Adoção de *Domain-Driven Design* para o Domínio de Pagamentos

Pedro da Cruz Chagas[1], **Carlos Danilo Luz**[1], **Edson OliveiraJr**[1]

[1]Universidade Estadual de Maringá.

ra94028@uem.br, carlos.danilo.luz@gmail.com, edson@din.eum.br

Abstract. *It is very common in the software engineering market that there is no definition regarding the domains present in the context of a given company, making the long term maintenance of these domains a challenge, and may even culminate in the need to re-implement the system as a whole. Given a company in payment's context with several unencapsulated domains distributed in different applications, the purpose of this work is to document the application of Domain-Driven Design concepts and good practices that will be used in a specific domain of this company. This will be done by the team responsible for maintaining the aforementioned domain and application, encapsulating it within the aforementioned application, which will be responsible for implementing the domain model that will be studied.*

Resumo. *É muito comum no mercado de engenharia de software que não se tenha uma definição quanto aos domínios presentes no contexto de uma determinada empresa, fazendo com que seja um desafio a manutenção desses domínios a longo prazo, podendo culminar até mesmo na necessidade de reimplementar o sistema como um todo. Dada uma empresa no contexto de pagamentos com vários domínios desencapsulados e distribuídos em diversas aplicações, o que se propõe com este trabalho é documentar a aplicação dos conceitos e boas práticas do Domain-Driven Design que serão empregados em um domínio específico desta empresa. Isso será realizado pela equipe responsável pela manutenção do domínio e da aplicação já citados, encapsulando o mesmo dentro somente desta aplicação, que será responsável por implementar o modelo do domínio que será estudado.*

1. Introdução

Domain-Driven Design, ou DDD, trata-se de uma filosofia de modelagem e design de domínios, apresentada formalmente por Eric Evans em seu livro *Domain-Driven Design*: *Tackling Complexity in the Heart of Software*, em 2003. Esta abordagem sintetiza princípios para reconhecimento do domínio em um determinado contexto, bem como práticas de iterações de design que possibilitam enriquecer o domínio em questão, no que diz respeito ao entendimento de processos e regras que o compõe [Evans 2009].

O "Domínio" consiste no coração do negócio em questão. Este é composto de um conjunto de ideias, regras e processos. Para que se possa definir e, ao longo do tempo, incrementar um modelo de um domínio de forma satisfatória, deve-se desenvolver primeiramente o que Evans chama de "Linguagem Ubíqua", que se refere à terminologia da realidade do negócio [ADZIC 2010]. Esta linguagem deve ser amplamente utilizada no

software a ser implementando/mantido e, também, entre os especialistas de domínios e profissionais responsáveis pelo mesmo [Fowler 2020].

Um domínio é formado por um ou mais "*Bounded Contexts*", ou contextos delimitados, que se relacionam entre si. Este conceito configura-se como um padrão essencial para o DDD, sendo o foco da seção de design estratégico dessa filosofia [Fowler 2020]. Cada contexto delimitado possui suas responsabilidades claramente definidas que, por sua vez, podem ter sua própria Linguagem Ubíqua.

No presente trabalho, está sendo abordado um módulo específico de uma empresa fornecedora de pagamentos para empresas internacionais que queiram comercializar no Brasil e na América Latina. Este contexto possui diversas particularidades, pode-se citar como exemplo a captura de informações que serão utilizadas no pagamento, o processamento do pagamento em si junto a um parceiro comercial, a conversão de moeda baseado na cotação diária das moedas envolvidas, análise de risco da transação a ser efetuada e do comprador, notificações a serem enviadas tanto ao comprador quanto ao parceiro comercial, entre outros contextos.

A principal motivação deste trabalho se dá por evitar a *Big Ball of Mud*, ou a Grande Bola de Lama, conceito que diz respeito a sistemas com pouca definição do modelo, seus agregados e relacionamentos entre as entidades. O que se tem, portanto, trata-se de um software confuso e mal gerenciado. Geralmente, esses sistemas necessitam de um número muito seleto de desenvolvedores que possuem um vasto conhecimento da aplicação em questão para evitar que a mesma entre em colapso [Vernon 2016].

O DDD se propõe a evitar a criação da *Big Ball of Mud*, se as técnicas do mesmo forem empregadas tanto no domínio a ser modelado, como na comunicação do mesmo com outros domínios. O que motiva este trabalho é a aplicação destes conceitos para a modelagem de um domínio específico da empresa em questão. Para que, posteriormente a este processo, seja possível obter um modelo consistente, profundo, simples e fácil de receber alterações.

