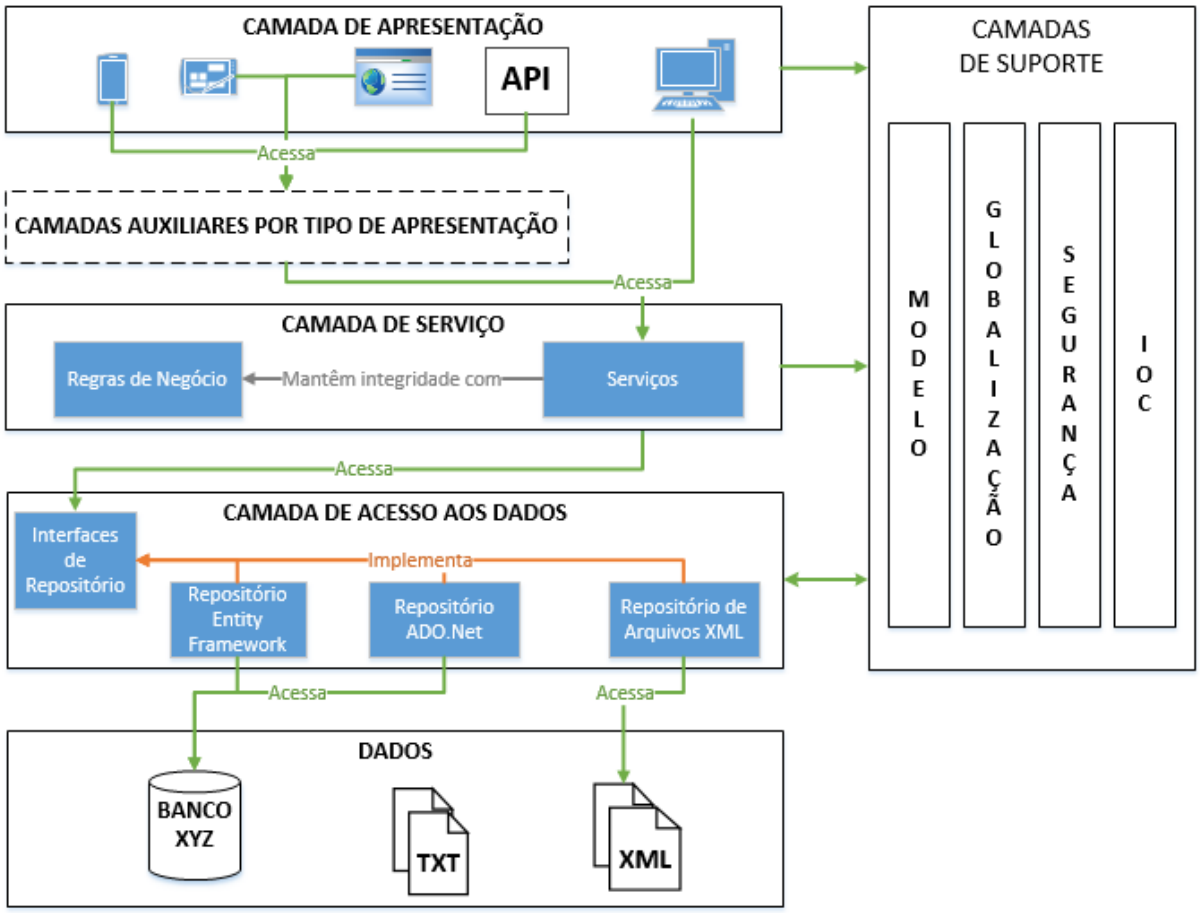


Figura 28 - Funcionamento da arquitetura de software

Fonte: Elaborado pelo autor

4.1.1 Camada de acesso aos dados

A responsabilidade atribuída a esta camada é, interagir com o local que armazena os dados das aplicações, seja um banco de dados, acessado via

frameworks ORM e bibliotecas de dados como ADO.NET, uma coleção de arquivos serializados, dados mantidos em memória ou outras formas de armazenamento.

Esta camada é acessada pela camada de serviço, sendo a porta de entrada o projeto[16] que mantém as interfaces dos repositórios, sem a classe concreta de manipulação dos dados. As classes responsáveis pela comunicação com o local de armazenamento de dados devem estar em outros projetos, de modo que, a referência para tecnologias especificas de acesso aos dados mantenham-se somente nos projetos específicos de cada, de tal forma que os projeto acessado pelos serviços estão desacoplados das tecnologias; na Figura 29 é possível observar a estratégia apresentada.

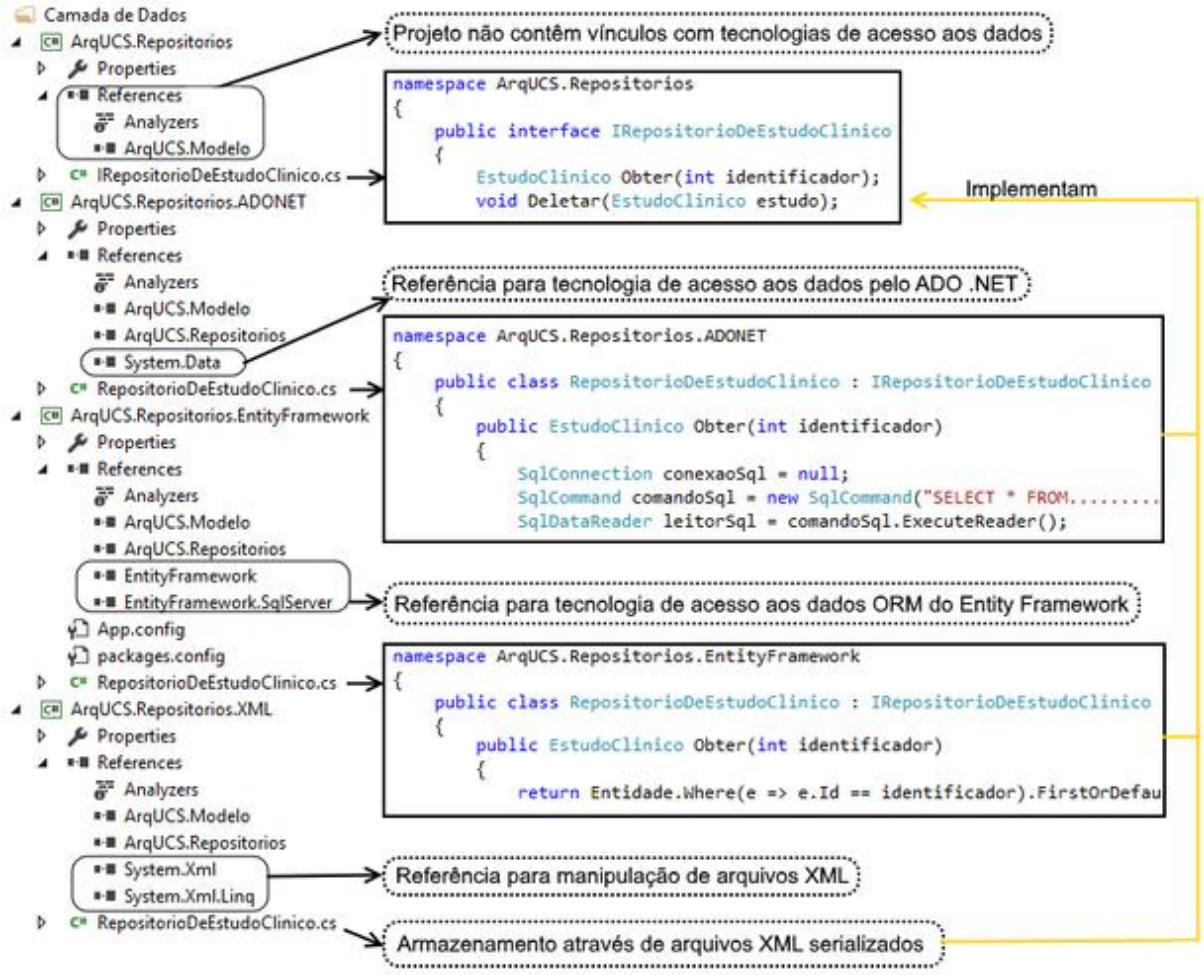


Figura 29 - Exemplo com múltiplas tecnologias na camada de acesso aos dados

Fonte: Elaborado pelo autor

[16] O termo projeto no contexto do .NET refere-se ao que seria equivalente a pacotes, criando assim uma separação lógica.

Os três projetos de acesso a dados visualizados na Figura 29, não são obrigatórios na implementação de todas as aplicações, que vierem a utilizar esta arquitetura. Grande parte das aplicações define uma tecnologia/forma de acesso aos dados e trabalha apenas com esta. Caso necessário esta arquitetura de *software* permite utilizar mais do que uma tecnologia/forma de acesso aos dados na mesma aplicação. É importante destacar que a tecnologia/forma de acesso aos dados não implica em limitar o tipo de banco de dados (SQL Server, PostgreSQL, MySQL, etc.), por exemplo: é viável implementar repositórios utilizando *frameworks* ORM (NHibernate, Entity Framework) e conectá-los em diferentes tipos de bancos de dados (SQL Server, PostgreSQL, MySQL, etc.), sendo o *framework* ORM apenas um facilitador para interagir com o banco de dados.

A separação de responsabilidades e baixo acoplamento desta camada, permitem que uma aplicação troque o meio de acesso aos dados sem refatorações, pois todo o acesso realizado pelos serviços é feito pelas interfaces e não as classes concretas. Para exemplificar uma possível em uma aplicação que faz uso do Entity Framework, e por algum motivo qualquer foi necessário alterar para o NHibernate, a refatoração consiste nas seguintes tarefas:

1. Criar o novo projeto na camada de dados utilizando o NHibernate;
2. Criar classes que implementem as interfaces referentes aos objetos de modelo;
3. Alterar[17] os registros de vínculo de interfaces com as novas classes criadas.

É importante destacar no exemplo anterior que não foi necessário a alteração da camada de serviço, imediatamente acima desta. Este comportamento é possível principalmente através da utilização do padrão de injeção de dependência.

4.1.2 Camada de serviço

Esta camada mantem as regras de negócio da aplicação isoladas, através de duas partes, sendo a primeira ao fazer uso do padrão de projeto de serviços, e a

[17] Possui relação de 1 para 1, se houver 20 objetos de modelo (cliente, usuário, pedido, etc.) utilizados em 400 locais resulta em 20 alterações.

segunda com as regras de negócio da aplicação. Os serviços são responsáveis por manter íntegro as operações realizadas pelo usuário, através das regras de negócio, que se bem-sucedido permite interagir com a camada de dados. Os serviços implementados são o meio de acesso desta camada pelas camadas superiores.

Existe uma separação entre serviços e regras de negócio dentro da própria camada, de modo a melhor separar responsabilidades. As regras de negócio compõem o real patrimônio de uma aplicação, e estando separadas dos serviços tornam-se altamente testáveis[18] além de evidenciar quais são e onde estão as regras de negócio da aplicação.

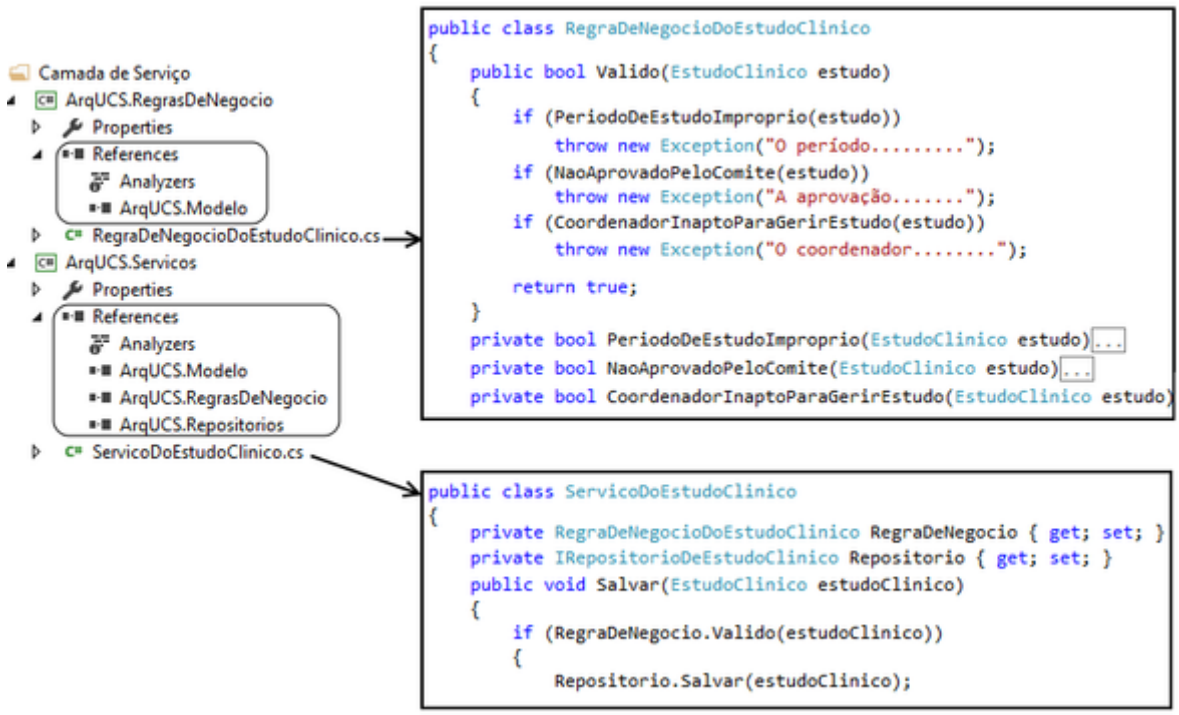


Figura 30 - Exemplo da camada de serviço

Fonte: Elaborado pelo autor

O conteúdo das classes referentes as regras de negócio são de implementação opcional, variando de cada objeto da aplicação, por outro lado o serviço é de caráter obrigatório.

A aplicação DotNetNuke (DNN) e o padrão de projeto de modelo de domínio, explicados nas seções de fundamentos e trabalhos correlatos, adotam a estratégia de

[18] Através de projetos de testes automatizados como NUnit, Specflow e etc.

criar uma camada específica para regras de negócio. Em um cenário de inexistência destas regras, seria necessário criar classes nestas camadas possibilitando continuar o fluxo da arquitetura, pois as camadas não devem ser ignoradas; porém se a camada fosse ignorada, em função da inexistência de regras de negócio no momento da implementação, e posteriormente fosse necessário implementá-las, a refatoração é maior, pois seria incluída uma nova camada de regras de negócio entre outras duas já existentes.

É possível desenvolver regras de negócio com o auxílio de interfaces[19] e atributos[20] de validação. Esta abordagem foi evitada por gerar acoplamento com tecnologias e dificultar a intercambialidade das camadas, principalmente de acesso aos dados como por exemplo, o Entity Framework considera os atributos e interfaces no momento da persistência, já o ADO.NET não.

4.1.3 Camadas auxiliares por tipo de apresentação

As camadas auxiliares por tipo de apresentação representam o nível da arquitetura de software em que é permitido, sem ferir os objetivos do trabalho, incluir novas camadas conforme o tipo de apresentação utilizado. A utilização do padrão de transferência de dados DTO[21] (Data Transfer Object), frequentemente utilizado em apresentações Web como forma de minimizar o tráfego de informações, seria implementado neste nível.

Estas camadas variam conforme as necessidades da aplicação e da forma de interação com o usuário.

[19] IValidatableObject disponível em https://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.ivalidatableobject.aspx
[20] ValidationAttribute disponível em https://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.validationattribute.aspx
[21] O padrão DTO é detalhado em FOWLER (2006, p. 380)

4.1.4 Camada de apresentação

A camada de apresentação comporta as formas de acesso do usuário final à aplicação. Uma mesma aplicação poderá conter “N” formas de acesso, como por exemplo: *desktop*, *mobile* e *web*. Ambos os pontos de acesso fazem uso da mesma camada de serviço, não havendo necessidade de que se reimplemente as regras de negócio.

Considerando o exemplo de uma apresentação *desktop*, suas telas podem acessar diretamente aos serviços, pois não existe o tráfego de informações através do protocolo HTTP. Uma apresentação web com padrão MVC pode possuir o mesmo comportamento, o acesso aos serviços aconteceria por meio do controlador (*Controller*), gerando uma visão (*View*), processada no servidor para ser enviada para o browser do usuário. Para uma apresentação web utilizando o AngularJS, haveria necessidade de implementação de camadas auxiliares, para converter os dados do domínio em JSON. Todos os exemplos utilizariam os mesmos serviços implementados.

4.1.5 Camadas de suporte

As camadas de suporte trabalham de forma a auxiliar a aplicação, em vários níveis da arquitetura. De acordo com os requisitos e necessidades das aplicações poderão ser incluídas ou removidas.

4.1.5.1 Camada de modelo

Esta camada apoia-se no funcionamento do modelo de domínio, mantém-se aqui classes com valor para o negócio, como por exemplo pessoa, cliente, pedido, etc. O funcionamento é semelhante no sentido de abrigar as definições das classes, porém o padrão tradicional também comporta as regras de negócio.

O foco em separar definições e regras é que ao aplicar o modelo de domínio, este trabalha com um controle restrito de acessos, e por ser uma camada, seriam permitidos apenas acessos da camada imediatamente superior. Ao considerar o funcionamento deste padrão, com o objetivo de preservar a regra de negócio mesmo que com “N” formas de apresentação, o modelo de domínio seria acessado no mínimo por cada tipo de apresentação. Os acessos seriam evitados ao criar uma camada entre o domínio e apresentações, porém, as classes de domínio deveriam ser convertidas para uma nova classe. A conversão ocasionaria que estas camadas definissem novas classes e mapeadores, responsáveis pelas transições das informações, onerando desenvolvimento e consumindo processamento.

```
public class Telefone
{
    public int DDD { get; set; }
    public int Numero { get; set; }
    public EnumeradorDeTipoDeTelefone Tipo { get; set; }
}

public class Pessoa
{
    public String Nome { get; set; }
    public IList<Telefone> Telefones { get; set; }
    public String NomeDaMae { get; set; }
```

Figura 31 - Exemplo de implementação na camada de modelo

Fonte: Elaborado pelo autor

O exemplo da Figura 31 define um modelo de classes de pessoa e telefone, ambas utilizando tipos primitivos da linguagem ou criados na própria camada, sendo que as regras de negócio relacionadas a pessoa estão na camada de serviço.

```
[Table(Name = "Pessoas")]
public class Pessoa
{
    [Column(Name = "Nome")]
    public String Nome { get; set; }
```

Figura 32 - Exemplo não recomendado de implementação na camada de modelo

Fonte: Elaborado pelo autor

O exemplo da Figura 32 é uma abordagem a ser evitada, pois os atributos Table e Column são utilizados no Entity Framework, para mapear o objeto a uma tabela e a propriedade para uma coluna, sendo que, se alterado a tecnologia de acesso aos dados eles perdem o sentido de funcionamento. Ainda que não seja alterado a forma de acesso aos dados outras camadas e tecnologias fazem uso desta classe; se cada uma acrescentar conteúdo pertinente a si a classe se torna poluída e de difícil manutenção.

4.1.5.2 Camada de globalização

A camada de globalização possibilita que aplicações operem com múltiplos idiomas, sendo implementado nesta camada um mecanismo que através de um valor chave e idioma possibilita obter uma palavra ou texto, que corresponde ao idioma acessado para aquela chave. Este comportamento é semelhante ao funcionamento de um dicionário, onde ao usar uma chave e obtêm-se um valor.

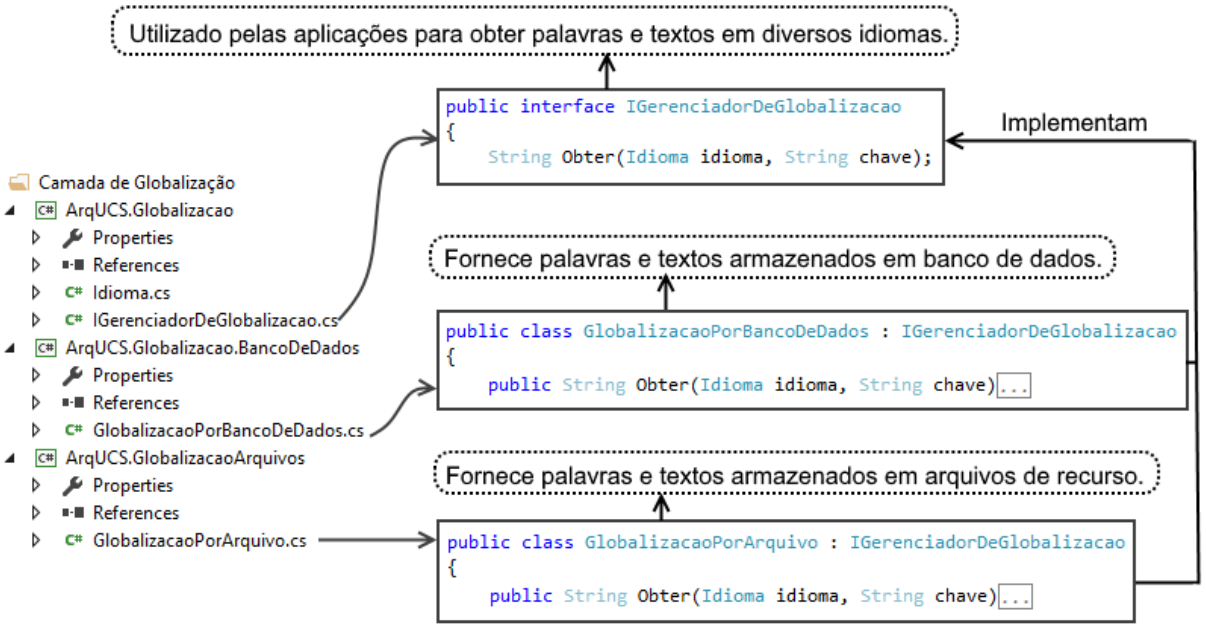


Figura 33 - Exemplo de implementação da globalização

Fonte: Elaborado pelo autor

O conteúdo globalizado que é exibido ao usuário final de acordo com seu idioma é abordado de duas formas, sendo por arquivos de recurso[22] ou por tabelas no banco de dados. A camada de globalização faz uso do padrão de projeto de injeção de dependência, tornando maleável a escolha de qual meio será utilizado para armazenar o conteúdo dos idiomas, como por exemplo banco de dados ou arquivos, com o mínimo de refatorações, conforme é visualizado na Figura 33.

A camada tem seu funcionamento embasado no processamento de um idioma e sua chave pela implementação do gerenciador de globalização, sendo o idioma alterado em nível de aplicação ou um atributo de um usuário por exemplo. A Figura 34 ilustra o diagrama de classes de globalização.

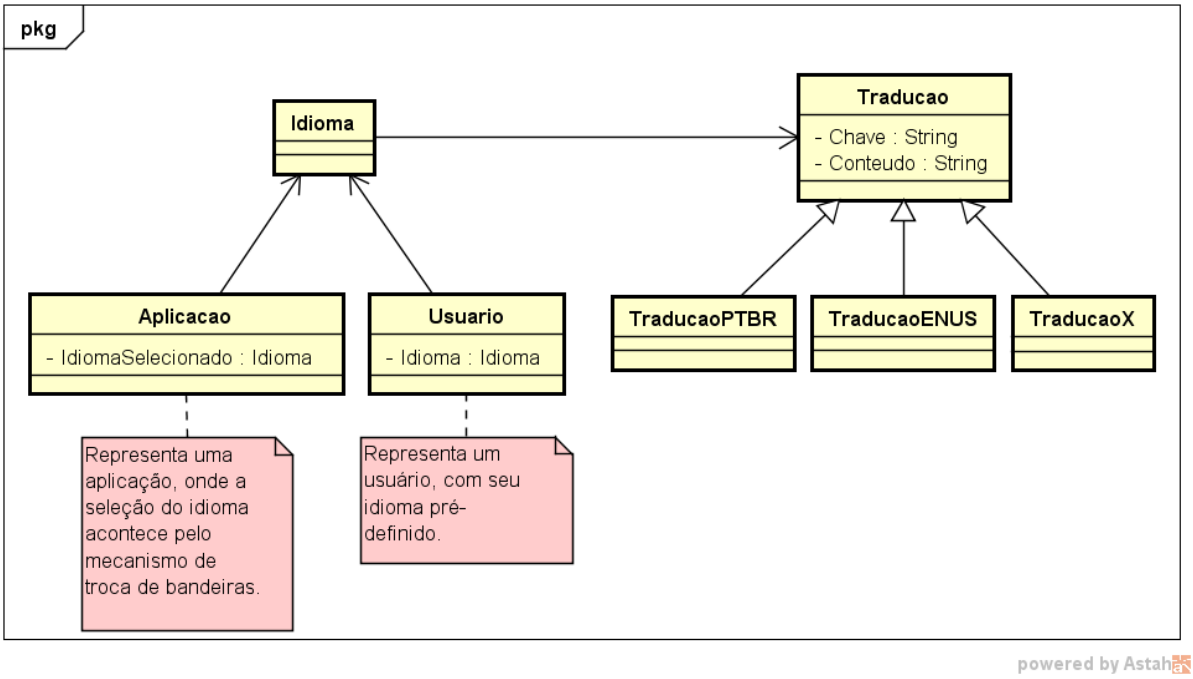


Figura 34 – Diagrama de classes de globalização

Fonte: Elaborado pelo autor

Além de critérios como o conteúdo escrito a globalização possui uma segunda particularidade não tão visível ao usuário final, que compreende por exemplo

[22] Arquivos de recurso no .Net funcionam como um dicionário chave e valor. Define-se seu conteúdo e depois de compilado obtêm-se as informações através da chave.

separadores de milhar, símbolo monetário ou formato da data e hora, além de outros. Esta parte será delegada para o .NET, através dos mecanismos já existentes[23].

4.1.5.3 Camada de segurança

A camada de segurança funciona através de uma divisão entre quem tem acesso, e sobre quais funcionalidades possui acesso. O quesito de quem tem acesso é definido por mecanismos de login. Quais funcionalidades estão disponíveis para os usuários, funciona por meio de regras para um determinado grupo de usuários, chamado de perfil. Um usuário poderá participar de “N” perfis, a Figura 35 representa a de tela referente ao cadastro de usuários das aplicações.

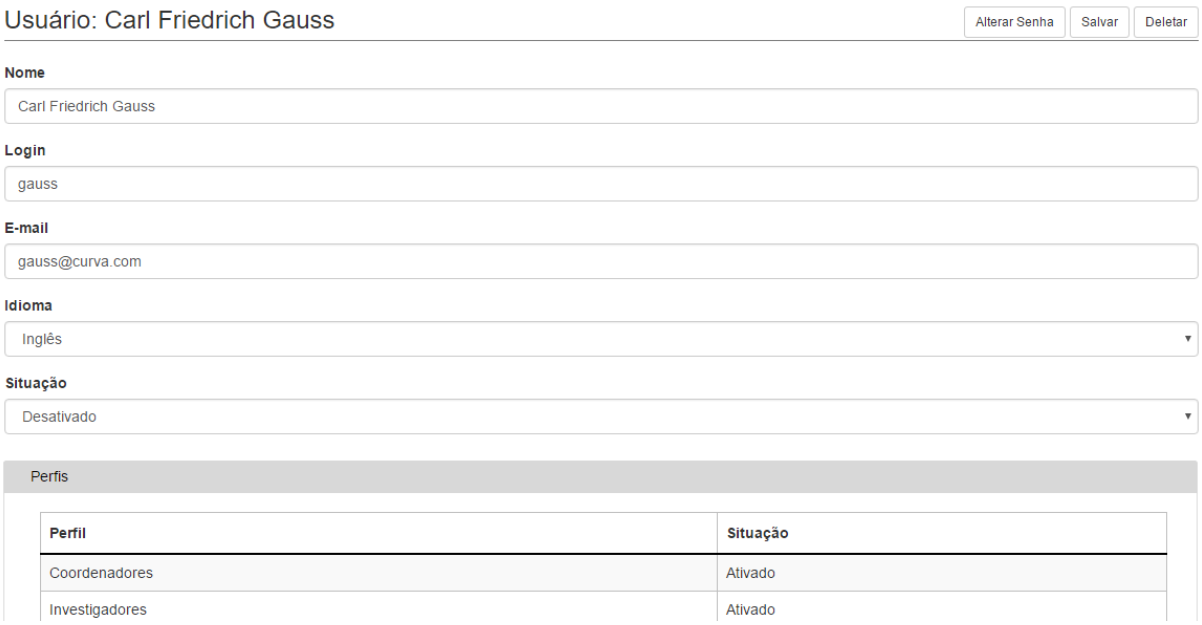


Figura 35 - Tela do cadastro de usuário

Fonte: Elaborado pelo autor

As tarefas de login e execução das regras serão delegados para camada de apresentação, que conforme a tecnologia dispõe de mecanismos prontos[24] para isso. Além de tais mecanismos, o tratamento varia conforme o tipo de apresentação envolvida, em um cenário onde as telas de uma aplicação são acessadas somente

[23] Conjunto de classes responsáveis pela globalização no .Net Framework, são armazenadas no System.Globalization, em https://msdn.microsoft.com/en-us/library/system.globalization.aspx
[24] Caso do ASP.NET, em https://docs.asp.net/en/latest/security/authorization/index.html

via menu, a apresentação *desktop* pode controlar a liberação de menus, e uma *web* deve controlar a liberação de menus e da própria página, pois o usuário pode digitar a URL e acessar a página sem o menu.

É responsabilidade desta camada fornecer um meio para que as camadas de apresentação realizem o controle de acesso através dos perfis. As permissões associam-se as telas, onde cada tela possui “N” funcionalidades, com as possibilidades de possuir ou não permissão. Na Figura 36 é visualizado a tela do perfil, permitindo gerenciar as funcionalidades da tela de “Pacientes” para um determinado perfil. O Apêndice A armazena as figuras de tela detalhadas, para os cadastros de perfil e usuário.

