JÚLIO CESAR LIMA

Roteiro para a Utilização da Arquitetura Hexagonal no Projeto Dirigido pelo Domínio

São Paulo
2016

JÚLIO CESAR LIMA

Roteiro para a Utilização da Arquitetura Hexagonal no Projeto Dirigido pelo Domínio

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do MBA em Tecnologia de Software

São Paulo
2016

JÚLIO CESAR LIMA

Roteiro para a Utilização da Arquitetura Hexagonal no Projeto Dirigido pelo Domínio

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do MBA em Tecnologia de Software

Área de concentração:

PECE/POLI

Orientador:
Prof. Dr. Paulo Sérgio Muniz Silva

São Paulo
2016

AGRADECIMENTOS

Ao professor Paulo Sérgio Muniz Silva, pelo aprendizado em suas aulas e durante as reuniões de orientação deste trabalho.

À Andréa Britto Mattos Lima, minha esposa, por todo apoio, carinho e incentivo em meus estudos e para a elaboração deste trabalho.

RESUMO

O controle da complexidade do desenvolvimento do software não é uma tarefa trivial. Um dos fatores fundamentais para conter essa complexidade é a elaboração de um modelo apropriado do domínio da aplicação, que deve ser isolado dos vários mecanismos necessários à implementação do software que não dizem respeito ao domínio (persistência, controle de transações, etc.). Essa é a essência do Projeto Dirigido pelo Domínio (*Domain Driven Design* - DDD). Esse isolamento implica a seleção de uma arquitetura apropriada para tratar a interação de um domínio da aplicação não só com outros domínios, mas com interações heterogêneas tais como protocolos e suportes de comunicação distintos. Além disso, esses elementos normalmente evoluem com o tempo, aumentando a complexidade do software a ser tratada pela arquitetura. Um estilo arquitetônico flexível proposto para solucionar esses problemas no âmbito do DDD é a Arquitetura Hexagonal. No entanto, o desenvolvedor iniciante carece de uma orientação para utilizar essa arquitetura, de modo aderente ao DDD, em projetos de software que tenham tais características de complexidade. Para contribuir no preenchimento dessa lacuna, o presente trabalho propõe um roteiro para a utilização da Arquitetura Hexagonal como base para a integração entre domínios e com flexibilidade para o suporte de dispositivos que se comunicam com um domínio-alvo.

Palavras-chave: Arquitetura Hexagonal. Integração de domínios. Projeto Dirigido pelo Domínio (DDD).

ABSTRACT

Controlling the complexity of software development is not a trivial task. One of the key factors to minimize this complexity is the development of an appropriate model of the application domain, which must be isolated from various mechanisms required for the implementation of the software that are unrelated to the domain (persistence, transaction control, etc.). This is the essence of Domain Driven Design (DDD). Such isolation implies on the selection of an appropriate architecture capable of addressing the interaction of an application domain not only with other areas, but also with heterogeneous interactions, such as different communication protocols and supports. Furthermore, these elements typically evolve over time, increasing the complexity of software to be treated by the architecture. A flexible architectural style proposed to solve these problems within the DDD is the Hexagonal Architecture. However, an inexperienced developer lacks orientation on how to use this architecture, adhering to the DDD model, in software designs that have such complexity characteristics. This work aims to contribute on filling this gap by proposing a roadmap for using the Hexagonal Architecture as the basis for integration across domains with flexibility to support devices that communicate with a target domain.

Keywords: Hexagonal Architecture. Integration domains. Project Managed by domain (DDD).

LISTA DE ILUSTRAÇÔES

Figura 1 - Modelo geral de Arquitetura em Camadas.....15
Figura 2 - Arquitetura SOA.....16
Figura 3 - Arquitetura Hexagonal 17
Figura 4 - Método Fábrica.....20
Figura 5 - Fábrica Abstrata.....20
Figura 6 - Padrão Adaptador.....21
Figura 7 - Padrão Fachada.....22
Figura 8 - Padrão Estado 22
Figura 9 - Padrão Estratégia 23
Figura 10 - Mapa Contextual.....27
Figura 11 - Núcleo Compartilhado.....28
Figura 12 - Camada Anticorrupção 30
Figura 13 - Agregado 32
Figura 14 - Mapa Contextual de exemplo 37
Figura 15 - Mapa Contextual de exemplo com Padrão Estratégico de Integração....38
Figura 16 - Estereótipo da Porta conceitual 39
Figura 17 – Notação da Porta conceitual 39
Figura 18 - Agregado Amostragem 43
Figura 19 - Agregado Questionário 44
Figura 20 - Mapa contextual do domínio 46
Figura 21 - Mapa contextual de integração 47
Figura 22 - HA/PA para o caso.....48
Figura 23 - Integração entre Módulos 49
Figura 24 - Integração entre Módulos com portas conceituais.....50
Figura 25 - onda.core 51

Figura 26 - onda.core.variaveldemografica 54
Figura 27 - onda.core.questionariorespondido 55
Figura 28 - onda.amostra 56
Figura 29 - gerenciador.questionario.core 58
Figura 30 - onda.aplicacao.backoffice 58
Figura 31 - onda.aplicacao.web 59
Figura 32 - onda.aplicacao.mobile 60
Figura 33 - onda.aplicacao.impresso 61
Figura 34 - onda.persistencia 62

LISTA DE TABELAS

Tabela 1 - Padrões de Projeto.....19

LISTA DE SIGLAS E ABREVIATURAS

ACL	Anticorruption Layer
API	Application Programmed Interface
DDD	Domain Driven Design
HA/PA	Hexagonal Architecture / Ports and Adapters
SOA	Service Oriented Architecture

SUMÁRIO

1. INTRODUÇÃO 11
1.1. Motivação 11
1.2. Objetivo 12
1.3. Método de Trabalho 13
1.4. Estrutura do Trabalho 13
2. PADRÕES DE PROJETO DE SOFTWARE 14
2.1. Padrões de Projeto Arquitetônico 14
2.1.1. Arquitetura em Camadas 14
2.1.2. Arquitetura Orientada a Serviço 15
2.1.3. Arquitetura Hexagonal 16
2.2. Padrões do Projeto Detalhado 18
2.2.1. Principais Padrões Adotados neste Trabalho 19
2.2.2. Princípios de Projeto Fundamentais 23
3. PROJETO DIRIGIDO PELO DOMÍNIO 25
3.1. Projeto Dirigido pelo Domínio (DDD) 25
3.2. Linguagem Onipresente 26
3.3. Projeto Estratégico 26
3.3.1. Contexto Delimitado 26
3.3.2. Mapa Contextual 26
3.3.3. Relações entre Contextos Delimitados 27
3.4. Projeto Tático 30
3.4.1. Entidade 30
3.4.2. Objeto Valor 31
3.4.3. Serviços 31
3.4.4. Módulo 32
3.4.5. Agregado 32

3.4.6. Fábrica....33
3.4.7. Repositório....33
4. ROTEIRO PARA UTILIZAÇÃO DA ARQUITETURA HEXAGONAL....34
4.1. Resumo da pesquisa bibliográfica....34
4.2. Roteiro proposto para a utilização da Arquitetura Hexagonal....34
4.2.1. Definição do domínio....34
4.2.2. Projeto de alto nível....35
4.2.3. Projeto Detalhado....36
4.3. Definição dos passos....36
4.3.1. Passo 1: Definição dos Contextos Delimitados e da Linguagem Onipresente....37
4.3.2. Passo 2: Projeto Estratégico – Integração....38
4.3.3. Passo 3: Projeto Estratégico – Arquitetura Hexagonal – Definição das Portas 39
4.3.4. Passo 4: Projeto Tático – Integração....39
4.3.5. Passo 5: Projeto Tático – Detalhamento dos Módulos....40
5. UM EXEMPLO DE APLICAÇÃO DA ARQUITETURA HEXAGONAL....41
5.1. Descrição do domínio do problema....41
5.2. Passo 1: Definição dos Contextos Delimitados e Linguagem Onipresente..42
5.2.1. Descrição dos Contextos Delimitados....42
5.2.2. Mapa Contextual do Domínio....46
5.3. Passo 2: Projeto Estratégico – Integração....47
5.3.1. Onda-Amostragem....47
5.3.2. Onda-Questionário....47
5.3.3. Mapa Contextual com os Padrões Estratégicos de Integração....47
5.4. Passo 3: Projeto Estratégico – Arquitetura Hexagonal – Definição das Portas 48
5.5. Passo 4: Projeto Tático – Integração....49

5.5.1. Modelo de Integração entre Módulos 49
5.5.2. Modelo da Integração entre módulos com as Portas conceituais da HA/PA 49
5.6. Passo 5: Projeto Tático – Detalhamento dos Módulos.....50
5.6.1. Utilização de padrões 50
5.6.2. Detalhamento dos Módulos 51
6. CONCLUSÃO 63
6.1. Considerações gerais.....63
6.2. Trabalhos futuros 64
REFERÊNCIAS.....65

1. INTRODUÇÃO

1.1. Motivação

Controlar a complexidade do desenvolvimento de softwares não é trivial (EVANS, 2010, p. xxi). Portanto, é fundamental dedicar-se para alcançar um bom modelo que permita o máximo de aproveitamento do software, de maneira a conter essa complexidade. Entretanto, a modelagem de domínios é também uma tarefa complexa, por isso é importante considerar continuamente o modelo conceitual juntamente com as tarefas de implementação. Portanto, os modelos de domínio não devem ser primeiro projetados para então serem implementados: bons modelos devem ser adaptados e evoluídos em conjunto ao longo do projeto, tornando-se enriquecidos após várias iterações de projeto (*design*). Isso constitui a pedra fundamental do Projeto Dirigido pelo Domínio (*Domain Driven Design* - DDD) (EVANS, 2010).

Devido à complexidade do domínio, um software torna-se mais difícil de ser compreendido em todos os seus estados e mais difícil de ser extensível sem efeitos colaterais (DA-WEI, 2007). Além do mais, um software torna-se mais complexo por suas próprias implicações técnicas, dentre as quais as dependências dos mecanismos de persistência, de controle de transações, de segurança, dentre outros (GHAZARIAN, 2015). Para organizar estas implicações em configurações de componentes, uma arquitetura de software visa domar essa complexidade. No entanto, é necessário selecionar a arquitetura adequada, tendo-se o entendimento dos vários estilos arquitetônicos, para que se tenha sucesso no tratamento da complexidade do software (GARLAN e SHAW, 1994, p. 2).

O problema da complexidade de software aumenta quando se defronta com a integração de subsistemas que devem implementar requisitos de software de domínios distintos. Em (JAIN, *et al.*, 2008) foi realizado um estudo para avaliar os impactos da complexidade no processo de integração de sistemas, em que foi apontado que as arquiteturas de integração de sistemas, se não bem planejadas, podem levar um sistema à falha, especialmente se o escopo da integração não foi bem definido. Requisitos de projeto da arquitetura e do plano de integração são algumas das entradas do processo de integração de sistemas. Uma conclusão é que o domínio da aplicação tem impacto direto no projeto de uma arquitetura de sistema

e, consequentemente, na integração das arquiteturas dos subsistemas que o compõem.

Outro problema importante que surge na escolha da arquitetura apropriada para um determinado domínio da aplicação é a flexibilidade para tratar vários subsistemas clientes possuindo interações heterogêneas (por exemplo, protocolos e suportes de comunicação distintos), que podem se alterar ao longo do tempo, ou mesmo a inclusão ou exclusão destes clientes. A questão é como produzir uma simetria nesse tratamento, de tal modo que as interações sejam vistas de modo similar e que a aplicação não seja afetada por essa diversidade. Um estilo arquitetônico proposto para solucionar esse problema no âmbito do DDD, assim como para facilitar a integração de domínios distintos, é a Arquitetura Hexagonal (COCKBURN, 2005), dando foco no domínio-alvo para atender à heterogeneidade dos subsistemas clientes e liberar o projeto do domínio-alvo das questões de integração. A integração é tratada num segundo momento do projeto, permitindo uma flexibilidade na evolução do sistema.

No entanto, a pesquisa bibliográfica feita parece indicar que, aparentemente, o desenvolvedor iniciante carece de uma orientação relativamente detalhada para utilizar essa arquitetura, de modo aderente ao DDD, em projetos de software que tenham tais características de complexidade.

1.2. Objetivo

O objetivo deste trabalho é propor um roteiro para a utilização da Arquitetura Hexagonal como base para a integração entre domínios e com flexibilidade para que dispositivos possam se comunicar com um domínio-alvo. O roteiro baseia-se em técnicas do Projeto Dirigido pelo Domínio (*Domain Driven Design* – DDD) e pretende ser utilizado por desenvolvedores iniciantes no DDD. A Arquitetura Hexagonal foi escolhida porque mantém o enfoque no domínio-alvo, aplicando o DDD e liberando seu projeto da complexidade adicional imposta pela integração e a diversidade dos dispositivos.

O roteiro é exercitado em um fragmento significativo de uma aplicação de software real de grande porte.

1.3. Método de Trabalho

O método de trabalho utilizado nesta monografia seguiu os seguintes passos:

- Definição de um caso de negócio a ser projetado;
- Definição do DDD como técnica de modelo de domínio;
- Estudo do DDD (pesquisa bibliográfica);
- Estudo da Arquitetura Hexagonal como padrão arquitetônico;
- Definição de um roteiro para o emprego da Arquitetura Hexagonal no DDD.
- Projeto (*design*) do Modelo Estratégico e Tático para a Arquitetura Hexagonal.

1.4. Estrutura do Trabalho

CAPÍTULO 2 – PADRÕES DE PROJETO DE SOFTWARE

Descreve alguns padrões de projeto, tanto arquitetônico quanto detalhado, que serão abordados como possíveis soluções de projeto de sistemas. A ênfase é dada aos padrões utilizados no presente trabalho.

CAPÍTULO 3 – PROJETO DIRIGIDO PELO DOMÍNIO

Provê os principais pontos do projeto estratégico e do projeto tático como abordagem para a modelagem de um sistema baseado no domínio de negócio.

CAPÍTULO 4 – ROTEIRO PARA UTILIZAÇÃO DA ARQUITETURA HEXAGONAL

Descrição de um roteiro para a utilização da Arquitetura Hexagonal em conjunto com o Projeto Dirigido pelo Domínio.

CAPÍTULO 5 – UM EXEMPLO DE APLICAÇÃO DA ARQUITETURA HEXAGONAL

Apresenta um caso real de uma empresa como exemplo para elaborar um projeto de sistema baseado no roteiro proposto no capítulo 4.

CAPÍTULO 6 – CONCLUSÃO

Conclusão do trabalho e trabalhos futuros.

2. PADRÕES DE PROJETO DE SOFTWARE

Descrevem-se neste capítulo alguns Padrões de Projeto Arquitetônico e alguns Padrões de Projeto Detalhado de interesse para este trabalho.

Neste capítulo, entende-se por Padrões de Projeto soluções técnicas para problemas específicos de uma implementação de software. São utilizados em sistemas de software com a finalidade de dar flexibilidade, reutilização e permitir extensões de componentes, além de facilitar a novos integrantes de uma equipe de desenvolvimento a familiarização com as técnicas implementadas.

2.1. Padrões de Projeto Arquitetônico

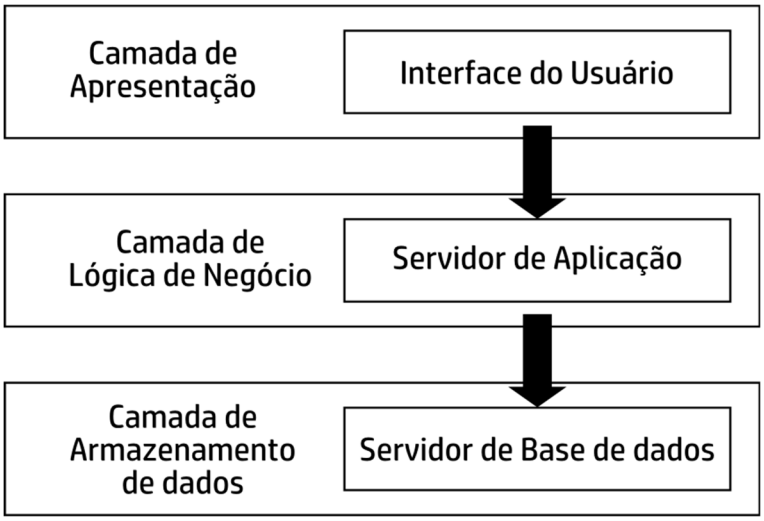
Os padrões arquitetônicos de software (BUSCHMANN, *et al.*, 1996, p. 26) são modelos nos quais desenvolvedores se apoiam na construção de um software, dividindo-o em partes e componentes para a definição de responsabilidades. Não há um modelo de arquitetura correto para todos os softwares. Para um determinado caso, um padrão de arquitetura de software pode ser mais adequada que outra para fornecer a melhor solução de um problema e a melhoria no desenvolvimento de um software.