O objetivo principal deste trabalho é realizar a modelagem de um domínio específico de um sistema de pagamentos internacionais. Nesta modelagem, espera-se obter a compreensão a a extração da Linguagem Ubíqua do contexto delimitado a ser modelado e, posteriormente, aplicados os *domain model patterns* para estruturar a aplicação com enfoque no domínio, definindo as entidades de agregadores, os objetos de valor, os repositórios, serviços, etc.

1.1. Organização do Trabalho

Logo após esta Introdução é apresentada a fundamentação teórica para o presente trabalho. Nesta etapa são levantados os conceitos referentes ao *Domain-Driven Design*, tanto no que diz respeito ao DDD estratégico quanto a etapa tática do mesmo.

Em seguida os materiais e métodos utilizados no trabalho são evidenciados na seção 3. Na seção 4 é apresentada a modelagem propriamente dita, primeiramente o estudo e análise do contexto de Transação, que é o objeto de estudo deste trabalho e, logo após, a execução da modelagem em si.

Na seção 5 tem-se a apresentação dos resultados obtidos da modelagem e uma discussão sobre a mesma. E, na seção 6, as conclusões do trabalho e sugestões de trabalhos

futuros.

2. Fundamentação Teórica

Nesta seção apresentaremos os principais conceitos envolvidos para a concepção da proposta.

2.1. Domain-Driven Design

Domain-Driven Design foi discutido pela primeira vez em 2003 por Eric Evans, em seu livro *Domain-Driven Design: Tackling Complexity in the Heart of Software*. De lá para cá, a metodologia foi amadurecendo pelo próprio autor, bem como foi difundida pela comunidade de Engenharia de Software, com destaque para o arquiteto de software e autor Vaugh Vernon, que publicou duas obras tratando do assunto, *Implementing Domain-Driven Design*, em 2013 [Vernon 2013], e *Domain-Driven Design Distilled*, em 2016 [Vernon 2016].

Esta metodologia trata-se de um pacote de técnicas avançadas para auxiliar desenvolvedores a modelar, desenvolver e manter projetos de software complexos que entregam alto valor, tanto estrategicamente, quanto taticamente. Em suma, DDD diz respeito ao ato de modelar um determinado domínio da maneira mais explícita possível e evitar o desenvolvimento de aplicações monolíticas [Vernon 2016].

O que se pretende buscar utilizando as ferramentas do DDD é um modelo profundo e um design flexível que possibilite mudanças sem muito esforço, conforme o modelo em questão for sendo amadurecendo ao decorrer do tempo. Esse aprofundamento também pode ocorrer por meio de uma sequencia de pequenas refatorações no modelo em si, conforme os responsáveis identifiquem a necessidade de revisitar a modelagem, seja por uma demanda externa, ou por melhoria interna dos artefatos que compõem o sistema que está sendo mantido [Evans 2009].

Evans defende que o modelo não seja desconectado da implementação. Uma vez que uma oportunidade de avanço for aproveitada, a mesma deve refletir no código as alterações que foram feitas no modelo. Modelagem e implementação devem estar coesas entre si durante todo o processo [Evans 2009].

Como citado anteriormente, o DDD possui duas etapas subsequentes, a etapa estratégica e a etapa tática. Ambas serão apresentadas em seguida, bem como os conceitos e práticas utilizados neste trabalho que compõem as mesmas.

2.1.1. DDD Estratégico

Trata-se da etapa inicial da modelagem de um domínio. Vernon exemplifica este processo como as pinceladas largas que um pintor realiza em uma tela para dar a forma inicial de uma obra, antes de entrar nos detalhes da mesma [Vernon 2016]. Esta etapa destaca graus de importância no sistema a ser modelado, evidenciando, dessa forma, as necessidades a serem endereçadas.

Evans apresenta três temas concernentes à esta etapa, são eles contexto, destilação e estrutura em larga escala. O contexto determina que um modelo bem definido deve ser consistente do início ao fim, sem contradições ou sobreposições. A destilação dá clareza

ao que precisa ser realizado, concentrando a atenção dos modeladores de forma adequada às partes vitais do sistema, ao domínio principal e alocando cada subdomínio em suas devidas funções. E a estrutura em larga escala permite que se tenha uma visão do sistema com um todo, entendendo-o em linhas gerais por meio de um conjunto de conceitos e/ou regras avançadas. Esses três princípios, quando utilizados em conjunto, possibilitam a produção de um design de qualidade na etapa estratégica do *Domain-Driven Design Distilled* [Evans 2009].