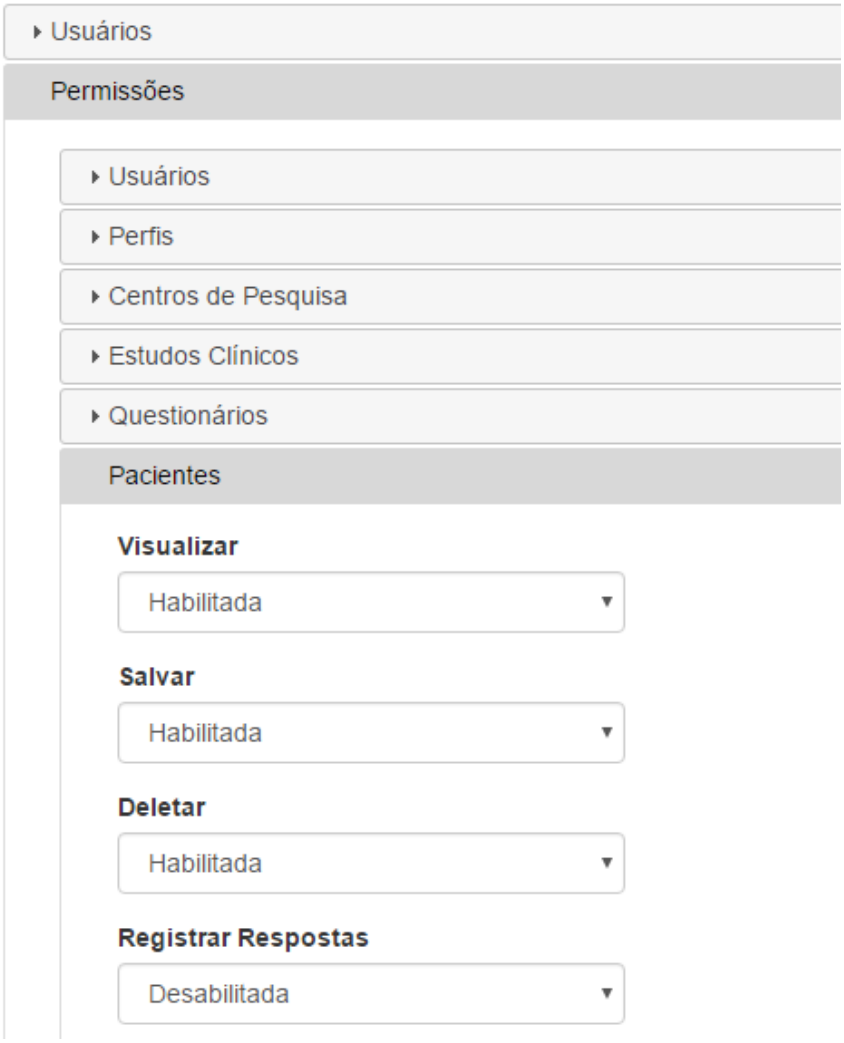


Figura 36 – Cadastro de perfil

Fonte: Elaborado pelo autor

O cadastro tem seu funcionamento de forma dinâmica para se adaptar a cada aplicação (com “N” telas, e cada uma delas com “N” funcionalidades) e também, para novas telas e funcionalidades que vierem a ser acrescentadas nas aplicações. O conteúdo utilizado para compor as permissões e funcionalidades devem ser implementados em cada aplicação através de interfaces padrão, um exemplo deste processo é ilustrado na Figura 37, através da implementação de um gerenciador de permissões para uma determinada aplicação; considerando que cada aplicação implementa um gerenciador de permissões, se necessário as informações básicas para a criação de um perfil poderiam vir através de um banco de dados.

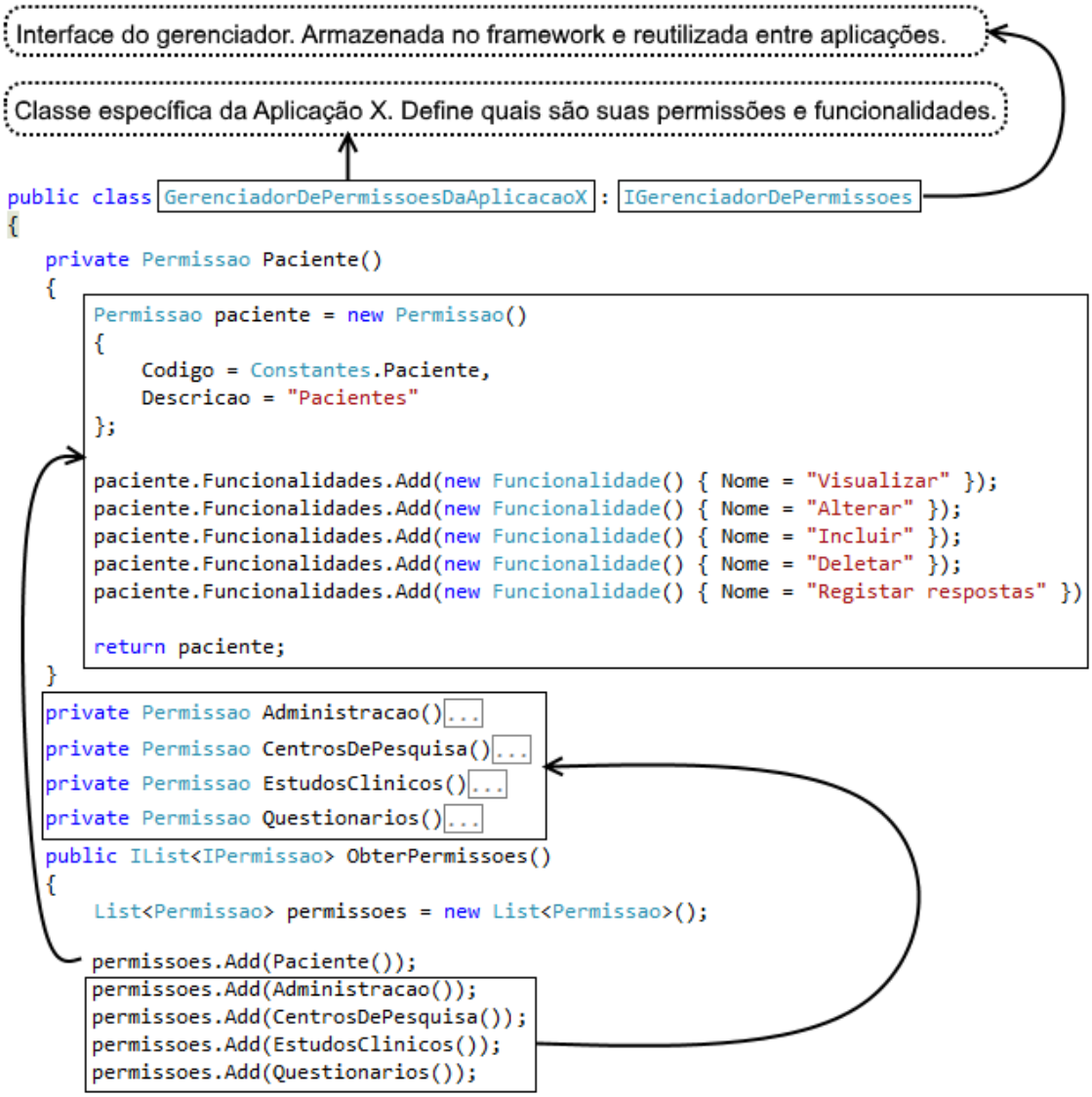


Figura 37 - Exemplo das possíveis permissões de telas e funcionalidades

Fonte: Elaborado pelo autor

A estrutura da camada de segurança tem suas classes visualizadas na Figura 38.

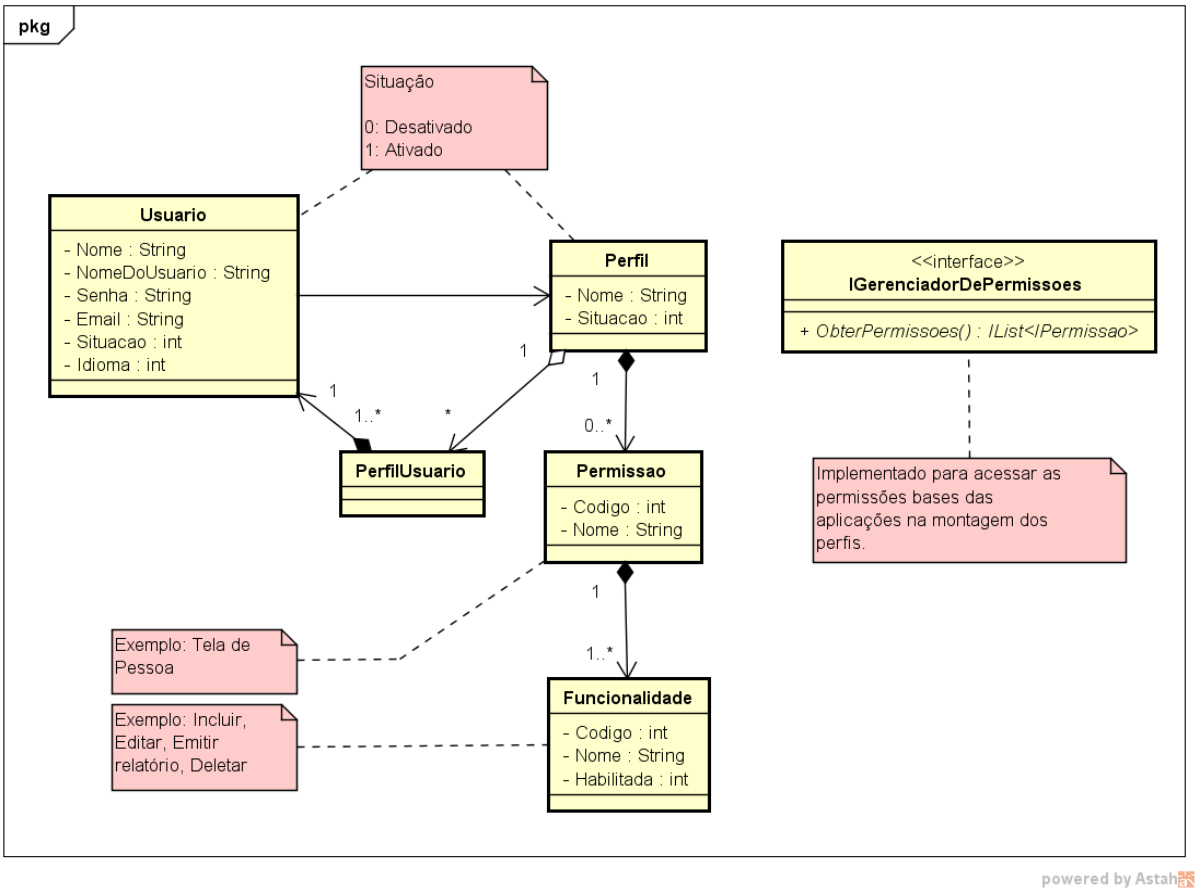


Figura 38 - Diagrama de classes básico para a camada de segurança

Fonte: Elaborado pelo autor

4.1.5.4 Camada de Injeção de Dependência (IOC)

Esta camada possui a incumbência de coordenar a inversão de controle/injeção de dependência na aplicação. A injeção de dependência é utilizada na camada de acesso aos dados auxiliando o desacoplamento com as tecnologias de acesso. A utilização pode acontecer também em outros níveis da arquitetura, como é o caso da camada de globalização.

O funcionamento desta camada acontece por meio de vínculos entre interface classe, e no momento da utilização das interfaces cabe a ela o “preenchimento” da classe concreta, sendo este processo visualizado na Figura 39.

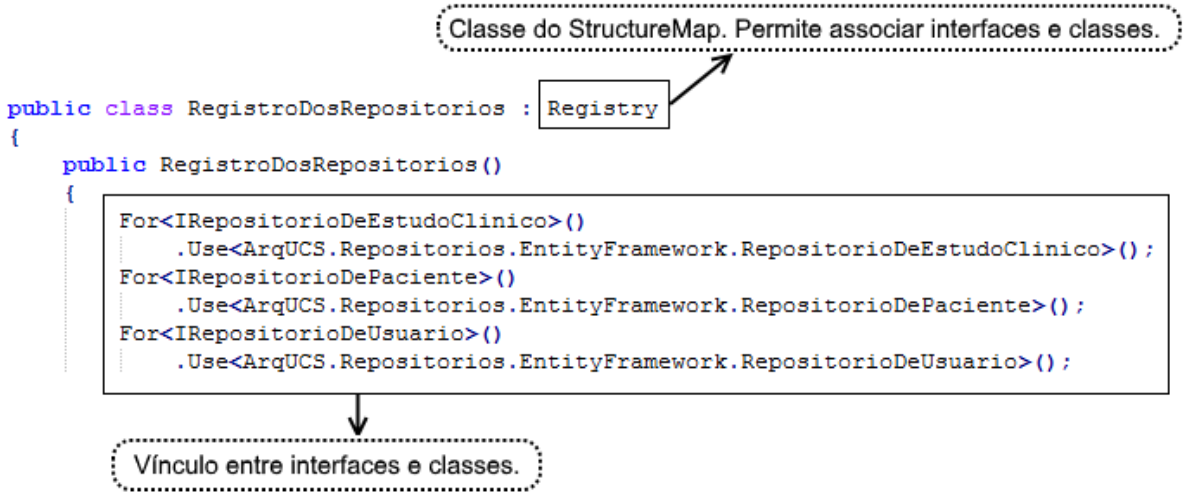


Figura 39 - Exemplo de injeção de dependência

Fonte: Elaborado pelo autor

O uso de bibliotecas como o StructureMap não tem caráter obrigatório para o funcionamento da camada, sendo exemplificado na Figura 40 outra abordagem de implementação, onde a injeção de dependência acontece por meio do padrão de fábricas, onde o retorno seja sempre a *interface*, alterando somente o tipo criado.

```
public class FabricaDeRepositorios
{
    public IRepositorioDeEstudoClinico ObterRepositorioDeEstudoClinico()
    {
        return new Repositorios.ADONET.RepositorioDeEstudoClinico();
```

Figura 40 - Exemplo de injeção de dependência através de fábricas

Fonte: Elaborado pelo autor

4.2 PROVAS DE CONCEITO

As provas de conceito desenvolvidas neste trabalham tiveram o objetivo de validar a arquitetura de software projetada. Foram desenvolvidas as aplicações de CRUD e de Acompanhamento de Indicadores Clínicos.

4.2.1 Aplicação CRUD

Esta aplicação tem em seus fins ser modelo de implementação e testou a arquitetura de software, sendo seus requisitos as quatro operações básicas (adicionar, visualizar, editar e excluir), representadas na Figura 41, tendo como foco um cadastro de pessoas, como é visualizado no diagrama da Figura 42. A aplicação tem duas[25] interfaces com o usuário, uma via web utilizando a tecnologia ASP.NET e outra desktop através do Windows Forms.

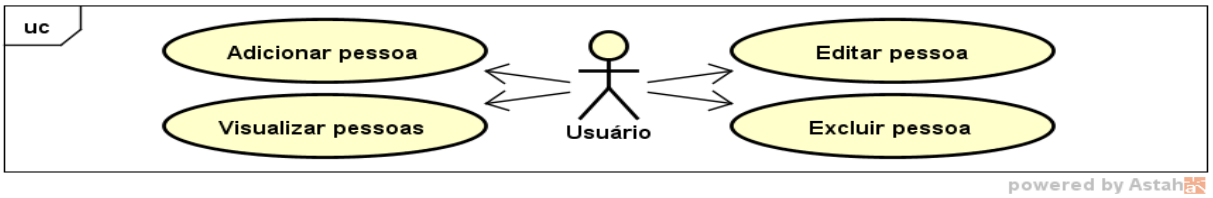


Figura 41 - Diagrama de Caso de Uso da Aplicação CRUD

Fonte: Elaborado pelo autor

A Aplicação CRUD possibilita armazenar informações em um banco de dados do tipo PostgreSQL, utilizando duas formas diferentes de acesso, a primeira pelo ADO.NET, e sendo a segunda por meio de um framework ORM, acessando o mesmo banco de dados PostgreSQL. O funcionamento geral desta aplicação é ilustrado na Figura 43, e no Apêndice B contêm os diagramas de sequência, que detalham o funcionamento.

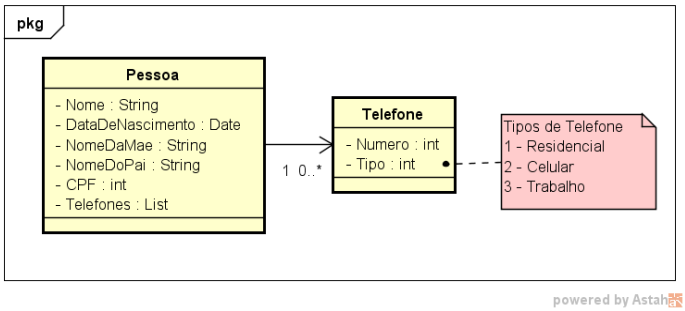


Figura 42 - Diagrama de classes da Aplicação CRUD

Fonte: Elaborado pelo autor

[25] Conforme Fowler (2008) se ao acrescentarmos uma camada radicalmente diferente, como Web e linha de comando por exemplo, for necessário duplicar funcionalidades é um sinal de que a lógica vazou para a camada de apresentação; esta mesma estratégia pode ser aplicada ao local que as informações, frequentemente um banco de dados utilizando alguma tecnologia de acesso.

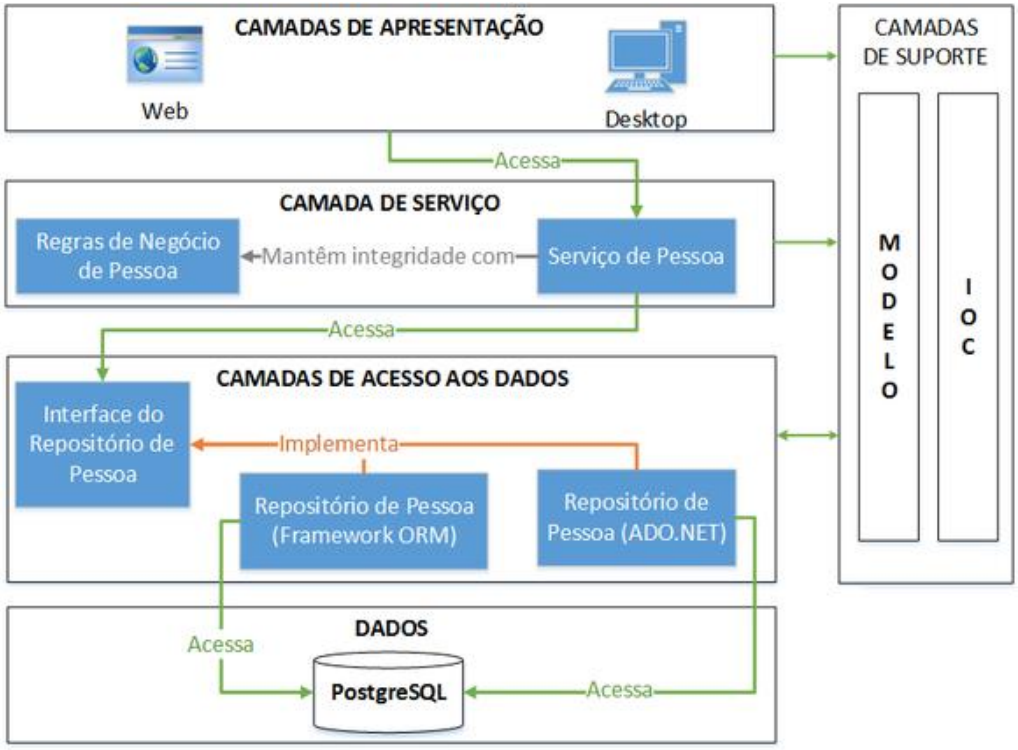


Figura 43 - Funcionamento geral da Aplicação CRUD

Fonte: Elaborado pelo autor

Como regra de negócio para esta aplicação, foi determinado que a pessoa tenha ao menos um telefone celular informado, e caso não o tiver esta pessoa é inválida para o negócio.

Esta aplicação possui duas camadas de apresentação, e duas camadas de acesso aos dados, e utilizou-se da mesma regra de negócio para qualquer uma das combinações possíveis entre as tecnologias de acesso aos dados e formas de apresentação.

4.2.2 Acompanhamento de Indicadores Clínicos

Este sistema surgiu a partir da necessidade de pesquisadores gerenciarem estudos clínicos, através da aplicação de questionários criados de forma específica para as patologias tratadas nestes estudos, sendo as respostas analisadas

estatisticamente. Os requisitos que correspondem ao incremento inicial da aplicação são visualizados na Figura 44.

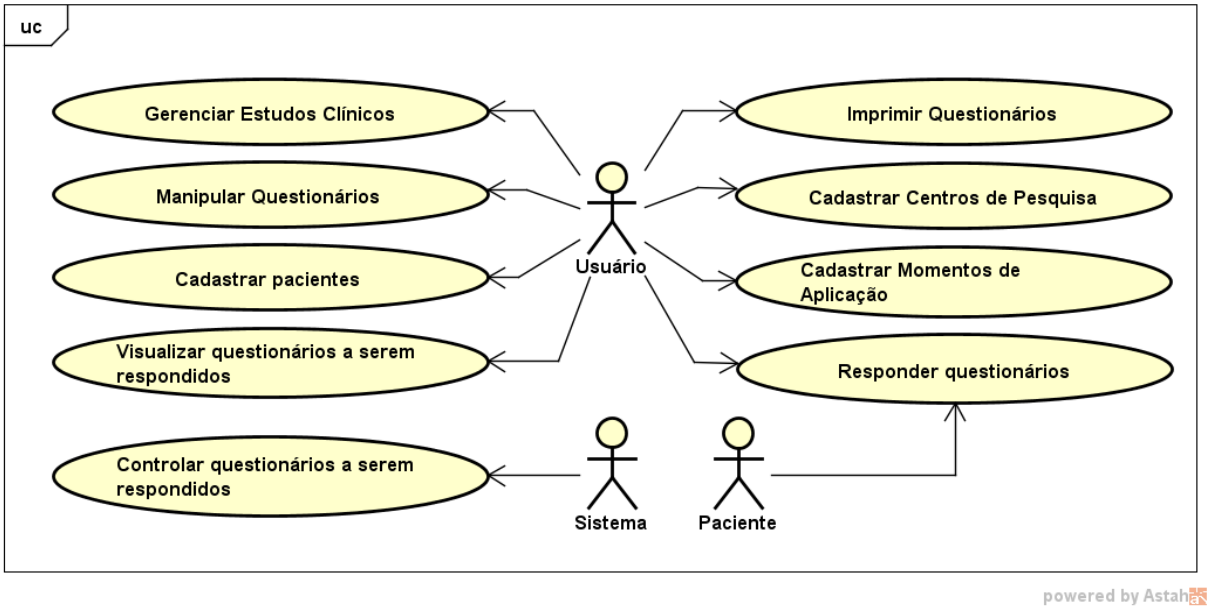


Figura 44 - Casos de uso de Acompanhamento de Indicadores Clínicos

Fonte: Elaborado pelo autor

Esta aplicação suporta mais de um idioma além um controle de permissões flexível, sendo ambas as funcionalidades abstraídas em frameworks, permitindo serem reutilizadas em futuros projetos do Laboratório. O funcionamento geral desta aplicação é visualizado na Figura 45, e detalhes da modelagem, como por exemplo casos de uso, diagrama de classes e figuras dos cadastros são armazenados no Apêndice C, sendo que informações referentes a modelagem de segurança e globalização localizam-se nas seções 4.1.5.2 Camada de globalização e 4.1.5.3 Camada de segurança.

A aplicação tem uma camada de apresentação *web,* e o controle de permissões de páginas e métodos foi delegado aos mecanismos[26] do ASP.NET, que interpreta as permissões criadas com a camada de segurança.

[26] Informações sobre o controle de autorização disponível em https://docs.asp.net/en/latest/security/authorization/index.html

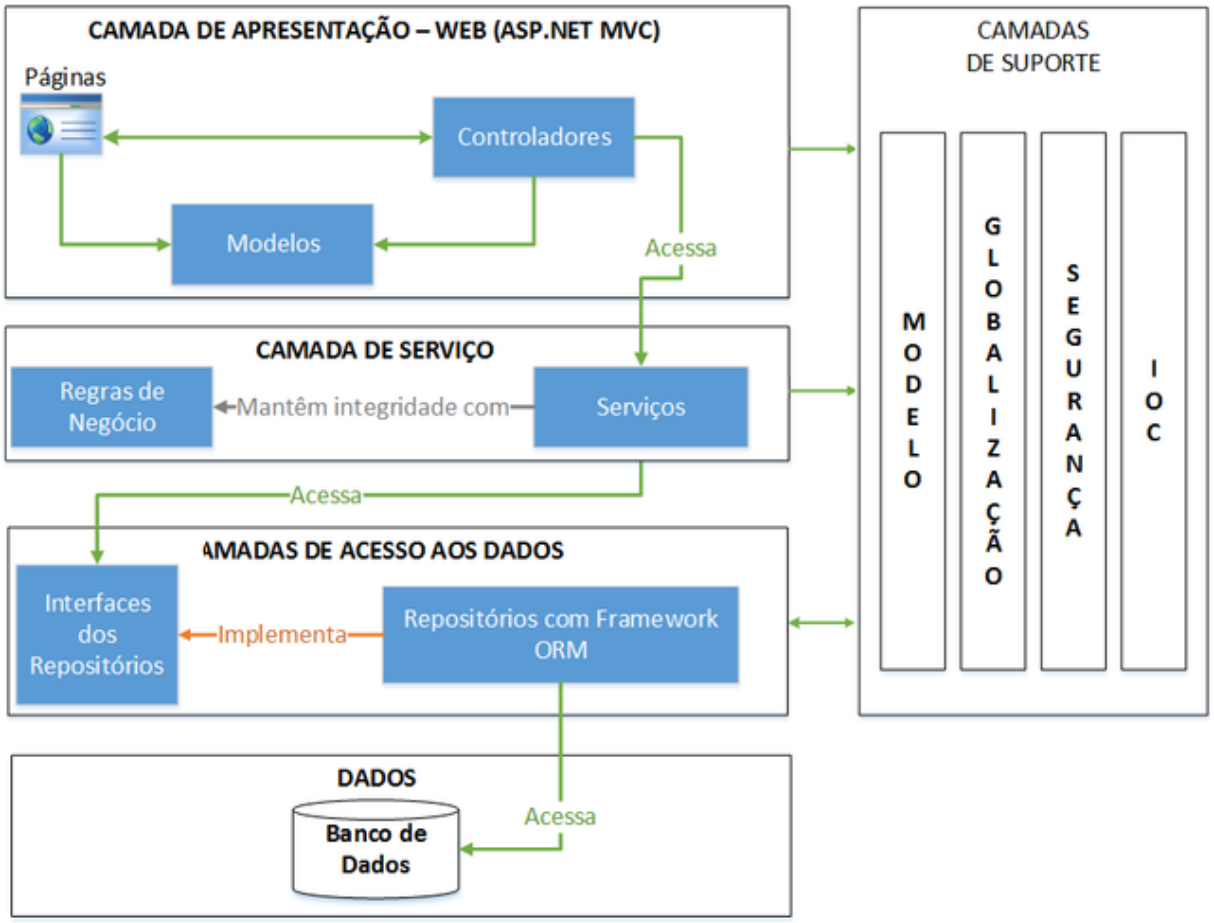


Figura 45 - Funcionamento geral da aplicação de Indicadores Clínicos

Fonte: Elaborado pelo autor

No Apêndice E está localizado um guia relacionado aos passos necessários para a instalação desta aplicação.

Foi apresentado neste capítulo a arquitetura de *software* reutilizável criada neste trabalho, além das aplicações que foram desenvolvidas como provas de conceito da arquitetura. As documentações de modelagem estão armazenadas nos Apêndices A, B e C. O próximo capítulo trata sobre a aplicação e resultados deste trabalho, estando as informações pertinentes à utilização da arquitetura mantida no Apêndice D.

5 DESENVOLVIMENTO

Esta seção do trabalho apresenta os resultados obtidos nas implementações, além de guiar o desenvolvimento de outras aplicações utilizando esta mesma arquitetura de *software* e *frameworks* criados. Detalhes e explicações sobre a arquitetura de *software* e suas camadas estão especificadas no projeto deste trabalho, apresentado na seção 4.

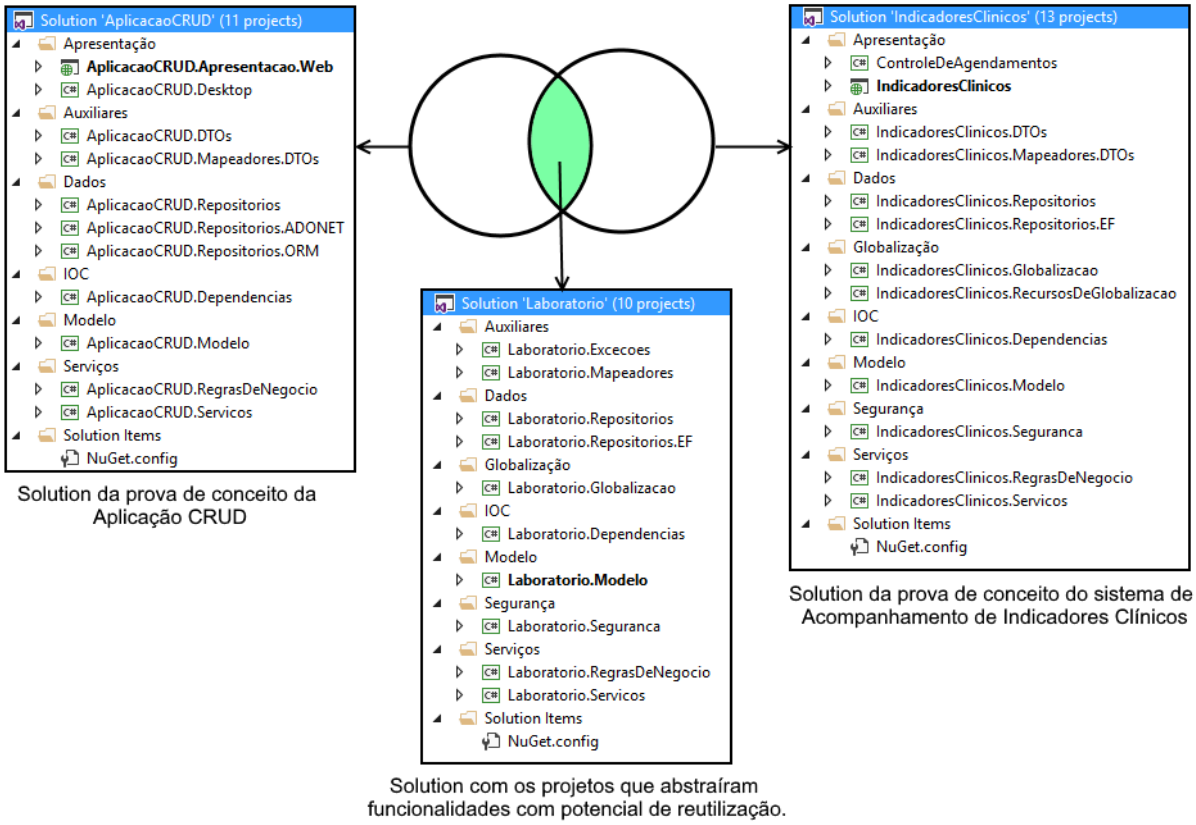


Figura 46 - Ilustração dos projetos desenvolvidos

Fonte: Elaborado pelo autor

A Figura 46 ilustra as estruturas das aplicações das provas de conceito, além da estrutura criada para abstrair as funcionalidades comuns entre elas, possibilitando que outras aplicações que vierem a ser desenvolvidas façam uso destas funcionalidades, e principalmente da própria arquitetura de *software*.