2.1.1. Arquitetura em Camadas

A Arquitetura em Camadas (BUSCHMANN, *et al.*, 1996, p. 31) é mais conhecida pela divisão em 3 camadas, dividida normalmente em Apresentação, Negócio e Acesso a Dados (FOWLER, *et al.*, 2002, p. 19) (TIE, JIN e WANG, 2011), como representado na Figura 1.

A dependência entre camadas é de cima para baixo (EVANS, 2010, p. 65). A vantagem desse modelo é a de preparar a aplicação para potenciais alterações em determinada camada para que mudanças específicas não sejam necessárias em outras camadas. Por exemplo, no caso de um servidor estar sobrecarregado com o gerenciamento de dados e lógica de negócio, a divisão de carga poderia ser dividida entre dois servidores, sendo um apenas para dados e outro para lógica de negócio, além da apresentação. Este modelo também auxilia na separação da aplicação por especialistas, como um desenvolvedor web, programador e um especialista em banco de dados.

Figura 1 - Modelo geral de Arquitetura em Camadas



Fonte (TIE, JIN e WANG, 2011)

Algumas variações do modelo de camadas são as responsabilidades que cada camada possui, ou a adição de camadas de apresentação superiores utilizando a camada de negócio/dados. Neste modelo, a forte dependência das camadas superiores dificulta os testes na camada de negócio, além da adição de camadas de dados conforme necessário. Ao adicionar uma camada abaixo, será necessário levar essa dependência para as camadas acima.

2.1.2. Arquitetura Orientada a Serviço

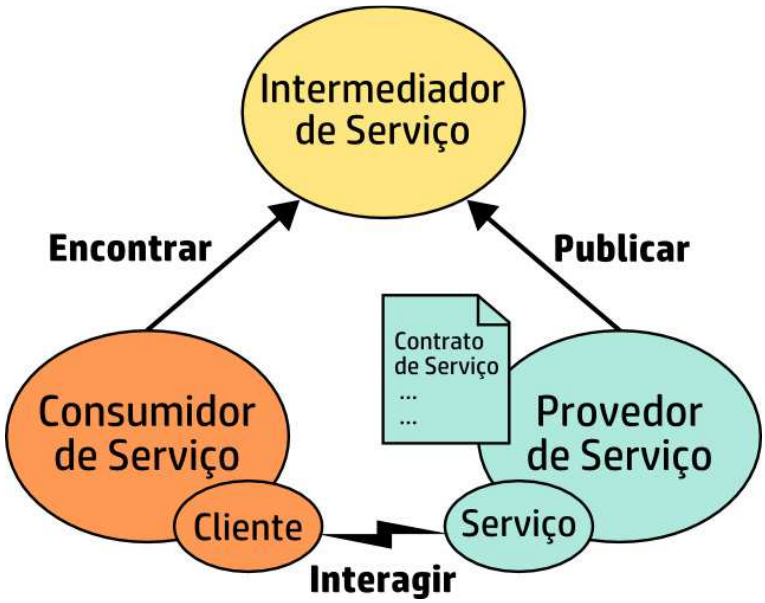
A Arquitetura Orientada a Serviço ou *Service Oriented Architecture* (SOA) (OASIS SOA REFERENCE MODEL TC, 2006) tem por objetivo fornecer serviços através de tecnologias que permitam a comunicação entre partes de uma ou várias aplicações. As tecnologias XML, SOAP, WSDL, UDDI e HTTP permitem que diferentes plataformas sejam integradas, abstraindo da aplicação-cliente a implementação por trás do serviço (WANG e LIAO, 2009).

O projeto do serviço pode utilizar outras tecnologias para permitir escalabilidade ou um fluxo de processamento específico, localizado na mesma rede de uma empresa ou na internet. A proposta da SOA é ser flexível e reutilizável, embora necessite de um controle efetivo para que as aplicações dependentes não sofram com alterações não previstas. Apesar das vantagens da integração, não é de simples desenvolvimento.

Na Figura 2, é apresentada a relação entre os principais agentes da SOA, sendo eles:

O Consumidor de Serviço, fazendo o papel de cliente, que para utilizar serviços, primeiramente, precisa encontrar os serviços disponíveis através de um Intermediador de Serviço. O Intermediador de Serviço conhece os Serviços, pois um Provedor o publicou para entrar no catálogo. Para que o Cliente de Serviço possa utilizar um serviço, ele deve respeitar o contrato e então efetivamente interagir com o Serviço.

Figura 2 - Arquitetura SOA



Fonte (HAAS, 2003)

Os serviços disponibilizados também devem ser desenvolvidos adequadamente para os propósitos que foram concebidos, para que representem o serviço de negócio que a empresa espera que seja resolvido.

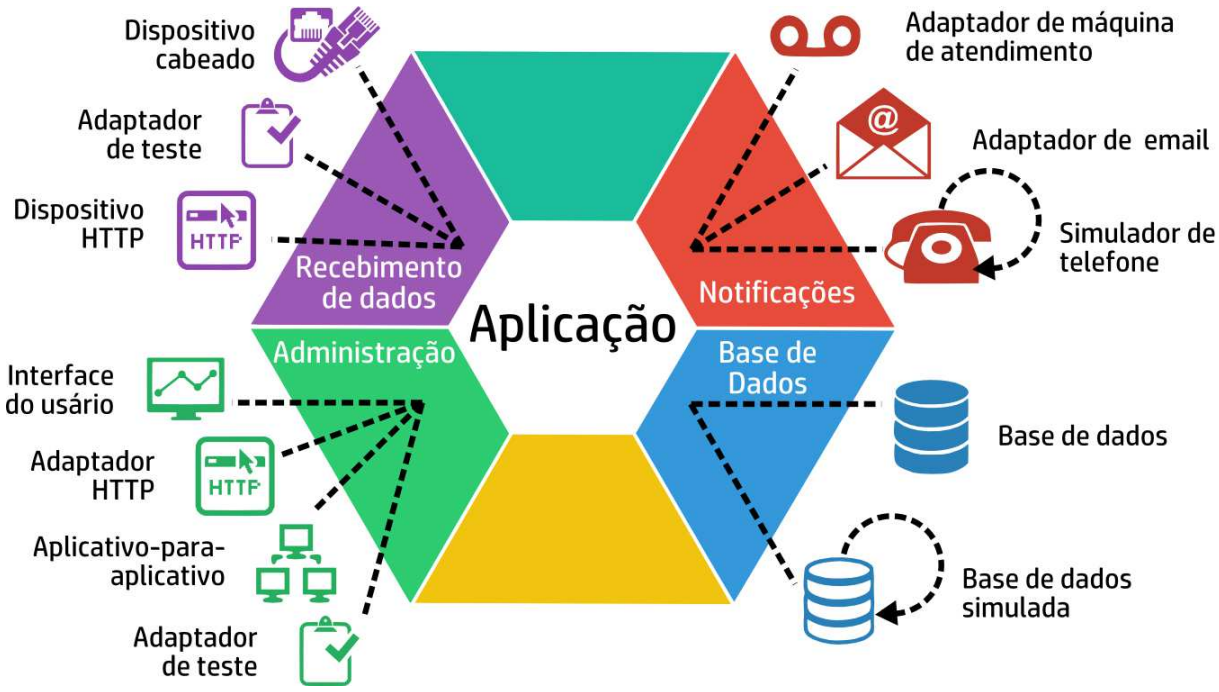
2.1.3. Arquitetura Hexagonal

A Arquitetura Hexagonal (COCKBURN, 2005) tem por proposta separar o domínio da aplicação da infraestrutura técnica da aplicação (PRYCE, 2009). A metáfora do hexágono enfatiza as múltiplas possibilidades em que os clientes do software interagem com ele, além de delimitar duas áreas principais: a externa e a interna, a qual contém efetivamente o projeto dos serviços oferecidos pelo software. Cada lado representa um tipo de porta de comunicação do domínio com o mundo externo e vice-versa, ou seja, cada lado do hexágono tem um objetivo de acesso de uma parte do domínio específico. Apesar de ser representada por um hexágono, esta representação não é limitada a seis lados apenas, mas a quantos lados forem necessários para a comunicação do lado interno com o lado externo. Dada uma finalidade de um lado do

hexágono, ela não está restrita a um tipo de comunicação, pois vários dispositivos podem ser integrados àquela porta através de adaptadores.

Na Figura 3, é representada a Arquitetura Hexagonal com exemplos das finalidades dos lados do hexágono, metaforicamente, e com tipos de dispositivos com o qual se comunicam. Cada lado do hexágono é uma Camada de Aplicação, que neste exemplo possui 4 pontos de comunicação com o domínio: Administração do sistema, Recebimento de dados, Envio de notificações e Base de Dados.

Figura 3 - Arquitetura Hexagonal



Fonte (COCKBURN, 2005) / Adaptado (o autor)

Esta arquitetura é também conhecida como Portas e Adaptadores. Neste trabalho, será utilizada a abreviatura HA/PA do inglês *Hexagonal Architecture* / *Ports and Adapters*.

Diferentemente da Arquitetura em Camadas tradicional (seção 2.1.1), em que a camada superior acessa a camada inferior, a HA/PA centraliza o negócio e abstrai as demais camadas, tornando-se disponível para ser utilizado com o mínimo de dependências. Isso facilita os testes automatizados e promove a flexibilidade, pois novas camadas podem ser plugadas através de portas e adaptadores.

A flexibilidade da HA/PA é fornecida por suas portas, vistas como APIs (*Application Programmed Interface*), que são desenvolvidas em volta do centro. Não apenas a

flexibilidade, mas também os testes automatizados são essenciais para garantir que o domínio esteja íntegro e que a nova API esteja funcionando. Técnicas de testes amplamente abordadas são apoiadas pela a HA/PA (FREEMAN e PRYCE, 2009).

Com o foco no domínio, a eliminação de regras de negócio na camada de apresentação ou base de dados é encorajada para mostrar que o domínio trata de todas as regras esperadas.

2.1.3.1. Porta

Uma Porta da HA/PA não tem uma definição formal, sendo compreendida como um acesso à parte domínio, ou ao centro do hexágono. Em sua proposta (COCKBURN, 2005) propõe que de alguma face do hexágono há a comunicação com a parte externa e que cada face tem o objetivo de limitar a comunicação necessária com o domínio (COCKBURN, 2006).

Para garantir o funcionamento da Porta, testes são necessários para validar o modelo de domínio acessado por aquela porta (FREEMAN e PRYCE, 2009, p. 10).

As Portas da HA/PA podem ser caracterizadas como Camadas de Aplicação, onde interage entre o domínio e com dispositivos e meios de comunicação.

2.1.3.2. Adaptador

Um adaptador para a HA/PA é qualquer artifício da arquitetura que permita modificar algo que tente se comunicar com o centro do hexágono e precisa ser modificado para seu correto funcionamento, permitindo uma tradução para o domínio. Do ponto de vista do *design*, o Adaptador da HA/PA pode assumir o papel de um Adaptador (seção 2.2.1.3) do (GAMMA, *et al.*, 2007).

A comunicação com uma base de dados MySQL, por exemplo, seria através de uma API para fazer as requisições. A maneira como a aplicação utiliza a API do MySQL deve ser transparente para o domínio, podendo utilizar para tanto um Adaptador. O padrão Adaptador, dos Padrões de Projetos (GAMMA, *et al.*, 2007), é bastante apropriado para ser utilizado pela HA/PA.

2.2. Padrões do Projeto Detalhado

“Os padrões de projeto ajudam a escolher alternativas de projeto que tornam um sistema reutilizável e a evitar alternativas que comprometam a reutilização. Os

padrões de projeto podem melhorar a documentação e a manutenção de sistemas ao fornecer uma especificação explícita de interações de classes e objetos e o seu objetivo subjacente. Em suma, ajudam um projetista a obter mais rapidamente um projeto adequado.” (GAMMA, *et al.*, 2007, p. 18).

2.2.1. Principais Padrões Adotados neste Trabalho

Tipos de padrões

Os Padrões de Projeto são divididos por finalidade (são três) e escopo (GAMMA, *et al.*, 2007), conforme a Tabela 1.

Tabela 1 - Padrões de Projeto

		Finalidade		
		Criação	Estrutural	Comportamental
Escopo	Classe	Método Fábrica	Adaptador (classe)	Interprete Método Modelo
	Objeto	Fábrica Abstrata Construtor Protótipo Instância Única	Adaptador (objeto) Ponte (*Bridge*) Compósito (*Composite*) Decorador Fachada Peso-mosca (*Flyweight*) Proxy	Cadeia de Responsabilidade Comando Iterador Mediador Memento Observador Estado Estratégia Visitante

Fonte (GAMMA, *et al.*, 2007, p. 26)

Criação

Os padrões de criação abstraem o processo de instanciação. Eles usam a herança para variar a classe que é instanciada. Desta maneira, o sistema pode funcionar de maneira independente de quantos filhos (quantas generalizações) uma classe pai pode ter.

Estruturais

Os padrões estruturais resolvem problemas de composição de classes e objetos dando flexibilidade à expansão do sistema.

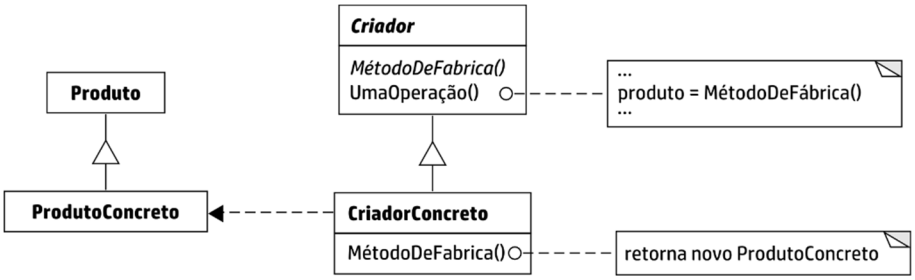
Comportamentais

Os padrões comportamentais resolvem problemas de tempo de execução e mostram como montar estruturas para que o fluxo de comunicação seja simplificado.

2.2.1.1. Método Fábrica

O Método Fábrica instancia uma subclasse de outra classe abstrata na subclasse que precisa de uma instância. A classe concreta de uma classe abstrata é quem define qual subclasse é utilizada quando for necessária, como ilustrado na Figura 4.

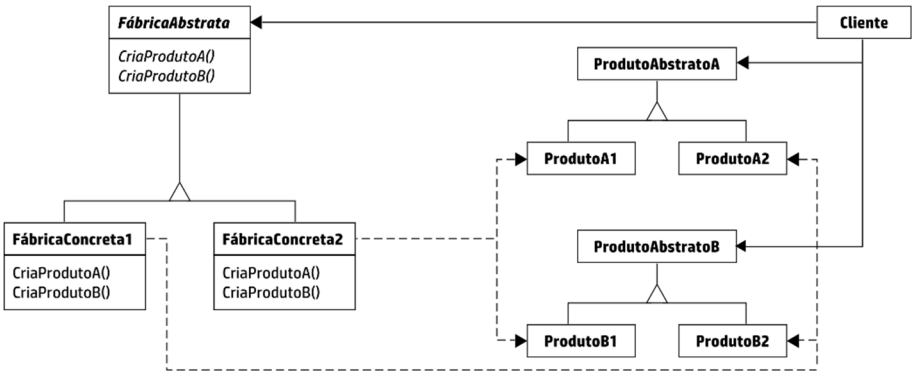
Figura 4 - Método Fábrica



Fonte (GAMMA, *et al.*, 2007, p. 113)

2.2.1.2. Fábrica Abstrata

Figura 5 - Fábrica Abstrata



Fonte (GAMMA, *et al.*, 2007, p. 97)

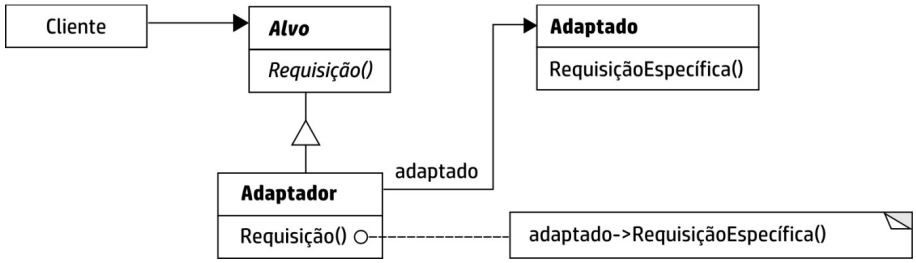
O padrão Fábrica Abstrata, apresentado na Figura 5, permite um cliente utilizar uma classe criadora de objetos sem especificar qual implementação é utilizada pela sua

interface. Classes de fábricas concretas definem as classes concretas de outras generalizações que serão instanciadas.