Dois conceitos essenciais ao DDD são aprofundados durante o desenvolvimento dos temas citados contidos neste processo, tratam-se dos *Bounded Contexts*, ou Contextos Delimitados, e a *Ubiquitous Language*, traduzindo tem-se Linguagem Ubíqua ou, como Evans define, Linguagem Onipresente [Evans 2009].

Segundo Vernon, um Contexto Delimitado trata-se de um limite contextual semântico, em que cada componente dentro deste limite tem um significado e responsabilidade específica para o modelo. No começo do processo de modelagem de um domínio, o contexto delimitado pode ser algo conceitual porém, conforme as etapas avançam, o mesmo transiciona para se tornar, na verdade, onde o modelo será implementado, no que diz respeito à relação do modelo com o código fonte do projeto [Vernon 2016].

O contexto *Ubiquitous Language* deve refletir a linguagem descoberta, amadurecida e também falada por todos os membros do time responsável pelo desenvolvimento do mesmo, bem como dos especialistas do domínio a ser modelado [Vernon 2013]. Essa linguagem recebe nome de *Ubiquitous Language*, em português Linguagem Ubíqua ou, como Evans define, Linguagem Onipresente. Sem a Linguagem Onipresente os desenvolvedores precisam traduzir os conceitos para os especialistas de domínio e vice versa, a comunicação torna-se, desta forma, lenta e improdutiva [Evans 2009]. O objetivo principal da Linguagem Onipresente é aproximar desenvolvedores de especialistas de domínio, fazendo com que ambos estejam na mesma página no que diz respeito à linguagem falada.

2.1.2. DDD Tático

Logo após a modelagem estratégica de uma organização ter sido concluída, faz-se necessário aplicar as ferramentas disponibilizadas pela etapa tática do DDD. Voltando à analogia do pintor e seus pincéis apresentada por Vernon, se o DDD estratégico diz respeito às pinceladas largas iniciais que o artistas realiza em sua obra, o DDD tático seria relacionado às pinceladas finas e detalhadas que o pintor emprega em sua obra para dar os toques finais à pintura [Vernon 2016].

Nesta etapa é posto em prática as definições da etapa estratégica, fazendo com que o espaço da solução proposta ganhe forma, baseada naquilo que foi levantado como espaço do problema do contexto que se procura modelar

3. Materiais e Métodos

Após a análise do contexto atual da empresa em questão, foram aplicados os conhecimentos obtidos, juntamente com os conceitos do DDD estudados, para que se chegasse à uma modelagem do domínio de Transação que pudesse manter os sistemas atuais em funcionamento, isolasse o modelo antigo e que fosse condizente com a expectativas dos

domain experts. Isto envolveu vários atores como, por exemplo, os desenvolvedores responsáveis pela modelagem, os especialistas de domínio e colaboradores mais experientes, que possuem maior conhecimento sobre os artefatos legados da empresa que precisaram ser analisados. Em relação à execução dessa dinâmica, as sessões da mesma foram realizadas *on-line*, utilizando uma ferramenta de colaboração visual chamada Miro. Para a análise de código foram utilizados os ambientes integrados de desenvolvimento PHPStorm, da JetBrains, e o Visual Code Studio, da Microsoft.

Para o presente trabalho foi utilizado o método de pesquisa exploratória, com a finalidade de se aprofundar os conhecimentos relacionados ao *Domain-Driven Design*, realizando-se uma fundamentação teórica sobre o assunto e documentando a aplicação dos conceitos estudados em um contexto empresarial.

Logo após, foi realizado o estudo e análise do contexto que foi modelado. Para isso, foram realizadas, ao longo de duas semanas, várias sessões de *Event Storming* com os pares envolvidos na remodelagem do domínio em questão. Essas reuniões foram realizadas *on-line*, utilizando-se da ferramenta Miro, apresentada anteriormente. Destas sessões obteve-se insumos sobre os eventos de Transação, a máquina de estados do domínio e as dependências do domínio em relação a outros domínios.