A Figura 49 exibe os projetos que foram criados e seus respectivos pacotes, sendo estes importantes por possibilitarem deploy, atualização e instalação automatizados por meio do NuGet, facilitando a reutilização dos mesmos em outras aplicações.

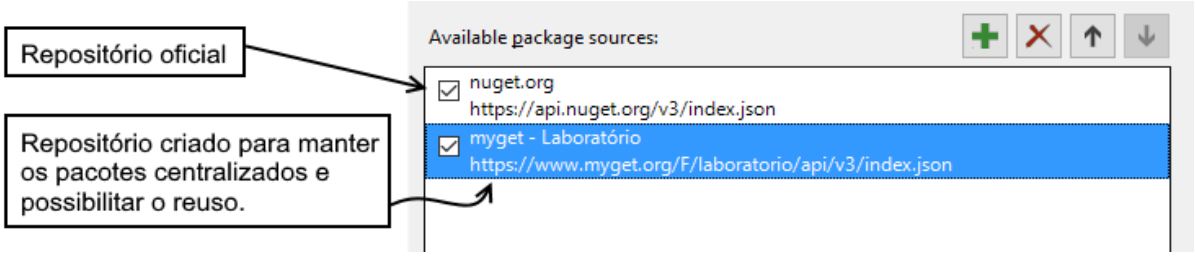


Figura 47 - Configurações de pacote do Visual Studio

Fonte: Elaborado pelo autor

Os pacotes da Figura 49 foram publicados na estrutura gratuita do myget e utilizados pelo gerenciador de pacotes padrão do Visual Studio, como pode ser observado na Figura 47. Na Figura 48 é visualizado o gerenciador de pacotes padrão utilizado pelo Visual Studio no desenvolvimento das aplicações, onde é possível visualizar que a fonte de pacotes (*package source*) em uso é aquela definida na Figura 47. Ainda que esteja selecionado a fonte de pacotes do laboratório é possível alterar para a fonte oficial do NuGet, de modo a conseguir buscar e utilizar todos os pacotes que estão publicados e mantidos em seu repositório.

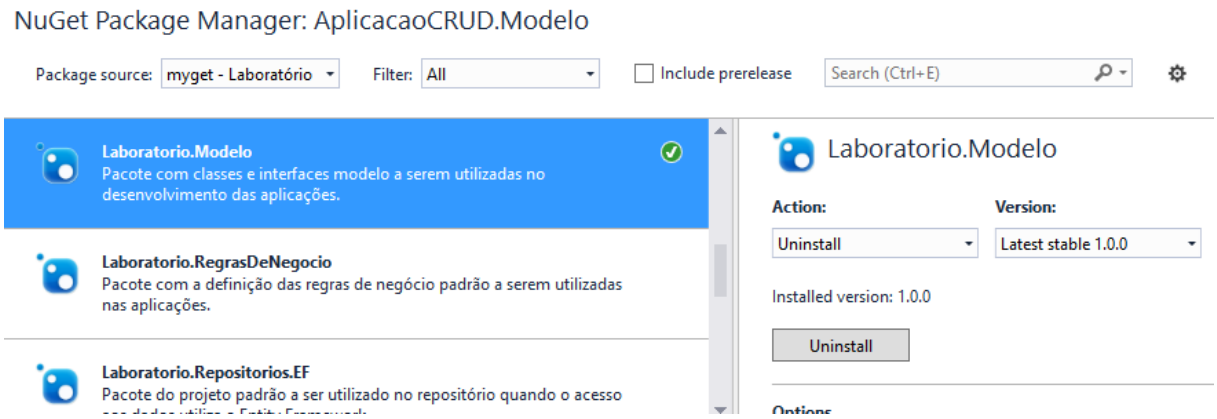


Figura 48 - Gerenciador de pacotes do Visual Studio para Aplicação CRUD

Fonte: Elaborado pelo autor

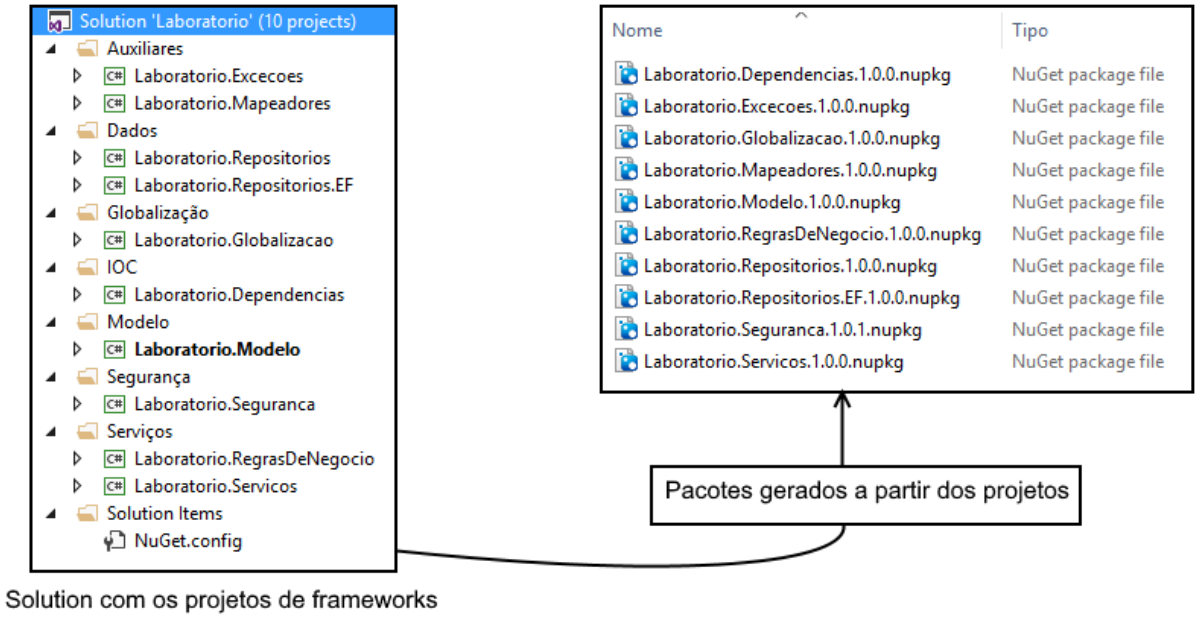


Figura 49 - Pacotes com as funcionalidades abstraídas

Fonte: Elaborado pelo autor

O uso dos *frameworks* não é requisito para utilização ou implementação da arquitetura de *software* desenvolvida, este funciona como um facilitador abstraindo funcionalidades e possibilitando sua reutilização, além de otimizar o desenvolvimento.

Ao considerarmos que os *frameworks* desenvolvidos estão sob a estrutura de gerenciamento de pacotes do NuGet, é possível que futuros desenvolvimentos de projetos incrementem novas funcionalidades ou melhorias, sendo possível uma atualização e instalação em massa, por meio do gerenciador de pacotes.

O desenvolvimento de ambas as provas de conceito tem seu pilar sobre a arquitetura de software desenvolvida neste trabalho, variando a utilização dos *frameworks*. Por estarem sob a mesma arquitetura de software as inclusões de funcionalidades nestas aplicações seguiram o mesmo modo de trabalho, que é representado na Figura 50.

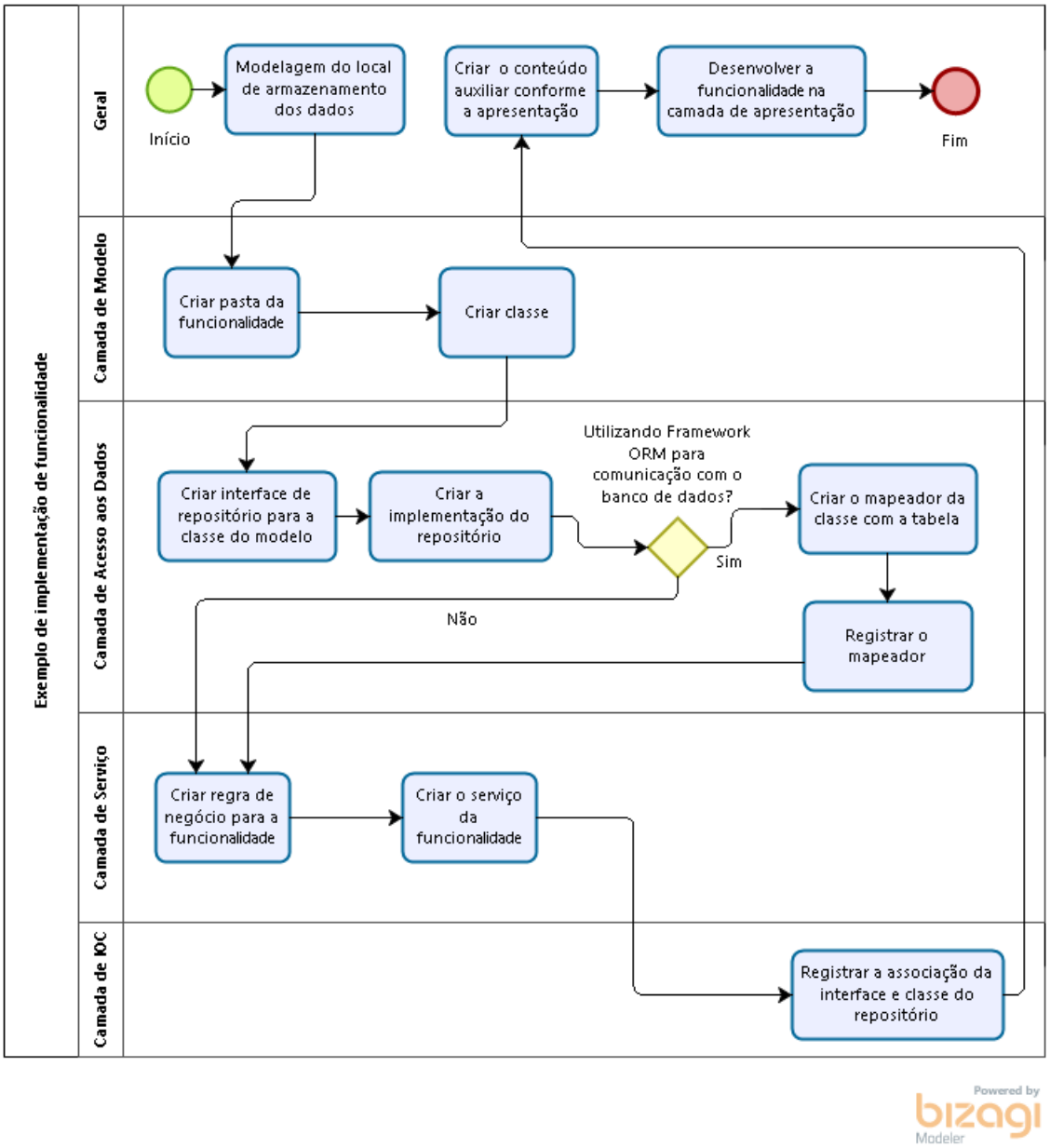


Figura 50 - Fluxo de implementação de funcionalidades

Fonte: Elaborado pelo autor

Apesar do processo representado na Figura 50 ter sido utilizado na implementação das funcionalidades, para cada uma das provas de conceito foi necessário realizar a criação da estrutura básica de desenvolvimento, sendo este processo exibido na Figura 51 e seus sub processos detalhados no Apêndice D – Guia de Desenvolvimento. No Apêndice D também está detalhado as atividades de implementação de funcionalidades, visualizadas na Figura 50.

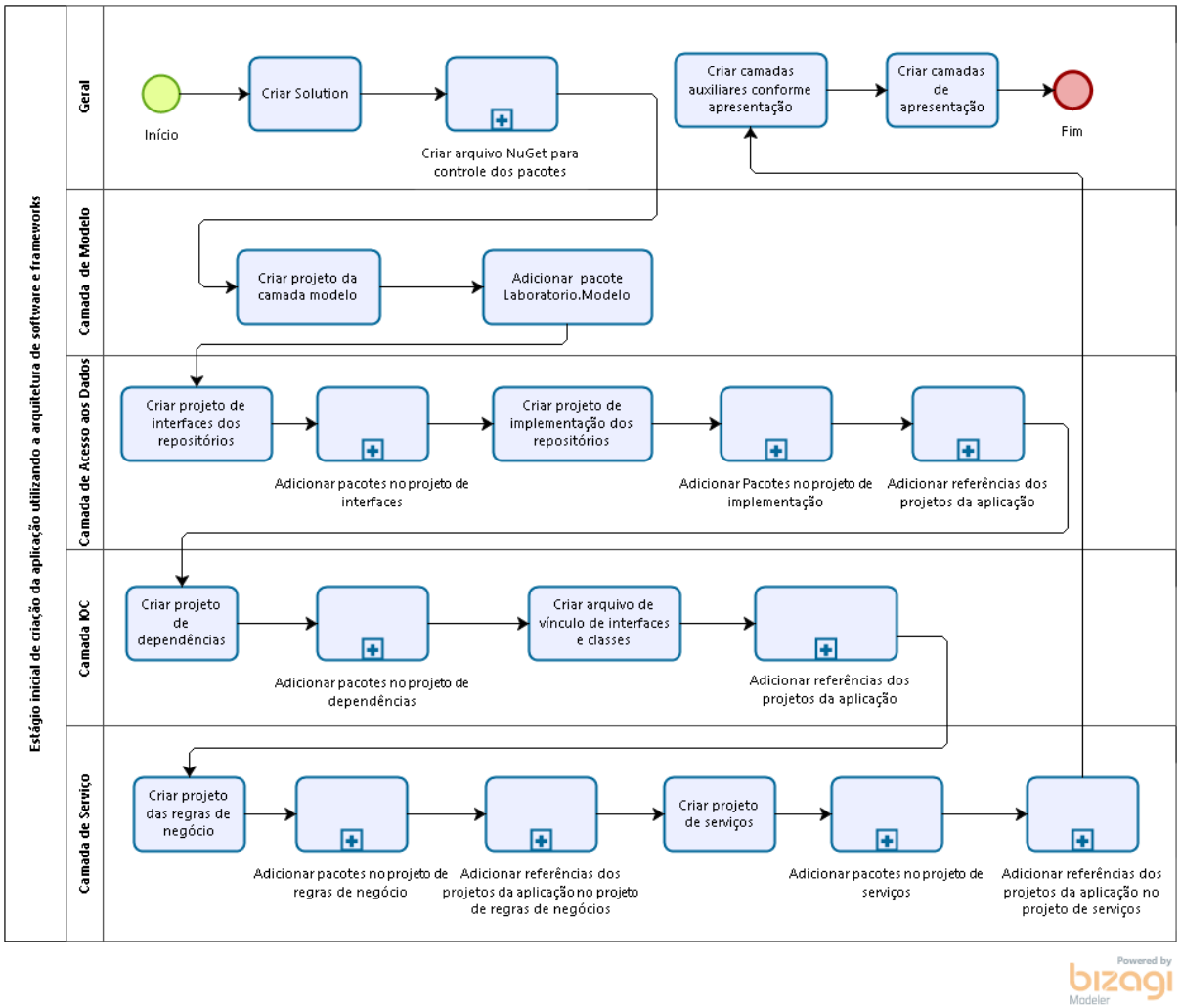


Figura 51 - Fluxo de criação da estrutura de aplicações

Fonte: Elaborado pelo autor

O fluxo de trabalho representado na Figura 51 é o mínimo necessário para a implementação da estrutura de trabalho, sendo descartado deste processo as responsabilidades das camadas de segurança e de globalização. O processo da Figura 50 não aborda assuntos de segurança e globalização por serem variáveis de um projeto para outro.

A implementação da estrutura básica de trabalho para uma aplicação que tem por requisito ser globalizada é detalhada Figura 52, e na Figura 53 é exibido o fluxo de trabalho para utilizar as funcionalidades de segurança, estando também seus sub processos detalhados no Apêndice D.

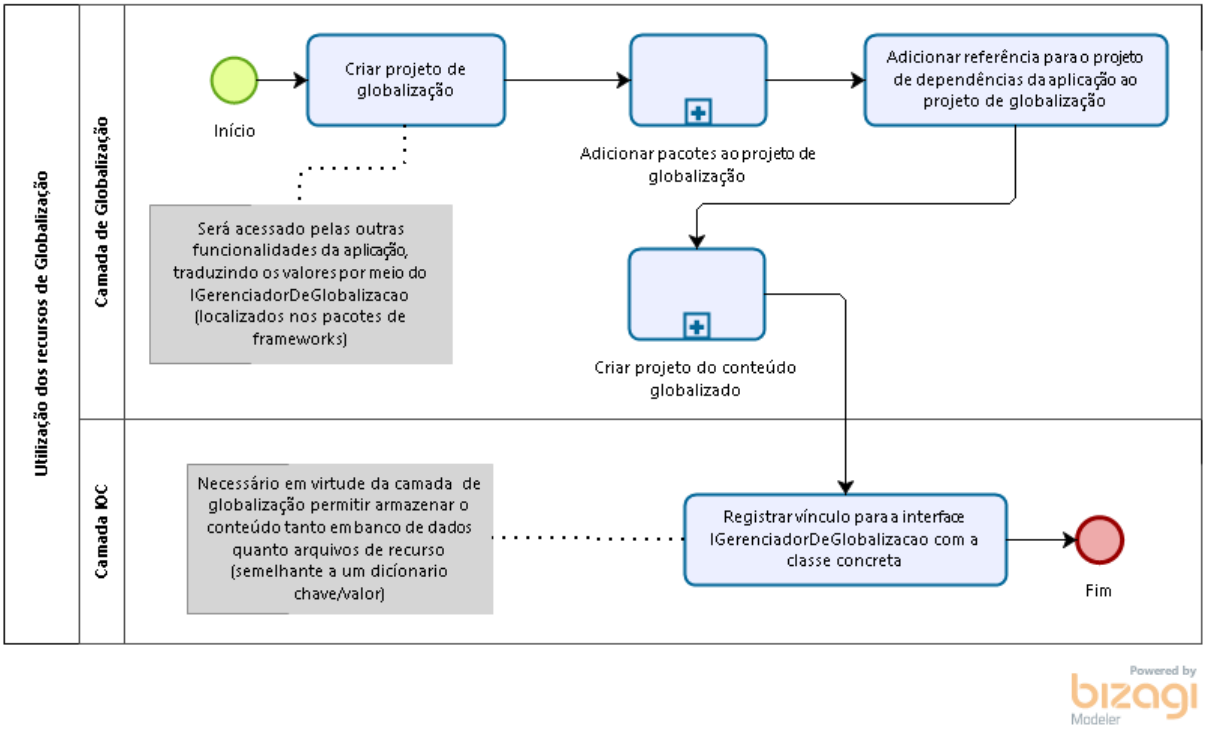


Figura 52 - Fluxo de utilização dos recursos de globalização

Fonte: Elaborado pelo autor

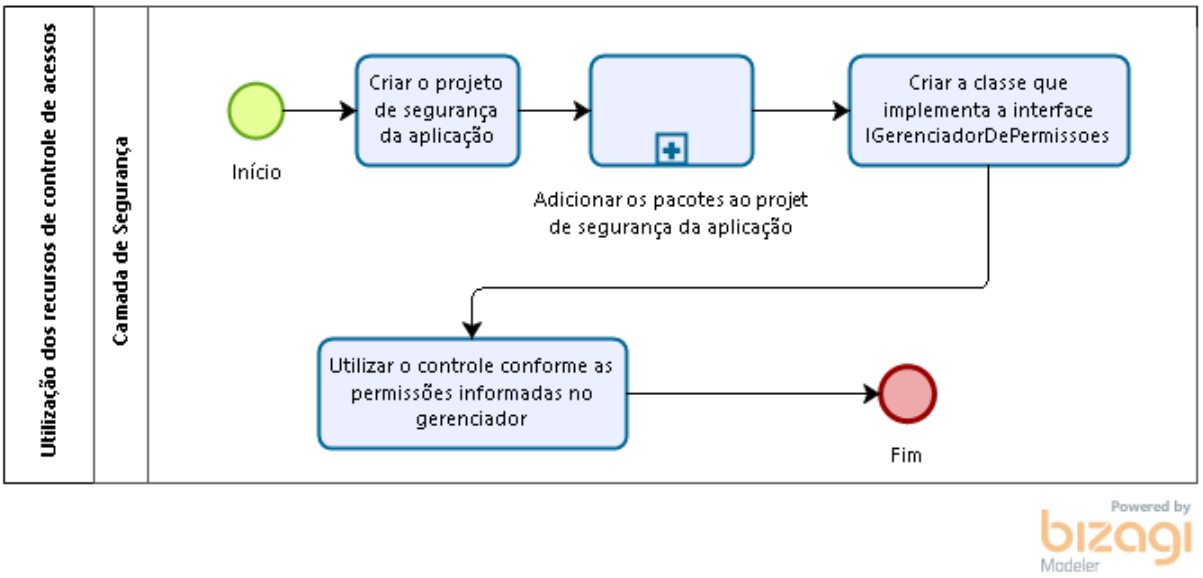


Figura 53 - Fluxo de utilização dos recursos de segurança

Fonte: Elaborado pelo autor

Neste capítulo foi apresentado o resultado do desenvolvimento das provas de conceito utilizando a arquitetura de *software* e os frameworks produzidos, além de fornecer uma versão superficial do processo de trabalho a ser seguido em projetos futuros; a versão detalhada das atividades do processo e seus sub processos estão armazenadas no Apêndice D. O capítulo a seguir trata sobre as conclusões finais deste trabalho.

6 CONSIDERAÇÕES FINAIS

Este trabalho obteve como principal resultado o desenvolvimento e aplicação de uma arquitetura de *software* reutilizável, que preserva as regras de negócio dos sistemas, indiferente da tecnologia de acesso aos dados e de interface com o usuário (*Web*, *Desktop*). A arquitetura foi aplicada em dois projetos, um utilizado como exemplo e que possui seus fins ser um modelo de desenvolvimento para outros projetos, que vierem a utilizarem a arquitetura proposta; o outro projeto desenvolvido é utilizado por usuários reais, na aplicação de questionários para pacientes que participam de estudos clínicos voltados a área da medicina.

Outro resultado obtido foram as abstrações de funcionalidades em *frameworks,* sendo estes utilizados em ambos os projetos, e possibilitando que futuros desenvolvimentos também os utilizem. Ainda que a arquitetura e os *frameworks* possuam relação entre si, é possível serem utilizados de forma independente, de modo que se use a arquitetura proposta neste trabalho e não seja utilizado os *frameworks,* ou o inverso.

Durante o desenvolvimento deste trabalho ficou evidenciado a carência de arquiteturas de *software* documentadas, disponibilizadas e aplicadas. Entre as arquiteturas elencadas na seção de trabalhos correlatos o Order Pad foi o principal pilar que direcionou este trabalho, ainda que seja desenvolvido em Java seus princípios serviram de base para mostrar um caminho focado em objetivos, e não na utilização de tecnologias específicas para acessar dados ou interagir com usuários, ainda que estes fatores sejam de real importância.

De acordo com os princípios da arquitetura de *software,* que foca na organização dos sistemas em seus componentes e os fundamentos que vão guiar o design e evolução, é possível utilizar esta arquitetura de *software* para desenvolver sistemas que preservem a integridade das regras de negócio, que é o real patrimônio do software, sendo as tecnologias uma forma de conectar usuários ao patrimônio.

6.1 TRABALHOS FUTUROS

No decorrer deste trabalho ficou evidenciado duas linhas de trabalhos futuros que permitem dar continuidade a este, sendo:

1) Refatoração para utilizar a estrutura do .Net Core: conforme mencionado no trabalho, principalmente na seção de fundamentos, o .Net está sendo revisado para tornar-se cross platform sendo possível utilizar os códigos em ambientes Linux e Mac. A partir do lançamento da versão estável do .Net Core seria possível readequar os resultados deste trabalho para usufruir das novas funcionalidades.

2) Incremento dos *frameworks*: de modo a provar que a arquitetura é reutilizável entre aplicações foi projetado e implementado duas provas de conceito, sendo as funcionalidades comuns entre elas ou com potencial de reutilização futura abstraídos em *frameworks.* Os conteúdos empacotados nestes *frameworks* ficaram limitados ao conjunto de funcionalidades de ambas as provas de conceito, de modo que, seria possível incrementar novos *frameworks* para utilização em futuros desenvolvimentos de projetos no Laboratório. Um critério a ser respeitado e mantido nesta linha de trabalho é, manter a relação entre arquitetura e os frameworks não obrigatória, sendo possível utilizar um ou ambos, sem ferir as responsabilidades de cada um.

REFERÊNCIAS

ALEXANDER, Christopher et al. **A pattern language**. Nova York: Oxford University Press, 1977.

ALUR, Deepak; CRUPI, John; MALKS, DAN. **Core J2EE patterns:** as melhores práticas e estratégias de design. Rio de Janeiro: Elsevier , 2004.

ASP.NET. **Get Started with ASP.NET Web Sites.** [s. d.]a. Disponível em <https://www.asp.net/get-started/websites>. Acesso em: 7 abr. 2016.

_____. **Mobile Apps & Sites with ASP.NET.** [s. d.]b. Disponível em <https://www.asp.net/mobile>. Acesso em: 7 abr. 2016.

_____. **Learn About ASP.NET Web API.** [s. d.]c. Disponível em <https://www.asp.net/web-api>. Acesso em: 7 abr. 2016.

_____. **Introduction to SignalR.** 2014a. Disponível em <https://www.asp.net/signalr/overview/getting-started/introduction-to-signalr>. Acesso em: 7 abr. 2016.

_____. **Introduction to ASP.NET Web Programming Using the Razor Syntax (C#).** 2014b. Disponível em <http://www.asp.net/web-pages/overview/getting-started/introducing-razor-syntax-c>. Acesso em: 13 abr. 2016.

BASS, Len; CLEMENTS, Paul; KAZMAN, Rick. **Software Architecture in Practice**. 3. ed. Addison-Wesley, 2012.

BROOKS, Frederick P. Jr. **No silver bullet**: essence and accident in software engineering. University of North Carolina. 1986.

BOOCH, Grady; JACOBSON, Ivar; RUMBAUGH, James. **The Unified Modeling Language User Guide**. Addison-Wesley. 1998.

BOOCH, Grady et al. **Object-Oriented Analysis and Design with Applications**. 3 ed. Addison-Wesley. 2007.

BUILD. **Introducing ASP.NET Core 1.0,** 2016a. Disponível em <https://channel9.msdn.com/Events/Build/2016/B810>. Acesso em: 9 abr. 2016.

BUILD. **.NET Overview,** 2016b. Disponível em <https://channel9.msdn.com/Events/Build/2016/B891>. Acesso em: 23 abr. 2016.

CHAUDHARY, Gaurav. **How to Build an Enterprise Framework for .NET Based Web Application**. 2014. Disponível em <http://www.codeproject.com/Articles/717656/How-to-Build-an-Enterprise-Framework-for-NET-Based>. Acesso em: 19 abr. 2016.

CLEMENTS, Paul; SHAW, Mary. **The Golden Age of Software Architecture**. IEEE Software, vol. 23, no. 2, 2006.

CLEMENTS, Paul; SHAW, Mary. **The Golden Age of Software Architecture - Revisited**. IEEE Software, 2009.

COCKBURN, Alistair. **Surviving Object-Oriented Projects**. AddisonWesley, 1998.

CODEPLEX. **Entity Framework**. [s. d.] Disponível em <https://entityframework.codeplex.com/>. Acesso em: 14 abr. 2016.

CUNNINGHAM, Ward. 1992. **The WyCash portfolio management system.** OOPSLA '92 Experience Report. Disponível em <http://c2.com/doc/oopsla92.html>. Acesso em 05 mar. 2016.

DEVART. **Entity Framework Code-First Migrations support for Oracle, MySQL, PostgreSQL and SQLite**, 2012. Disponível em <http://blog.devart.com/entity-framework-code-first-migrations-support-for-oracle-mysql-postgresql-and-sqlite.html>. Acesso em: 14 abr. 2016.

DNN. **About DotNetNuke**. [s. d]a. Disponível em <http://www.dnnsoftware.com/about>. Acesso em: 18 abr. 2016.

_____. **DNN Open Source Platform Technology**. [s. d]b. Disponível em <http://www.dnnsoftware.com/platform/start/architecture>. Acesso em: 18 abr. 2016.

_____. **Module Architecture**. [s. d]c. Disponível em <http://www.dnnsoftware.com/docs/developers/about-modules/module-architecture.html>. Acesso em: 18 abr. 2016.

EVANS, Eric. **Domain-driven design:** atacando as complexidades no coração do software. Rio de Janeiro: Alta Books, 2009.

FERNANDES MONTEIRO, Paula Alexandra. **Model-based Transformations for Software Architectures**: a pervasive application case study. 2005. 75 f. Dissertação (Mestre em Informática) – Universidade do Minho, Braga, 2005.

FOWLER, Martin. **Design - Who needs an architect?** IEEE Software, vol. 20, no 5, 2003a.

FOWLER, Martin. **Technical debt**. 2003b. Disponível em <http://www.martinfowler.com/bliki/TechnicalDebt.html>. Acesso em 05 mar. 2016.