2.2.1.3. Adaptador

O padrão de projeto Adaptador é responsável por converter uma interface em outra interface que se pretende utilizar. Quando o cliente utiliza uma interface, mas a implementação estaria implementada em outra interface, uma implementação intermediária, o adaptador, faria a tradução para a chamada da outra interface. Para aplicar este padrão, a classe adaptador deve implementar a interface alvo e utilizar, por associação, a interface adaptada, como representado na Figura 6.

Figura 6 - Padrão Adaptador

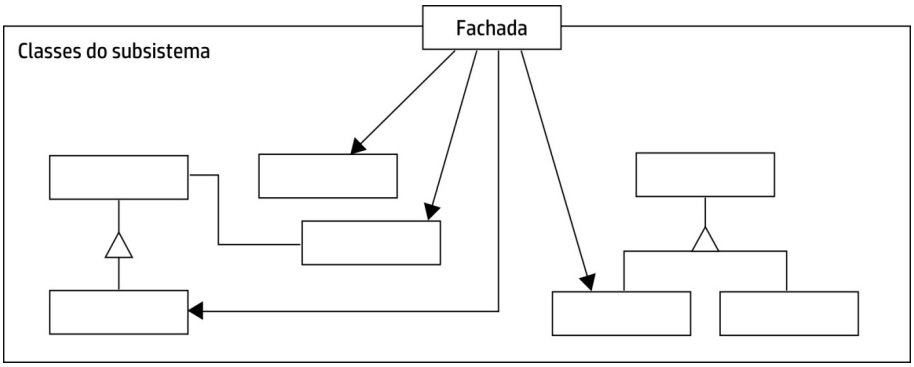


Fonte (GAMMA, *et al.*, 2007, p. 142)

2.2.1.4. Fachada

O padrão Fachada abstrai as dependências de diversas interfaces para um cliente de modo a reduzir as dependências. Por exemplo, ao executar um caso de uso, que possui uma ordem de execução, a Fachada teria as referências das interfaces dependentes e executaria as chamadas para as outras interfaces, deixando o cliente livre do conhecimento de diversas interfaces uma vez que o cliente tem apenas a Fachada como referência ao subsistema. A Figura 7 ilustra este padrão.

Figura 7 - Padrão Fachada

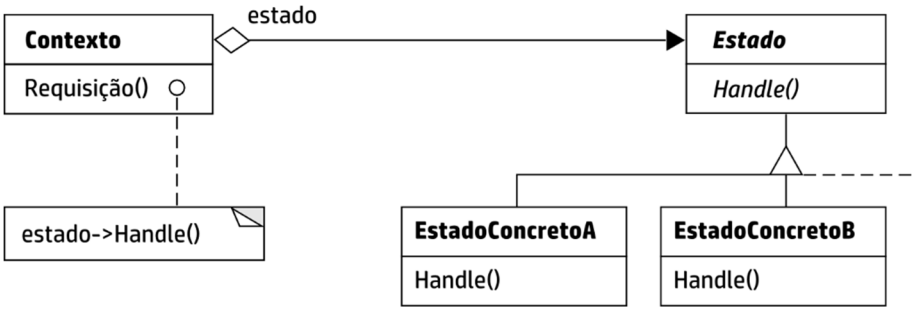


Fonte (GAMMA, *et al.*, 2007, p. 181)

2.2.1.5. Estado

O padrão Estado resolve por delegação o comportamento de uma classe de acordo com seu estado interno. O contexto de uma classe possui uma associação à classe abstrata Estado. Suas subclasses respondem pelo comportamento, como representado na Figura 8.

Figura 8 - Padrão Estado

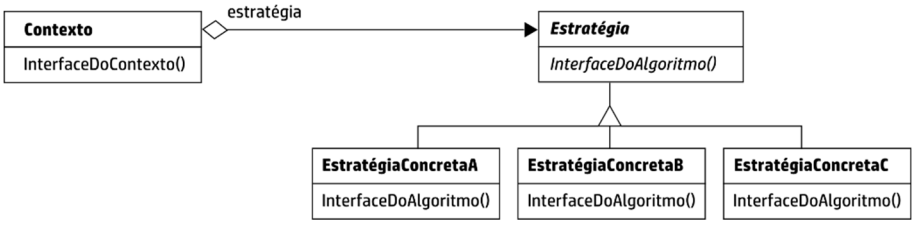


Fonte (GAMMA, *et al.*, 2007, p. 285)

2.2.1.6. Estratégia

O padrão Estratégia, representado na Figura 9, encapsula em cada subclasse de uma interface os algoritmos que podem mudar dependendo da instância. O contexto de uma classe tem associação com uma interface, a qual tem subclasses concretas com diferentes algoritmos.

Figura 9 - Padrão Estratégia



Fonte (GAMMA, *et al.*, 2007, p. 294)

2.2.2. Princípios de Projeto Fundamentais

Os Princípios de Projeto (MARTIN, 2002, p. 86) esclarecem os motivos dos Padrões de Projeto, uma vez que sozinhos solucionam problemas específicos. Os Princípios definem de forma mais ampla o que se procura resolver na orientação a objeto.

2.2.2.1. Princípio da Responsabilidade Única

Uma classe deve ter apenas uma responsabilidade para não sofrer com modificações por mais de um motivo. Caso sejam necessárias modificações, o desenvolvedor não deve se preocupar em alterar outros comportamentos senão o da classe em modificação (MARTIN, 2002, p. 95).

2.2.2.2. Princípio Aberto – Fechado

Uma aplicação deve ser projetada para ser aberto para extensão, quando novos requisitos são desenvolvidos, mas fechado para modificações, para não comprometer um comportamento já em funcionamento. Para atingi-lo, a chave do projeto está na abstração e polimorfismo de classes, para que novos comportamentos sejam desenvolvidos em subclasses e a abstração não sofra alterações (MARTIN, 2002, p. 99).

2.2.2.3. Princípio da substituição de Liskov

Uma classe que possui referência para uma classe-base não deve conhecer as subclasses. Este princípio permite forçar o princípio aberto-fechado, pois uma classe que referencie qualquer subclasse não permitirá estender o sistema (MARTIN, 2002, p. 111).

2.2.2.4. Princípio da segregação de interface

Uma interface deve atender a apenas um comportamento específico. Quando um cliente utilizar a interface, ele utiliza as operações da interface por completo para o

escopo que ela foi projetada. Caso um cliente utilize uma interface parcialmente, o cliente poderá sofrer impactos com alterações que não fariam sentido em operações que não são utilizados (MARTIN, 2002, p. 135).

2.2.2.5. Princípio da inversão de dependência

Para reduzir dependências, a separação de Módulos e, portanto, suas classes devem respeitar hierarquias, para que o módulo superior não dependa de Módulo inferior. Módulos inferiores devem depender das abstrações dos Módulos superiores (MARTIN, 2002, p. 127).

3. PROJETO DIRIGIDO PELO DOMÍNIO

Neste capítulo descreve-se algumas características do Projeto Dirigido pelo Domínio (DDD), tais como: Linguagem Onipresente, Contexto Delimitado, Projeto Estratégico e Projeto Tático.

3.1. Projeto Dirigido pelo Domínio (DDD)

O Projeto Dirigido pelo Domínio, DDD (*Domain Driven Design*) (EVANS, 2010), propõe padrões para implementar software utilizando modelos de domínio, cujo objetivo é trazer o software, no código, o mais próximo possível da linguagem do negócio. O benefício de juntar especialistas no domínio e desenvolvedores é o de que eles tenham a mesma interpretação do que é proposto em um projeto de software, facilitando o desenvolvimento através de técnicas focalizadas no domínio.

O DDD surgiu do reconhecimento de que a modelagem e o *design* são fundamentais no desenvolvimento de software. Ele provê diretivas que orientam a escolha de um modelo de software que deve atender os requisitos de um domínio.

O modelo e o núcleo do *design* dão forma um ao outro. A conexão entre o modelo e a implementação é que torna o modelo relevante, garantindo sua aplicação ao produto final. Essa ligação é também muito útil para as atividades de manutenção e desenvolvimento contínuo do código, que pode ser interpretado com base na compreensão do modelo.

O modelo é a principal base da linguagem utilizada por todos os membros da equipe. Uma vez que modelo e implementação estejam conectados, os desenvolvedores podem conversar sobre o programa nessa linguagem particular. Eles podem comunicar-se como especialistas no domínio sem ter, idealmente, a necessidade de tradução. Como a linguagem de comunicação é baseada nesse modelo, a ideia motivadora do DDD é que a capacidade linguística natural dos membros da equipe (negócio e software) pode ser usada para refinar o próprio modelo.

O modelo é um conhecimento refinado, constituindo a forma aceita pela equipe para estruturar o conhecimento do domínio e distinguir os elementos de maior interesse. A linguagem de comunicação compartilhada permite que os desenvolvedores e os especialistas do domínio trabalhem efetivamente em conjunto à medida em que colocam informações nessa forma.

O DDD refina os caminhos da Modelagem Dirigida por Modelo. Fowler indica que o Modelo de Domínio é separado por uma camada isolada, onde as regras de negócio são transformadas em objetos e serviços (FOWLER, *et al.*, 2002, p. 116). No DDD são apresentados os Padrões de Domínio que devem ser utilizados no *design* (SOARES, *et al.*, 2015).

3.2. Linguagem Onipresente

Base fundamental do DDD, a Linguagem Onipresente utiliza os jargões dos especialistas do domínio no software e seus modelos. Quando desenvolvedores conversam e querem tirar dúvidas sobre o domínio com os especialistas, não precisariam traduzir as nomenclaturas de classes e métodos que apenas eles compreenderiam, mas sim desenvolver um modelo do software com as nomenclaturas do domínio, para que haja fluidez no entendimento entre todos os envolvidos (EVANS, 2010, p. 22).

3.3. Projeto Estratégico

O Projeto Estratégico auxilia no trabalho de como o modelo do domínio será projetado. No Projeto Estratégico, identifica-se como criar os Contextos Delimitados e modularizar o domínio, para então se chegar a integração. Apesar de gerar integrações entre Módulos, deixa-se claro a responsabilidade de cada parte e como serão integradas, assim como no domínio da vida real (EVANS, 2010, p. 317) .

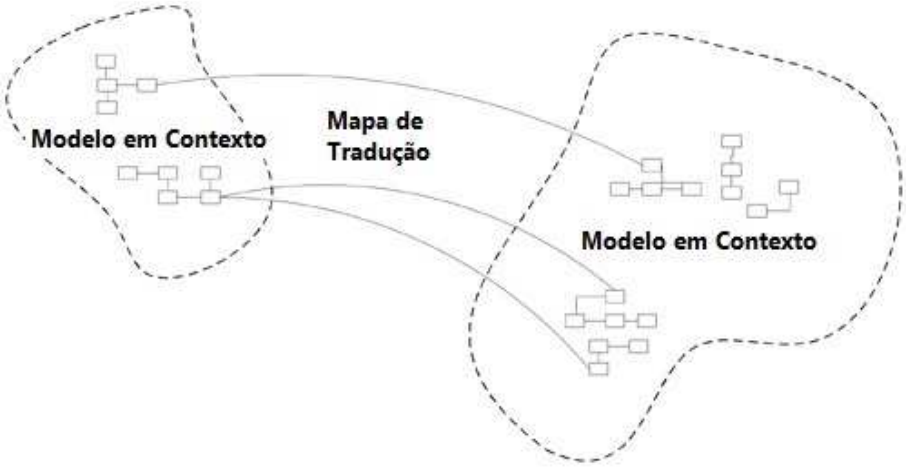
3.3.1. Contexto Delimitado

O Contexto Delimitado define claramente até onde o modelo do domínio tem seu escopo. Um projeto pode ter um modelo muito grande e com jargões semelhantes para cenários diferentes. Para que o modelo não sofra com uma grande carga de informação e torne-se demasiadamente complexo, a definição do Contexto Delimitado de um modelo é essencial para compreender o que está sendo abordado (EVANS, 2010, p. 321).

3.3.2. Mapa Contextual

O Mapa Contextual auxilia as equipes responsáveis em seus subsistemas, assim como no projeto de um novo subsistema, na compreensão do limite de seu escopo. Auxilia de forma visual a compreender a separação e a associação entre Contextos Delimitados (EVANS, 2010, p. 329), como ilustrado na Figura 10.

Figura 10 - Mapa Contextual



Fonte (EVANS, 2010, p. 329)

3.3.3. Relações entre Contextos Delimitados

As relações entre os Contextos Delimitados podem ser definidas como padrões de integração de subsistemas e equipes, de maneira a esclarecer como é tratada a abordagem de integração no nível estratégico (EVANS, 2010, p. 337).

Nas descrições que seguem, os termos “mais acima” e “mais abaixo”, relativos a contextos delimitados, significam que o contexto mais abaixo depende do mais acima. Em outras palavras, uma modificação no contexto mais acima desconsidera o contexto mais abaixo, podendo gerar efeito colateral neste último.

As relações constituem um mapa de traduções entre os elementos dos domínios.

3.3.3.1. Contexto Delimitado Único

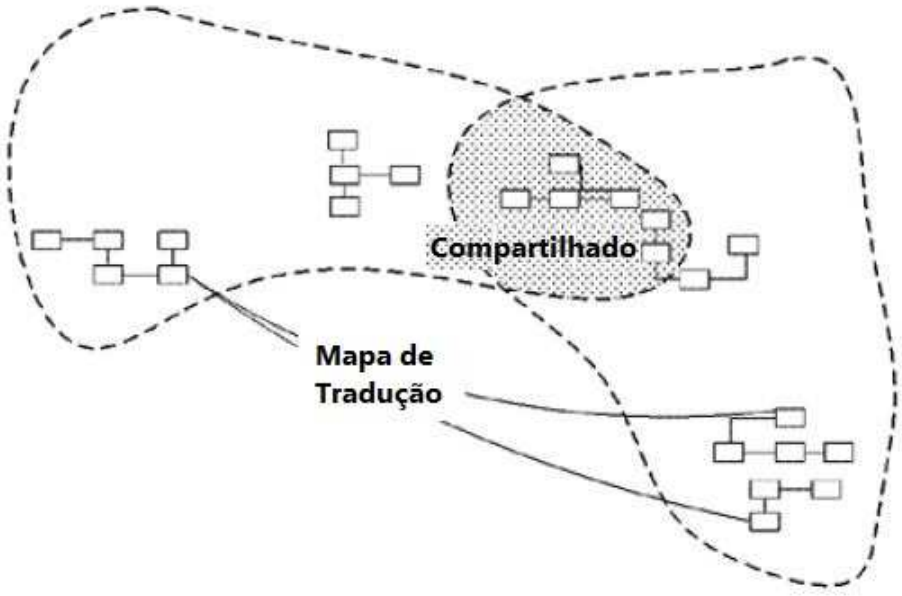
O Contexto Delimitado Único é aplicado quando a Linguagem Onipresente é única em todos os envolvidos e a solução é uma só. É um padrão que requer cuidados, pois poderá ocorrer alguma tradução no seu contexto e isto pode mostrar a necessidade da separação de um contexto distinto. O entendimento do sistema é um só e não de integração com qualquer outro contexto para satisfazer seu propósito (EVANS, 2010, p. 326).

3.3.3.2. Núcleo Compartilhado

O Núcleo Compartilhado (EVANS, 2010, p. 338) é utilizado quando dois ou mais Contextos Delimitados têm parte da Linguagem Onipresente igual dentre os envolvidos, cujas diferenças não justificariam um novo Contexto Delimitado completo. O núcleo é mantido e os dois contextos o utilizam. Um contexto pode ser dependente do outro ou ambos dependem de um, caracterizando-o como o domínio principal.

A Figura 11 ilustra como dois contextos compartilham o mesmo núcleo.

Figura 11 - Núcleo Compartilhado



Fonte (EVANS, 2010, p. 338)

Um exemplo é uma camada de infraestrutura compartilhada entre Contextos Delimitados.

3.3.3.3. Cliente / Fornecedor

O padrão Cliente/Fornecedor (EVANS, 2010, p. 340) é definido por “contexto acima” e “contexto abaixo”. Alterações no contexto mais acima devem ser informadas ao contexto mais abaixo, e um requisito do contexto abaixo deve ser solicitado ao contexto acima. Possuem linguagens diferentes, mas com forte dependência entre eles. Pode ocorrer quando o público alvo destes dois contextos é diferente, mas dependentes. A relação entre as equipes de desenvolvimento pode ser levada em consideração, se a gestão das equipes for distinta.

Por exemplo, suponha-se que já exista um sistema de transporte de cargas e a empresa deseja construir um sistema de análises que permitam o melhor aproveitamento do espaço de navios com as cargas. O sistema de reserva de cargas já está desenvolvido e um novo sistema de análise precisa utilizar o domínio existente, construído com tecnologias distintas. O novo sistema de análise seria um cliente que utiliza o sistema de reservas em sua linguagem como fornecedor.