Após a modelagem, foi requisitado ao *tech lead* da equipe responsável pela modelagem um relato livre escrito sobre o impacto dos conceitos do DDD no processo, para que se pudesse mensurar o sentimento dos desenvolvedores envolvidos com relação a modelagem realizada

4. Desenvolvimento

Neste capítulo será apresentado o desenvolvimento da documentação da modelagem proposta.

4.1. Estudo e Análise do Contexto a ser Modelado

A empresa em que o estudo foi realizado trata-se de uma organização que possui um grande sistema de intermediação de pagamentos para que empresas do exterior possam processar pagamentos no Brasil e na América Latina como um todo. Isso se dá da seguinte forma, a empresa foco deste estudo solicita os dados do pagamento para o cliente, após o mesmo informar tais dados, a empresa irá repassar esses dados para um parceiro comercial responsável por fazer o processamento do pagamento em si, neste momento é criada uma transação.

Nesta intermediação são feitos os cálculos das taxas cobradas do cliente e as devidas conversões de moeda, no caso de operações internacionais. Este parceiro informará o resultado do processamento do pagamento, a empresa então atualizará o estado da transação com o resultado o pagamento e notificará as partes interessadas.

A modelagem proposta para este trabalho será realizada em um *Core Domains* da empresa em questão, mais precisamente será realizado no domínio de Transação. Domínio este responsável por gerenciar o ciclo de vida de uma transação de pagamento, desde sua criação, que ocorre quando os dados do pagador são coletados por outro contexto delimitado, até a notificação ao cliente do resultado do pagamento, como ser visto na Figura 1.

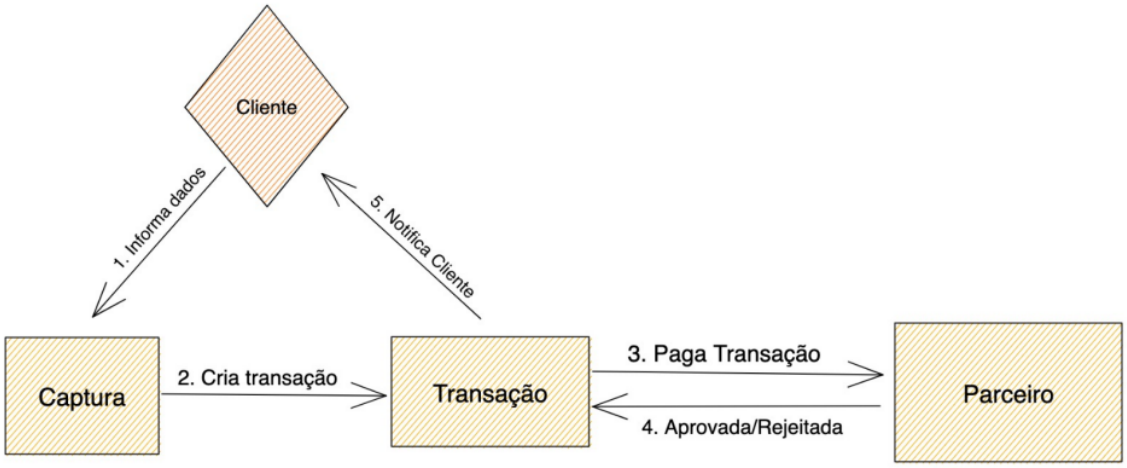

Figura 1. Ciclo de vida de uma transação

Uma vez que o domínio a ser modelado foi escolhido, pôde-se dar início ao processo de modelagem do mesmo. Antes de iniciar esta etapa propriamente dita, fez-se necessário realizar uma análise e o levantamento do estado atual da implementação do modelo deste domínio nos diversos módulos que compõe o sistema todo, pois o mesmo encontrou-se desencapsulado e presente em lugares diferentes dentro deste mesmo sistema. Esta análise se propôs a entender a modelagem atual do domínio, bem como sua máquina de estados e o modelo utilizado na persistência das instâncias do Agregado de Transação.

4.2. Execução da Modelagem

Tendo todos os insumos necessários provenientes da análise, a equipe de desenvolvimento pôde iniciar a modelagem do domínio de transação. A implementação da mesma foi realizada na aplicação que foi convencionada a ser a responsável por conter todas as implementações pertinentes ao domínio de transação, tanto como por exemplo do modelo novo cuja modelagem está sendo documentada neste trabalho, quanto da modelagem antiga.