FOWLER, Martin. **Inversion of Control Containers and the Dependency Injection pattern.** 2004. Disponível em <http://martinfowler.com/articles/injection.html>. Acesso em 07 jun. 2016.

FOWLER, Martin. **Inversion of Control.** 2005. Disponível em < http://martinfowler.com/bliki/InversionOfControl.html >. Acesso em 07 jun. 2016.

FOWLER, Martin. **Expositional architecture**. [s. d]a. Disponível em <http://martinfowler.com/bliki/ExpositionalArchitecture.html>. Acesso em 17 abr. 2016.

FOWLER, Martin. **Padrões de arquitetura de aplicações corporativas**. Porto Alegre: Bookman, 2006.

FOWLER, Martin. **Design stamina hypothesis**. 2007. Disponível em <http://www.martinfowler.com/bliki/DesignStaminaHypothesis.html>. Acesso em: 05 mar. 2016.

GAMMA, Erich et al. **Padrões de projeto:** soluções reutilizáveis de software orientado a objetos. São Paulo: Bookman, 2000.

GARLAN, David; SHAW, Marry. **An Introduction to Software Architecture**. Carnegie Mellon University: School of Computer Science. 1994

GARLAN, David. **Software Architecture**: a Roadmap. Carnegie Mellon University 2000.

GARLAN, David; PERRY, Dewayne. **Software Architecture: Practice, Potential, and Pitfalls**. IEEE. 1994.

HIBERNATE. **What is Object/Relational Mapping?** [s. d.]. Disponível em <http://hibernate.org/orm/what-is-an-orm>. Acesso em: 14 abr. 2016.

International Organization for Standardization; IEEE. **ISO/IEC 42010 IEEE Std 1471-2000 Systems and software engineering** — Recommended practice for architectural description of software-intensive systems. 2007.

_____. **ISO/IEC/IEEE 24765 Systems and software engineering** — Vocabulary. 2010.

JOHNSON, Ralph; FOOTE, Brian. **Designing Reusable Classes.** Journal of Object-Oriented Programming. 1988.

LARMAN, Craig. **Utilizando UML e Padrões** - Uma Introdução Á Análise e ao Projeto Orientados a Objetos e Desenvolvimento Iterativo. 3. ed. Porto Alegre: Bookman, 2007.

LUCRÈDIO, Daniel; FORTES, Renata P. M.;ALMEIDA, Eduardo S.; MEIRA, Silvio L. **Designing Domain Architectures for Model-Driven Engineering**. IEEE Software, 2010.

MARTINS AZEVEDO, Rafael Pereira. **Seleção de padrões para a arquitetura de software**: uma abordagem baseada em procura de termos e sinônimos. 2014. 92 f. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Viçosa, Programa de Pós-Graduação em Ciência da Computação, Viçosa, 2014.

MAVERICK. **Framework Maverick.NET**. [s. d.]a. Disponível em <http://mavnet.sourceforge.net>. Acesso em: 17 abr. 2016.

MAVERICK. **Framework Maverick.NET**. [s. d.]b. Disponível em <https://sourceforge.net/projects/mavnet>. Acesso em: 17 abr. 2016.

Microsoft Developer Network – MSDN. **Welcome to visual studio 2015**. [s. d.]a. Disponível em <https://msdn.microsoft.com/en-us/library/dd831853.aspx>. Acesso em: 3 mar. 2016.

_____. **Overview of the .net framework**. [s. d.]b. Disponível em <https://msdn.microsoft.com/en-us/library/zw4w595w.aspx>. Acesso em: 3 mar. 2016.

_____. **Getting Started with the .NET Framework**. [s. d.]c. Disponível em <https://msdn.microsoft.com/en-us/library/hh425099.aspx>. Acesso em: 6 abr. 2016.

_____. **Visual Studio IDE**. [s. d.]d. Disponível em <https://msdn.microsoft.com/en-us/library/dn762121.aspx>. Acesso em: 6 abr. 2016.

_____. **Cross-Platform Development in Visual Studio**. [s. d.]e. Disponível em <https://msdn.microsoft.com/en-us/library/dn771552.aspx>. Acesso em: 6 abr. 2016.

_____. **Package: NuGet Packager**. [s. d.]f. Disponível em <https://msdn.microsoft.com/en-us/Library/vs/alm/Build/steps/package/nuget-packager>. Acesso em: 7 abr. 2016.

_____. **Package: NuGet Publisher**. [s. d.]g. Disponível em <https://msdn.microsoft.com/en-us/library/vs/alm/build/steps/package/nuget-publisher>. Acesso em: 7 abr. 2016.

MOJO PORTAL. **Architecture - mojoPortal**. [s. d.]a. Disponível em <https://www.mojoportal.com/about>. Acesso em: 17 abr. 2016.

_____. **Architecture - mojoPortal**. [s. d.]b. Disponível em <https://www.mojoportal.com/architecture.aspx>. Acesso em: 17 abr. 2016.

_____. **Why I Don't Use an Object Relational Mapper in mojoPortal**. [s. d.]c. Disponível em <https://www.mojoportal.com/why-i-dont-use-an-or-mapper-in-mojoportal.aspx>. Acesso em: 17 abr. 2016.

_____. **Generating Code for mojoPortal with Codesmith**. [s. d.]d. Disponível em <https://www.mojoportal.com/generatingcodewithcodesmith.aspx>. Acesso em: 17 abr. 2016.

MILES. Rob; FOWLER, Martin. **The Architecture of the Morrison's OrderPad**. 2014. Disponível em < http://martinfowler.com/articles/orderPad>. Acesso em: 21 abr. 2016.

MYGET. **Personal and Enterprise NuGet hosting**. [s. d.] Disponível em <http://myget.org/>. Acesso em: 13 abr. 2016.

NHIBERNATE. **NHibernate** - The object-relational mapper for .NET. [s. d.]a Disponível em <http://nhibernate.info/>. Acesso em: 14 abr. 2016.

NHIBERNATE. **Your first NHibernate based application**. [s. d.]b Disponível em <http://nhibernate.info/doc/tutorials/first-nh-app/your-first-nhibernate-based-application.html>. Acesso em: 14 abr. 2016.

NUGET. **NuGet Overview.** [s. d.]a Disponível em <http://docs.nuget.org/consume/overview>. Acesso em: 7 abr. 2016.

NUGET. **NuGet.Server.** [s. d.]b Disponível em <https://www.nuget.org/packages/NuGet.Server>. Acesso em: 7 abr. 2016.

OMNISHARP. **OmniSharp** - .NET and IntelliSense on any platform with your editor of choice. [s. d.] Disponível em <http://www.omnisharp.net>. Acesso em: 12 abr. 2016.

ORACLE. **Oracle® Communications Unified Inventory Management Developer's Guide Release 7.2.2:** 3 Using the Persistence Framework. 2013. Disponível em <https://docs.oracle.com/cd/E36032_01/doc.722/e36039/dev_persistence_fw.htm#g1042312>. Acesso em: 14 abr. 2016.

PERRY, Dewayne E.; WOLF, Alexander L. **Foundations for the Study of Software Architecture**. ACM SIGSOFT SOFTWARE ENGINEERING NOTES, vol 17, no 4. 1992

PEREIRA E SILVA, Ricardo. **Suporte ao desenvolvimento e uso de frameworks e componentes**. 2000. 262 f. Tese (Doutorado em Ciência da Computação) –

Universidade Federal do Rio Grande do Sul, Program de Pós-Graduação em Computação, Porto Alegre, 2000.

PRESSMAN, Roger S; MAXIN, Bruce R. **Engenharia de Software** – Uma abordagem profissional. 8. ed. São Paulo: Bookman, 2016.

POLLAK, Mark. **Introduction to Spring.NET**. 2008. Disponível em <http://www.infoq.com/presentations/pollack-intro-spring>. Acesso em: 17 abr. 2016.

ROSENBERG, Doug. STEPHENS, Matt. **Use Case Driven Object Modeling with UML:** Theory and Practice. United States of America: Apress, 2007.

SAUVÉ, Jacques Philippe. **Frameworks** – O que é um framework? [s. d.]a Disponível em <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/frame/oque.htm>. Acesso em: 5 abr. 2016.

SAUVÉ, Jacques Philippe. **Frameworks** – Vantagens e desvantagens no uso de frameworks [s. d.]b. Disponível em < http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/frame/porque.htm >. Acesso em: 5 abr. 2016.

SAUVÉ, Jacques Philippe. **Frameworks** – Tipos de frameworks [s. d.]c. Disponível em < http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/frame/tipos.htm>. Acesso em: 5 abr. 2016.

SILVA CARNEIRO, Cristiane Marise Pérez. **Frameworks de Aplicações Orientadas a Objetos** - Uma Abordagem Iterativa e Incremental. 2003. 121 f. Dissertação (Mestrado em Redes de Computadores) – Universidade Salvador, Salvador, 2003.

SPRING.NET. **Spring.NET framework**. 2011. Disponível em <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/frame/tipos.htm>. Acesso em: 17 abr. 2016.

SOMMERVILLE, Ian. **Engenharia de software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011.

TOGAF. **The Open Group Architecture Framework**, 2011. Disponível em <http://pubs.opengroup.org/architecture/togaf9-doc/arch/>. Acesso em: 12 abr. 2016.

INTRODUÇÃO AOS APÊNDICES

Os capítulos dos apêndices a seguir abordam temas específicos relacionados a este trabalho de forma mais detalhada. No Apêndice A é armazenado as telas que abrangem a camada de segurança utilizada pelas aplicações que fizeram uso do *framework.*

As modelagens das aplicações que foram utilizadas como provas de conceito encontram-se no Apêndice B para a Aplicação CRUD, e no Apêndice C para o *software* de Acompanhamento de Indicadores Clínicos e as diretrizes para sua instalação no Apêndice E.

O Apêndice D mantém um guia de desenvolvimento para nortear a utilização dos *frameworks* criados e da arquitetura de software.

APÊNDICE A - CAMADA DE SEGURANÇA

Armazena-se neste apêndice as figuras de telas dos cadastros e listagens de usuário e perfil, referente a camada de segurança da arquitetura de *software* elaborado neste trabalho, que será abstraído em um framework, possibilitando a utilização em outros projetos.

Figura 1 – Tela dos perfis de uma aplicação

Fonte: Elaborado pelo autor

A Figura 2 representa a tela do cadastro de perfis, sendo possível visualizar o agrupamento de Usuários e Permissões que compõem o determinado perfil.

Figura 2 – Cadastro de perfil: Informações Gerais

Fonte: Elaborado pelo autor

Figura 3 – Cadastro de perfil: Usuários

Fonte: Elaborado pelo autor

Visualiza-se na Figura 3 os usuários que fazem parte de determinado perfil, e na Figura 4 a listagem dos usuários cadastrados na aplicação.

Figura 4 – Lista de usuários criados

Fonte: Elaborado pelo autor

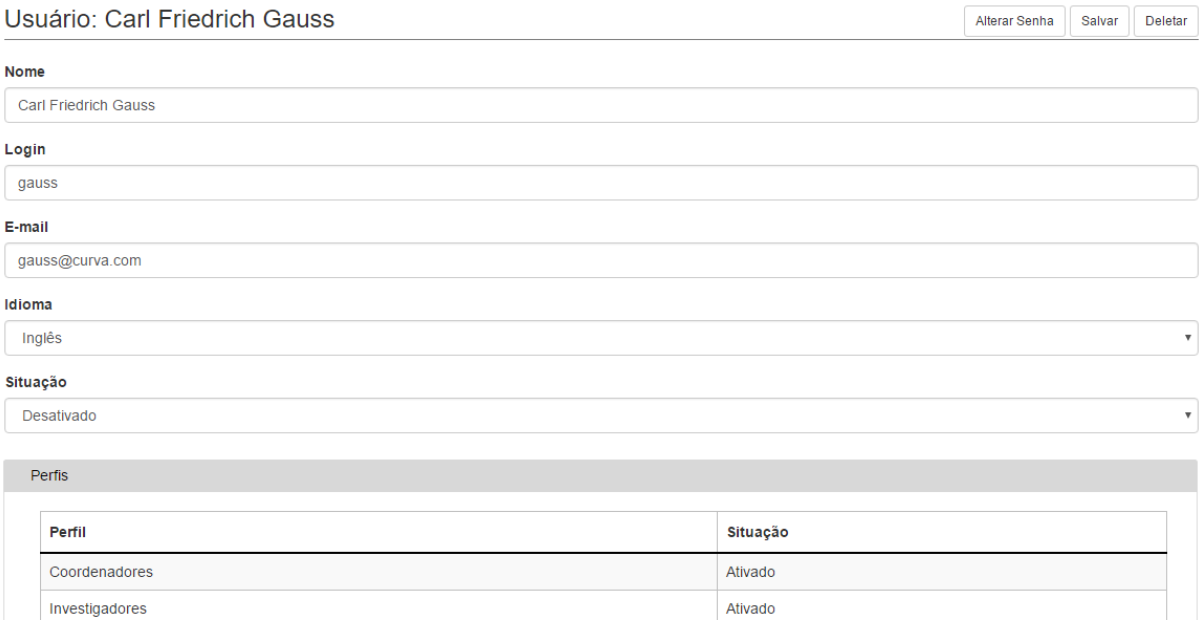


Figura 5 – Castro de usuário

Fonte: Elaborado pelo autor

A Figura 5 corresponde a tela do cadastro de usuário de uma determinada aplicação que faz uso do mecanismo de segurança.

APÊNDICE B - MODELAGEM DA APLICAÇÃO CRUD

Armazenam-se neste apêndice os diagramas de sequência, de acordo com a modelagem o ICONIX, para a aplicação *web* e para a *desktop*, visualizadas na Figura 1 e na Figura 2.

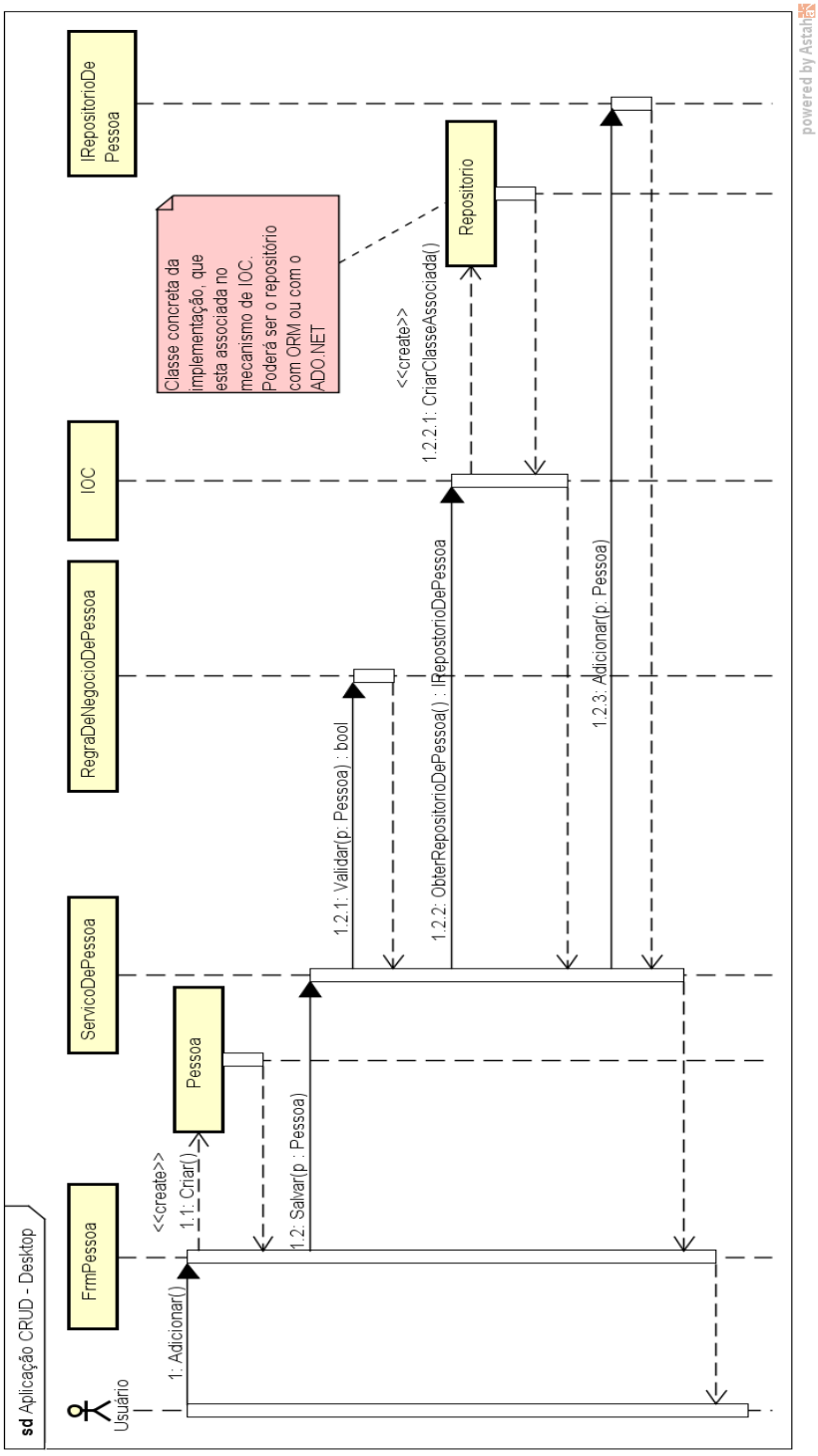


Figura 1 – Diagrama de sequência da camada desktop

Fonte: Elaborado pelo autor

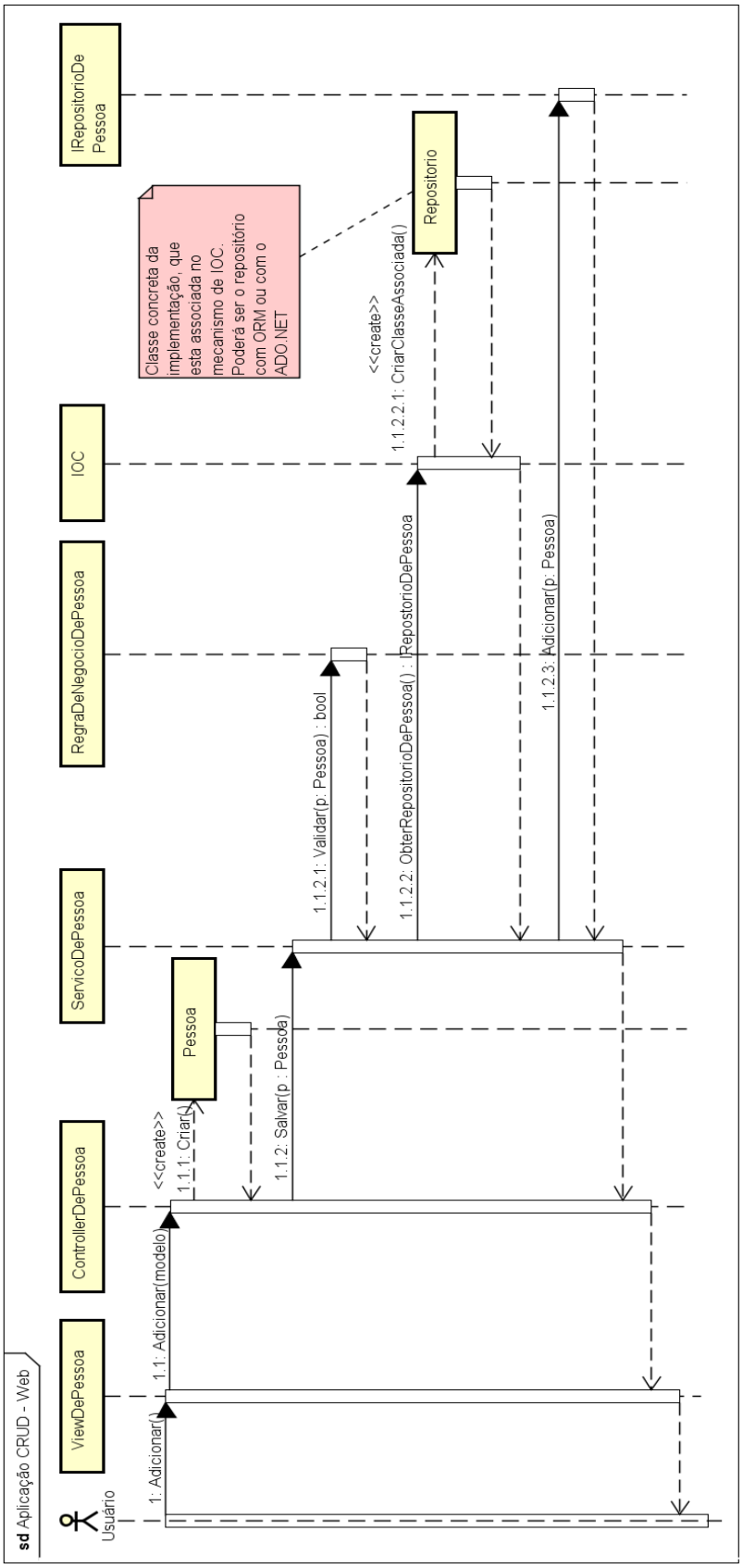


Figura 2 – Diagrama de sequência da camada *web*

Fonte: Elaborado pelo autor

APÊNDICE C - ACOMPANHAMENTO DE INDICADORES CLÍNICOS

Este apêndice complementa a modelagem da aplicação na seção 4.2.2 Acompanhamento de Indicadores Clínicos, de acordo com a metodologia ICONIX.

1 MODELO DE DOMÍNIO

A Figura 1 retrata o modelo de domínio da aplicação. As classes de usuário e perfis não estão representadas porque utiliza-se as da camada de segurança.

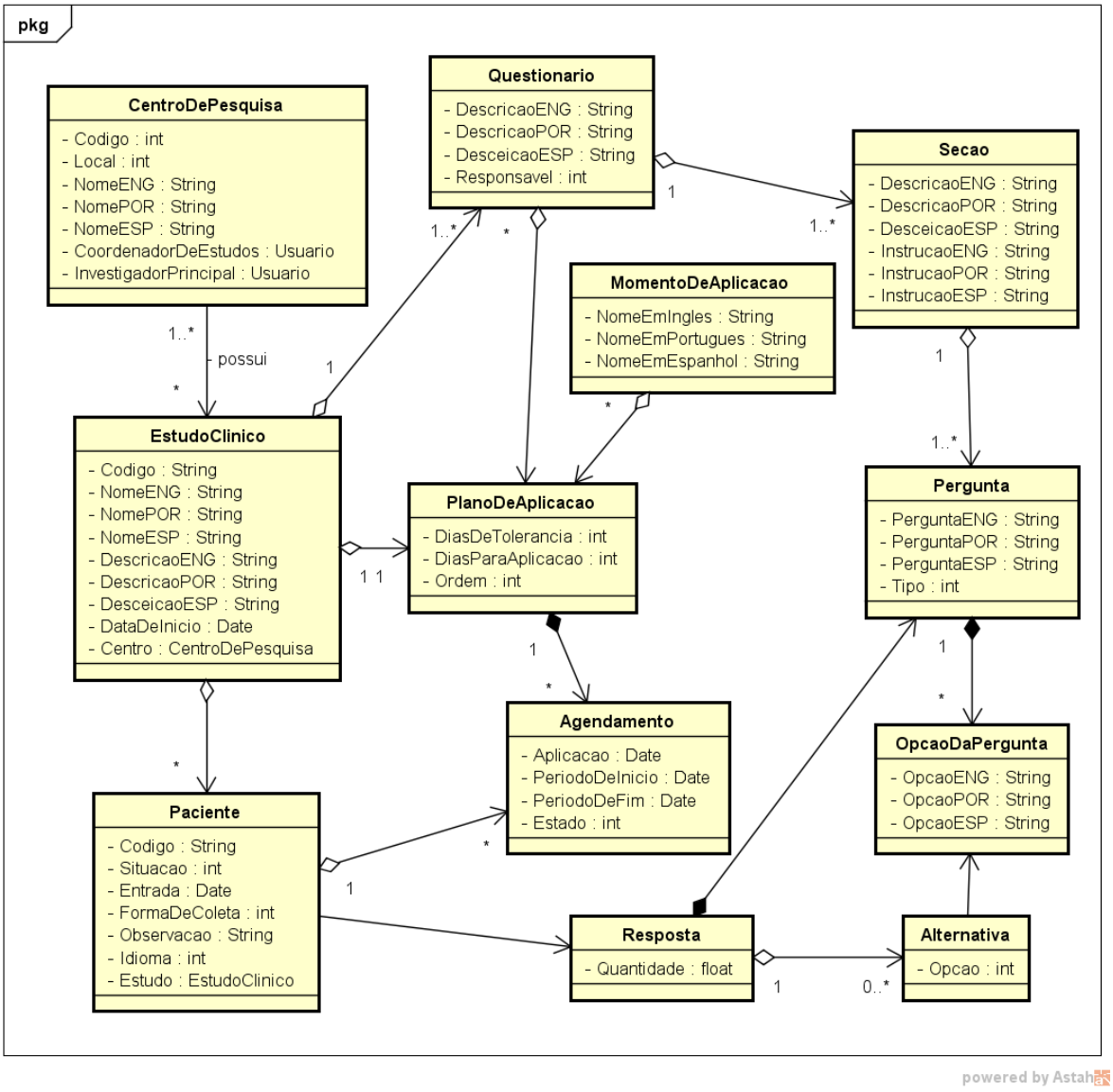


Figura 1 – Diagrama de classes referentes aos requisitos de negócio da aplicação

Fonte: Elaborado pelo autor

2 CASOS DE USO

Nesta seção são detalhados todos os casos de uso visualizados no diagrama na seção 4.2.2 Acompanhamento de Indicadores Clínicos. Criou-se somente um diagrama de sequência para exemplificar o funcionamento da aplicação, de acordo com a arquitetura de software, não sendo criado para todos os casos de uso pela similaridade de funcionamento.

CADASTRAR CENTROS DE PESQUISA **Ator: Usuário**
CURSO BÁSICO: O usuário preenche as informações do cadastro, sendo obrigatório os campos código, coordenador de estudos, investigador principal e ao menos um nome.
CURSO ALTERNATIVO: **Investigador principal e coordenador de estudos são o mesmo usuário:** Não permite prosseguir com a operação, informando ao usuário a mensagem “O Investigador principal e Coordenador de estudos não podem ser o mesmo usuário”.
PRECONDIÇÕES: Usuário autenticado na aplicação. Usuário com permissão para acessar a funcionalidade. Possuir no mínimo dois usuários cadastrados no sistema.
PÓS CONDIÇÕES: Centro de pesquisa cadastrado

Quadro 1 – Caso de uso: Cadastrar centros de pesquisa

Fonte: Elaborado pelo autor

No Quadro 1 detalha-se o caso de uso de cadastro de centros de pesquisa, e seus cadastros correspondem a Figura 2 e a Figura 3.

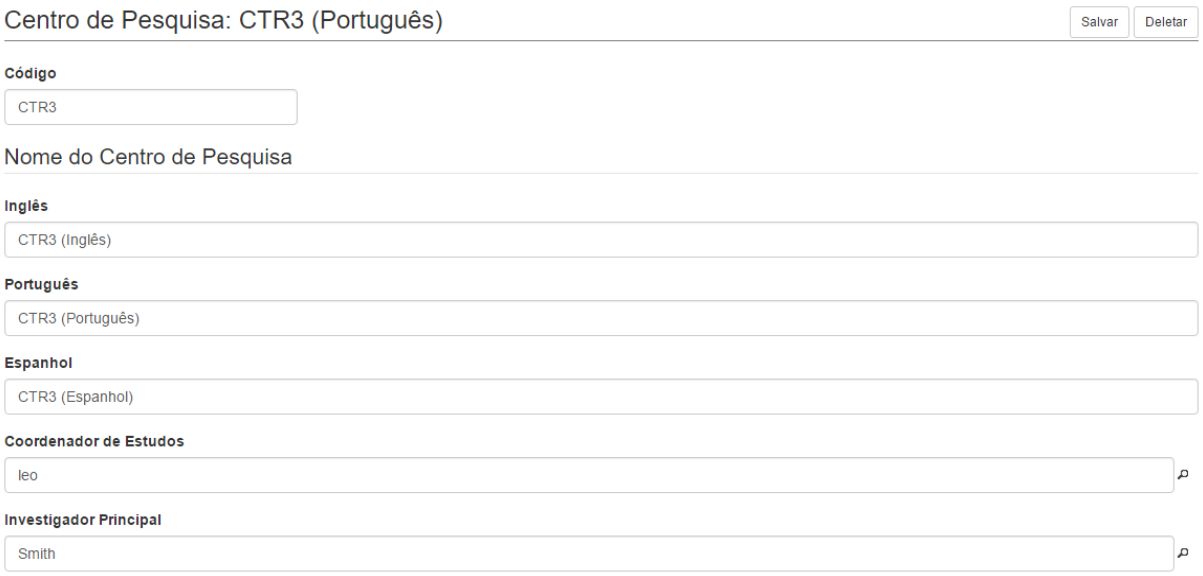


Figura 2 – Cadastro de centros de pesquisa

Fonte: Elaborado pelo autor

Figura 3 – Exibição dos centros de pesquisa cadastrados

Fonte: Elaborado pelo autor

No Quadro 2 detalha-se o caso de uso referente ao cadastro de pacientes. A Figura 4 corresponde a tela de cadastro de pacientes.

CADASTRAR PACIENTES **Ator: Usuário**
CURSO BÁSICO: Ao cadastrar um novo paciente o sistema sugere como situação o valor “Participando”. O usuário preenche as informações do paciente, sendo obrigatório

o preenchimento dos campos código, situação, data da entrada, forma de coleta e idioma.
CURSO ALTERNATIVO: **Forma de coleta pelo próprio paciente:** A página deve habilitar um campo e-mail, que tem seu preenchimento obrigatório.
PRECONDIÇÕES: Usuário autenticado na aplicação. Usuário com permissão para acessar a funcionalidade. Selecionado estudo clínico.
PÓS CONDIÇÕES: Paciente cadastrado.