3.3.3.4. Serviço de Host Aberto

Ao expor seu Contexto Delimitado a diversas integrações, um esclarecimento da utilização pode ser necessário e a disponibilização de serviços específicos são fornecidas para possíveis integrações (EVANS, 2010, p. 357). O contexto em desenvolvimento é quem proverá serviços a outros contextos para integração.

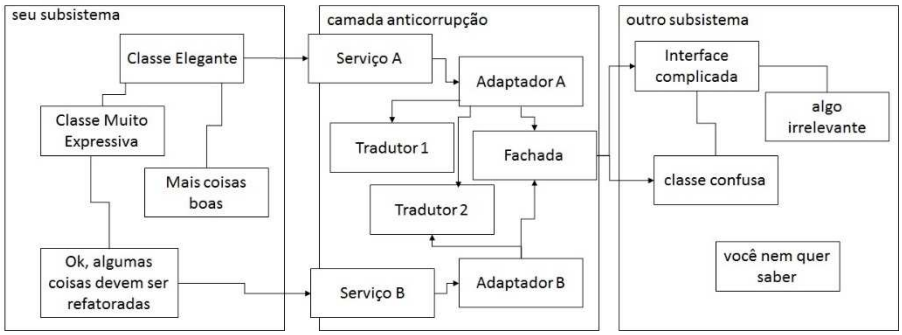
3.3.3.5. Linguagem Publicada

A Linguagem Publicada (EVANS, 2010, p. 358) é a disponibilização de um Contexto Delimitado de forma bem documentada para que a integração não necessite ter traduções, mas que saiba exatamente como utilizar a outra parte.

3.3.3.6. Camada Anticorrupção (Anticorruption Layer – ACL)

A camada anticorrupção pode ser utilizada quando não há o controle do Contexto Delimitado que está sendo trabalhado e outro contexto a ser integrado. Quando o Contexto Delimitado que está sendo trabalho prevê integração com outro contexto que não se tem controle, ou é um legado de difícil manutenção e a integração é necessária, pode ser criada uma camada de tradução que proteja o contexto utilizando a Linguagem Onipresente, chamada de ACL (*Anticorruption Layer*) (EVANS, 2010, p. 348). A ACL é utilizada para que a interpretação da integração seja clara para o Contexto Delimitado em construção/integração, de modo independente de como o outro foi desenvolvido. A Linguagem Onipresente na ACL ainda é a do Contexto Delimitado trabalhado, mas com adaptações / implementações para integrar com o outro contexto. A Figura 12 ilustra as linhas gerais deste padrão.

Figura 12 - Camada Anticorrupção



Fonte (EVANS, 2010, p. 351)

Por exemplo, para desenvolver um novo aplicativo de reservas que se comunica com um sistema legado, deve-se construir uma camada de tradução entre o novo aplicativo e o sistema legado para não contaminar o novo domínio em desenvolvimento (EVANS, 2010, p. 353).

3.3.3.7. Conformista

Quando um contexto “mais abaixo” não pode contar com alterações do contexto “mais acima” e há forte dependência do “mais abaixo” com o “mais acima” bem projetado, a abordagem do Conformista (EVANS, 2010, p. 345) seria a mais indicada. O contexto “mais abaixo” realiza uma forte integração, inclusive na Linguagem Onipresente, com o contexto “mais acima”. Deve ser usado com muita cautela, pois alterações no contexto “mais acima” não seriam mais previstas e corromperiam por completo o contexto “mais abaixo”.

3.4. Projeto Tático

O Projeto Tático no DDD implementa o modelo estratégico para a visão do projeto já mais próximo da implementação. Fornece conceitos para o entendimento profundo do domínio para permitir a implementação mais clara do modelo. Os conceitos no projeto tático auxiliam o desenvolvedor a refinar seu grau de conhecimento para o fortalecimento do domínio implementados no software.

3.4.1. Entidade

Entidade é um modelo que representa algo distinguível por uma identidade. Mesmo que através do ciclo de vida ele se transforme, ele deve permanecer único e rastreável. A identidade única deve corresponder à mesma distinção na vida real, em

que atributos podem ser iguais, mas de maneira que se possa identificar unicamente um elemento.

Por exemplo, tome-se uma transação bancária. Uma transação bancária tem um identificador único para diferenciar, por exemplo, dois depósitos ocorridos no mesmo dia, na mesma conta da mesma origem. Para diferenciá-los, possuem um identificador e neste domínio seria uma Entidade (EVANS, 2010, p. 84).

3.4.2. Objeto Valor

São objetos que não possuem uma identidade, ou seja, não precisam ser identificáveis, mas trazem um significado ao domínio. Podem ser copiados dentro do sistema, desde que se tenham as devidas precauções para que a alteração em uma referência seja aplicada a outros objetos. Por exemplo, uma Entidade que tenha uma referência para um Objeto Valor.

Como exemplo, seguindo com a transação bancária, enquanto a TransacaoBancaria seria uma Entidade, uma propriedade do tipo Dinheiro seria um Objeto Valor, uma vez que o dinheiro não precisa de um identificador, mas representa algo importante no domínio como representação monetária (EVANS, 2010, p. 92).

3.4.3. Serviços

3.4.3.1. Serviço de Domínio

O Serviço de Domínio (EVANS, 2010, p. 100) é normalmente responsável por executar uma tarefa, ou um conjunto de tarefas, que tem um significado no domínio não pertinente a uma Entidade ou Objeto Valor. Separa uma ação que não teria sentido no modelo de objetos para uma abstração separada, que representa uma ação da Linguagem Onipresente. Não possui um estado e se relaciona com o domínio.

Por exemplo, um serviço que possa transferir fundos de uma conta para outra em um contexto bancário.

3.4.3.2. Serviço de Aplicação

Os Serviços de Aplicação (EVANS, 2010, p. 101) e (FOWLER, *et al.*, 2002, p. 133-134), definem uma fronteira da aplicação com um conjunto de serviços que define operações disponíveis e coordena a resposta da aplicação a cada operação. Ela abstrai a implementação do contexto delimitado, isolando e protegendo a integridade

do modelo de domínio. Por exemplo, os serviços de aplicação realizam o comportamento de casos de uso coordenando a execução da lógica de domínio do ponto de vista transacional, segurança, etc.

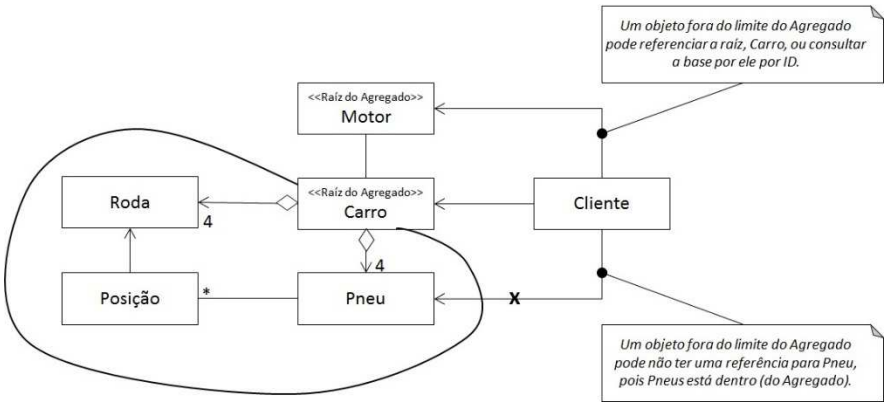
3.4.4. Módulo

Um Módulo (EVANS, 2010, p. 104) é um agregador de classes que podem juntas fazer algum sentido. Com a aplicação do DDD, um Módulo pode tornar-se parte da definição do domínio, tornando-se referência na Linguagem Onipresente. A separação de um contexto de um conjunto de classes pode fisicamente estar separada em Módulos, permitindo melhor compreensão do que faz parte ou não do domínio e como o domínio é ligado a outras partes.

3.4.5. Agregado

O Agregado (EVANS, 2010, p. 119) é um conjunto de objetos que juntos têm forte significado ao domínio. Possui uma Entidade raiz e outras Entidades ou Objetos de Valor associados que, sozinhos não teriam significado para o domínio. O Agregado deve ter um limite de associações bem definido e não podem ser acessados a não ser pela Entidade raiz. Auxilia a manter objetos inalterados além do seu objeto raiz, a fim de defender o modelo de alterações simultâneas no sistema.

Figura 13 - Agregado



Fonte (EVANS, 2010, p. 121)

Por exemplo, a Figura 13 ilustra as relações entre as classes que compõem um Carro. Como Carro é que contém as classes Roda, Pneu e Posição, então, neste contexto, estas partes não fariam sentido sem o Carro. Portanto, Carro é a raiz do Agregado. Outro exemplo é um Motor. Ele é constituído por várias partes, sendo, portanto, a raiz

de um Agregado. Assim definidos, esses dois agregados podem ter uma associação um com o outro, sempre relacionados por suas raízes.

3.4.6. Fábrica

As Fábricas (EVANS, 2010, p. 129) servem para montar conjuntos de objetos complexos, inclusive Agregados, a fim de remover a complexidade das associações de objetos. Deixando a responsabilidade técnica a cargo da fábrica, o código cliente não sofrerá com o aumento do alto acoplamento e baixa coesão.

Por exemplo, a construção de um Agregado Carro pode ser muito complexa para que um cliente tenha tanta responsabilidade em criar este Agregado. Então, pode-se criar uma FabricaCarro, responsável por montar o carro, motor, pneu e todas as dependências que crie um Agregado consistente.

3.4.7. Repositório

O Repositório (EVANS, 2010, p. 140) encapsula a complexidade da utilização da infraestrutura de dados para não enfraquecer o modelo do domínio. Se para a reconstituição de um objeto são necessárias diversas consultas à base de dados, *frameworks* de recuperação podem ser utilizados, assim como a utilização de Fábricas, para remontar Agregados. A responsabilidade da entrega de um modelo persistido em Agregado é do Repositório. Por exemplo, um RepositorioPedidoComercial representa a coleção de PedidoComercial.

4. ROTEIRO PARA UTILIZAÇÃO DA ARQUITETURA HEXAGONAL

Propõe-se aqui um roteiro para a modelagem do domínio, aplicando os conceitos do Projeto Estratégico e Projeto Tático de (EVANS, 2010) utilizando a Arquitetura Hexagonal como base de padrão arquitetônico como solução de projeto.

Este trabalho não tem foco em processos de desenvolvimento de software, mas sim nas atividades sequenciais para se chegar no projeto detalhado de um modelo de software.

4.1. Resumo da pesquisa bibliográfica

Da pesquisa bibliográfica utilizada nos capítulos 2 e 3 pode-se dizer que, em resumo, Evans considera que essencialmente é necessário avaliar o Mapa Contextual, a Linguagem Onipresente, o Domínio Principal, o Projeto (*Design*) Dirigido por Modelos, além de outros dois pontos que envolvem equipes (EVANS, 2010, p. 464). No entanto, não há explicitamente a descrição para aplicar o DDD, particularmente, à arquitetura. Mostra-se mais claramente como lidar com a questão arquitetônica através de dois estilos focalizados em equipes: um que não possui um projeto/padrão de arquitetura antes de iniciar o desenvolvimento; e outro com a equipe de arquitetura focalizada no cliente; isto é, uma equipe centralizada que possa ajudar às demais equipes (EVANS, 2010, p. 465).

4.2. Roteiro proposto para a utilização da Arquitetura Hexagonal

O roteiro apresentado a seguir propõe, em primeiro lugar a análise do contexto do sistema a ser projetado antes do início do projeto detalhado. Tendo a visão do contexto do sistema, é possível definir-se uma estratégia do relacionamento entre sistemas para evitar, ou minimizar, problemas de integração arquitetônica (JAIN, *et al.*, 2008). Somente com a visão estratégica definida é que se deve avançar nos Projeto Estratégico e Projeto Tático do DDD.

4.2.1. Definição do domínio

Para a definição do modelo de domínio, (EVANS, 2010) inicia seu livro apresentando o Projeto Tático. Na Parte IV – Projeto (*Design*) Estratégico do livro, descreve-se o Contexto Delimitado, o Mapa Contextual e o emprego da Linguagem Onipresente para a relação entre contextos. Também, é nesta parte que descreve seu ponto de vista sobre arquitetura de software.

Para a definição da Linguagem Onipresente, é necessário entender o contexto que está sob análise. Através dos Contextos Delimitados e suas relações pelo Mapa Contextual, é possível compreender sob qual escopo de negócio o sistema está em projeto, e então definir a Linguagem Onipresente. Portanto, para a utilização do DDD requer-se, inicialmente, desses elementos. Assim, tem-se o Passo 1: Definição dos Contextos Delimitados e da Linguagem Onipresente.

4.2.2. Projeto de alto nível

Com o Mapa Contextual elaborado no primeiro passo, é possível avaliar os padrões de integração entre Contextos Delimitados. A avaliação de integração somente é possível após ter-se em mãos os Contextos Delimitados no Mapa Contextual, utilizando a Linguagem Onipresente. Este é o nível do Projeto Estratégico de Evans.

Assim, tem-se o Passo 2: Projeto Estratégico – Integração.

Ainda no Projeto Estratégico, a escolha de uma arquitetura deve levar em consideração como uma empresa trabalha: com ou sem uma equipe de arquitetura focalizada. Para este trabalho, a abordagem selecionada pelas características do domínio utiliza a Arquitetura Hexagonal (HA/PA) no DDD, adotando-se a abordagem de definir-se uma arquitetura antes do desenvolvimento (equipe de arquitetura focalizada). A HA/PA foi a arquitetura escolhida por enfocar a resolução de problemas de domínio, de diversidade de dispositivos que interagem com o domínio-alvo e de integração. Do ponto de vista do DDD, as integrações entre Contextos Delimitados são solucionadas através dos Padrões Estratégicos. A flexibilidade dessa arquitetura fornece uma solução para integrações no mesmo modelo de domínio entre vários contextos e dispositivos que requerem padrões de integração distintos.

O modelo do domínio é centralizado no hexágono, que pode ser testado através de testes automatizados, conectado a novos dispositivos que venham a utilizá-lo, assim como integrá-lo a outros contextos.

Uma vez que o desenvolvedor compreende a abrangência da HA/PA, uma visão arquitetônica diferenciada é dada antes mesmo de o projeto ser iniciado, levando-o a focalizar primeiro na interpretação do domínio e depois nas possíveis integrações entre sistemas. Outros pontos técnicos e de integração podem ser resolvidos através das portas e adaptadores, uma vez que o domínio esteja íntegro, pois todos os lados

do hexágono utilizarão os mesmos critérios para tais integrações. Um benefício imediato disso é o de potencialmente poderem contar com especialistas em tecnologia de integração que não conheçam bem o domínio, mas que precisam atuar focalizados em um lado do hexágono (uma integração específica ou outras similares).

Após a avaliação da integração entre contextos, pode-se então definir o Passo 3: Projeto Estratégico – Arquitetura Hexagonal – Definição das Portas.

4.2.3. Projeto Detalhado

Como a visão arquitetônica definida, pode-se então partir para o Projeto Detalhado, para refinar a visão do sistema mais próxima da implementação. Como se tem em mãos as definições das portas da HA/PA, pode-se avaliar a integração detalhada e identificar os módulos do sistema. Neste nível de abstração, utilizando a Linguagem Onipresente, identifica-se o Passo 4: Projeto Tático – Integração.

Com o projeto tático de integração entre módulos, pode-se refinar e identificar as classes através do Passo 5: Projeto Tático – Detalhamento dos Módulos.

4.3. Definição dos passos

Dados os conceitos do DDD e da Arquitetura Hexagonal (HA/PA), e as considerações expostas no tópico anterior, propõe-se o roteiro que segue para a utilização da Arquitetura Hexagonal como solução para os problemas descritos. As seções que seguem detalham os passos do roteiro.

- **Passo 1: Definição dos Contextos Delimitados e da Linguagem Onipresente**
- **Passo 2: Projeto Estratégico – Integração**
- **Passo 3: Projeto Estratégico – Arquitetura Hexagonal – Definição das Portas**
- **Passo 4: Projeto Tático – Integração**
- **Passo 5: Projeto Tático – Detalhamento dos Módulos**

4.3.1. Passo 1: Definição dos Contextos Delimitados e da Linguagem Onipresente

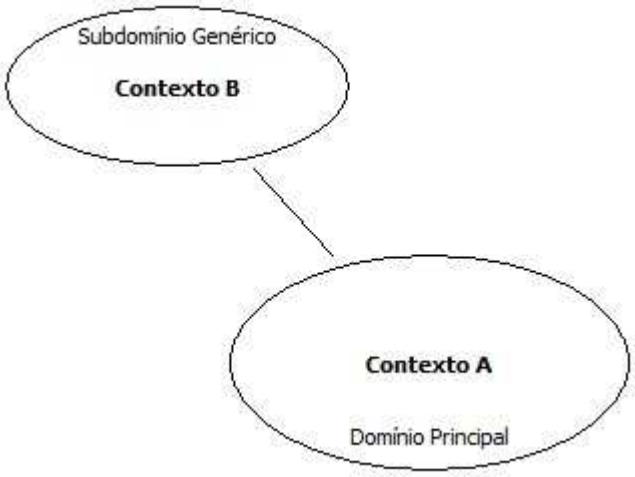
Entrada:

- Descrição do domínio.