4.2.1. Conceitos do Domain-Driven Design utilizados na modelagem

Como se trata de uma re-modelagem de um domínio, com um modelo previamente implementado e disponibilizado em produção, algumas decisões precisaram ser tomadas pelo time de desenvolvimento responsável. Como por exemplo, foi decido que o antigo modelo não seria descontinuado no momento em que o novo modelo estivesse disponível para uso, por conta da atual utilização do mesmo e toda estrutura de persistência já implementada. A equipe de desenvolvedores convencionou que a melhor abordagem para este caso seria, ao final de todo o processo, isolar o modelo antigo e não permitir que o mesmo seja acessado por nenhum outro agente.

Assim que a nova modelagem, mais rica e profunda, estivesse disponível, todo o fluxo de entrada da aplicação seria convertido a utilizar o novo modelo, portanto a API, *Application Programming Interface*, da aplicação em questão viria a expor os termos e

conceitos do novo modelo para os interessados neste contexto.

Outro conceito do DDD que foi utilizado neste caso é a **Camada de Anticorrupção**. Para esta finalidade, a ferramenta foi usada em um contexto diferente do que a literatura apresenta. Evans apresenta a Camada de Anticorrupção como sendo um mecanismo responsável por traduzir objetos conceituais e ações de um Contexto Delimitado para outro [Evans 2009]. Neste caso específico da empresa em questão, a Camada Anticorrupção foi utilizada como tradução entre duas versões de modelos correspondentes a um mesmo Agregado. O mesmo conceito, aplicado à um contexto diferente, mas que se mostra igualmente eficiente. Um diagrama ilustrando como isso se daria é mostrado abaixo, na Figura 2.

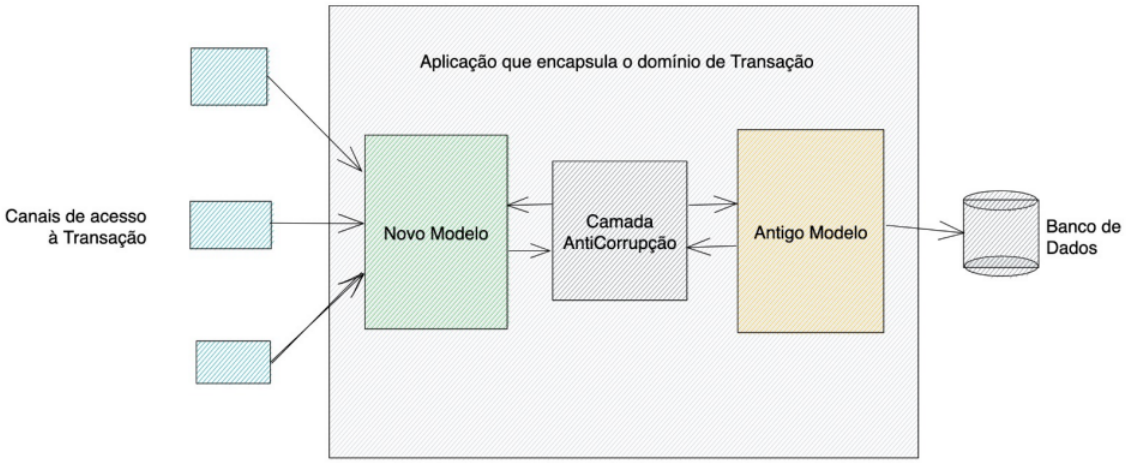

Figura 2. Camada de AntiCorrupção entre as versões do modelo de Transação

Outra decisão tomada pela equipe de desenvolvimento teve relação com os ***Value Objects***. Pois na literatura não há distinção entre tipos de *value objects*, enquanto que durante este processo de modelagem houve a necessidade de se fazer esta distinção. Esses atributos se configuram como sendo *value objects* por si só, pois são valores mensuráveis importantes para a Entidade [Evans 2009]. Porém o endereço como um todo também se encaixa como sendo *value object*, pois o mesmo não possui uma identidade em si mesmo.

Com isso dito, a estrutura do modelo do Agregado de Transação foi montada a partir desses *value objects* e, também, atrelando ao modelo um identificador no formato UUID - *Universally unique identifier*, para cumprir os requisitos do DDD para um Entidade. Essa estrutura foi definida tendo como base nos *payloads* que irão compor os Comandos e Eventos do domínio em questão, que foram estabelecidos antes que a modelagem tivesse início, utilizando-se de novas sessões de*Event Storming* entre os membros da equipe de desenvolvimento responsável pela iniciativa.