Quadro 2 – Caso de uso: Cadastrar pacientes

Fonte: Elaborado pelo autor

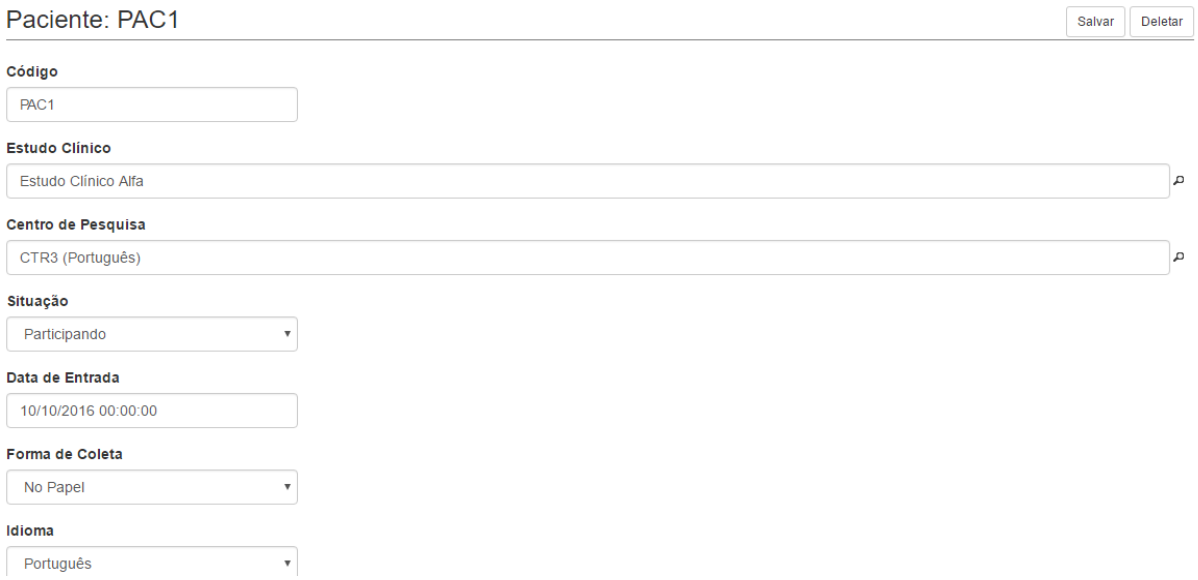


Figura 4 – Cadastro de pacientes Informações gerais

Fonte: Elaborado pelo autor

Agendamentos

Questionário	Estado	Data de Resposta	Prazo	Responsável	
1. Período Pré-Operatório	Respondido	17/10/2016	12/10/2016 até 17/10/2016	Outros	
1. Período Pré-Operatório	Respondido	18/10/2016	15/10/2016 até 20/10/2016	Paciente	
2. Informações Cirúrgicas	Aguardando		20/10/2016 até 30/10/2016	Paciente	
3. Alta Hospitalar	Aguardando		09/11/2016 até 19/11/2016	Outros	
4. Análise de Custo	Aguardando		09/11/2016 até 19/11/2016	Outros	
5. Período Pós-Operatório	Aguardando		09/11/2016 até 16/11/2016	Paciente	
5. Período Pós-Operatório	Aguardando		08/01/2017 até 18/01/2017	Paciente	
5. Período Pós-Operatório	Aguardando		08/04/2017 até 22/04/2017	Paciente	
5. Período Pós-Operatório	Aguardando		05/10/2017 até 21/10/2017	Paciente	

Figura 5 – Cadastro de pacientes: Agendamentos

Fonte: Elaborado pelo autor

O cadastro da Figura 5 corresponde aos agendamentos programados para um determinado paciente, e o da Figura 6 representa a lista de pacientes cadastrados na aplicação. O diagrama de sequência, para o cadastro de pacientes, visualiza-se na Figura 7

Pacientes +

Código	Situação	Estudo Clínico	Centro de Pesquisa	Forma de Coleta	Entrada
PAC1	Participando	Estudo Clínico Alfa	CTR3 (Português)	No Papel	10/10/2016
PAC2	Participando	Estudo Clínico Alfa	CTR3 (Português)	Próprio Paciente	22/10/2016

Figura 6 – Exibição dos pacientes cadastrados

Fonte: Elaborado pelo autor

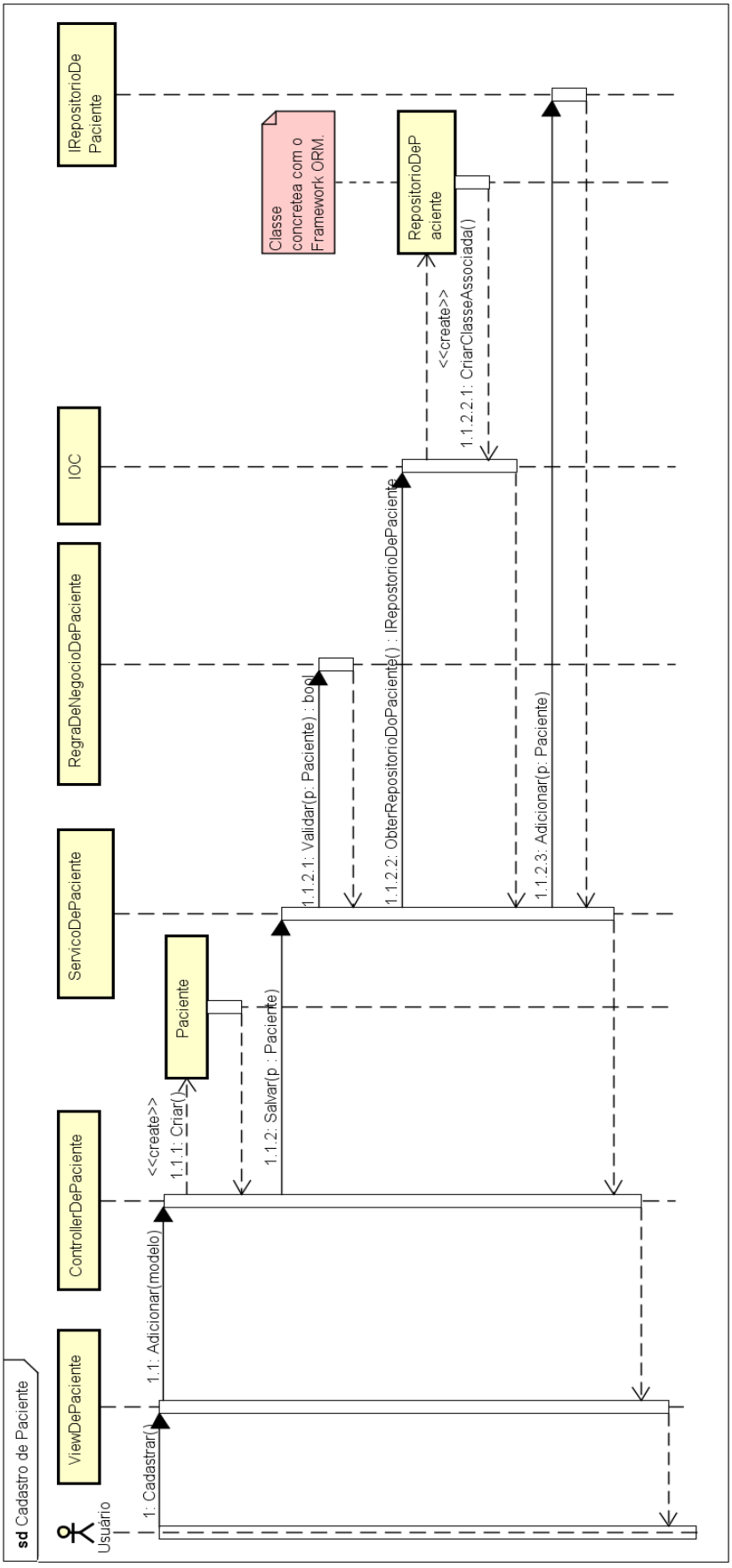


Figura 7 – Diagrama de sequência do cadastro de pacientes

Fonte: Elaborado pelo autor

GERENCIAR ESTUDOS CLÍNICOS **Ator: Usuário**
CURSO BÁSICO: O usuário deve preencher as informações, sendo obrigatórios os campos centro de pesquisa coordenador, código, data de início do estudo.
CURSO ALTERNATIVO: **Estudo clínico com centros de pesquisa participantes:** Na região “Centros de Pesquisa Participantes” é possível adicionar múltiplos centros de pesquisa que estão participando ativamente daquele determinado estudo clínico. **Criar plano de aplicação:** Na região “Plano de Aplicação” vincula-se os questionários informando a ordem de aplicação, os dias utilizados para aplicar este questionário aos pacientes (que considera a data de entrada de um paciente), os dias de tolerância para aceitar as respostas. Todos os campos são obrigatórios. **Vincular pacientes ao estudo clínico:** Na região “Pacientes” vincula-se os pacientes cadastrados para aquele determinado estudo clínico. O sistema deverá criar os agendamentos do paciente automaticamente, respeitando o plano de aplicação do estudo, sendo que, estes agendamentos são visualizados na aba de agendamentos do cadastro de pacientes. **Vincular um paciente já existente ao estudo clínico:** O sistema deve informar ao usuário a mensagem “O paciente Código X já está vinculado ao estudo”.
PRECONDIÇÕES: Usuário autenticado na aplicação. Usuário com permissão para acessar a funcionalidade.
PÓS CONDIÇÕES: Estudo clínico cadastrado.

Quadro 3 – Caso de uso: Gerenciar estudos clínicos

Fonte: Elaborado pelo autor

No Quadro 3 detalha-se o caso de uso de gerenciar estudos clínicos.

Figura 8 – Exibição dos estudos clínicos cadastrados

Fonte: Elaborado pelo autor

A Figura 8 e a Figura 9 correspondem as telas para gerenciamento de estudos clínicos.

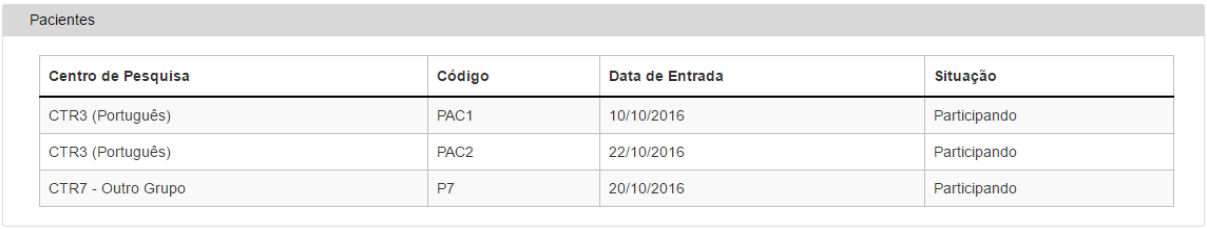
Figura 9 – Cadastro de estudo clínico: Informações gerais

Fonte: Elaborado pelo autor

Figura 10 – Cadastro de estudo clínico: Plano de aplicação

Fonte: Elaborado pelo autor

Ilustra-se na Figura 10 o cadastro do plano de aplicação dos questionários de um estudo clínico, e na Figura 11 os pacientes associados ao estudo que participaram do plano de aplicação.


Pacientes

Centro de Pesquisa	Código	Data de Entrada	Situação
CTR3 (Português)	PAC1	10/10/2016	Participando
CTR3 (Português)	PAC2	22/10/2016	Participando
CTR7 - Outro Grupo	P7	20/10/2016	Participando

Figura 11 – Cadastro de estudo clínico: Pacientes do estudo

Fonte: Elaborado pelo autor

MANIPULAR QUESTIONÁRIOS **Ator: Usuário**
CURSO BÁSICO: O usuário preenche as informações básicas do questionário, sendo obrigatório os campos código e responsável. O sistema deve sugerir para o campo responsável o valor “Paciente”.
CURSO ALTERNATIVO: **Adicionar seções do questionário:** Ao adicionar uma seção para o questionário o usuário deve preencher ao menos um título e uma descrição para o mesmo, sendo estes correspondentes ao mesmo idioma. Por exemplo: preencher o título em inglês e a descrição em espanhol é considerado inválido e o sistema deve exibir uma mensagem informando ao usuário. As seções funcionam como agrupamentos de perguntas. **Adicionar perguntas do questionário:** O sistema deve solicitar o tipo da pergunta a ser criada (múltipla escolha, única escolha ou numérica). Ao escolher o tipo, o sistema deve adicionar uma nova pergunta na seção selecionada. Para uma pergunta é obrigatório o preenchimento do campo título. **Remover perguntas e seções do questionário:** O usuário deve selecionar a seção ou pergunta que deseja excluir, e o sistema deve solicitar confirmação, antes

de remover uma seção ou pergunta, sendo que, ao remover uma seção todas as perguntas associadas a ela são excluídas por consequência. **Adicionar novas opções de resposta:** O usuário adiciona uma nova opção e informa o título para a opção. O sistema inclui a nova opção no conjunto da pergunta. **Remover uma opção de resposta:** O usuário solicita que determinada opção de resposta deve ser excluída. O sistema deve pedir confirmação e caso afirmativo, remover a opção selecionada.
PRECONDIÇÕES: Usuário autenticado na aplicação. Usuário com permissão para acessar a funcionalidade.
PÓS CONDIÇÕES: Questionário cadastrado.

Quadro 4 – Caso de uso: Manipular questionários

Fonte: Elaborado pelo autor

No Quadro 4 detalha-se o caso de uso de manipulação de questionários, e na Figura 12 o cadastro dos questionários cadastrados na aplicação.

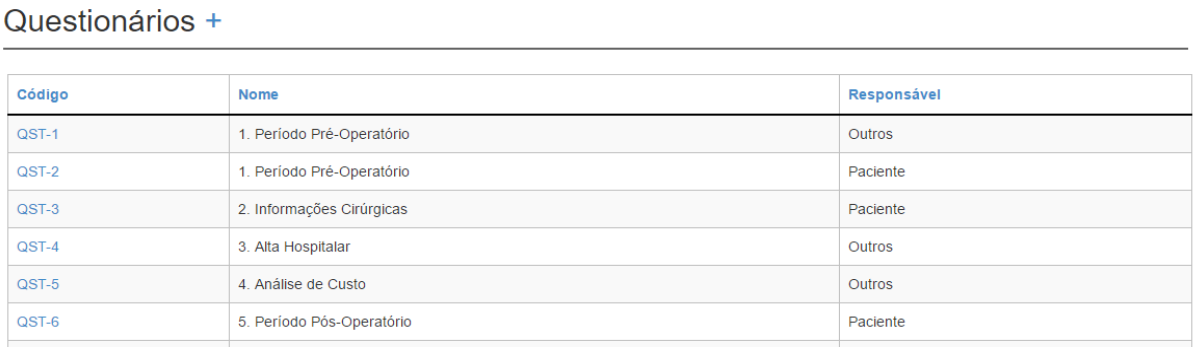


Figura 12 – Exibição dos questionários cadastrados

Fonte: Elaborado pelo autor

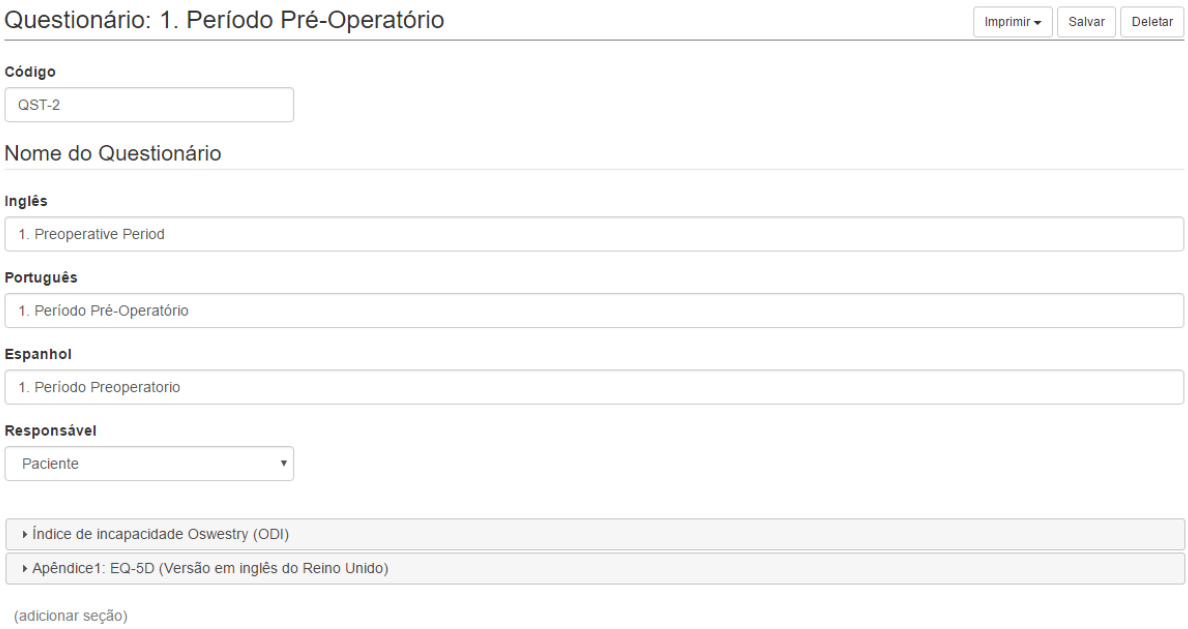


Figura 13 – Cadastro de questionário

Fonte: Elaborado pelo autor

A Figura 13 e a Figura 14 correspondem aos cadastros que permitem manipular questionários. A Figura 13 ilustra o questionário em uma visão com as seções recolhidas, e na Figura 14 visualiza-se o mesmo questionário com as seções expandidas, permitindo que sejam incluídas ou removidas novas perguntas, bem como opções destas.

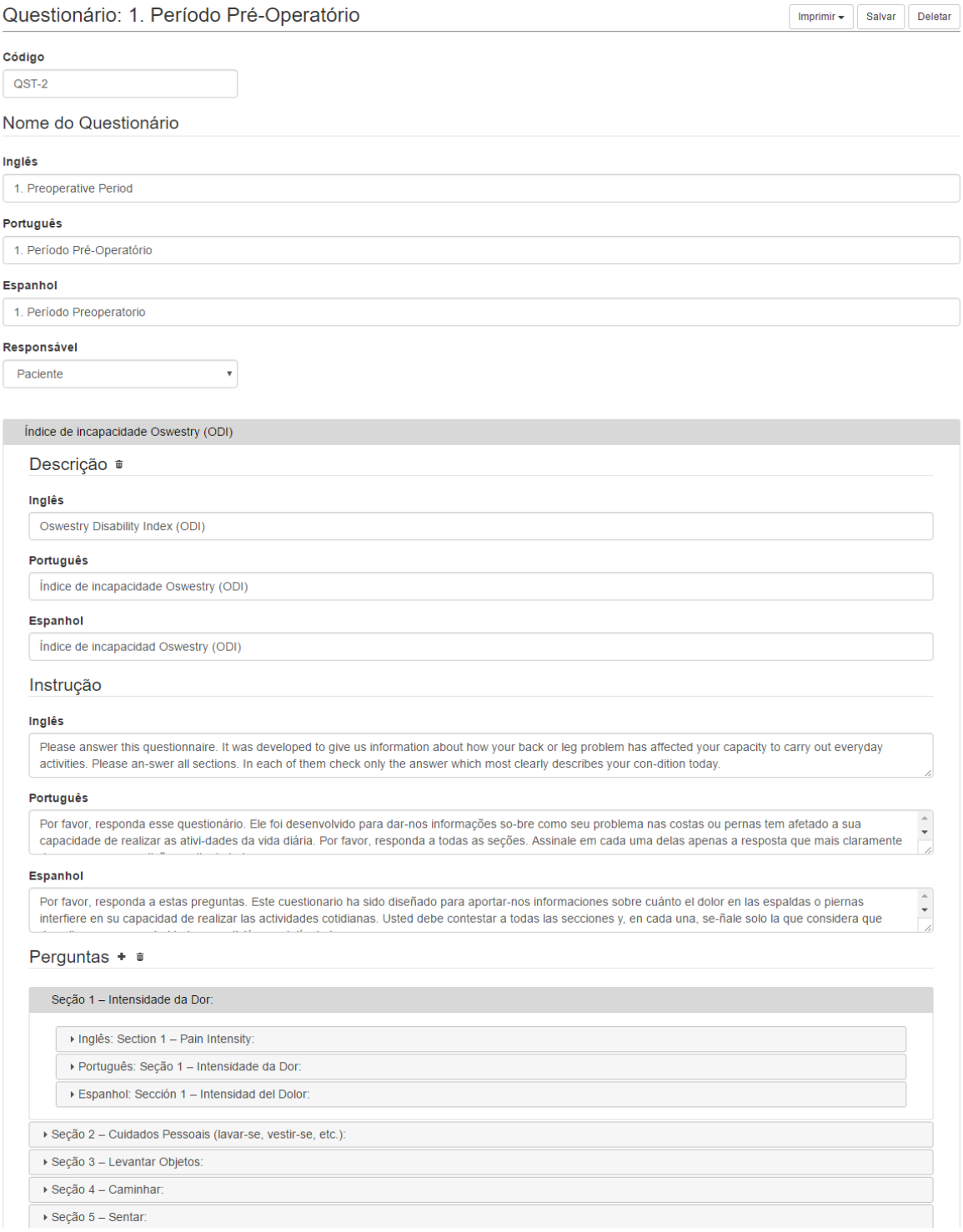


Figura 14 – Manipulação do questionário com seção das perguntas expandidas

Fonte: Elaborado pelo autor

IMPRIMIR QUESTIONÁRIOS **Ator: Usuário**
CURSO BÁSICO:

O usuário informa o idioma para a impressão. O sistema gera uma página de impressão em PDF com o conteúdo do questionário no determinado idioma.
CURSO ALTERNATIVO: **Idioma informado não preenchido no questionário:** O sistema deve informar ao usuário que o questionário não possui conteúdo no determinado idioma da impressão.
PRECONDIÇÕES: Usuário autenticado na aplicação. Usuário com permissão para acessar a funcionalidade. Idioma do paciente deve estar preenchido no questionário.
PÓS CONDIÇÕES: Relatório emitido para o idioma correspondente ao selecionado.

Quadro 5 – Caso de uso: Imprimir questionários

Fonte: Elaborado pelo autor

O Quadro 5 representa os detalhes do caso de uso referente a impressão de questionários.

RESPONDER QUESTIONÁRIOS **Ator: Usuário e Paciente**
CURSO BÁSICO: O paciente abre o link único, enviado para seu e-mail pelo sistema, e responde o questionário. O sistema gera o questionário com o conteúdo e informações de tela a partir do idioma do paciente.
CURSO ALTERNATIVO: **Prazo para responder vencido:** O usuário ao abrir o link de resposta é informado que perdeu o prazo para preencher o questionário. **Coleta das respostas via papel:** O sistema exibe o questionário para que sejam computadas as respostas que foram fornecidas em um questionário impresso. O usuário transcreve as respostas do relatório impresso para o computador.

PRECONDIÇÕES: Usuário autenticado na aplicação. Usuário com permissão para acessar a funcionalidade. Paciente com agendamento vigente.
PÓS CONDIÇÕES: Questionário respondido.

Quadro 6 – Caso de uso: Responder questionários

Fonte: Elaborado pelo autor

O Quadro 6 representa o caso de uso responder questionários, e a Figura 15 corresponde a interface para coletar as respostas na visualização do médico registrando para o paciente.

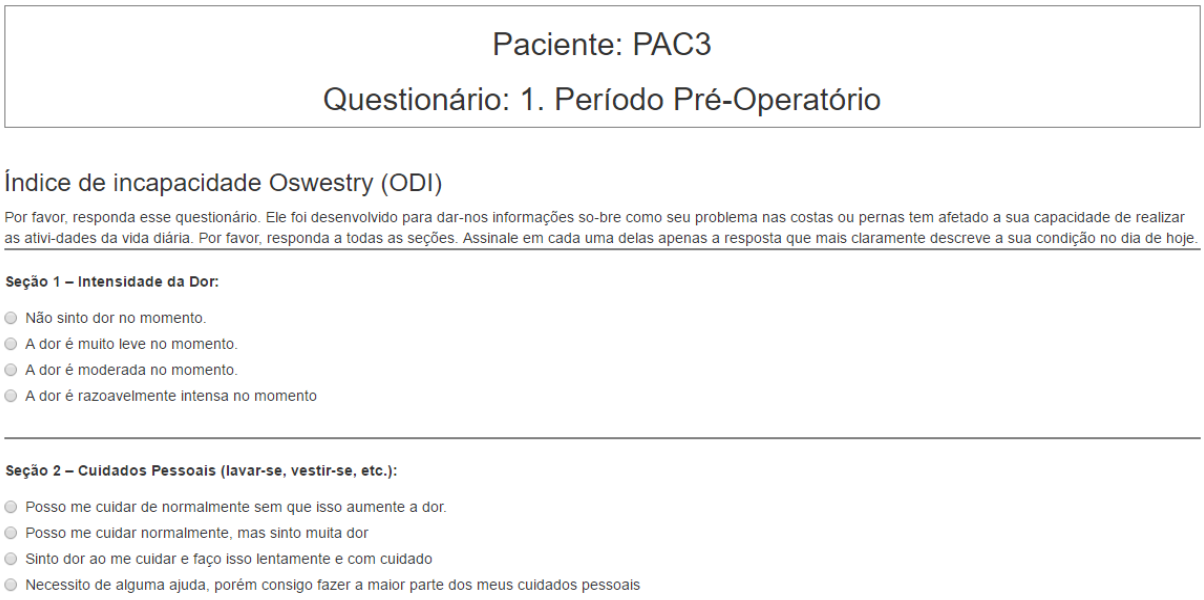


Figura 15 – Tela de aplicação dos questionários visão do médico

Fonte: Elaborado pelo autor

A Figura 16 mostra o registro das respostas na visualização o próprio paciente respondendo, sendo o caso onde ele recebeu por e-mail um link único que direciona para o questionário.

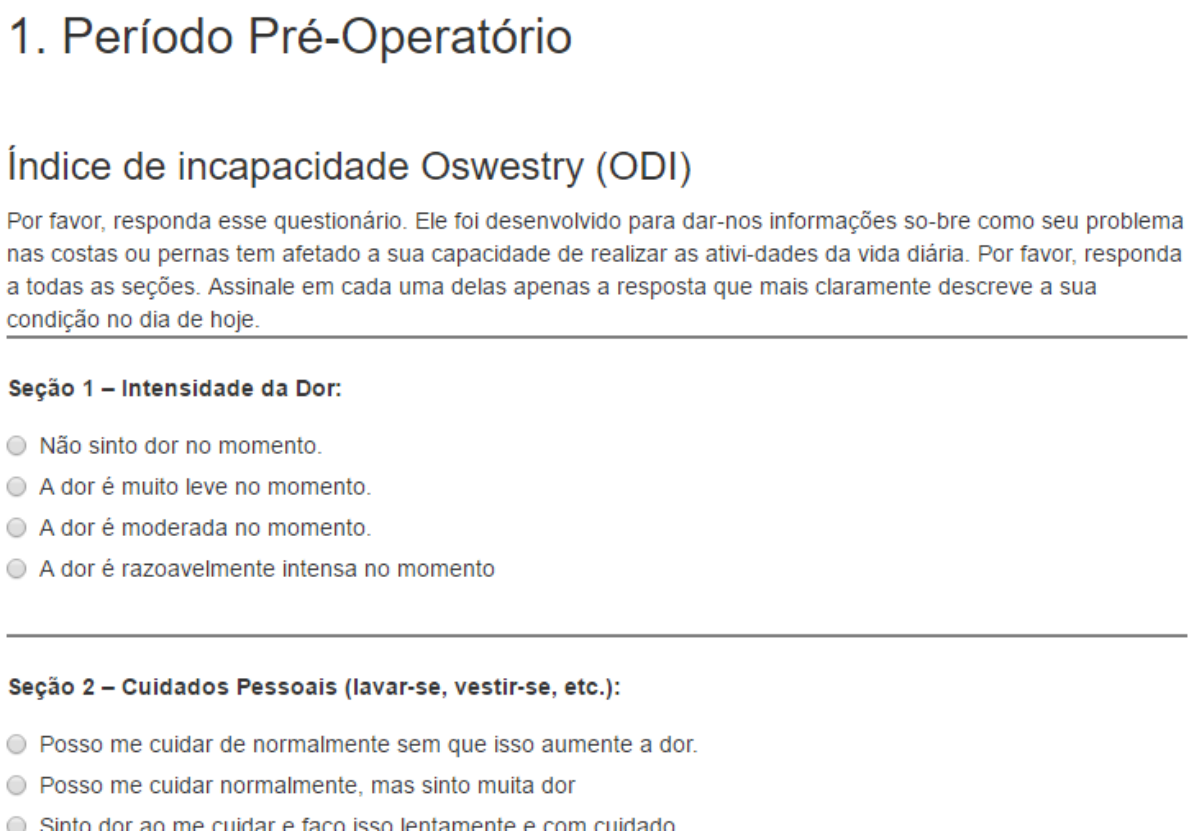


Figura 16 – Tela de aplicação dos questionários visão do paciente

Fonte: Elaborado pelo autor

VISUALIZAR QUESTIONÁRIOS A SEREM RESPONDIDOS **Ator: Usuário**
CURSO BÁSICO: O sistema deve exibir em uma agenda, todos os questionários a serem respondidos, indiferentemente da responsabilidade (paciente ou outros). O sistema deve considerar o prazo de aplicação dos questionários e informar no compromisso qual é o paciente e o questionário a ser respondido. Os questionários agendados que perderam o prazo ou foram respondidos não devem ser visualizados nesta agenda.
PRECONDIÇÕES: Usuário autenticado na aplicação. Usuário com permissão para acessar a funcionalidade. Selecionado estudo clínico.

PÓS CONDIÇÕES:
Questionários pendentes visualizados na agenda, informando o paciente e detalhes do agendamento.

Quadro 7 – Caso de uso: Visualizar questionários a serem respondidos

Fonte: Elaborado pelo autor

Figura 17 – Agenda de visualização dos questionários a serem respondidos

Fonte: Elaborado pelo autor

O Quadro 7 representa o caso de uso de visualização dos questionários a serem respondidos, sendo ilustrado na Figura 17 a tela que organiza os agendamentos.

No Quadro 8 detalha-se o caso de uso de cadastro de momentos de aplicação de um item do plano de aplicação do estudo.