Saída:

- Descrição dos Contextos Delimitados;
- Modelo conceitual com a Linguagem Onipresente.
- Mapa Contextual do Domínio;

Figura 14 - Mapa Contextual de exemplo



Fonte (o autor)

Descrição: Descrever o domínio em análise para a compreensão dos contextos envolvidos. Desta maneira, revelam-se os sistemas que potencialmente correspondem a estes contextos, gerando o Mapa Contextual do domínio, como ilustrado um exemplo na Figura 14. O Mapa Contextual contém a indicação do domínio que está sendo projetado, identificando-o como “Domínio Principal”. Os demais contextos que serão utilizados são identificados como “Subdomínio Genérico”. Uma vez identificados outros contextos, é necessário descrevê-los para que posteriormente se pense na provável integração entre eles. Basta um modelo conceitual dos contextos envolvidos para compreender seus limites.

4.3.2. Passo 2: Projeto Estratégico – Integração

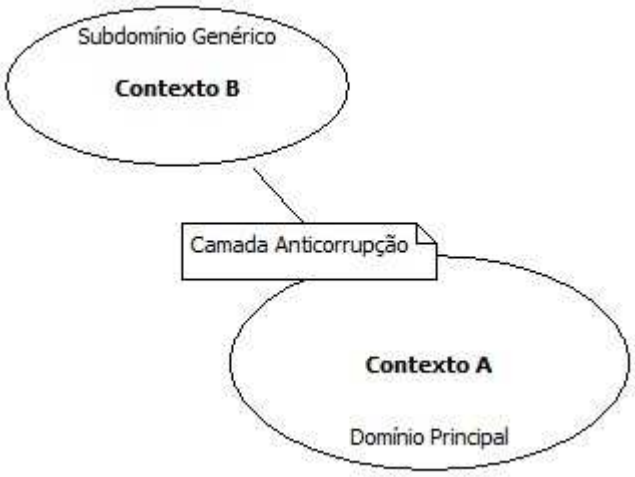
Entrada:

- Descrição dos Contextos Delimitados;
- Mapa Contextual do Domínio;
- Diagrama do modelo conceitual com a Linguagem Onipresente.

Saída:

- Mapa Contextual do Domínio com os Padrões Estratégicos de Integração.

Figura 15 - Mapa Contextual de exemplo com Padrão Estratégico de Integração



Fonte (o autor)

Descrição: Com a identificação dos Contextos Delimitados envolvidos, principalmente em projeção, verificar qual padrão estratégico de integração é mais adequado nos pontos de integração, como visto em 3.3.3. Indicar o Padrão Estratégico de integração adotado nas associações entre os Contextos Delimitados dentro do Mapa Contextual elaborado no Passo 1. A Figura 15 ilustra a utilização do padrão Camada Anticorrupção (ACL).

4.3.3. Passo 3: Projeto Estratégico – Arquitetura Hexagonal – Definição das Portas

Entrada:

- Mapa Contextual do Domínio com Padrão Estratégico de Integração.

Saída:

- Projeto da Arquitetura Hexagonal com definição das portas.

Descrição: Uma vez definidos os padrões estratégicos de integração, as portas conceituais da HA/PA têm relação direta com as estratégias pré-definidas. Identificar outras portas de acesso ao domínio que podem estar associadas apenas ao contexto em projeto. O Projeto da HA/PA faz parte do Projeto Estratégico.

4.3.4. Passo 4: Projeto Tático – Integração

Entrada:

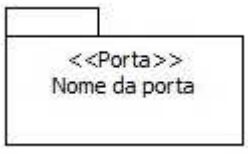
- Mapa Contextual do Domínio com Padrão Estratégico de Integração
- Projeto da Arquitetura Hexagonal com definição das portas.

Saída:

- Modelo de Integração entre Módulos;
- Modelo da Integração entre módulos com as Portas conceituais da HA/PA.

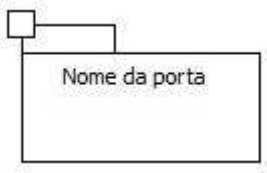
Descrição: Com o projeto da HA/PA e suas portas definidas, iniciar a projeção em diagramas UML, avaliando os Módulos nomeados pela Linguagem Onipresente, como cada Módulo é integrado no seu domínio, assim como na aplicação (interface de usuário, base de dados, etc.).

Figura 16 - Estereótipo da Porta conceitual



Fonte (o autor)

Figura 17 – Notação da Porta conceitual



Fonte (o autor)

Para representar as portas da HA/PA no modelo, o estereótipo <<Porta>> é colocado na representação do módulo (pacote) da UML, e dentro dele os módulos físicos projetados, como na Figura 16. Para facilitar a identificação da representação gráfica de uma Porta, a Figura 17 ilustra a substituição do estereótipo <<Porta>> pela marcação de um quadrado no topo do pacote do projeto.

4.3.5. Passo 5: Projeto Tático – Detalhamento dos Módulos

Entrada:

- Diagrama de Integração entre Módulos;
- Diagrama do modelo conceitual com a Linguagem Onipresente;
- Descrição do Contexto Delimitado;

Saída:

- Detalhamento dos Módulos

Descrição: Com os Módulos inicialmente definidos, aplicar o projeto tático do DDD com as Entidades, Objetos Valor, Serviços de Domínio, Repositórios e outros conceitos táticos necessários. Utilizar os padrões de projeto detalhado nos módulos definidos.

5. UM EXEMPLO DE APLICAÇÃO DA ARQUITETURA HEXAGONAL

Neste capítulo será aplicado o roteiro do capítulo 4 em uma parte do projeto arquitetônico de um software para uma empresa real. Apresenta-se uma descrição do domínio do problema e aplica-se o roteiro proposto apresentando os resultados em diagramas e descrições como saída de cada passo do roteiro.

5.1. Descrição do domínio do problema

Uma empresa de pesquisa de mercado, chamaremos de Research Corp por motivos confidenciais, deseja um sistema que permita realizar entrevistas com diversas pessoas para gerar informação de consumo para seus clientes. A Research Corp é especialista em controle de amostra, sistema que controla a quantidade de indivíduos necessários para representar um determinado perfil demográfico. A Research Corp já possui um sistema que realiza a gestão de indivíduos candidatos a participar de uma pesquisa e seleciona aleatoriamente, dado um filtro demográfico, um grupo de indivíduos para responder a uma “onda” de questionário.

Por onda entende-se a aplicação de um mesmo questionário (as mesmas perguntas), por certo período de tempo, em indivíduos selecionados para representar os perfis demográficos.

A cada onda, um questionário diferente é elaborado para ser utilizado. Ao receber as respostas, é feita uma consolidação de dados para a geração de relatórios com informações de consumo e características do público que participou da onda.

A Research Corp também possui um sistema para gerenciamento de questionários, mas toda entrega de questionários, coleta das respostas e transcrição de resultados é feita por uma empresa terceira.

A Research Corp patrocinou um projeto de software para ela mesma ter o sistema de pesquisa e consolidação de dados para entregar aos seus clientes. A Research Corp também está interessada em realizar as entrevistas em diferentes formas para permitir mais opções de resposta dos indivíduos da amostra, pois há pessoas de classes sociais mais baixas que não possuem recursos de responder digitalmente. Algumas possibilidades de se fazer entrevistas serão descritas, mas a empresa deseja flexibilidade para adicionar novas maneiras de se realizar as entrevistas.

De acordo com essa descrição, pode-se definir o seguinte escopo para o projeto de software:

1. Possibilidades de realização da entrevista:
    a. Entrevistador que bate de porta-em-porta para deixar um questionário em papel e recolher após certo período;
    b. Entrevistador com um *tablet* realizando entrevista face-a-face com o entrevistado;
    c. Entrevistado responder via *website*.
2. Permitir o controle das ondas e seus questionários.
3. Consolidar as respostas dos questionários.

5.2. Passo 1: Definição dos Contextos Delimitados e Linguagem Onipresente

Para compreender melhor a divisão do escopo e quais contextos estão envolvidos, o Mapa Contextual descreve como os contextos são delimitados e associados através do novo domínio a ser projetado. Segue a descrição dos conceitos dos contextos em sua Linguagem Onipresente.

5.2.1. Descrição dos Contextos Delimitados

5.2.1.1. Contexto Amostragem

O contexto Amostragem gerencia os indivíduos que podem participar de pesquisas para a empresa, inclusive para responder pesquisas via questionários. A amostra (indivíduos que estão registrados na listagem da empresa) não é exclusiva para seu próprio sistema, Amostragem, mas pode ser utilizada para diversos fins. Outros sistemas que precisem dela podem utilizar sua porta de comunicação padrão, através de serviços *web* (*web-service*). Não será detalhado como indivíduos são convidados a fazer parte da amostra, mas, uma vez que concordem em participar dela, eles são registrados neste sistema e representados no projeto conceitual. Este sistema é encarado como um legado, pois não haverá alterações de projeto.

Modelo conceitual

O modelo conceitual contém as classes que representam o sistema de amostra de modo simplificado. Para este trabalho, o conceito mais importante é o indivíduo que faz parte da amostra, que possui características que representarão outras pessoas

estatisticamente. Este modelo é o de um Agregado, pois suas partes só têm sentido quando unificados, como apresentado na Figura 18.

Figura 18 - Agregado Amostragem

Fonte (o autor)

Linguagem Onipresente

- Amostragem: um conjunto de indivíduos que juntos representam um universo de pesquisa;
- Indivíduo: uma pessoa que faz parte da amostragem;
- Variável demográfica: informação que descreve algo que represente um perfil, por exemplo, gênero, região onde mora, classe social;
- Peso: a quantidade de pessoas representadas através de um indivíduo.

Responsabilidades do Contexto Delimitado

- Gerenciar indivíduos que participam da amostra;
- Controlar indivíduos que participarão de uma amostra a partir de um período solicitado;
- Realizar cálculos de ponderação para o peso dos indivíduos.

5.2.1.2. Contexto Gerenciamento de Questionários

Para que uma pesquisa seja feita, é necessário ter um questionário definido. O contexto Gerenciamento de Questionários foi projetado para a responsabilidade de mesmo nome. Inicialmente, ele deveria controlar o andamento da pesquisa e gerar os resultados finais, mas, uma reflexão mais aprofundada revelou que o domínio para o gerenciamento de questionários é distinto daquele em que se aplica um questionário, com usos e usuários distintos.

Modelo conceitual

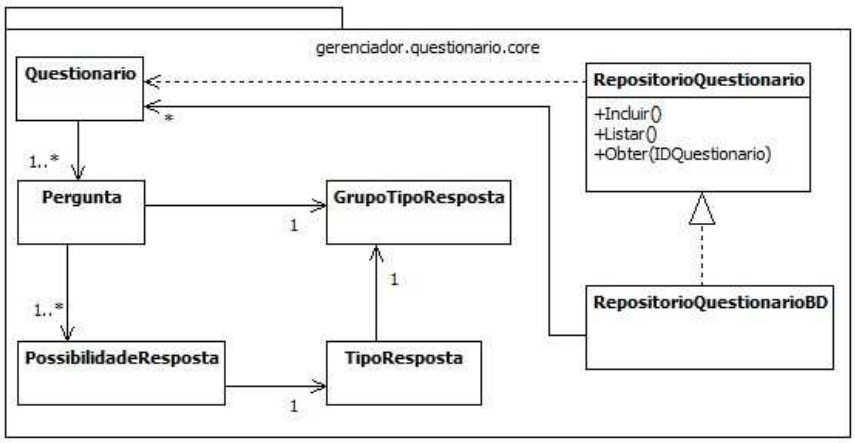
O modelo conceitual de um Questionário tem ao menos uma Pergunta com Possíveis Respostas atreladas a esta pergunta. As Possíveis Respostas têm as Possibilidades de Respostas, com um Tipo, e o Tipo pertence a um Grupo.

Por exemplo, um objeto *QuestionarioEleicao* possui uma *PerguntaQuantasTelevisoesHaEmCasa*. A própria *PerguntaQuantasTelevisoesHaEmCasa* já possui a associação com um *GrupoTipoRespostaUnica*. As possibilidades de *TipoRespostaUnica* são *PossibilidadeRespostaNenhuma*, *PossibilidadeRespostaUma*, *PossibilidadeRespostaDuas*, *PossibilidadeRespostaTresOuMais*.

O Questionário é armazenado no *RepositórioQuestionario*, que possui sua implementação *RepositorioQuestionarioBD*.

A implementação do Repositório está no mesmo Módulo. Possui a Linguagem Onipresente forte, mas alguns pontos de modularização não parecem ter tanta importância para a equipe de desenvolvimento deste sistema. A Figura 19 apresenta o modelo conceitual adotado.

Figura 19 - Agregado Questionário



Fonte (o autor)

Linguagem Onipresente

- Questionário: um conjunto de perguntas. Cada questionário é único, pois é necessário manter o histórico dos questionários já associados à onda para efeitos de auditoria;

- Pergunta: uma pergunta específica em um questionário;
- Possíveis Respostas: possíveis respostas para uma pergunta;
- Tipos de Respostas: Tipo de uma resposta para uma pergunta. Resposta única ou multi-seleção.

Responsabilidades do Contexto Delimitado

- Criar um questionário;
- Adicionar perguntas e definir seu tipo;
- Adicionar possíveis respostas em uma pergunta respeitando o tipo da pergunta;
- Fechar um questionário (fechamento) para indicar que ele está pronto para ser utilizado em uma onda.

5.2.1.3. Contexto Onda

O contexto Onda é o principal ponto a ser projetado nesta monografia. É o contexto quem cria o conceito de uma Onda, representando um período que um número determinado de indivíduos, que fazem parte da amostra, serão selecionados para responder um questionário num prazo determinado. Os indivíduos e suas respostas representarão estatisticamente um universo, permitindo que ao final do período de entrevistas as respostas sejam transformadas em informações significantes através de uma consolidação de dados.

Vocabulário

- Onda: período em que um questionário é utilizado para uma pesquisa;
- Indivíduo: pessoa que respondeu o questionário da Onda;
- Respostas: respostas das perguntas respondidas pelo indivíduo.

Requisitos do Contexto Delimitado

- Criar uma Onda num sistema administrativo:
    - Data de início e fim;
    - Perfis demográficos e quantidade de indivíduos necessários para atingir uma amostra especificada;
    - Questionário que será utilizado para a onda;
- Aplicar a Onda e seu questionário nos indivíduos através das plataformas de comunicação:

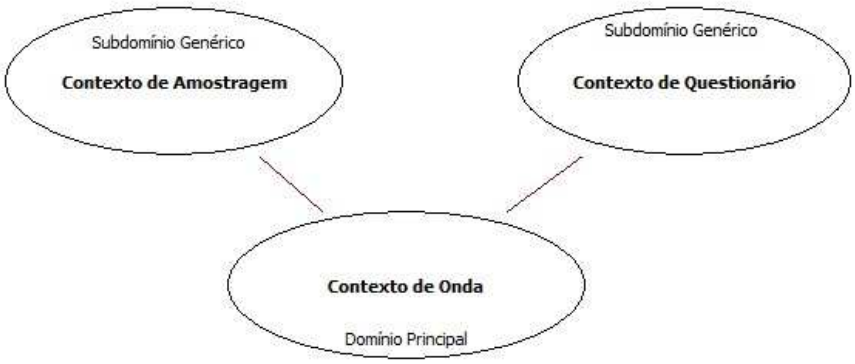
  - Entrevista face-a-face com um entrevistador utilizando um *tablet*;
  - Deixar o questionário impresso para buscar posteriormente;
  - Envio de *link* de questionário via e-mail para acesso e preenchimento via *web*;
- Controlar quantas entrevistas foram concluídas (respondidas por um entrevistado) e quantas ainda são necessárias;
- Fazer a projeção da quantidade de entrevistas necessárias por dia de acordo com o prazo de encerramento da onda e tamanho da amostra;
- Consolidar entrevistas no encerramento da onda e disponibilizar um relatório com os dados calculados.

5.2.2. Mapa Contextual do Domínio

O Mapa Contextual, como visto no Capítulo 3 e Passo 1 do Capítulo 4, auxilia visualmente como será abordado e como estão divididos os contextos derivados da descrição. A Figura 20 apresenta este mapa.