Essas sessões levaram em consideração dependências de outros contextos levantadas durantes a análise realizada previamente à modelagem, o que permitiu que a equipe fosse assertiva no que diz respeito às necessidades dos contextos que dependem do domínio de transação, e vice versa.

4.2.2. Integração entre Domínios

O processo de modelagem desse domínio foi realizado em conjunto com outras modelagens que estão ocorrendo na empresa em questão. A mesma possui contextos responsáveis pela captura das informações que serão utilizadas no processamento de uma transação, pelo processamento do pagamento em si junto à parceiros comerciais, pela análise de risco de uma transação e pela contabilidade financeira do que será repassado ao cliente posteriormente, sendo dependências entre esses domínios.

Outra decisão tomada pela empresa em um contexto geral, para que o setor de Engenharia de Software como um todo pudesse atualizar os seus modelos de uma forma integrada, compartilhando informações e necessidades. Tendo uma melhoria vista a partir da aplicação do DDD na empresa como um todo unificando uma Linguagem Onipresente que fosse comum a todos os contextos delimitados, o que possibilitou maior entendimento dos artefatos atuais e das mudanças que precisavam ser realizadas para que a empresa pudesse se organizar considerando esta nova linguagem.

5. Resultados e Discussão

Para que se pudesse ter uma visão interna deste processo foi requisitado um relato livre ao *tech lead* a equipe responsável por esta modelagem. Este profissional se configura como um desenvolvedor de software especialista em sistemas, que atua nesta empresa por volta de quatro anos liderando o desenvolvimento técnico do contexto de Transação estudado. Portanto, como resultado do presente trabalho, obteve-se como ponto positivo do presente estudo uma maior clareza sobre os diferentes contextos delimitados presentes na empresa estudada e dos relacionamentos entre cada um deles, por meio da análise realizada no princípio do estudo. Como segue na opinião dada pelo *tech lead* citado.

> *"...já é possível observar os ganhos de tais decisões, especialmente no que diz respeito à clareza de comunicação, confiança e velocidade com que as decisões são tomadas. É possível afirmar, portanto, que a aplicação dos conceitos táticos e estratégicos propostos pela literatura de DDD têm sido de grande relevância no processo de remodelagem realizado no contexto da companhia."*

Em decorrência da modelagem se tem maior assertividade por conta da fundamentação teórica e estudo do contexto, os conceitos do DDD puderam ser estudados e aplicados em um contexto onde os domínios estavam modelados erroneamente. A **Linguagem Onipresente** do domínio de Transação foi definida e passou a ser utilizada na implementação do novo modelo, mas também a linguagem onipresente da empresa como um todo foi desenvolvida. Ponto este também ressaltado pelo desenvolvedor, como segue.

> *"A decisão sobre utilizar os conceitos de DDD (Domain-Driven Design) se mostrou muito acertada, já que uma de suas principais propostas é a construção de uma linguagem coesa, compartilhada entre equipe de desenvolvimento e equipe de produto (linguagem ubíqua) acerca do contexto delimitado de um domínio, permitindo uma diminuição da carga cognitiva sobre a semântica dos termos falados."*

Uma das decisões acertadas envolvendo os conceitos do DDD que não estava previsto na literatura foi o isolamento do modelo antigo utilizando uma Camada de Anticorrupção, para reutilização posteriormente em casos já implementados que demandariam muito esforço para refatoração completa neste momento. Sobre isso, o *tech lead* da equipe relatou o que segue.

> *"...criar um bom isolamento entre os modelos, evitando a corrupção do novo modelo em função do modelo legado, a fim de que o trabalho de remodelagem não fosse em vão, passamos a visualizar a base de código legada não apenas como uma versão depreciada do mesmo modelo de domínio, mas sim como um modelo isolado, coexistindo com o novo modelo, de forma que este seria, nos termos do context mapping, um upstream para o novo modelo."*

Como pontos negativos do presente trabalho teve-se a necessidade de realizar esta modelagem partindo de um modelo previamente implementado, o que levou a equipe responsável a tomar decisões para contornar os problemas gerados por essa modelagem obsoleta.