CADASTRAR MOMENTOS DE APLICAÇÃO **Ator: Usuário**
CURSO BÁSICO: O usuário preenche o nome do momento de aplicação de um questionário, nos três idiomas.
PRECONDIÇÕES: Usuário autenticado na aplicação. Usuário com permissão para acessar a funcionalidade.

PÓS CONDIÇÕES:
Momento de aplicação cadastrado no sistema.

Quadro 8 – Caso de uso: Cadastrar Momentos de Aplicação

Fonte: Elaborado pelo autor

É possível visualizar na Figura 18 a tela de cadastro e na Figura 19 a tela de listagem.

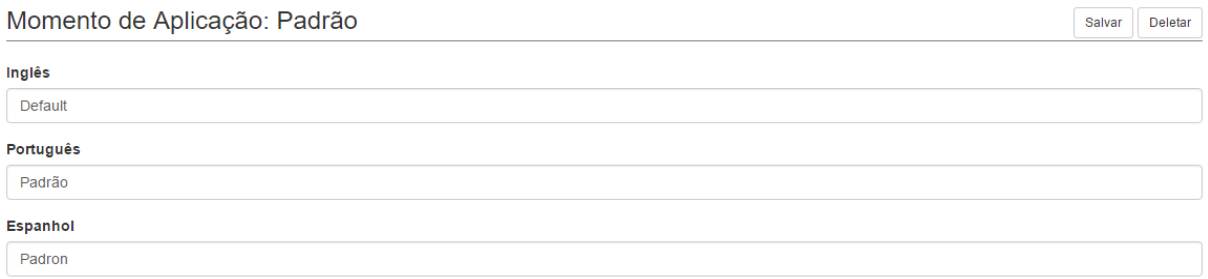


Figura 18 – Cadastro de momentos de aplicação

Fonte: Elaborado pelo autor

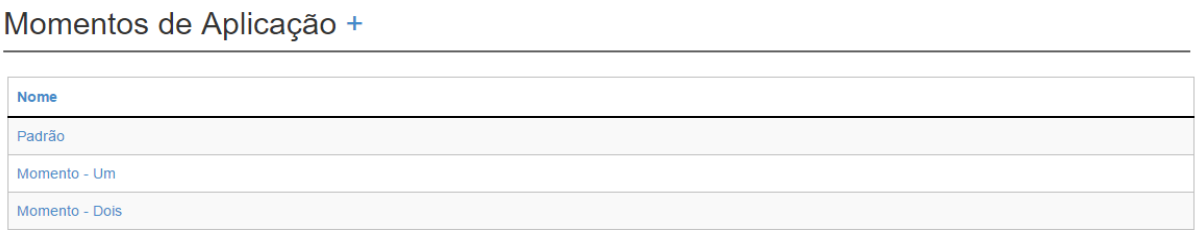


Figura 19 – Listagem dos momentos de aplicação

Fonte: Elaborado pelo autor

No Quadro 9 é detalhado o caso de uso referente ao controle dos agendamentos programados para um determinado paciente.

CONTROLAR QUESTIONÁRIOS A SEREM RESPONDIDOS **Ator: Sistema**
CURSO BÁSICO: O controle de agendamentos, que são os questionários a serem respondidos pelos médicos e pacientes, deve encontrar os que entraram em prazo de aceitar as respostas e enviar um e-mail avisando os responsáveis pelo seu preenchimento. No caso do paciente que responde por si próprio deve ser enviado um e-mail ao mesmo. Os agendamentos dos médicos devem ser enviados ao coordenador e investigador do centro de pesquisa cujo qual o paciente está associado.
CURSO ALTERNATIVO: **Agendamento próximo do prazo final:** Caso um agendamento não seja respondido e faltem três dias para o prazo final o sistema envia um e-mail informando que o prazo pode ser perdido. Os destinatários deste e-mail são o coordenador e investigador principais do centro de pesquisa que coordenam o estudo, além dos que interagem com o próprio paciente. Deve ser alterada a cor exibida para o agendamento de verde para amarela, sinalizando que o mesmo se encontra em prazo quase que limite para aceitar as respostas (esta cor é utilizada na agenda que exibe os questionários a serem respondidos) **Prazo perdido:** O controle de agendamentos deve alterar a situação do agendamento para “Perda de prazo” de modo que não seja aceite mais respostas.
PRECONDIÇÕES: Tarefa agendada para disparar o controle de agendamentos.
PÓS CONDIÇÕES: E-mails enviados a quem se destina responder o determinado questionário.

Quadro 9 – Caso de uso: Controlar questionários a serem respondidos

Fonte: Elaborado pelo autor

Na Figura 20 é ilustrado um exemplo de e-mail recebido pelo paciente para que responda um determinado questionário.

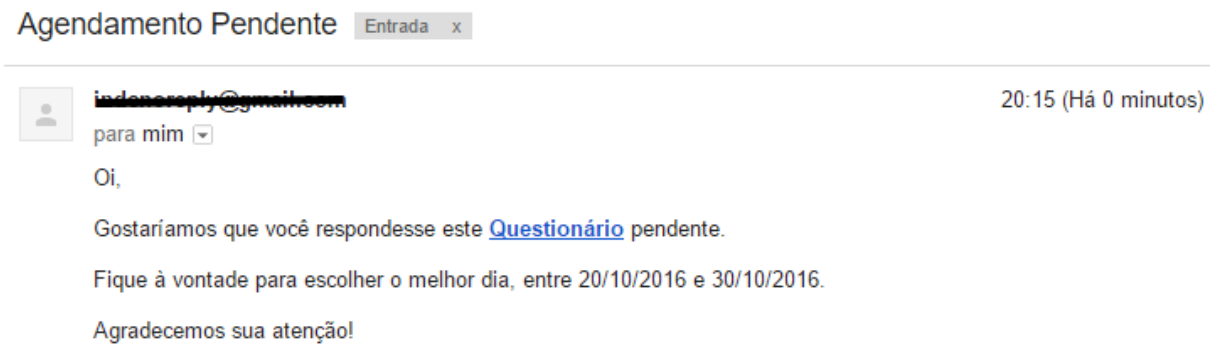


Figura 20 – Exemplo de e-mail para o paciente

Fonte: Elaborado pelo autor

O conteúdo do e-mail é traduzido de acordo com o idioma do paciente. A Figura 21 e a Figura 22 ilustram um exemplo de e-mail recebido pelos coordenadores e investigadores.

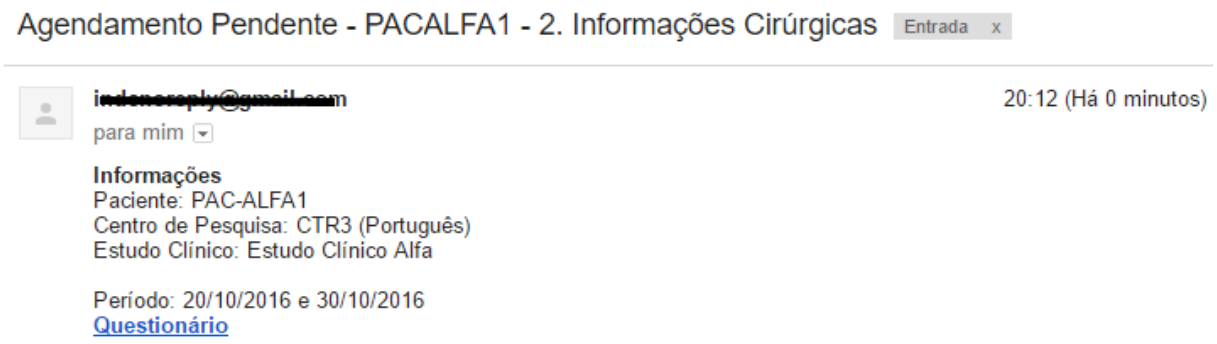


Figura 21 – Exemplo de e-mail para o médico

Fonte: Elaborado pelo autor

O conteúdo do e-mail é mais objetivo pois poderão haver diversos agendamentos pendentes para os coordenadores e investigadores responsáveis responderem. A Figura 22 ilustra o exemplo de e-mail recebido quando um agendamento está próximo da data final do prazo.

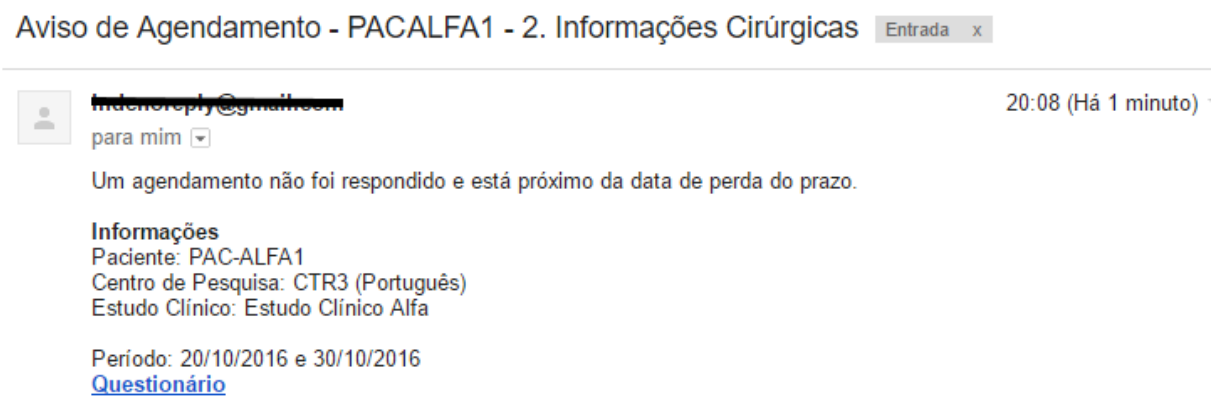


Figura 22 – Exemplo de e-mail de aviso para os médicos

Fonte: Elaborado pelo autor

3 DIAGRAMAS DE ROBUSTEZ

Armazenam-se nesta seção os diagramas de robustez dos casos de uso visualizados na seção 4.2.2 Acompanhamento de Indicadores Clínicos.

A Figura 17 representa o diagrama de robustez do caso de uso para cadastrar centros de pesquisa.

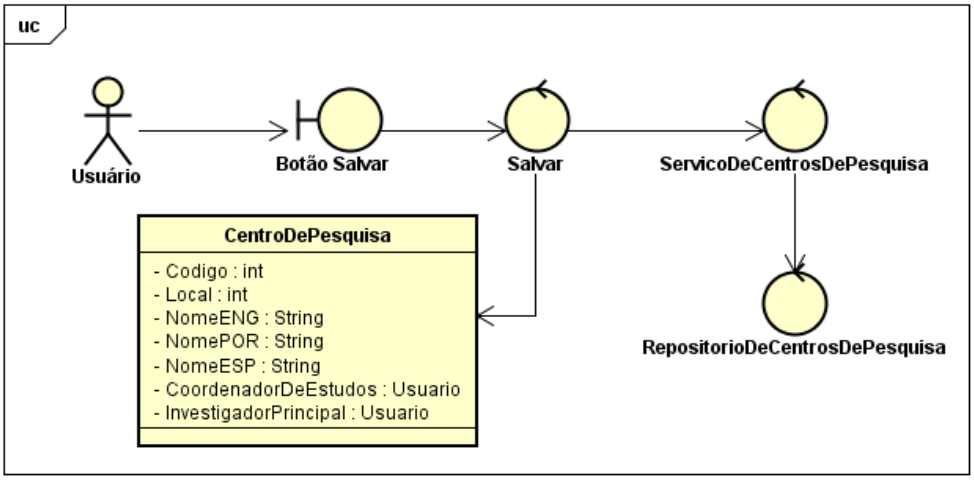


Figura 17 – Diagrama de robustez de centro de pesquisa

Fonte: Elaborado pelo autor

A Figura 18 representa o diagrama de robustez do caso de uso de gerenciar estudos clínicos.

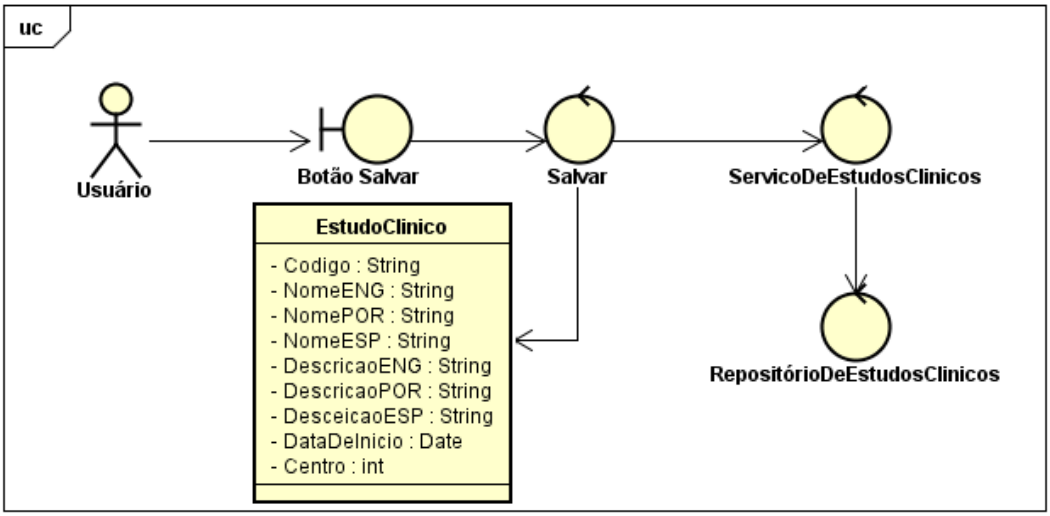


Figura 18 – Diagrama de robustez de gerenciamento de estudos clínicos

Fonte: Elaborado pelo autor

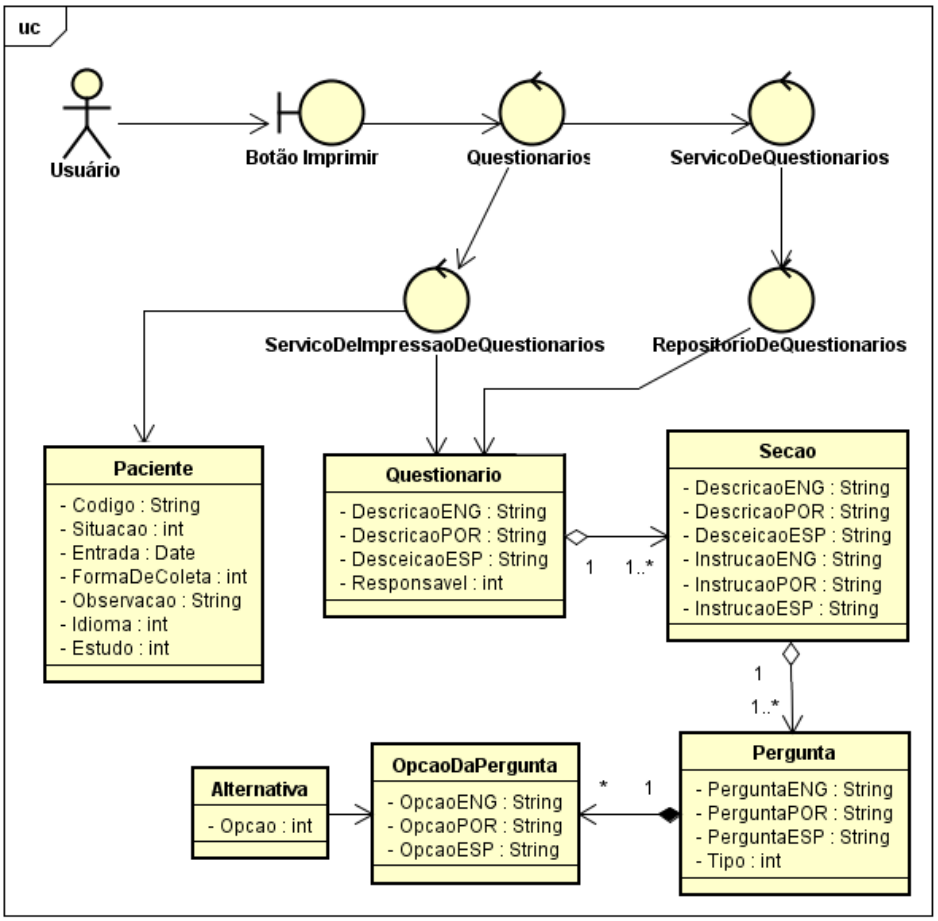


Figura 19 – Diagrama de robustez de impressão de questionários

Fonte: Elaborado pelo autor

A Figura 19 representa o diagrama de robustez do caso de imprimir questionários. A Figura 20 representa o diagrama de robustez do caso de uso de manipular questionários.

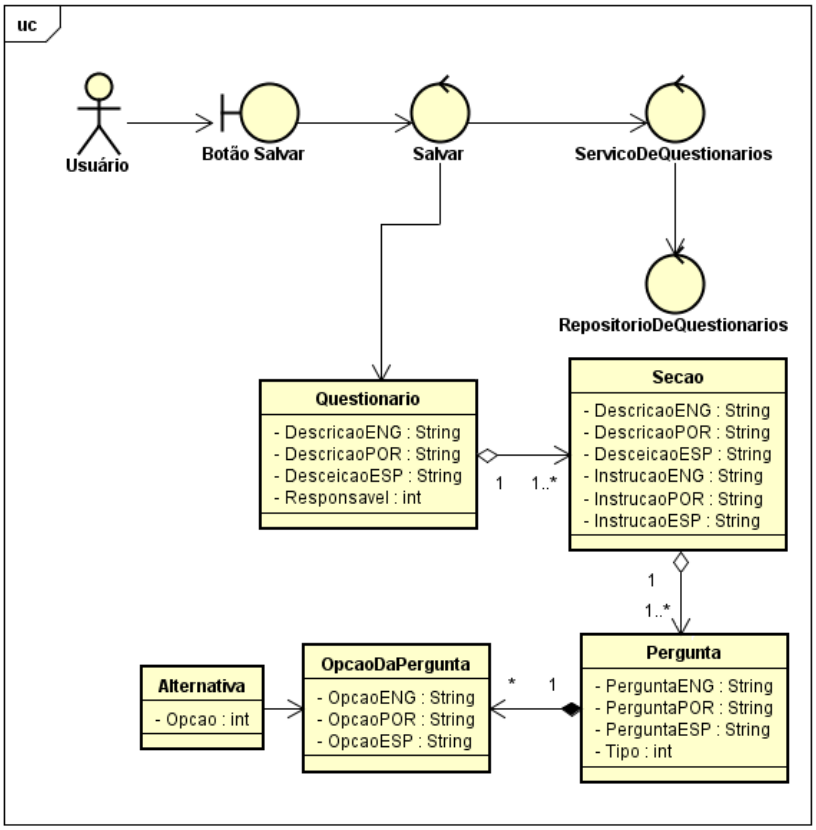


Figura 20 – Diagrama de robustez de manipulação de questionários

Fonte: Elaborado pelo autor

A Figura 21 representa o diagrama de robustez do caso de uso de cadastrar pacientes.

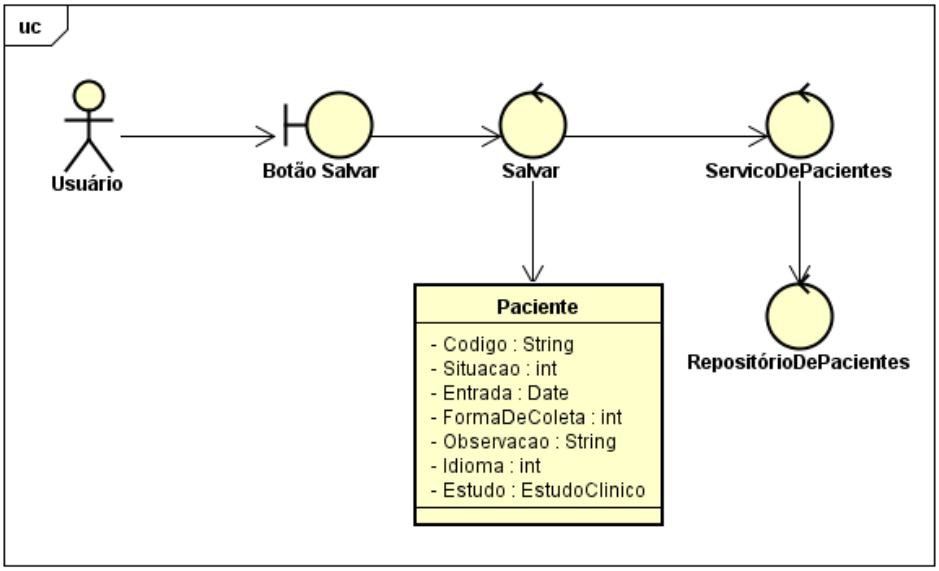


Figura 21 – Diagrama de robustez de pacientes

Fonte: Elaborado pelo autor

A Figura 22 representa o diagrama de robustez do caso de uso para visualizar questionários a serem respondidos.

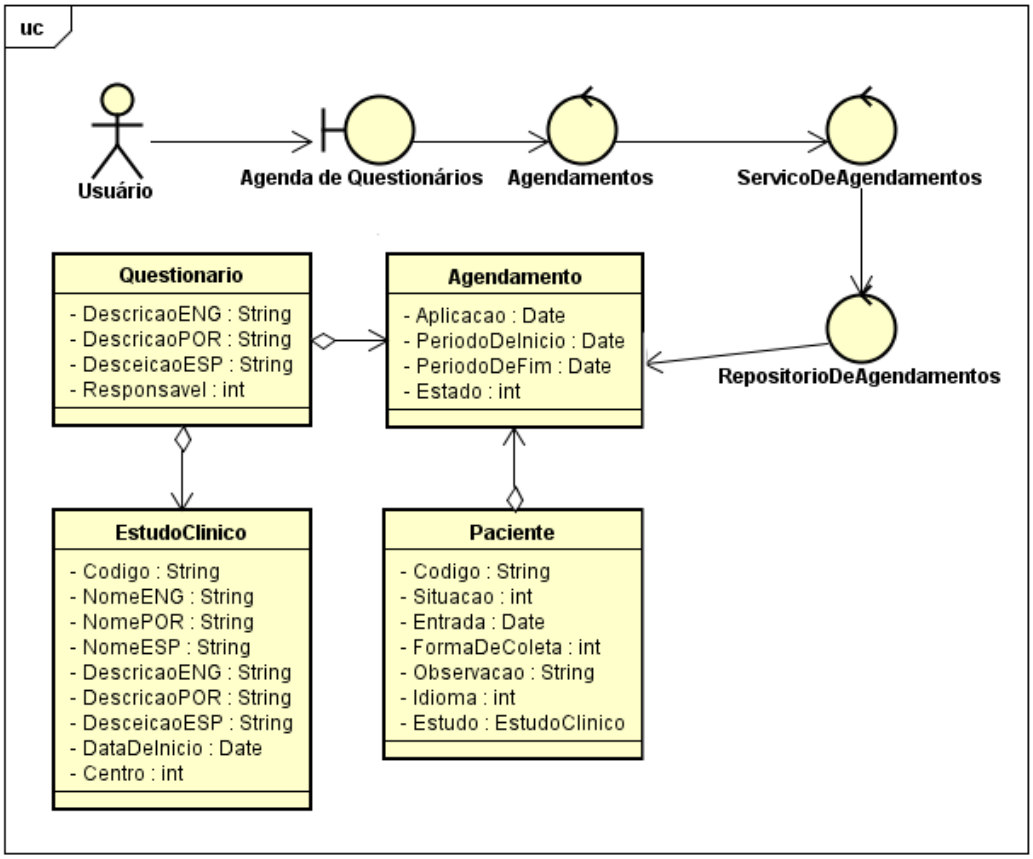


Figura 22 – Diagrama de robustez de questionários pendentes

Fonte: Elaborado pelo autor

A Figura 23 representa o diagrama de robustez do caso de uso de responder questionários.

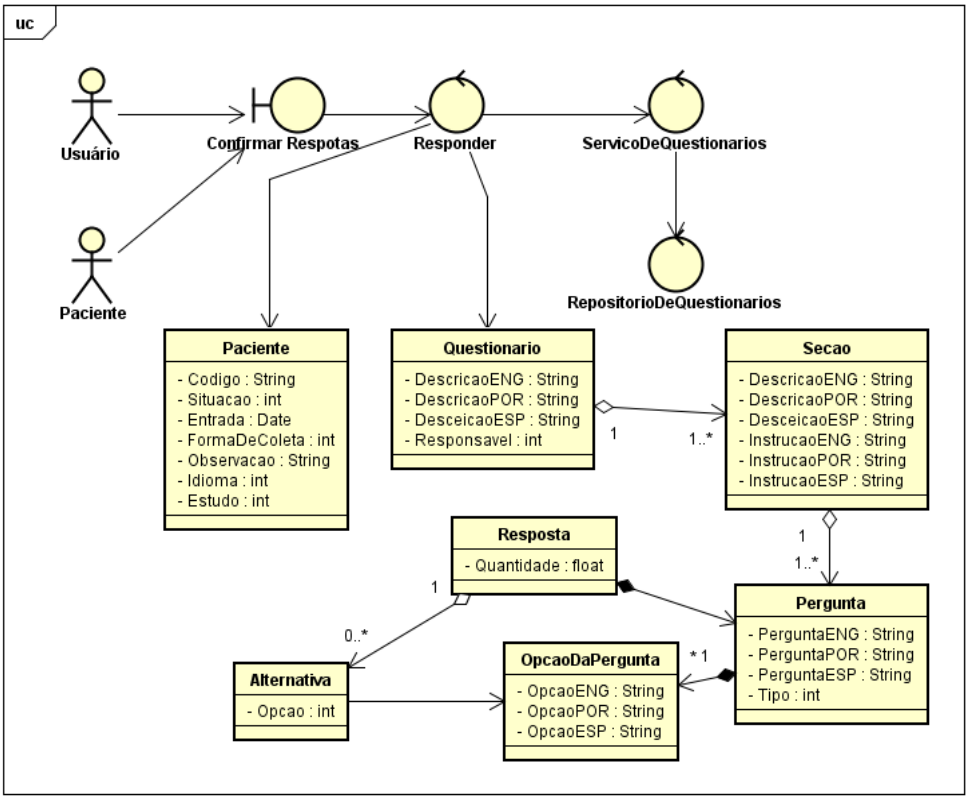


Figura 23 – Diagrama de robustez de responder questionários

Fonte: Elaborado pelo autor

A Figura 24 representa o diagrama de robustez do caso de uso para selecionar estudo clínico para realizar operações no sistema.

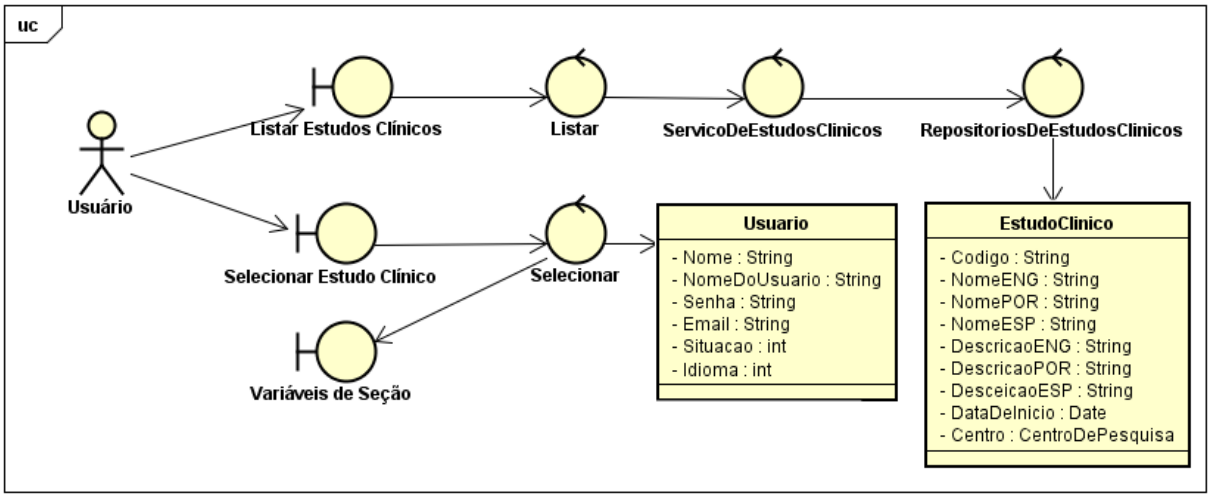


Figura 24 – Diagrama de robustez de selecionar estudo clínico

Fonte: Elaborado pelo autor

APÊNDICE D – GUIA DE DESENVOLVIMENTO

O conteúdo deste apêndice reúne informações sobre o desenvolvimento de aplicações utilizando a arquitetura de software proposta neste trabalho. Este guia espera certo grau de familiaridade com a ferramenta Visual Studio, para o entendimento completo das atividades a serem realizadas.

1 ESTRUTURA DE TRABALHO

As atividades que devem ser realizadas no início do desenvolvimento são representadas na Figura 1, sendo o resultado a estrutura de trabalho. A utilização da globalização e de mecanismos de segurança serão detalhados separadamente neste apêndice por serem utilizados sob a necessidade de cada projeto.