Identificados os Contextos Delimitados, a Onda, sendo o foco do projeto elaborado neste trabalho, está identificada como “Domínio Principal”. Os contextos Amostragem e Questionário serão utilizados pelo contexto Onda, sendo identificados como “Subdomínio Genérico”.

Figura 20 - Mapa contextual do domínio



Fonte (o autor)

5.3. Passo 2: Projeto Estratégico – Integração

Segue a utilização de Padrões Estratégicos nos Contextos Delimitados.

5.3.1. Onda-Amostragem

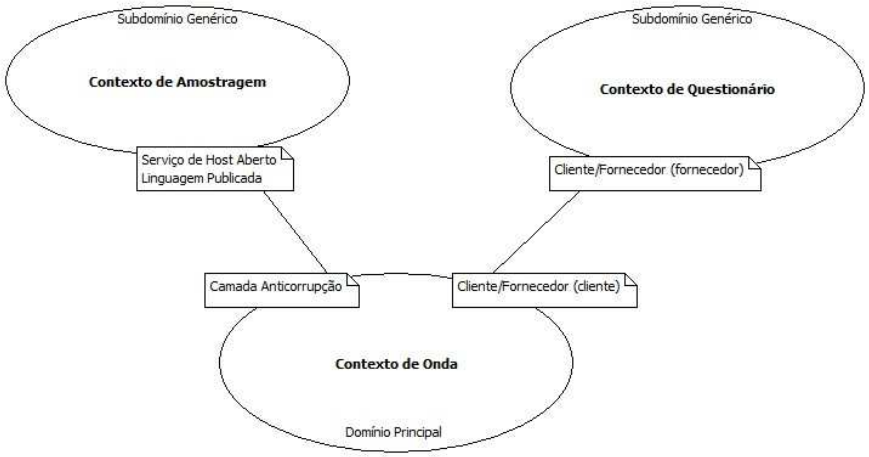
A integração entre os contextos Onda e Amostragem utilizará o padrão estratégico Camada Anticorrupção (visto em 3.3.3), do ponto de vista do Contexto Onda, pois o contexto Amostragem é um sistema legado existente na empresa que disponibiliza serviços para a integração com outros sistemas. Descartam-se alterações no sistema legado. A camada de tradução da linguagem será utilizada para não corromper o domínio principal (Contexto Onda).

5.3.2. Onda-Questionário

A integração entre os contextos Onda e Questionário utilizará o padrão estratégico Cliente/Fornecedor, pois haverá comunicação entre as equipes de desenvolvimento numa relação em que a equipe do contexto Questionário poderá ter que desenvolver funcionalidades para a equipe que desenvolve o contexto Onda. A integração é forte, mas não é o mesmo núcleo, pois o entendimento da montagem do questionário é diferente da utilização de um questionário pronto durante a aplicação da Onda. O contexto Questionário será fornecedor para o contexto Onda, cliente.

5.3.3. Mapa Contextual com os Padrões Estratégicos de Integração

Figura 21 - Mapa contextual de integração



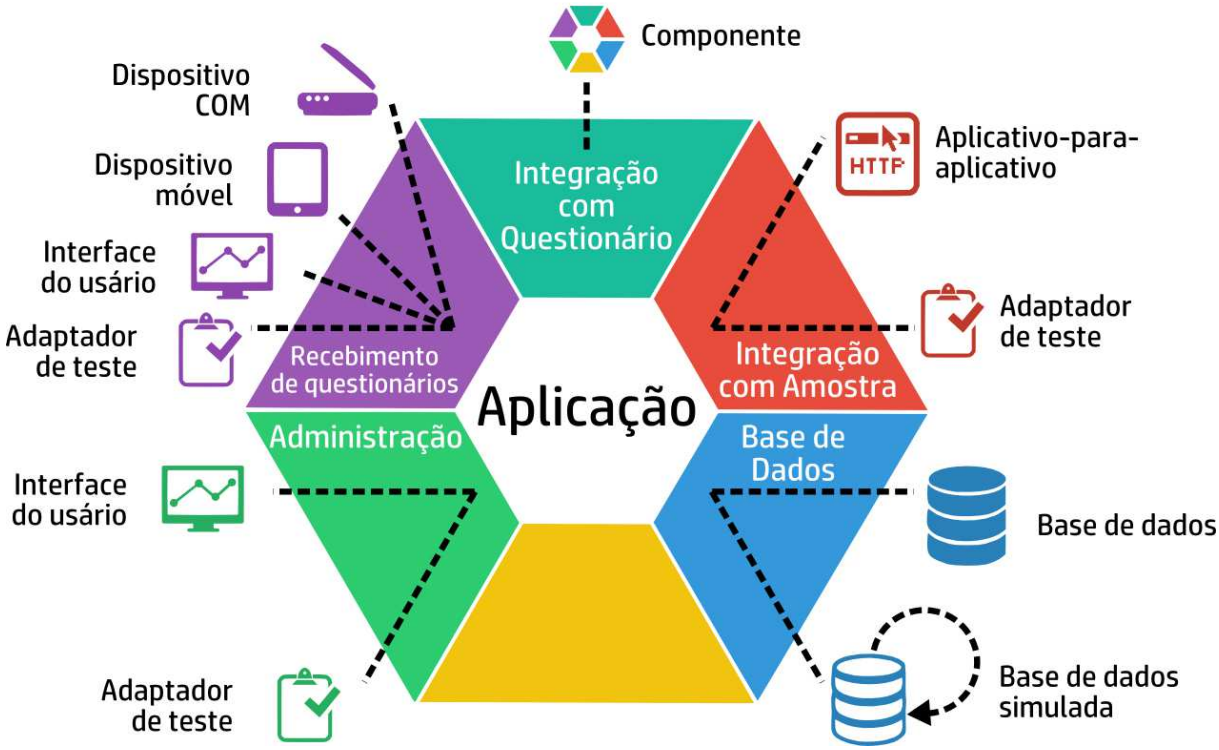
Fonte (o autor)

Após a descrição da relação entre os contextos Onda-Amostragem e Onda-Questionário, realizou-se a anotação das relações no Mapa Contextual, visualizado na Figura 21, como exemplificado em 4.3.2.

5.4. Passo 3: Projeto Estratégico – Arquitetura Hexagonal – Definição das Portas

A Figura 22 representa o projeto arquitetônico proposto para a HA/PA. Metaforicamente, cinco lados têm comunicação com o domínio por dispositivos distintos que potencialmente se integrarão com a “área interna” – a Aplicação – e com a “área externa” – outros domínios.

Figura 22 - HA/PA para o caso



Fonte (o autor)

Separando o domínio Onda, como proposto pelo DDD, com a HA/PA, o domínio fica livre das integrações e desconhece os dispositivos que o utilizarão, dando foco no Contexto Delimitado. As portas e adaptadores de apresentação, dados e integrações com dispositivos necessários para receber os questionários respondidos são desenvolvidos posteriormente.

5.5. Passo 4: Projeto Tático – Integração

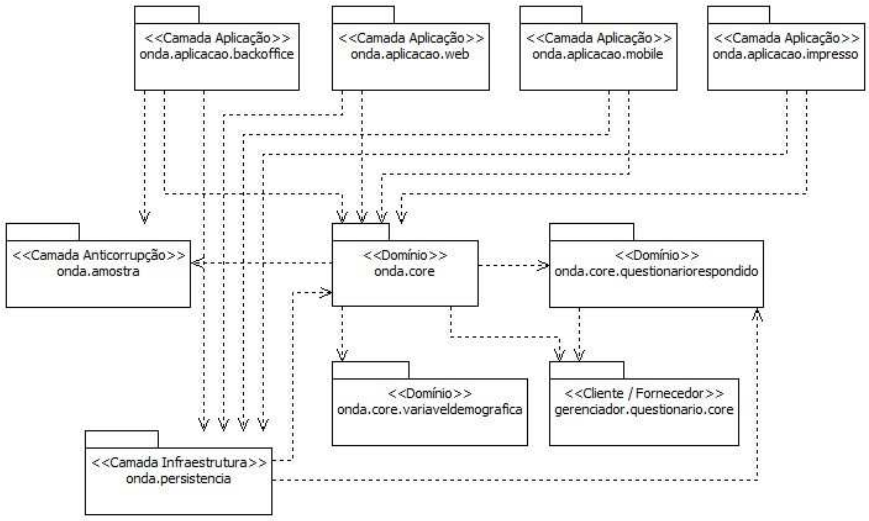
Utilizando a Arquitetura Hexagonal como base para comportar os detalhes do domínio e dar suporte às integrações posteriores, seja entre sistemas, ou para dispositivos, o projeto tático segue mais detalhado no núcleo para demonstrar sua importância e interdependências.

5.5.1. Modelo de Integração entre Módulos

Uma Onda, como descrito anteriormente, caracteriza-se principalmente por representar um período de tempo, coletando respostas de um único questionário para um conjunto de indivíduos.

Foi feita uma divisão em Módulos para minimizar o acoplamento e aumentar a coesão do domínio, representado na Figura 23.

Figura 23 - Integração entre Módulos



Fonte (o autor)

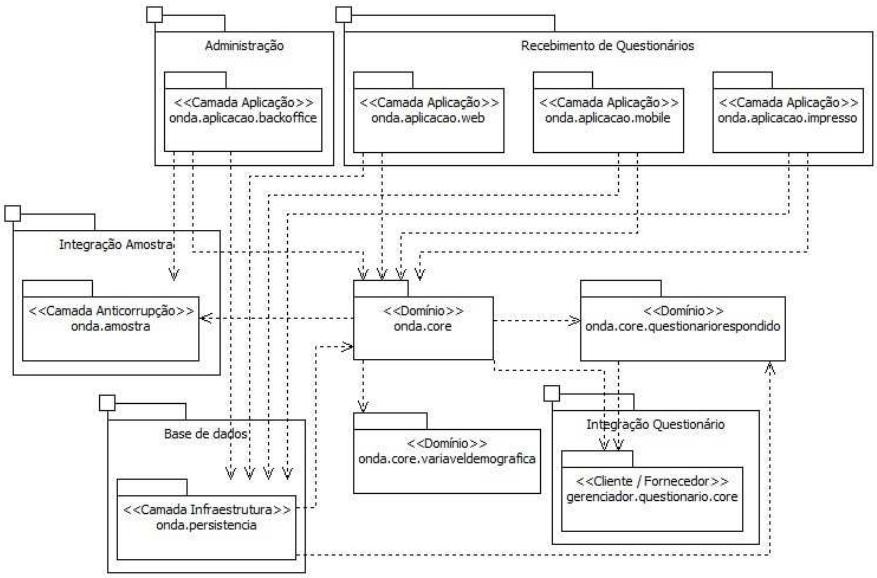
A anotação “Camada Aplicação” caracteriza os módulos em camadas de aplicação que se comunicam com dispositivos externos.

5.5.2. Modelo da Integração entre módulos com as Portas conceituais da HA/PA

Na integração entre Módulos pode-se verificar que ao centralizar o modelo de domínio, é possível estendê-lo com outros sistemas, permitindo seu uso por integrações que podem ser modificadas no futuro. Para adequá-lo à HA/PA, foi

adicionado ao modelo de integração entre Módulos as portas conceituais da arquitetura. A convenção utilizada para descrever as portas conceituais no diagrama é apresentada no Capítulo 4. A Figura 24 ilustra a integração entre módulos com a representação conceitual das portas da HA/PA.

Figura 24 - Integração entre Módulos com portas conceituais



Fonte (o autor)

5.6. Passo 5: Projeto Tático – Detalhamento dos Módulos

5.6.1. Utilização de padrões

Para este trabalho, foram utilizados alguns padrões detalhados de projeto. Para refinar a Arquitetura Hexagonal é necessário aumentar o nível de detalhe no projeto da solução.

5.6.1.1. Fábricas

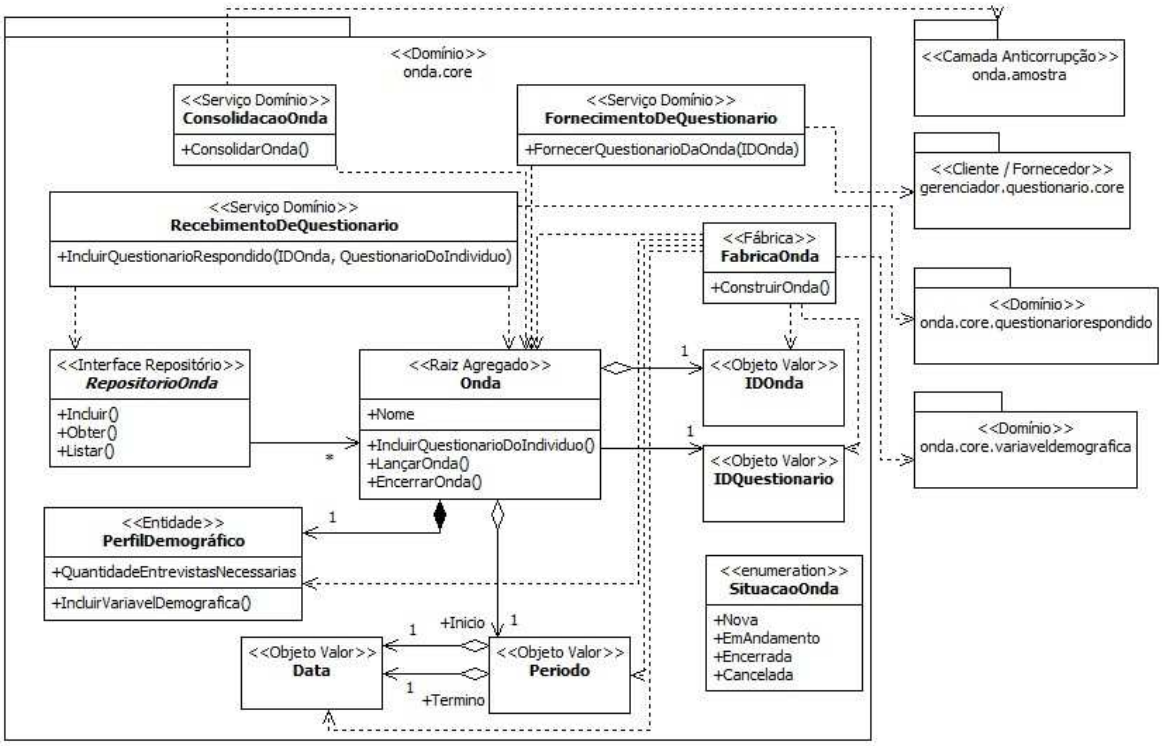
Além da definição feita no Capítulo 3, uma Fábrica cria uma raiz íntegra de um Agregado verificando as condições dos invariantes (o que é sempre verdadeiro). Caso a condição de um invariante seja violada, a Fábrica notificará um erro a quem a utiliza. As associações dos Agregados são verificadas pelas Fábricas. As Fábricas neste trabalho não são responsáveis pela remontagem de objetos já persistidos. Assume-se que isto é solucionado tecnicamente por *frameworks* de desenvolvimento, do tipo ORM (Mapeamento Objeto-Relacional).

5.6.2. Detalhamento dos Módulos

5.6.2.1. <<Domínio>> onda.core

O Módulo onda.core é o núcleo do contexto delimitado Onda e encontra-se no Agregado principal Onda, apresentado na Figura 25.

Figura 25 - onda.core



Fonte (o autor)

Seguem as descrições dos elementos do Módulo.

- <<Interface Repositório>> RepositorioOnda
    - Classe Abstrata de Repositório para armazenar (ainda não se trata da implementação, mas sim a representação das abstrações) o Agregado Onda. Para armazenar a Raiz Onda, o Repositório recebe o Agregado já criado pela FabricaOnda.
- <<Fábrica>> FabricaOnda
    - A classe Fábrica do DDD é responsável por criar a primeira instância do Agregado Onda e validar seus invariantes. Valida se a Data Início é menor que a Data Término.