> *"...o modelo previamente concebido não era mais considerado um bom modelo, principalmente porque não cumpria o importante papel de facilitar a comunicação entre as áreas envolvidas no processo de tomada de decisão sobre a evolução dos produtos"*

Como ponto de melhoria do processo de modelagem documentado tem-se que novas refatorações na aplicação que encapsula o modelo podem ser feitas no decorrer do tempo até que o modelo antigo seja totalmente descontinuado. Até este momento o mesmo servirá como provedor de serviços, visto que está em funcionamento, como pode ser visto no relato a seguir feito pelo desenvolvedor que liderou a modelagem.

> *"O processo de absorção do novo modelo, portanto, prevê apenas alterações de borda na aplicação que encapsula tal domínio, ou seja, pretendemos nos basear no novo modelo a fim de estabelecer um contrato público para a API de comandos e na definição dos eventos externalizados pela aplicação, evitando alterações drásticas na base de código existente."*

O que possibilitou essa refatoração foi a modelagem documentada neste trabalho, uma vez que a mesma isolou a modelagem antiga de agentes externos, garantindo que, ao realizar manutenção na mesma, não haverá impactos em outros contextos. Também observa-se a necessidade de que as etapas subsequentes sejam realizadas de forma coordenada, para que se possa obter o maior nível de aproveitamento dos conceitos do DDD em uma remodelagem, sem comprometer o funcionamento do sistema em questão ao longo do processo.

Portanto, o presente trabalho apresentou benefícios relevantes para o contexto estudado, como a clareza em relação aos contextos delimitados presentes na empresa e na comunicação entre esses contextos, utilizando-se da linguagem onipresente desenvolvida neste processo.

Pode-se citar, também, o aprofundamento do conhecimento e do modelo em si através da aplicação dos conceitos do DDD, como por exemplo a Camada de Anticorrupção. O presente trabalho evidenciou, como melhoria proposta, a possibilidade de se realizar refatorações ao longo do tempo para que o modelo antigo possa ser descontinuado e ser totalmente substituído pelo novo. Porém, este processo foi limitado no que diz respeito à liberdade para a modelagem, uma vez que foi necessário considerar a modelagem existente para desenvolver a nova.

6. Conclusões

O *Domain-Driven Design* se propõe a endereçar a complexidade de domínios de negócios, trazendo maior clareza sobre significados e termos concernentes a um determinando domínio, que podem ser utilizados em um modelo a ser possivelmente implementado em uma aplicação que seja responsável por gerenciar àquele domínio em um determinado contexto.

Tendo dito isso, o presente trabalho teve como objetivo documentar a modelagem do domínio de Transação de um sistema responsável pela intermediação de pagamentos internacionais. Para isso, foi realizado um levantamento bibliográfico sobre os conceitos do DDD que seriam utilizado neste processo e, também, foi feito um estudo do contexto atual do sistema em questão, elencando necessidades que precisariam ser endereçadas.

Mesmo com a modelagem possuindo pontos de pendência a serem melhorados, pôde-se observar ganhos obtidos com a adoção de DDD para o contexto estudado. Tem-se como exemplo a definição de uma linguagem onipresente, tanto para o domínio estudado, como para a empresa como um todo, trazendo maior clareza, coesão e, consequentemente, velocidade na comunicação na empresa estudada no presente trabalho.

Em relação às limitações deste trabalho, pode-se citar a impossibilidade de expor a solução da modelagem em si, por conta do sigilo imposto pela empresa no que diz respeito aos seus artefatos. Para a realização de trabalhos futuros, podem ser consideradas as seguintes alterações nos experimentos. A documentação e modelagem de um domínio desde sua concepção e, também, a adoção de *Domain-Driven Design* para um contexto que considere diferentes subdomínios e contextos delimitados, bem como possíveis comunicações entre os mesmos.

Referências

ADZIC, G. E. E. (2010). *Domain driven design redefined*. Disponível em: `https://gojko.net/2010/06/11/eric-evans-domain-driven-design-redefined/` - Acessado em: 20 set.

Evans, E. (2009). *Domain-driven design: atacando as complexidades no coração do software*. Alta Books.

Fowler, M. (2020). *DomainDrivenDesign. 2020*. Disponível em: `https://martinfowler.com/bliki/DomainDrivenDesign.html` - Acessado em: 20 set.

Vernon, V. (2013). *Implementing domain-driven design*. Addison-Wesley.

Vernon, V. (2016). *Domain-driven design distilled*. Addison-Wesley Professional.