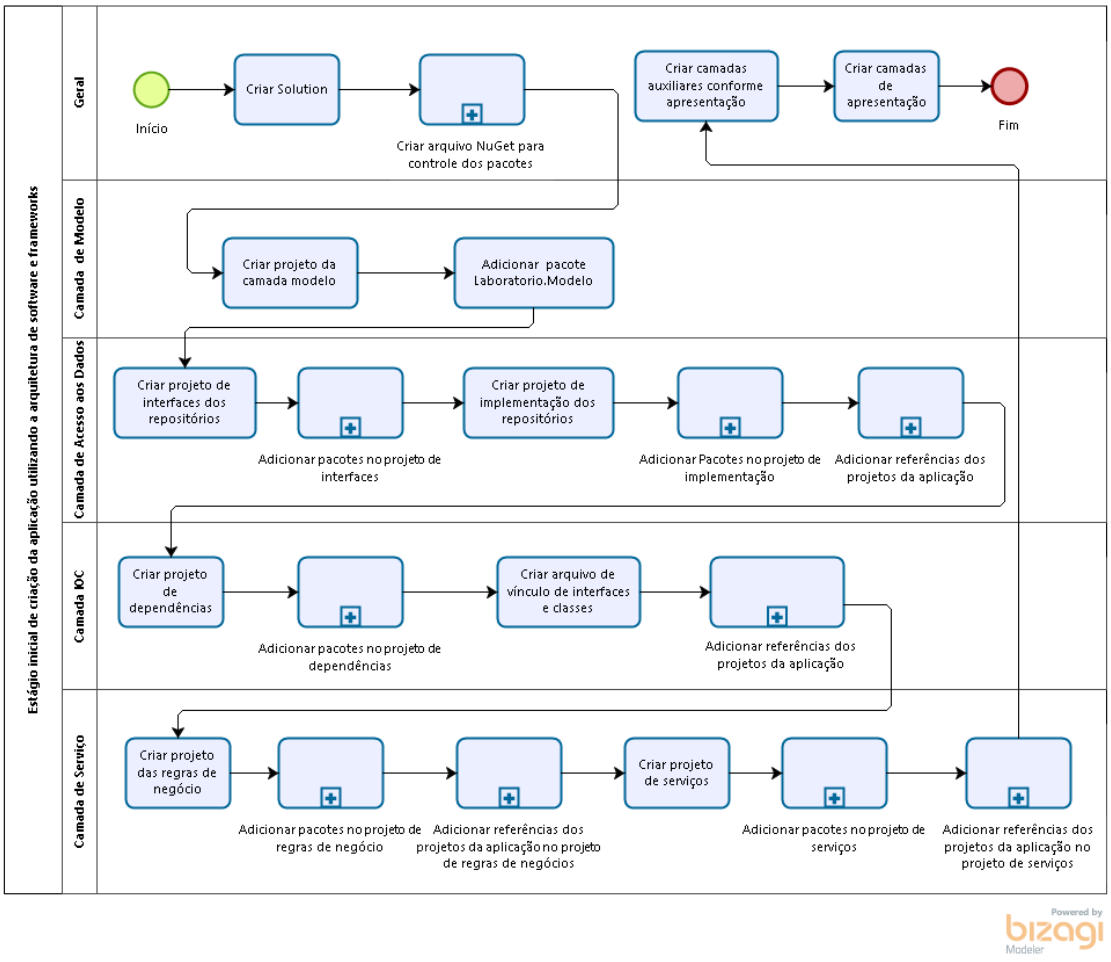


Figura 1 – Atividades para criação da estrutura de trabalho

Fonte: Elaborado pelo autor

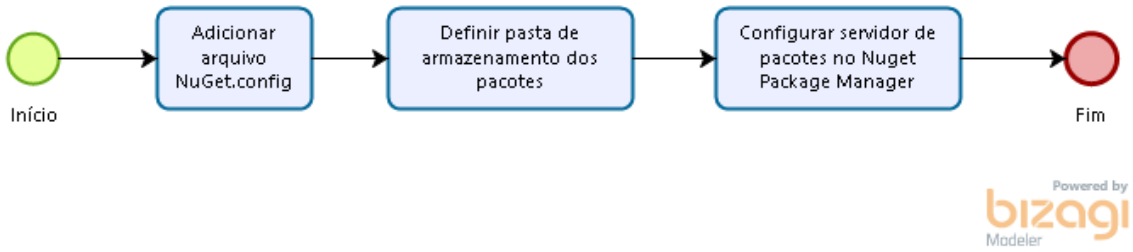


Figura 2 – Sub processo: Criar arquivo NuGet para controle dos pacotes

Fonte: Elaborado pelo autor

A Figura 2 representa o sub processo de criação do arquivo NuGet para controle dos pacotes. Através desta atividade é possível concentrar todas as bibliotecas instaladas em um único local, evitando a duplicação dos arquivos e otimizando futuras manutenções.

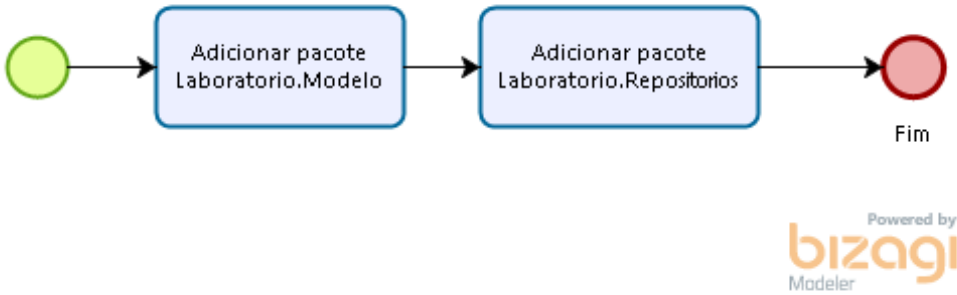


Figura 3 – Sub processo: Adicionar pacotes no projeto de interfaces

Fonte: Elaborado pelo autor

A Figura 3 ilustra os pacotes que devem ser adicionados no projeto de interfaces de repositório, localizado na camada de acesso aos dados. Cada aplicação possui seu próprio projeto de interfaces de repositórios, e devem estar separadas das classes concretas para permitir o baixo acoplamento da tecnologia de acesso aos dados com a aplicação.

Ainda na camada de acesso aos dados é necessário realizar no projeto com as classes concretas as atividades exibidas na Figura 4. Neste projeto da aplicação são implementadas as classes definidas no projeto de interfaces, sendo possível realizar o vínculo com a tecnologia de acesso aos dados neste projeto.

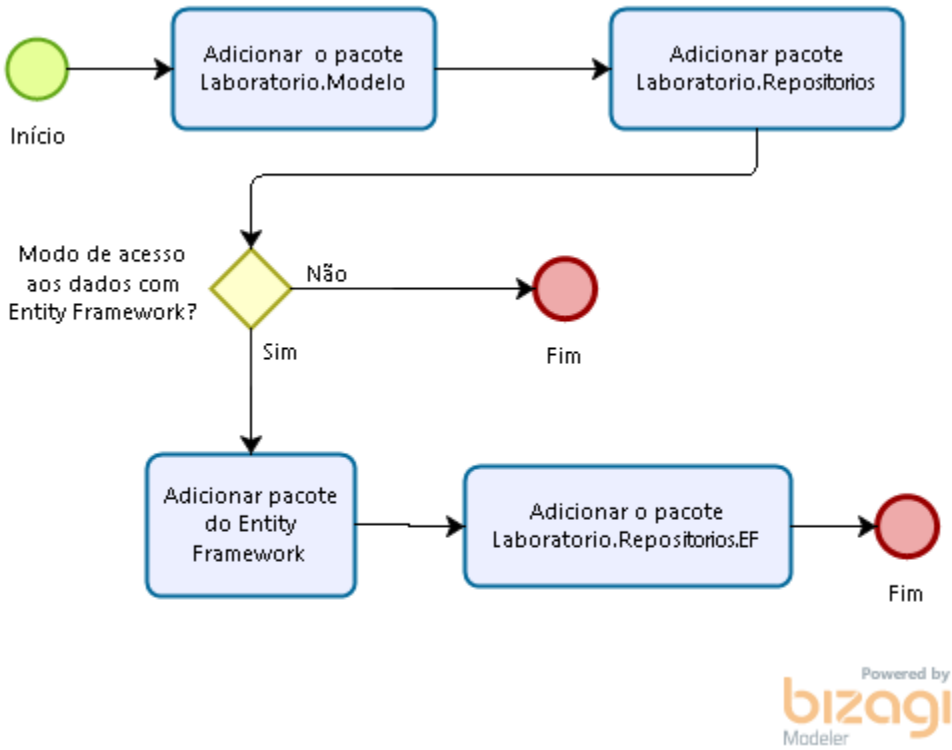


Figura 4 – Sub processo: Adicionar Pacotes no projeto de implementação

Fonte: Elaborado pelo autor

Além dos pacotes é necessário vincular referências dos próprios projetos da aplicação aos projetos de acesso aos dados, estando estas atividades representadas na Figura 5.

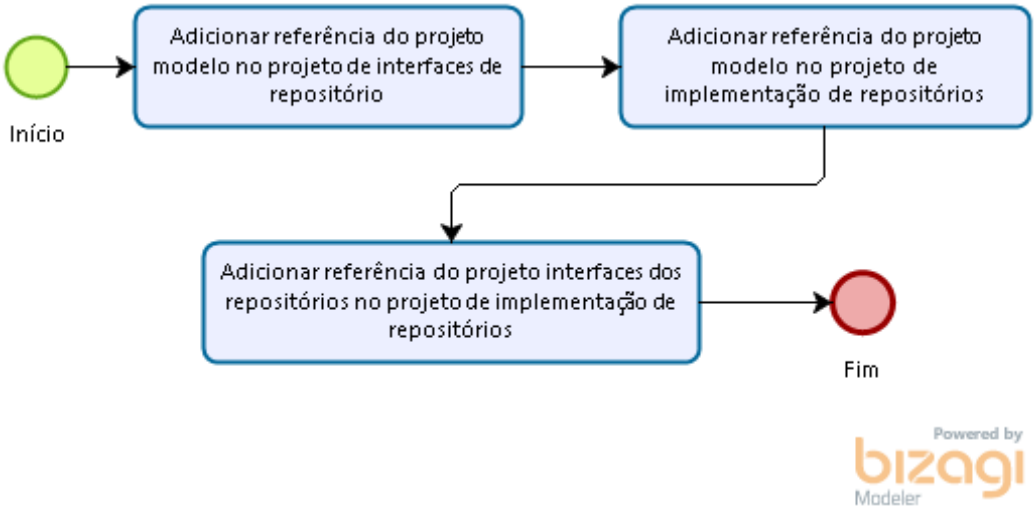


Figura 5 – Sub processo: Adicionar referências dos projetos da aplicação

Fonte: Elaborado pelo autor

O projeto da camada de injeção de dependência auxilia na tarefa de realizar os vínculos entre interfaces e classes, estando as atividades de adicionar pacotes necessários para o funcionamento utilizado o projeto representada na Figura 6.

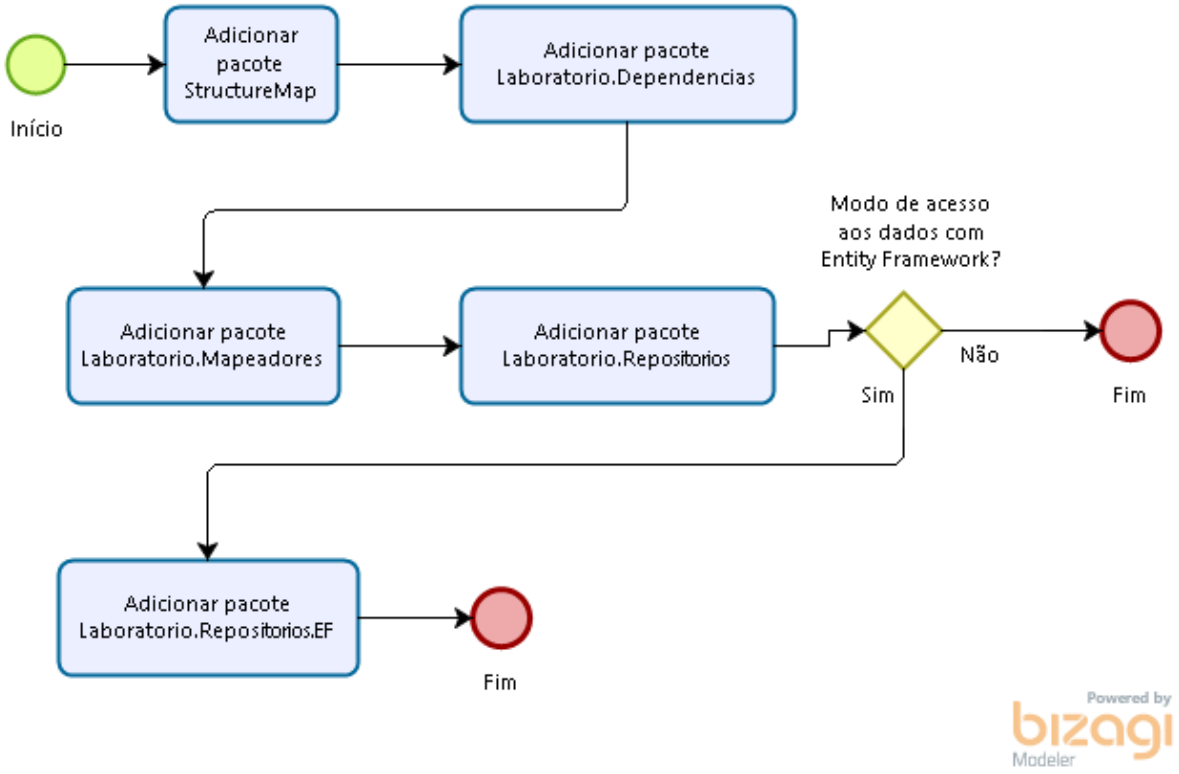


Figura 6 – Sub processo: Adicionar pacotes no projeto de dependências

Fonte: Elaborado pelo autor

Após a inclusão dos pacotes é necessário adicionar as referências dos próprios projetos da aplicação, atividade representada pela Figura 7.

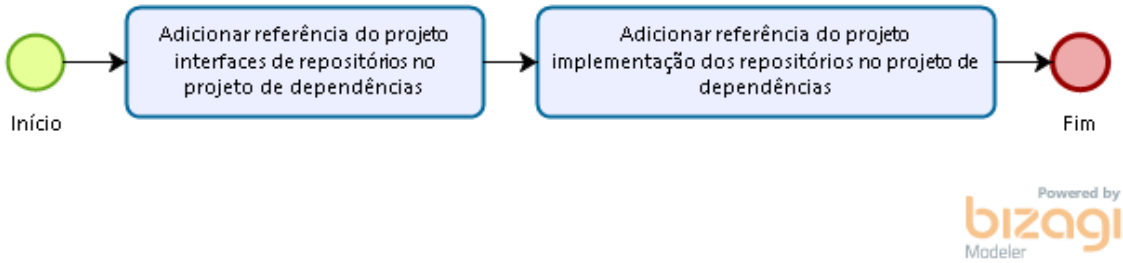


Figura 7 – Sub processo: Adicionar referências dos projetos da aplicação

Fonte: Elaborado pelo autor

Após a configuração do projeto de dependências e seguindo as atividades apresentado na Figura 1 se inicia o trabalho na camada de serviço, onde são adicionados os pacotes ao projeto de regras de negócio, como é representado na Figura 8.

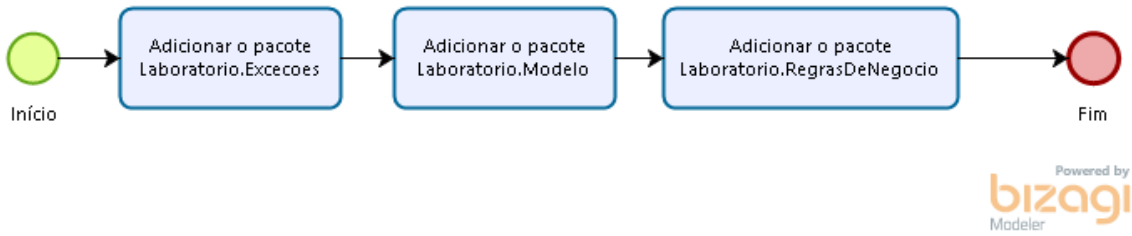


Figura 8 – Sub processo: Adicionar pacotes no projeto de regras de negócio

Fonte: Elaborado pelo autor

Posteriormente a adição dos pacotes é necessário adicionar as referências dos próprios projetos da aplicação, como é exibido na Figura 9.

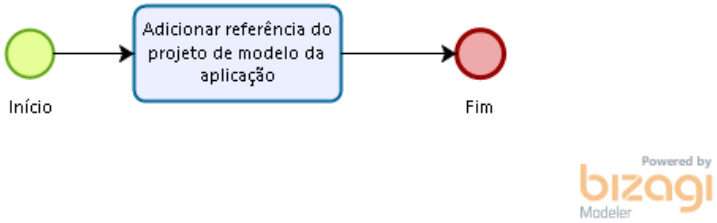


Figura 9 – Sub processo: Adicionar referências dos projetos nas regras de negócios

Fonte: Elaborado pelo autor

Realizadas as atividades no projeto de regras de negócio é necessário configurar o projeto de serviços.

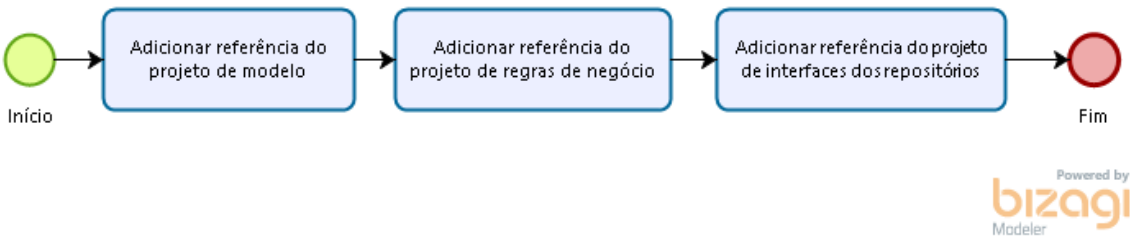


Figura 10 – Sub processo: Adicionar referências dos projetos da aplicação no projeto de serviços

Fonte: Elaborado pelo autor

As atividades visualizadas na Figura 10 correspondem a adição das referências dos projetos da aplicação ao projeto de serviços, também da própria aplicação.

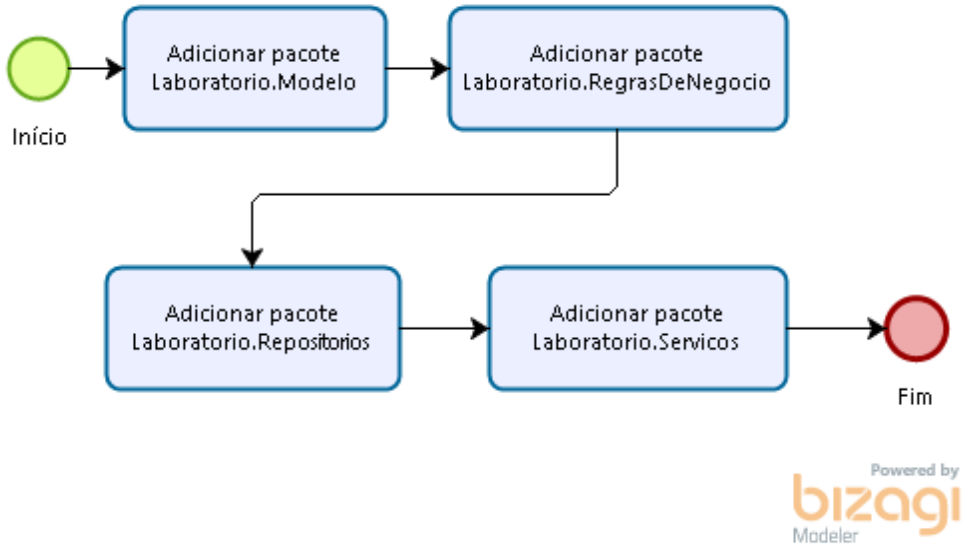


Figura 11 – Sub processo: Adicionar pacotes no projeto de serviços

Fonte: Elaborado pelo autor

A Figura 11 detalha os pacotes que devem ser adicionados ao projeto de serviços da aplicação. Ao concluir as atividades exibidas na Figura 1 se obtém como resultado a estrutura mínima suficiente para o desenvolvimento de *software,* possibilitando preservar as regras de negócio das aplicações e permitir a intercambialidade das camadas de apresentação e de acesso aos dados, tornando a aplicação flexível e protegendo o patrimônio das aplicações (suas regras de negócio).

2 ESTRUTURA DE TRABALHO: GLOBALIZAÇÃO

A camada de globalização tem sua utilização opcional delegando a decisão de utilização conforme a necessidade de cada aplicação. O processo geral de criação da estrutura é visualizado na Figura 12.

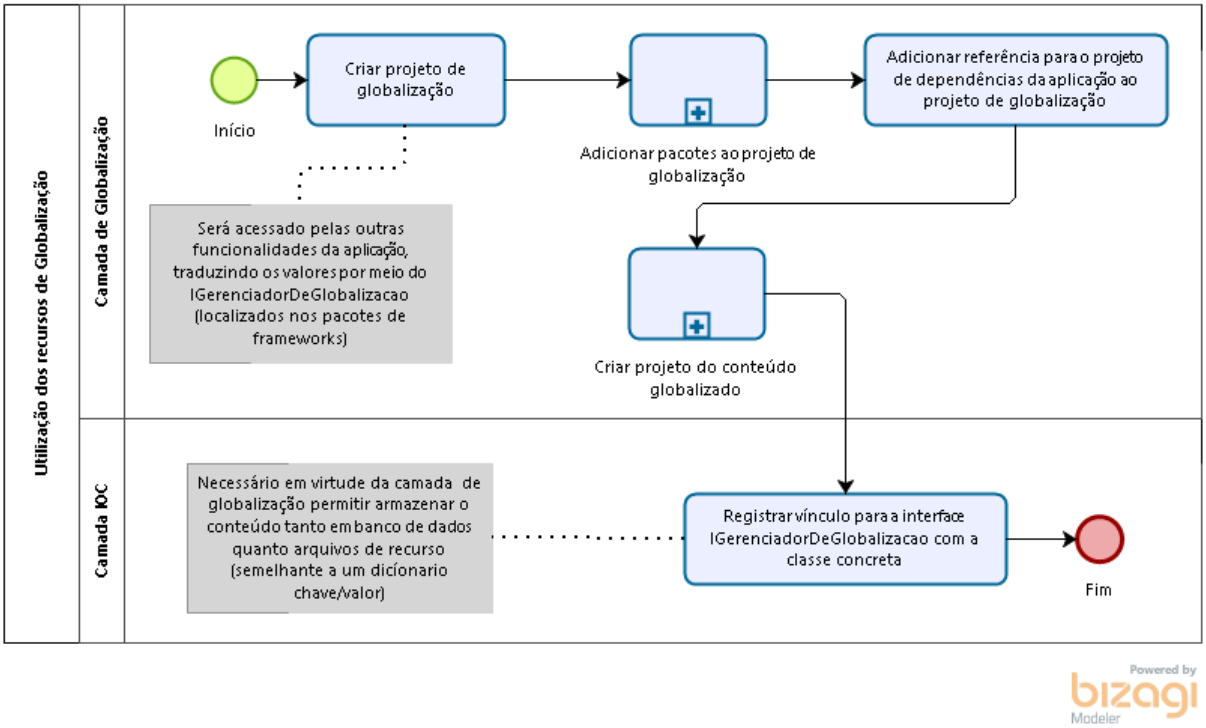


Figura 12 – Processo de criação da estrutura de globalização

Fonte: Elaborado pelo autor

A estrutura é gerida por dois projetos principais, sendo um a porta de entrada e o outro o conteúdo globalizado, foi utilizado esta estratégia para possibilitar que seja armazenado o conteúdo tanto em banco de dados quanto em arquivos de recursos.

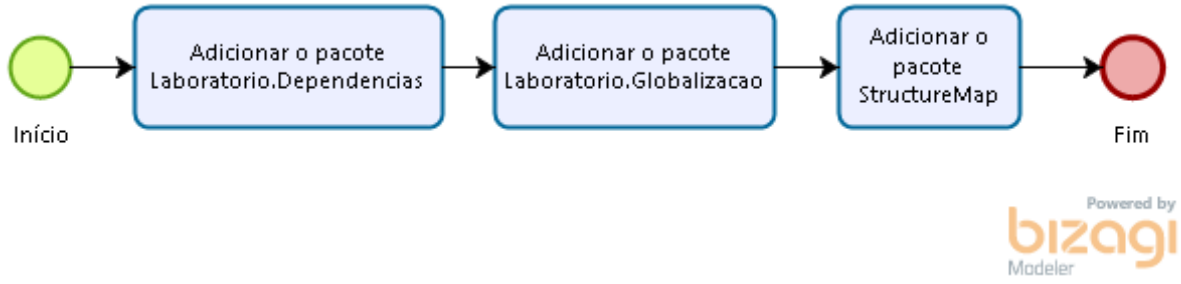


Figura 13 – Sub processo: Adicionar pacotes ao projeto de globalização

Fonte: Elaborado pelo autor

As atividades de inclusão dos pacotes e de criação do projeto que armazena o conteúdo são visualizados na Figura 13 e na Figura 14.

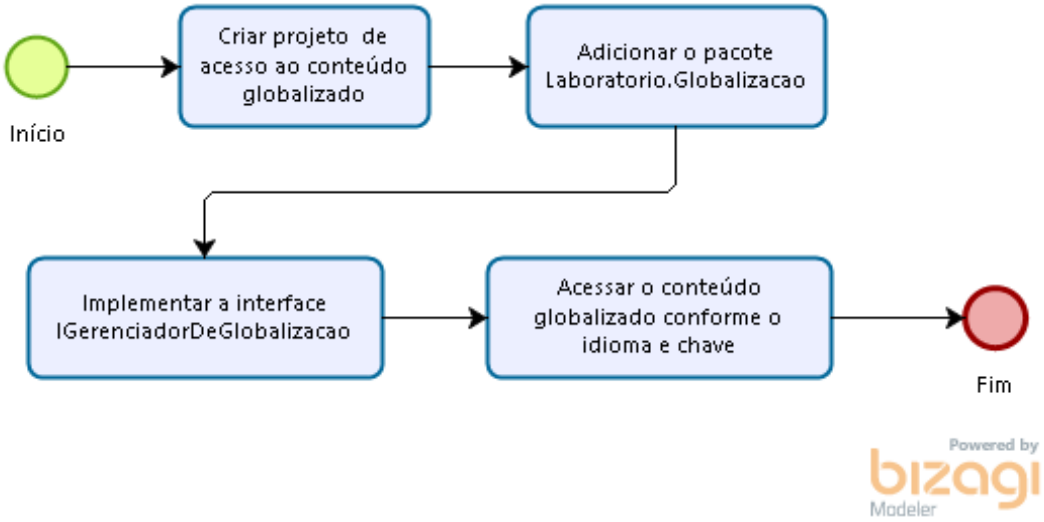


Figura 14 – Sub processo: Criar projeto do conteúdo globalizado

Fonte: Elaborado pelo autor

3 ESTRUTURA DE TRABALHO: SEGURANÇA

Ainda que possivelmente a grande maioria das aplicações possua controle de acesso a utilização dos recursos de segurança também é delegado as necessidades de cada uma das aplicações.

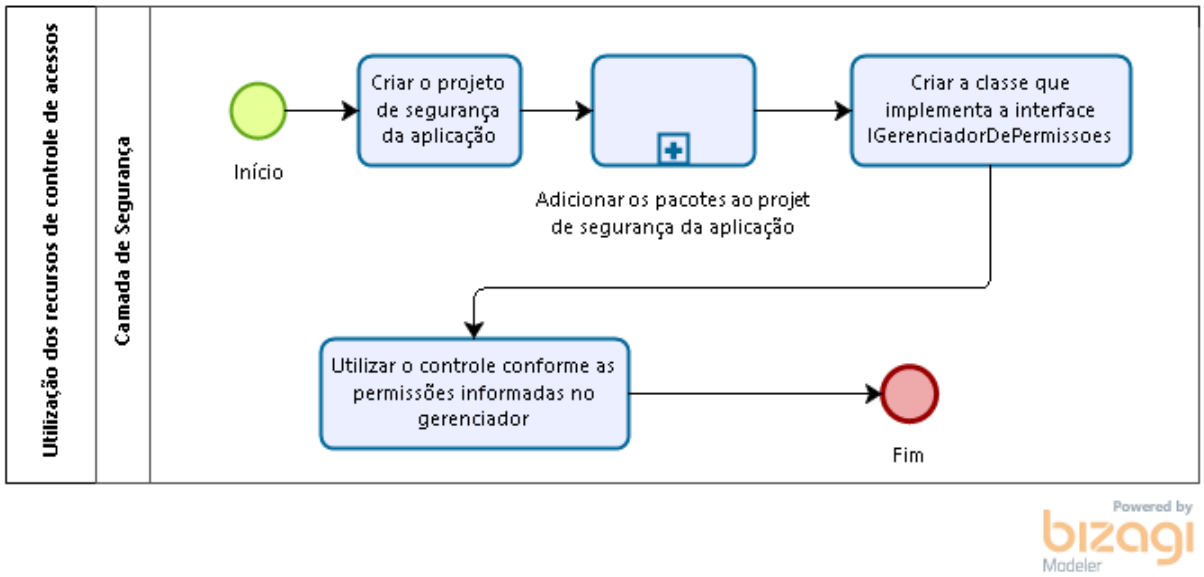


Figura 15 – Processo de criação da estrutura de segurança

Fonte: Elaborado pelo autor

A Figura 15 apresenta as atividades para a inclusão dos recursos de segurança na aplicação.

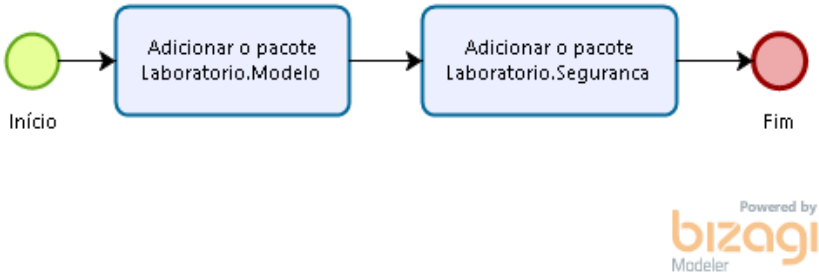


Figura 16 – Processo de criação da estrutura de segurança

Fonte: Elaborado pelo autor

A Figura 16 apresenta os detalhes do sub processo que adiciona os pacotes necessários para a utilização dos recursos já desenvolvidos.

4 IMPLEMENTAÇÃO DE FUNCIONALIDADE

O processo de incremento de funcionalidades nas aplicações que fazem uso da arquitetura e dos frameworks é visualizado na Figura 17, estando cada uma das atividades detalhadas em suas subseções, com exceção das atividades da camada de apresentação e sua auxiliar por serem específicas as tecnologias trabalhadas na apresentação.