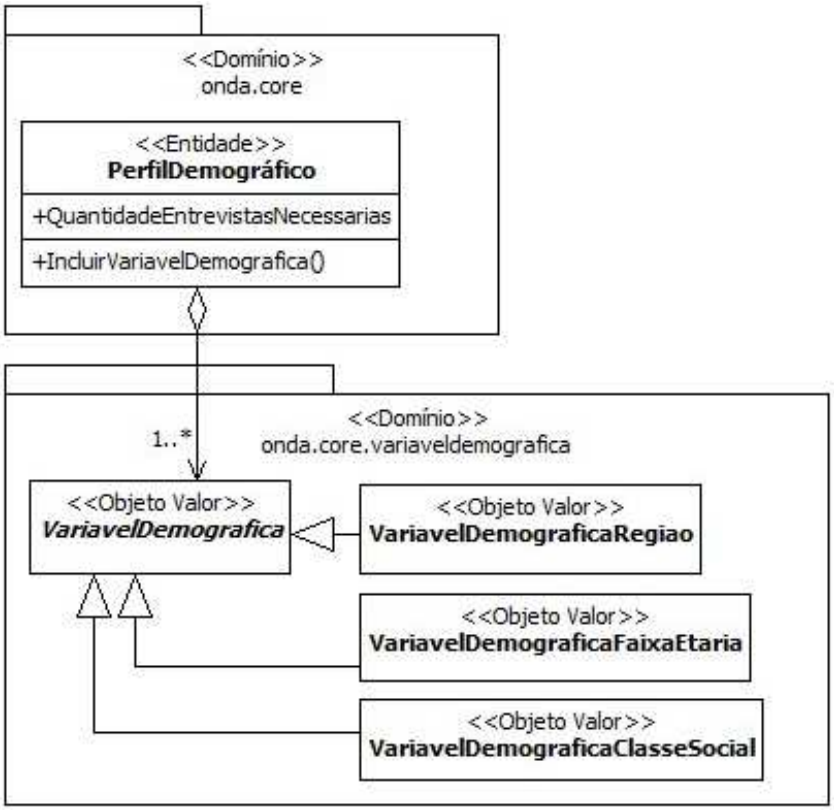
- <<Raiz Agregado>> Onda
    - A raiz do Agregado que modela a Onda do domínio, ou seja, um período específico e um questionário que representa uma pesquisa.
- <<Objeto Valor>> IDOnda
    - Objeto Valor para definir uma identidade para a Onda.
- <<Objeto Valor>> IDQuestionario
    - Um Objeto Valor para referenciar um Questionário específico do Contexto Delimitado Gerenciamento de questionários.
- <<Objeto Valor / Enumerador>> SituacaoOnda
    - Enumerador que tem as situações que uma onde pode conter:
        - Nova – ao concluir a criação da Onda, tem a situação Nova;
        - Em Andamento – Quando a data de início da Onda for a data calendário, a onda é considerada em andamento;
        - Encerrada – Quando o prazo atingir a data calendário, a onda é considerada encerrada;
        - Cancelada – Situação quando há a intervenção de um usuário interno da empresa e cancela a onda.
- <<Objeto Valor>> Periodo
    - Período em que a onda ocorre, utilizando a classe Data para ter Início e Fim.
- <<Objeto Valor>> Data
    - Classe que representa uma data (um dia do calendário).
- <<Entidade>> PerfilDemográfico
    - Entidade agregada à Onda para representar qual público alvo deverá ser atingido com a Onda para a realização de entrevistas.
- <<Serviço Domínio>> FornecimentoDeQuestionario
    - Serviço de domínio para disponibilizar o questionário associado à Onda para os Serviços de Aplicação. Como o Agregado Questionário encontra-se no outro sistema integrado pelo padrão estratégico Cliente/Fornecedor, o serviço utiliza o Repositório de questionário e seu Agregado para entregar o Questionário à aplicação.

- <<Serviço Domínio>> RecebimentoDeQuestionario
    - Serviço de Domínio que inclui um questionário respondido na Onda. Utiliza os Repositórios RepositorioOnda e RepositorioQuestionarioDoIndividuo para aplicar a persistência.
- <<Serviço Domínio>> ConsolidacaoOnda
    - Serviço de Domínio que realiza o procedimento de juntar o Agregado Onda e suas dependências, realizar validações e calcular a ponderação dos indivíduos que participaram (responderam) ao questionário, tornando-os representativos para o perfil demográfico definido na criação da Onda. Para realizar a consolidação, cálculos, são necessárias as informações do indivíduo que se encontram em outro sistema. Para isso, utiliza o serviço CalculoAmostral e o RepositorioAmostra na camada anticorrupção para se integrar com o outro sistema, Amostragem, e obter as informações dos indivíduos significantes para o contexto Onda.

5.6.2.2. <<Domínio>> onda.core.variaveldemografica

Módulo em que as generalizações da classe VariavelDemografica estão implementadas, representado na Figura 26.

Figura 26 - onda.core.variaveldemografica



Fonte (o autor)

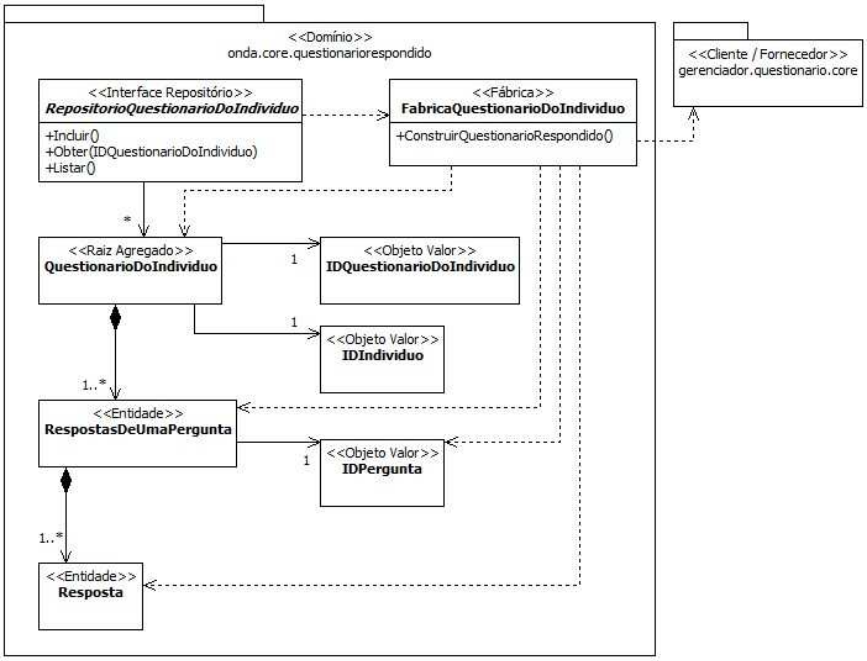
Seguem as descrições dos elementos do Módulo.

- <<Objeto Valor>> VariavelDemografica
  - A variável demográfica que faz parte do critério do público alvo.
- <<Objeto Valor>> VariavelDemograficaRegiao
  - Variável demográfica de uma região. Poderá ser de qualquer escala (país, estado, cidade, região metropolitana...). Outras generalizações podem ser acrescentadas se necessário.
- <<Objeto Valor>> VariavelDemograficaClasseSocial
  - Variável demográfica que caracteriza a classe social que será aplicada à Onda. Será o filtro na aplicação de Questionário para Indivíduo com as mesmas variáveis demográficas.
- <<Objeto Valor>> VariavelDemograficaFaixaEtaria
  - Variável demográfica que declara a faixa etária (faixa de idades) aceita na Onda para um indivíduo que responderá o Questionário.

5.6.2.3. <<Domínio>> onda.core.questionariorespondido

Módulo que contém as classes de um questionário respondido. Utiliza diretamente o Módulo (contexto) gerenciador.questionario.core utilizando o padrão estratégico Cliente/Fornecedor, como representado na Figura 27.

Figura 27 - onda.core.questionariorespondido



Fonte (o autor)

Seguem as descrições dos elementos do Módulo.

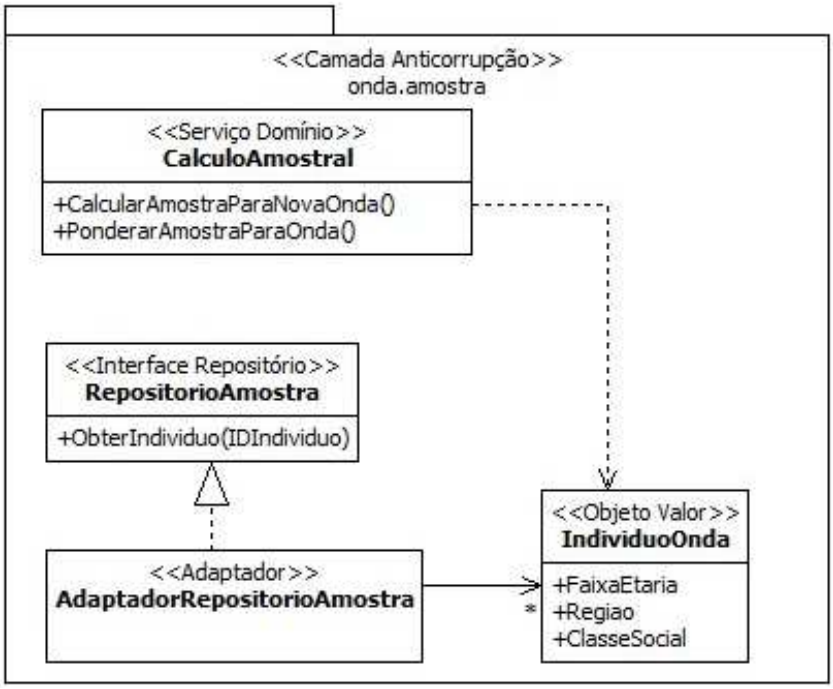
- <<Interface Repositório >> RepositorioQuestionarioDoIndividuo
    - Classe abstrata de Repositório para declarar os métodos definidos para interagir com o QuestionárioDoIndividuo. Para armazenar a Raiz QuestionarioDoIndividuo, o Repositório recebe o Agregado já criado pela FabricaQuestionarioDoIndividuo. A recuperação dos dados via persistência é implícita e poderia utilizar um *framework* de desenvolvimento ORM (Mapeamento objeto-relacional).
- <<Raiz Agregado>> QuestionarioDoIndividuo
    - Entidade raiz do Agregado que representa o questionário respondido por um indivíduo.
- <<Objeto Valor>> IDQuestionarioDoIndividuo
    - Objeto Valor que identifica o Agregado QuestionarioDoIndividuo

- <<Objeto Valor>> IDIndividuo
    - Objeto Valor do IDIndividuo do outro sistema “Amostragem”, que identifica o Agregado Individuo.
- <<Entidade >> RespostasDeUmaPergunta
    - Entidade que relaciona respostas do Individuo a uma Pergunta.
- <<Objeto Valor>> IDPergunta
    - Identificador da Pergunta do outro Contexto Delimitado Gerenciamento de Questionários.
- <<Entidade>> Resposta
    - Uma das respostas de um conjunto RespostasDeUmaPergunta de um indivíduo.
- <<Fábrica>> FabricaQuestionarioDoIndividuo
    - Fábrica para a construção do Agregado QuestionárioDoIndividuo.

5.6.2.4. <<Camada Anticorrupção>> onda.amostra

Módulo que encapsula a Camada Anticorrupção que faz a ligação entre os Contextos Delimitados Onda e Amostragem, conforme representado na Figura 28.

Figura 28 - onda.amostra



Fonte (o autor)

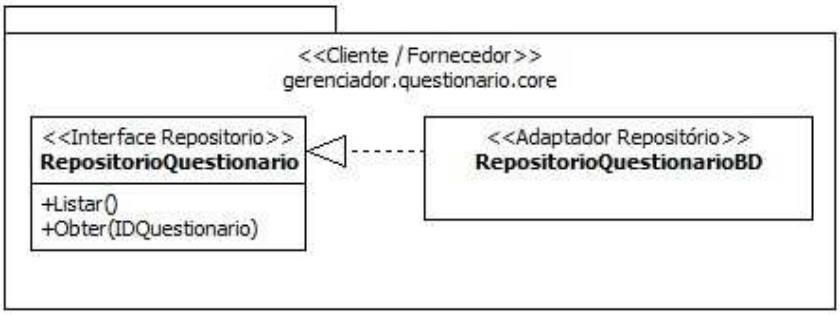
Seguem as descrições dos elementos do Módulo.

- <<Interface Repositório>> RepositorioAmostra
    - Repositório para utilizar o Contexto Delimitado Amostragem
- <<Adaptador>> AdaptadorRepositorioAmostra
    - Adaptador que faz a integração com o Contexto Delimitado Amostragem. O AdaptadorRepositorioAmostra encapsula a comunicação com o Contexto Amostragem, inibindo que o Contexto Onda tenha dependência com implementações técnicas ou altere a própria Linguagem Onipresente por causa do outro contexto. O Contexto Onda permanece íntegro em sua Linguagem Onipresente.
- <<Objeto Valor >> IndividuoOnda
    - Objeto Valor para trazer informações relevantes ao domínio Onda para a consolidação dos dados, independente de como seja seu projeto no outro sistema.
- <<Serviço Domínio> CalculoAmostral
    - Serviço que utiliza o contexto Amostragem para calcular a amostra da quantidade de indivíduos/entrevistas necessárias para uma Onda, dado o Perfil Demográfico. O cliente que utilizar esse Serviço de Domínio, deve enviar uma coleção de IndividuoOnda para o Cálculo Amostral.

5.6.2.5. <<Camada Cliente/Fornecedor>> gerenciador.questionario.core

Com o padrão estratégico Cliente/Fornecedor, há forte relação entre os contextos Onda e Questionário no nível de Módulo. Desta maneira, o contexto Onda utiliza diretamente o contexto Questionário, inclusive utilizando sua Linguagem Onipresente, como ilustrado na Figura 29.

Figura 29 - gerenciador.questionario.core



Fonte (o autor)

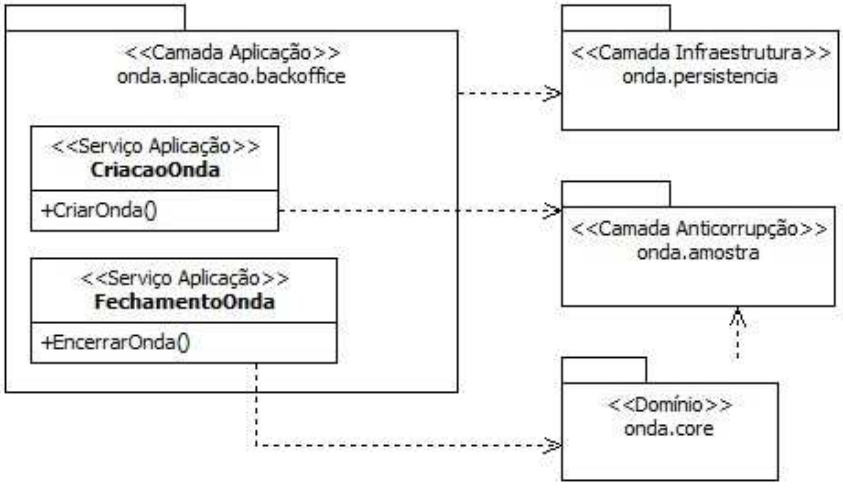
Seguem as descrições dos elementos do Módulo.

- <<Interface Repositório>> RepositorioQuestionario
    - Repositório para utilizar o Agregado Questionário do contexto Gerenciamento de Questionários.
- <<Adaptador Repositório>> RepositorioQuestionarioBD
    - Adaptador que faz a integração com o Contexto Delimitado Gerenciamento de Questionário.

5.6.2.6. <<Camada Aplicação>> onda.aplicacao.backoffice

Camada de Aplicação para o gerenciamento da Onda composta por Serviços de Aplicação. É utilizada por departamento interno da empresa para criação da Onda. No encerramento, faz a consolidação dos dados. A Figura 30 ilustra este Módulo.

Figura 30 - onda.aplicacao.backoffice

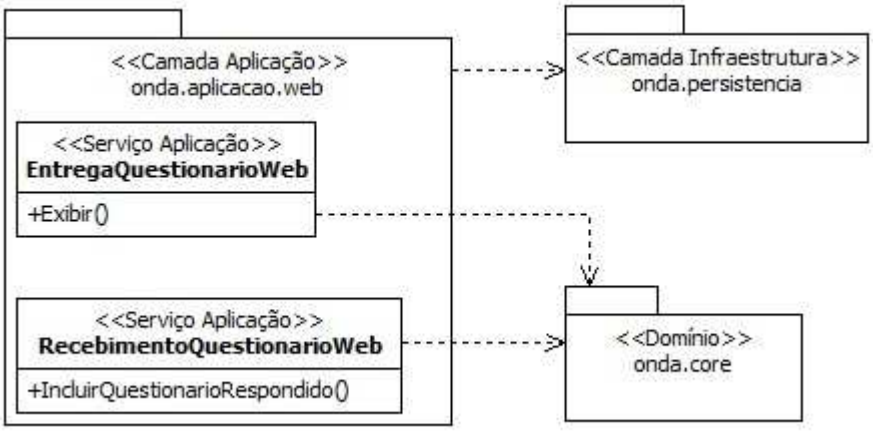


Fonte (o autor)

Seguem as descrições dos elementos do Módulo.

- <<Serviço Aplicação>> CriacaoOnda
    - Serviço de Aplicação para criar uma Onda. O serviço utiliza a FabricaOnda para montar o Agregado Onda e RepositorioOnda para a sua persistência.
- <<Serviço Aplicação>> FechamentoOnda
    - Serviço de Aplicação que utiliza SERVIÇOS DE DOMÍNIO para validações do encerramento da Onda.