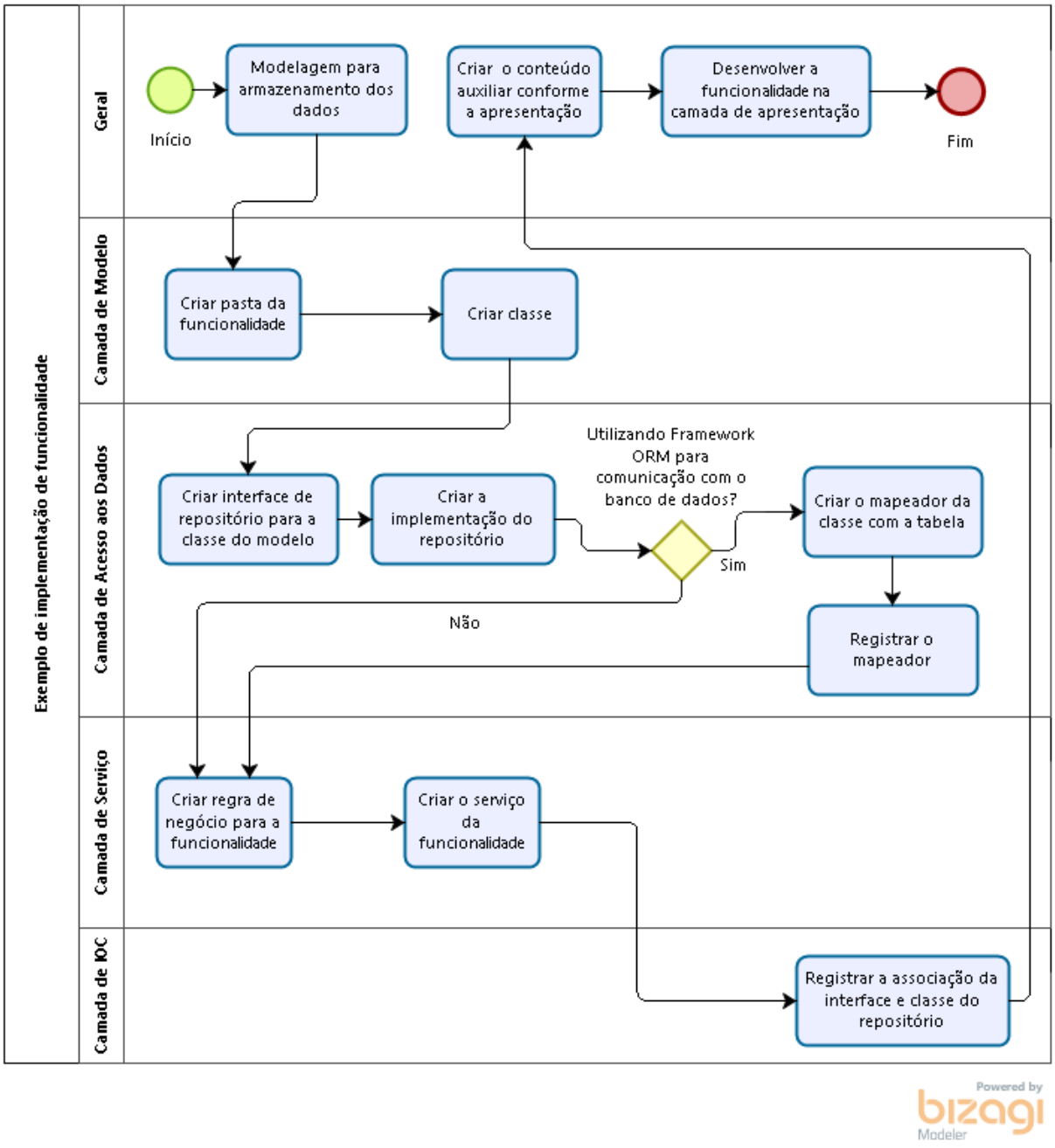


Figura 17 – Inclusão de funcionalidade na aplicação

Fonte: Elaborado pelo autor

4.1 Modelagem para armazenamento dos dados

Esta atividade tem por resultado final o meio em que serão armazenados os dados utilizados pela aplicação, sendo na grande maioria dos casos uma tabela banco de dados. Poderiam ser utilizados ainda arquivos serializados ou outros meios de armazenamento.

Considerando o exemplo de armazenar em um banco de dados o resultado final desta tarefa é a tabela no banco de dados, pronta para receber as informações, como pode ser observador na Figura 18.

Figura 18 – Visualização da estrutura de tabela de um cadastro

Fonte: Elaborado pelo autor

4.2 Atividades da camada de modelo

Após a criação da pasta e da classe esta deverá ser definida conforme os requisitos de negócio. Frequentemente esta classe é um reflexo da tabela do banco de dados, podendo conter outros atributos que importam ao negócio e que não estão na tabela. Um exemplo pode ser observado na Figura 19.

É importante observar na Figura 19 a ausência de atributos e tipos específicos de tecnologias terceiras, de modo que a classe utiliza apenas tipos primitivos da linguagem ou tipos complexos definidos na própria aplicação e seus frameworks (também de modelo que seguem as mesmas diretivas de funcionamento).

A estrutura da classe apresentada na Figura 19 é o reflexo da tabela no banco de dados.

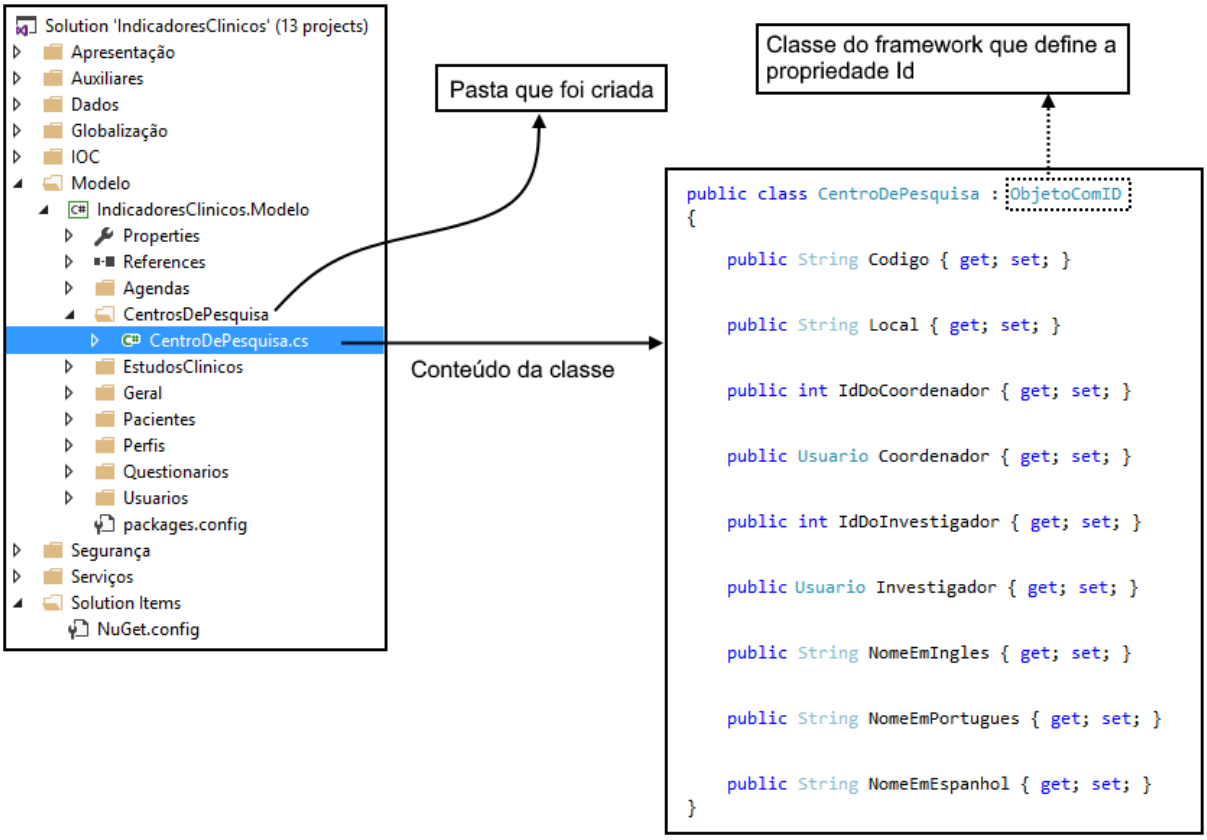


Figura 19 – Exemplo da classe de modelo criada

Fonte: Elaborado pelo autor

4.3 Criação da interface de repositório

A interface de repositório criada será utilizada pelo serviço para acessar os dados.

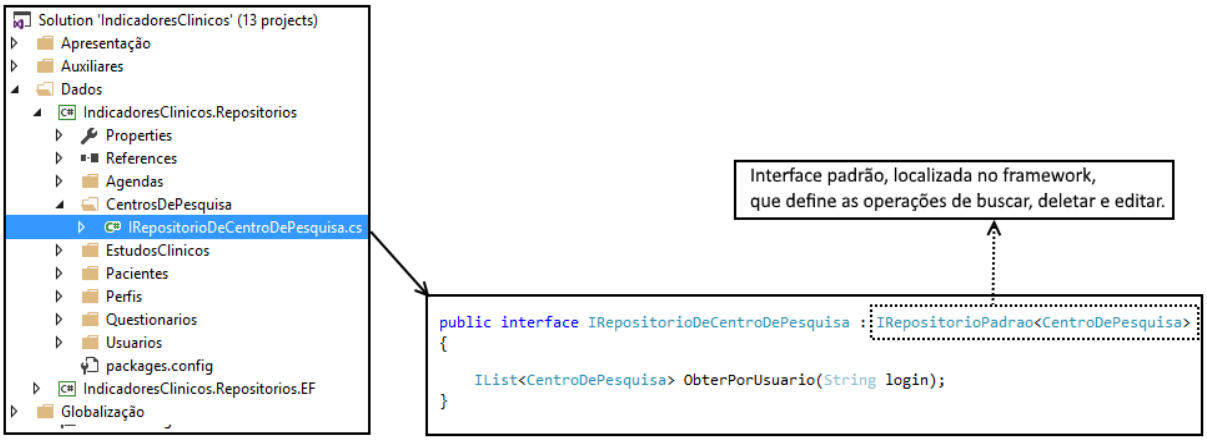


Figura 20 – Exemplo de interface de repositório

Fonte: Elaborado pelo autor

A interface funciona como um esqueleto que mantém apenas as definições dos métodos e funções, sendo implementadas posteriormente. A Figura 20 ilustra um exemplo de interface para repositório fazendo uso da interface padrão, que já define operações comuns como editar, incluir, excluir e buscar.

É importante que não seja vinculado a tecnologia de acesso aos dados na interface criada, de forma que ao trocar a tecnologia ou local de armazenar os dados a refatoração é a inclusão de uma nova camada não implicando em alterações em interfaces e serviços.

4.4 Implementação do repositório

Na criação da classe concreta de acessos aos dados é necessário implementar a interface que foi criada na atividade anterior. A Figura 21 ilustra um exemplo de implementação fazendo uso do Entity Framework para acessar o banco de dados, tecnologia onde não é necessário escrever comando SQL.

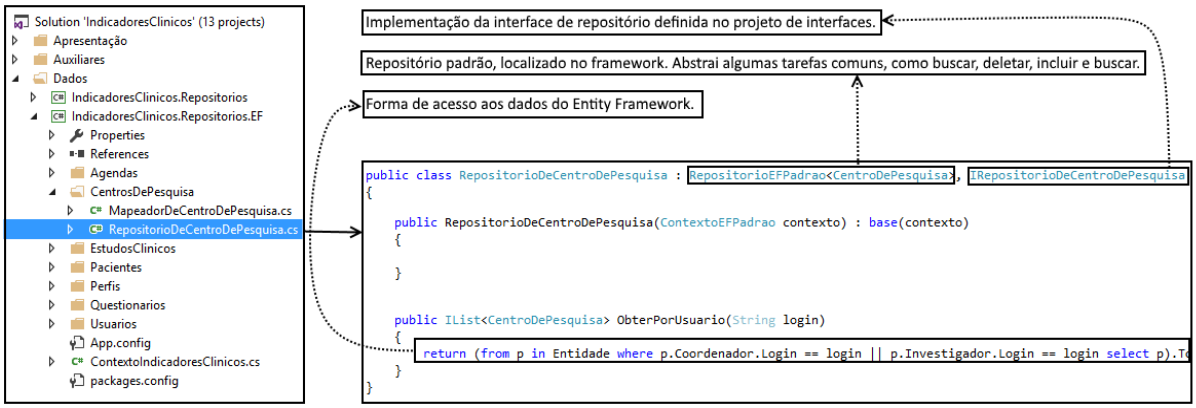


Figura 21 – Exemplo de implementação do repositório em Entity Framework

Fonte: Elaborado pelo autor

Na classe de implementação é que deve estar o vínculo com a tecnologia de acesso aos dados, como é observador na Figura 21. Não é visualizado a implementação de operações como buscar, deletar e incluir por terem sido abstraídas

no repositório padrão do Entity Framework, de forma que são reutilizados ao longo do desenvolvimento desta e de outras aplicações.

4.5 Mapeador da classe e tabela

O mapeamento entre uma classe e uma tabela de banco de dados é necessário somente quando existe a utilização de um framework ORM para acesso ao banco de dados, como por exemplo o Entity Framework ou NHibernate. Esta abordagem auxilia no objetivo de não poluir a classe modelo com a tecnologia de acesso ao banco, como atributos contendo nome de campos e da própria tabela.

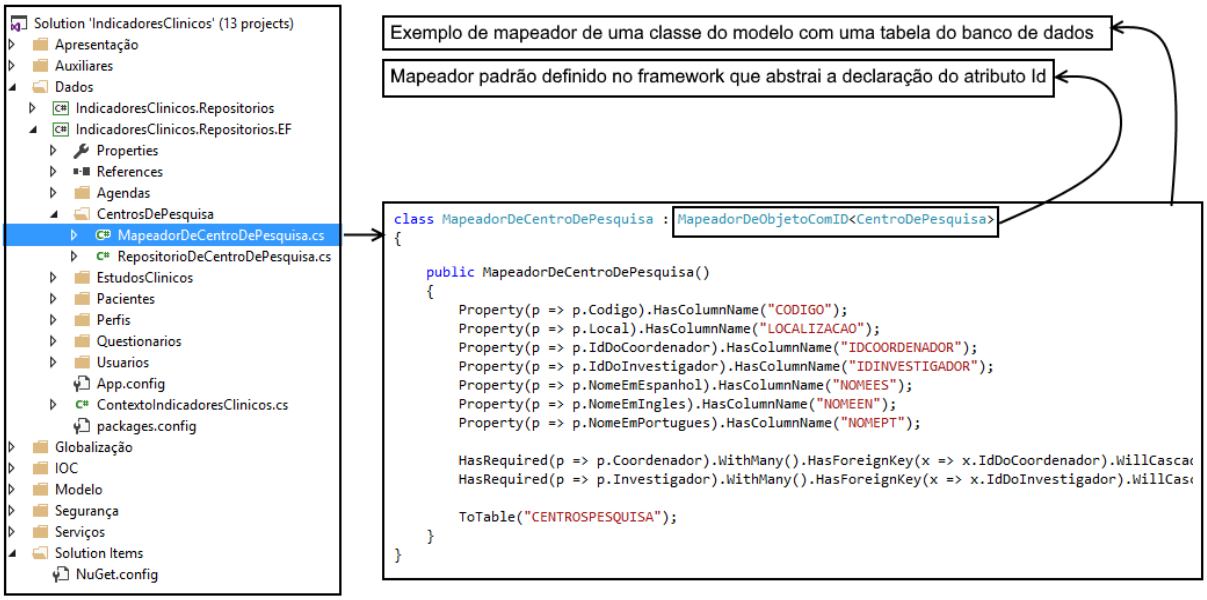


Figura 22 – Exemplo de implementação de um mapeador de classe e tabela

Fonte: Elaborado pelo autor

A Figura 22 ilustra um exemplo de mapeador, ao realizar o vínculo entre a classe definida no modelo com a estrutura da tabela no banco de dados. É delegado para esta classe de mapeador a associação com a tecnologia de acesso aos dados, tornando mais pura e reutilizável a classe do modelo.

4.6 Criação da regra de negócio

A regra de negócio da aplicação é o coração do software, onde existe margem para o maior valor agregado da aplicação. A Figura 23 exemplifica a criação e implementação da regra de negócio para uma determinada classe do modelo.

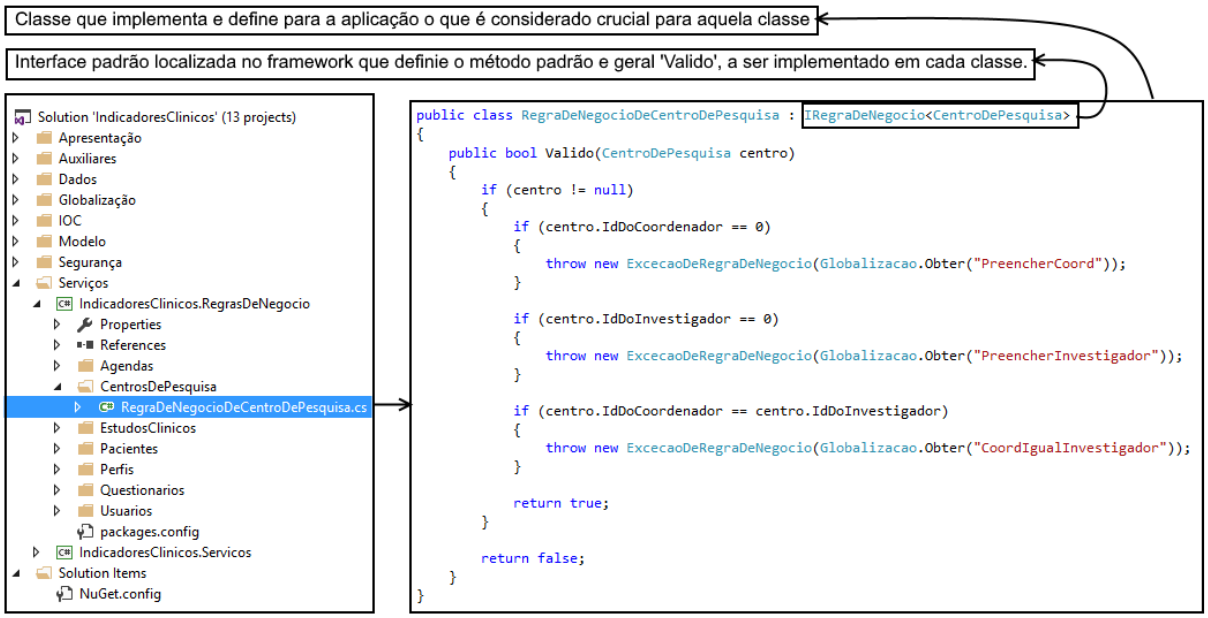


Figura 23 – Exemplo de implementação da classe de regra de negócio

Fonte: Elaborado pelo autor

Apesar da implementação da interface padrão, que é visualizado no exemplo da Figura 23, é importante observar que a classe poderá conter quantos métodos de validação forem necessários para proteger as regras de negócio, sendo o 'Valido' o mais comum, entretanto poderia uma determinada classe do modelo realizar um processamento de regra de negócio ao excluir uma informação, existindo assim um outro método para controle dos mesmos, que deveria ser utilizado no serviço.

A Figura 24 ilustra um outro exemplo de processamento de regras de negócio, sendo que para aquela aplicação não foi utilizado a globalização e o conteúdo da mensagem está transcrito diretamente na exceção.

```
public class RegraDeNegocioDePessoa : IRegraDeNegocio<Pessoa>
{
    public bool Valido(Pessoa pessoa)
    {
        if (pessoa != null && pessoa.Telefones != null)
        {
            if (pessoa.Telefones.Count == 0)
            {
                throw new ExcecaoDeRegraDeNegocio("É necessário que a pessoa tenha ao menos um telefone para contato.");
            }

            if (pessoa.Telefones.FirstOrDefault(x => x.Tipo == EnumeradorDeTipoDeTelefone.Celular) == null)
            {
                throw new ExcecaoDeRegraDeNegocio("Ao menos um dos telefones informados deve ser o Celular.");
            }
```

Figura 24 – Exemplo de implementação da classe de regra de negócio

Fonte: Elaborado pelo autor

4.7 Implementação do serviço

O serviço criado para cada funcionalidade também faz uso de um framework desenvolvido a partir da abstração das operações, tornando mais prático e limpo o código. A Figura 25 exibe um exemplo de implementação de serviço.

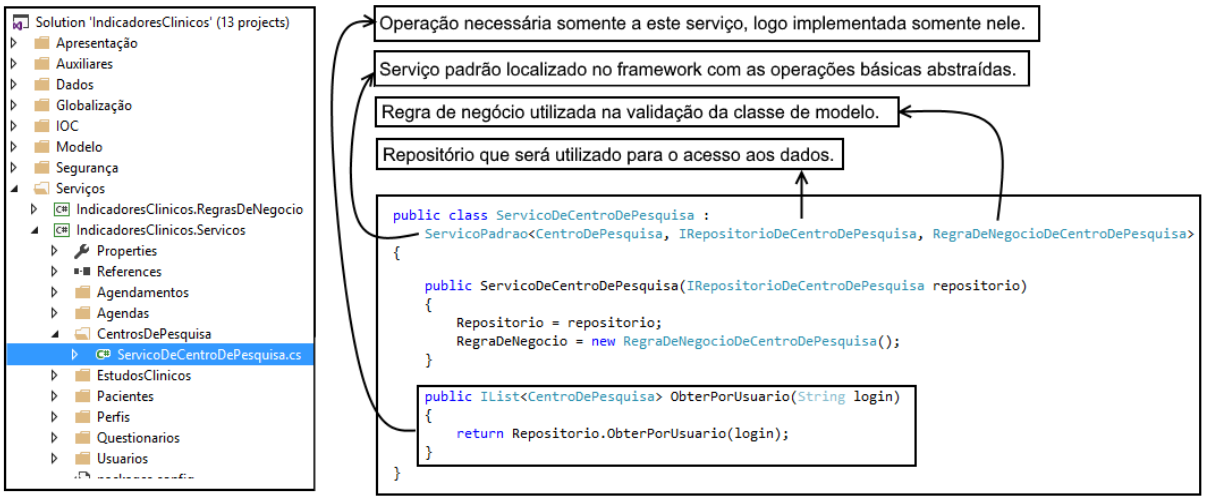


Figura 25 – Exemplo de implementação de um serviço

Fonte: Elaborado pelo autor

A Figura 25 mostra um serviço fazendo uso da interface de repositório para acessar os dados, além de uma regra de negócio correspondente a classe do modelo. Todos os métodos e funções comuns ficam abstraídos no serviço padrão, dentro do framework e estando este exibido na Figura 27.

A Figura 26 exibe outro exemplo de serviço em que não foi necessário a criação de novas operações, existindo apenas a necessidade as básicas.

```
public class ServicoDePessoa : ServicoPadrao<Pessoa, IRepositorioDePessoa, RegraDeNegocioDePessoa>
{
    public ServicoDePessoa(IRepositorioDePessoa repositorio)
    {
        Repositorio = repositorio;
        RegraDeNegocio = new RegraDeNegocioDePessoa();
    }
}
```

Figura 26 – Exemplo de implementação de um serviço

Fonte: Elaborado pelo autor

A simplificação de conteúdo dos serviços é possível por utilizarem do serviço padrão criado no framework, a partir das operações básicas e comuns, a definição deste serviço é representada na Figura 27.

```
public abstract class ServicoPadrao<TObjeto, TRepositorio, TRegraDeNegocio>
    where TObjeto : ObjetoComID
    where TRegraDeNegocio : IRegraDeNegocio<TObjeto>
    where TRepositorio : IRepositorioPadrao<TObjeto>
{

    protected TRegraDeNegocio RegraDeNegocio { get; set; }

    protected TRepositorio Repositorio { get; set; }
    public virtual TObjeto Obter(int identificador)
    {
        return Repositorio.Obter(identificador);
    }
    public virtual void Remover(int id)
    {
        Repositorio.Remover(id);
    }
    public IList<TObjeto> Obter()
    {
        return Repositorio.Obter();
    }
    public virtual TObjeto Salvar(TObjeto objeto)
    {
        if (RegraDeNegocio.Valido(objeto))
        {
            if (objeto.Id == 0)
            {
                return Repositorio.Adicionar(objeto);
            }
            else
            {
                Repositorio.Atualizar(objeto);
                return objeto;
            }
        }
```

Figura 27 – Definição do serviço padrão do framework

Fonte: Elaborado pelo autor

Ainda que utilizado o serviço padrão é possível sobrescrever os métodos, moldando-os da forma que for necessária a determinada funcionalidade. Ao sobrescrever os métodos é importante fazer uso da classe de regra de negócio, que

quando utilizado o serviço padrão é realizado automaticamente como pode ser visualizado na Figura 27.

4.8 Registro de interfaces e classes para injeção de dependência

A injeção de dependência é um dos pilares desta arquitetura e tem por objetivo “preencher o comportamento” ou o conteúdo nas interfaces utilizadas ao longo das camadas. Este padrão de projeto permite que o serviço, e o restante das camadas, não tenham conhecimento de qual forma e onde estão sendo salvas as informações, possibilitando que sejam alteradas.

A Figura 28 representa um exemplo de vínculo para injeção de dependência, fazendo uso da biblioteca StructureMap, que auxilia na injeção da classe para uma determinada interface.

```
namespace IndicadoresClinicos.Dependencias
{
    class VinculosDeRepositorios : Registry
    {
        public VinculosDeRepositorios()
        {
            For<Repositorios.CentrosDePesquisa.IRepositorioDeCentroDePesquisa>()
                .Use<Repositorios.EF.CentrosDePesquisa.RepositorioDeCentroDePesquisa>();
```

Figura 28 – Exemplo de vínculo entre interface e classe

Fonte: Elaborado pelo autor

De modo a complementar o exemplo da Figura 28 é possível observar na Figura 29 dois vínculos para uma mesma interface, estando um deles comentado de forma que é valido somente uma classe concreta por interface.

```
namespace AplicacaoCRUD.Dependencias
{
    class VinculosDeRepositorios : Registry
    {
        public VinculosDeRepositorios()
        {
            For<Repositorios.Pessoas.IRepositorioDePessoa>().Use<Repositorios.ORM.Pessoas.RepositorioDePessoa>();

            //For<Repositorios.Pessoas.IRepositorioDePessoa>().Use<Repositorios.ADONET.Pessoas.RepositorioDePessoa>();
```

Figura 29 – Exemplo de vínculo entre interface e classe

Fonte: Elaborado pelo autor

O exemplo da Figura 29 ilustra que uma possível troca entre tecnologia de acesso aos dados é possível apenas ao trocar o vínculo da classe, pois ambas implementam a mesma interface, de modo a garantir que as operações necessárias estão nas duas classes concretas, ainda que variando a forma de acessar o local de armazenamento das informações.

```
public class FabricaDeRepositorios
{
    public Repositorios.Pessoas.IRepositorioDePessoa ObterRepositorioDePessoa()
    {
        return new Repositorios.ADONET.Pessoas.RepositorioDePessoa();
```

Figura 30 – Exemplo de vínculo injeção de dependência por fábrica

Fonte: Elaborado pelo autor

Apesar de ambos os exemplos fazerem uso da biblioteca StructureMap é possível implementar através do padrão de projeto de fábricas, como é visualizado na Figura 30. Uma eventual troca no exemplo da Figura 30 seria correspondente a trocar a instanciação da classe, pois o retorno da função sempre trabalha com a interface.

APÊNDICE E – ACOMPANHAMENTO DE INDICADORES CLÍNICOS DEPLOY

A instalação do sistema deve ser realizada em um ambiente Microsoft como por exemplo IIS ou em computação em nuvem. O conteúdo a ser copiado para o ambiente está armazenado em um arquivo compacto chamado “IndicadoresClinicos.zip” na pasta de deploy. Para o controle dos agendamentos é necessário copiar o conteúdo do arquivo compactado “ControleAgendamento.zip”.

O controle de agendamento, em um cenário de ambiente do IIS pode ser executado uma vez por dia por tarefa agendada, caso o ambiente seja um serviço de computação em nuvem o mesmo pode ser disparado por um WebJob ou serviço do similar, também uma vez ao dia.

Após realizado a cópia dos arquivos compilados para os diretórios é necessário configurar a conexão com o banco de dados. O arquivo que mantem a conexão para o site dos Indicadores Clínicos é o “web.config” e para o Controle de Agendamentos o “app.config” ambos possuem a mesma estrutura de conexão, observada na Figura 1. Deve ser alterado apenas o nome do servidor, do banco de dados, usuário e senha para conexão com o banco de dados.

```
<connectionStrings>
  <add name="conexaoBD"
       connectionString="Server=DESKTOP-172R0R9\SQLEXPRESS;Database=IndicadoresClinicos;User Id=sa; Password=[redacted];"
       providerName="System.Data.SqlClient" />
```

Figura 1 – String de conexão das aplicações

Fonte: Elaborado pelo autor

Para proteger a integridade das informações no arquivo de configuração, como usuário e senha, é possível[27] sem comprometer o funcionamento da aplicação.

Para o controle de agendamentos é necessário configurar também no arquivo “app.config” é a configuração “EnderecoAplicacao” conforme ilustra a Figura 2.

[27] Mais informações em https://msdn.microsoft.com/en-us/library/bb986855.aspx

```
<appSettings>
  <add key="EnderecoAplicacao" value="DESKTOP-172R0R9/IndicadoresClinicos"/>
```

Exemplo utilizando o nome da máquina e do WebSite

```
<appSettings>
  <add key="EnderecoAplicacao" value="200.90.89.89/IndicadoresClinicos"/>
```

Exemplo utilizando o IP e nome do WebSite

Figura 2 – Configuração do endereço do site

Fonte: Elaborado pelo autor

Esta configuração do nome do site é necessária para que o sistema gere os links automáticos de preenchimento dos questionários antes de enviar por e-mail.

Como última tarefa de instalação é necessário popular o banco de dados. Deve ser utilizado um SQL Server, podendo ser a versão gratuita. Para a criação da estrutura e dos dados é necessário executar o arquivo “IndicadoresClinicosBD.sql”.

Para realizar o login na aplicação é possível utilizar o usuário “admin” e a senha “O2s&b9y” – sem aspas. É recomendado alterar a senha após o login.