Figura 31 - onda.aplicacao.web



Fonte (o autor)

5.6.2.7. <<Camada Aplicação>> onda.aplicacao.web

Camada de Aplicação para *web*, utilizando Serviços de Aplicação representados na Figura 31.

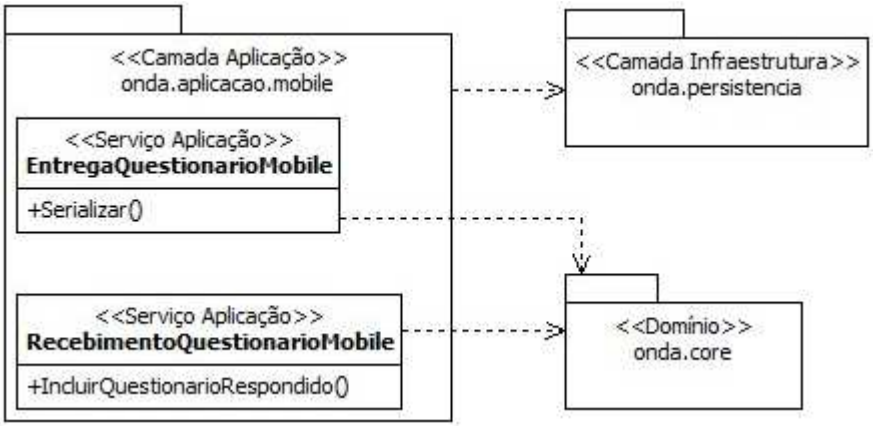
Seguem as descrições dos elementos do Módulo.

- <<Serviço Aplicação>> EntregaQuestionarioWeb
    - Serviço de Aplicação para a entrega do questionário para a *web* que provê acesso à camada da Interface de Usuário.
- <<Serviço Aplicação>> RecebimentoQuestionarioWeb
    - Serviço de Aplicação para o recebimento de um questionário respondido através da *web* (Interface de Usuário).

5.6.2.8. <<Camada Aplicação>> onda.aplicacao.mobile

Camada de Aplicação para aplicativos móveis, utilizando SERVIÇOS DE APLICAÇÂO como na Figura 32.

Figura 32 - onda.aplicacao.mobile



Fonte (o autor)

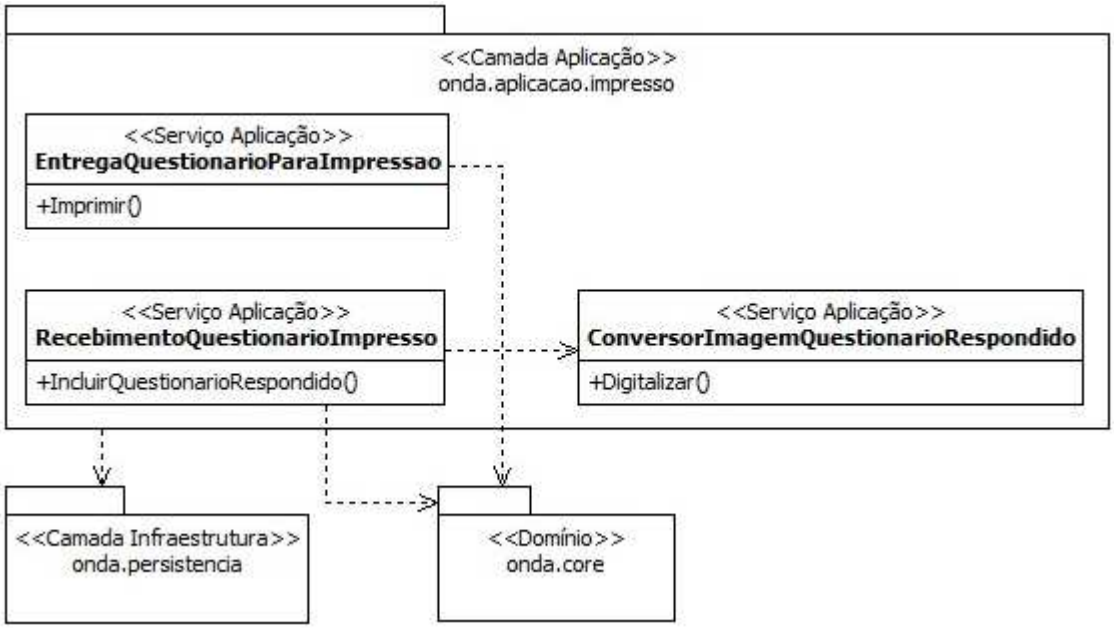
Seguem as descrições dos elementos do Módulo.

- <<Serviço Aplicação>> EntregaQuestionarioMobile
    - Serviço de Aplicação para a entrega do questionário para a *web*, provendo acesso à camada da Interface de Usuário.
- <<Serviço Aplicação>> RecebimentoQuestionarioMobile
    - Serviço de Aplicação para o recebimento de um questionário respondido através da web (da Interface de Usuário).

5.6.2.9. <<Camada Aplicação>> onda.aplicacao.impresso

Camada de Aplicação para lidar com questionários impressos, como representado na Figura 33.

Figura 33 - onda.aplicacao.impresso



Fonte (o autor)

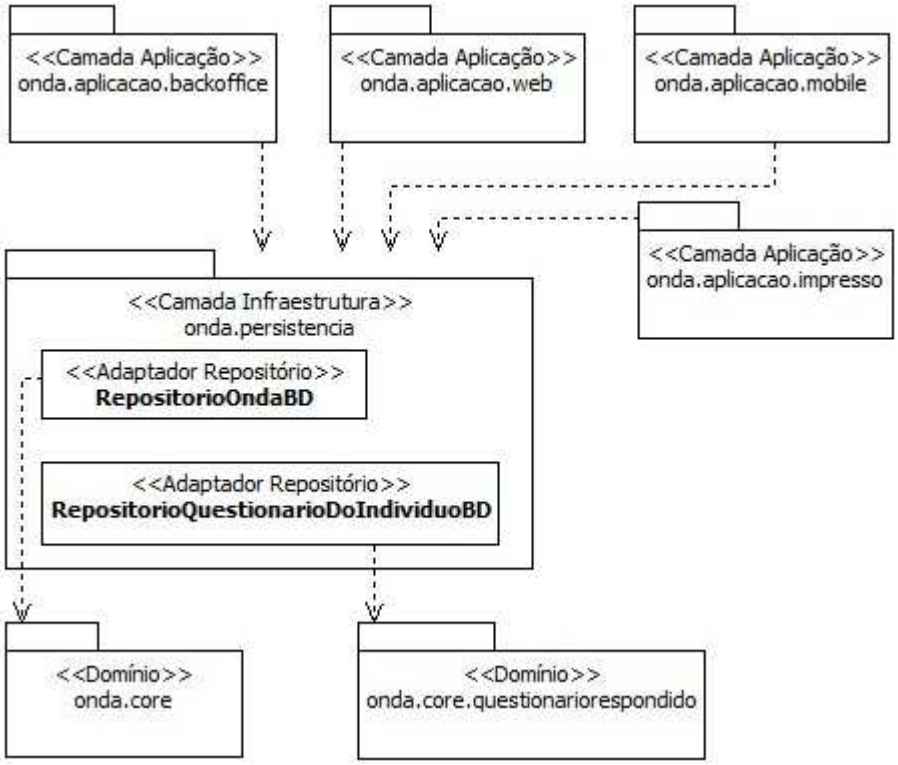
Seguem as descrições dos elementos do Módulo.

- <<Serviço Aplicação>> EntregaQuestionarioParaImpressao
    - Serviço de Aplicação para a entrega do questionário para a *web*, provendo acesso à camada da Interface de Usuário.
- <<Serviço Aplicação>> RecebimentoQuestionarioImpresso
    - Serviço de Aplicação para o recebimento de um questionário respondido através da web (da Interface de Usuário).
- <<Serviço Aplicação>> ConversosImagemQuestionarioRespondido
    - Serviço de Aplicação que lida com dispositivos de impressão e digitalização para questionários impressos.

5.6.2.10. <<Camada Infraestrutura>> onda.persistencia

Módulo de infraestrutura que contém as implementações dos Repositórios do sistema Onda, representado na Figura 34.

Figura 34 - onda.persistencia



Fonte (o autor)

Seguem as descrições dos elementos do Módulo.

- <<Adaptador Repositório>> RepositorioOndaBD
    - Implementação do Repositório RepositorioOnda para banco de dados
- <<Adaptador Repositório>> RepositorioQuestionarioDoIndividuoBD
    - Implementação do Repositório RepositorioQuestionarioDoIndividuo para banco de dados

6. CONCLUSÃO

6.1. Considerações gerais

O DDD auxilia no entendimento único do negócio tanto pelo solicitante do projeto quanto pelo desenvolvedor/projetista. O entendimento é definido pela Linguagem Onipresente e assim potenciais problemas de interpretação, como a não implementação de funcionalidades esperadas ou comportamentos diferentes, tendem a ser minimizados. A visão do Projeto Estratégico do DDD fornece caminhos de integração entre contextos a serem escolhidos, conforme as circunstâncias do negócio e as técnicas disponíveis. Centrando o foco no domínio-alvo em vez de questões técnicas, os desenvolvedores adquirem conhecimento do domínio da aplicação e ganham argumentos para discutir com os solicitantes do projeto, aumento a probabilidade de sucesso do software entregue.

A Arquitetura Hexagonal (HA/PA) tem o enfoque no domínio-alvo e deixa para resolver a integração através das portas conceituais, entendidas como pontos de acesso a partes do domínio. No essencial, essa arquitetura não deixa de ser uma Arquitetura em Camadas, mas com uma diferença fundamental: ela constitui um conjunto articulado de camadas que encapsulam as portas conceituais e organizado em torno de uma camada central que encapsula um único modelo do domínio da aplicação. Essa organização leva o desenvolvedor a primeiro pensar no domínio e, ao mesmo tempo, atender a alguns dos princípios de do projeto arquitetônico, como os Princípios da Inversão de Dependência, Aberto – Fechado e Responsabilidade Única. Embora (EVANS, 2010) aborde diversas técnicas para o projeto do domínio e da arquitetura do software, ele não fornece um roteiro para um projeto arquitetônico. A HA/PA utilizando o DDD permite um projeto arquitetônico flexível, uma vez que permite o atendimento da heterogeneidade das interações com subsistemas clientes e libera o projeto do domínio-alvo das questões de integração, mas (COCKBURN, 2005) também não fornece um roteiro para o seu emprego.

O roteiro proposto visa cobrir esta falta. Seguindo o roteiro, é esperado que um desenvolvedor/projetista menos experiente no DDD possa utilizá-lo para elaborar o projeto de um software real que requeira a integração entre domínios. Espera-se também que o roteiro permita a esse desenvolvedor/projetista assimilar mais facilmente os conceitos e o emprego do DDD.

Apesar de o roteiro cobrir os principais aspectos do DDD, utilizando a HA/PA como escolha arquitetônica, ele não aborda o DDD como um todo e equipes mais experientes podem sentir falta de mais passos. Apesar de a HA/PA ser aderente ao DDD, a falta de material bibliográfico mais detalhado sobre este estilo arquitetônico dificultou a elaboração do roteiro. O próprio DDD evoluiu desde a sua concepção original, refinando alguns conceitos como os Serviços de Aplicação e SERVIÇOS DE DOMÍNIO, havendo inclusive diferenças na definição dependendo da referência bibliográfica utilizada. Além disso, existem arquiteturas semelhantes à HA/PA, como a *Onion Architecture* (PALERMO, 2008), que também pretende resolver os mesmos problemas tratados neste trabalho. No entanto, a consideração de todas elas dificultaria a elaboração do roteiro.

Finalmente, uma lição importante apontada por (EVANS, 2010) (FOWLER, *et al.*, 2002): o emprego do DDD em um sistema que implementa um domínio potencialmente simples, com evolução e integração potencialmente simples, pode torná-lo desnecessariamente complexo.

6.2. Trabalhos futuros

Um dos trabalhos futuros, que estende o presente trabalho, é o estudo da integração do domínio-alvo com um sistema de terceiros. Por exemplo, uma empresa que fizesse a pesquisa por conta própria e precisasse utilizar os questionários criados no sistema da Research Corp e depois consolidá-los.

Como o roteiro apresentado neste trabalho não entrou no detalhe da implementação do software (código), poderia ser oportuno identificar mais passos para que o roteiro chegasse ao nível da codificação.

Por fim, é importante realizar um experimento – ou um estudo de caso – em que o roteiro proposto fosse aplicado por uma equipe experiente no DDD em um projeto real, para que se tivesse uma avaliação inicial da viabilidade prática do roteiro.

REFERÊNCIAS

BUSCHMANN, F. et al. **Pattern-Oriented Software Architecture:** A System of Patterns. Chichester, New York, Brisbane, Toronto, Singapore: John Wiley & Sons, Ltd, v. 1, 1996.

COCKBURN, A. Hexagonal Architecture. **alistair.cockburn.us**, 2005. Disponivel em: <http://alistair.cockburn.us/Hexagonal+architecture>. Acesso em: 21 abr. 2016.

COCKBURN, A. Ports And Adapters Architecture. **c2.com**, 2006. Disponivel em: <http://c2.com/cgi/wiki?PortsAndAdaptersArchitecture>. Acesso em: 21 abr. 2016.

DA-WEI, E. **The Software Complexity Model and Metrics for Object-Oriented**. 2007 International Workshop on Anti-Counterfeiting, Security and Identification (ASID). Xiamen, Fujian: IEEE. 2007. p. 464-469.

EVANS, E. **Domain Driven Design:** Atacando as Complexidades no Coração do Software. 2a. ed. Rio de Janeiro: Alta Books Editora, 2010.

FOWLER, M. et al. **Patterns of Enterprise Application Architecture**. Boston, San Francisco, New York, Toronto, Montreal, London, Munich, Paris, Madrid, Capetown, Sydney, Tokyo, Singapore, Mexico City: Addison-Wesley, 2002.

FREEMAN, S.; PRYCE, N. **Growing Object-Oriented Software, Guided by Tests**. Upper Saddle River, Boston, Indianapolis, San Francisco, New York, Toronto, Montreal, London, Munich, Paris, Madrid, Cape Town, Sydney, Tokyo, Singapore, Mexico City: Addison-Wesley, 2009.

GAMMA, E. et al. **Padrões de Projeto:** Soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2007.

GARLAN, D.; SHAW, M. **An Introduction to Software Architecture**. School of Computer Science, Carnegie Mellon University. Pittsburgh, PA. 1994. (CMU-CS-94-166).

GHAZARIAN, A. **A Theory of Software Complexity**. General Theory of Software Engineering (GTSE), 2015 IEEE/ACM 4th SEMAT Workshop on a. Florence: IEEE. 2015. p. 29-32.

HAAS, H. Designing the architecture for Web services. **w3.org**, 2003. Disponivel em: <https://www.w3.org/2003/Talks/0521-hh-wsa/>. Acesso em: 21 abr. 2016.

JAIN, R. et al. Exploring the Impact of Systems Architecture and Systems Requirements on Systems Integration Complexity. **IEEE Systems Journal**, v. 2, n. 2, p. 209-223, jun. 2008. ISSN ISSN: 1932-8184.

MARTIN, R. C. **Agile Software Development:** Principles, Patterns, and Practices. 2nd. ed. Upper Saddle River: Prentice Hall, 2002.

OASIS SOA REFERENCE MODEL TC. Reference Model for Service Oriented Architecture 1.0. **OASIS**, 2006. Disponivel em: <https://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf>. Acesso em: 29 maio 2016.

PALERMO, J. The Onion Architecture. **jeffreypalermo.com**, 2008. Disponivel em: <http://jeffreypalermo.com/blog/the-onion-architecture-part-1/>. Acesso em: 5 jun. 2016.

PRYCE, N. Visualising Test Terminology. **natpryce.com**, 2009. Disponivel em: <http://www.natpryce.com/articles/000772.html>. Acesso em: 21 abr. 2016.

SOARES, S. A. et al. **Dribbling complexity in model driven development using Naked Objects, domain driven design, and software design patterns**. Computing Conference (CLEI), 2015 Latin American. Arequipa: IEEE. 2015. p. 1-11.

TIE, J.; JIN, J.; WANG, X. **Study on application model of three-tiered architecture**. Mechanic Automation and Control Engineering (MACE), 2011 Second. Hohhot: IEEE. 2011. p. 7715-7718.

WANG, Y.-H.; LIAO, J. C. **Why or Why Not Service Oriented Architecture**. Services Science, Management and Engineering, 2009. SSME '09. IITA International Conference on. Zhangjiajie: IEEE. 2009. p. 65-68.