VLADMIR CRUZ

SOFTWARE ARCHITECTURE

DOMAIN-DRIVE DESIGN

AULA 04

SUMÁRIO

O QUE VEM POR AÍ?

Agora que já sabemos o que são os contextos delimitados, veremos como trabalhar com eles, o modo de trabalho dos times, quais as limitações e o uso de cada caso.

HANDS ON

Lembrando do nosso projeto escola, imagine que nosso time de desenvolvimento precisa trabalhar com vários contextos. Por exemplo, temos o contexto de admissões, de marketing e autenticação. Cada time trabalhando com seu contexto delimitado, como vimos na aula anterior.

Mas como realizar a integração entre esses vários contextos? Temos diversas formas de integração. Quais são elas e quais os cenários que ela melhor se aplica? E quando temos um sistema legado? Como ele pode ser integrado a todo esse cenário?

Esses pontos serão explicados no Hands On da aula 4. Assista a videoaula que preparamos para você!

SAIBA MAIS

Trabalhando com Contextos Delimitados

Contextos delimitados deixam claro cada parte que queremos construir, mas lembremos que são diversos contextos criados, e cada um tem sua linguagem própria, salvo raros casos, que iremos discutir futuramente.

O negócio não vive por conta de um único contexto, e sim de vários e de suas integrações. Agora vamos falar exatamente disso, como podemos integrar esses contextos, o que vamos observar, como vamos planejar e tratar cada caso.

Cada contexto delimitado será feito por um time ao mesmo tempo, e isso requer disciplina de comunicação e entendimento do que está sendo feito, pois a integração é parte chave de todo o processo de negócios.

Hoje trabalharemos com os padrões de desenho do DDD, que definem as integrações e como os contextos se integram.

Cooperação

Ao trabalhar com contextos delimitados, temos casos em que a comunicação é clara e objetiva entre os times e não existe precedência de um contexto a outro, tendo assim colaboração total entre os times.

Parceria

O primeiro modelo de desenho que temos é o de parceria, onde a comunicação flui constantemente e os times se notificam sobre mudanças que estão ocorrendo, dando tempo e documentação ao outro time de forma que adapte seu contexto para se manter conectado, e isso pode ser a forma de integração, a linguagem sendo utilizada (não a de programação, mas sim a que utiliza para conectar os contextos), tempo de resposta das adequações e muito mais.

Graficamente, o que temos está representado na figura 1 – “Times trabalhando nos contextos delimitados em formato de parceria”.

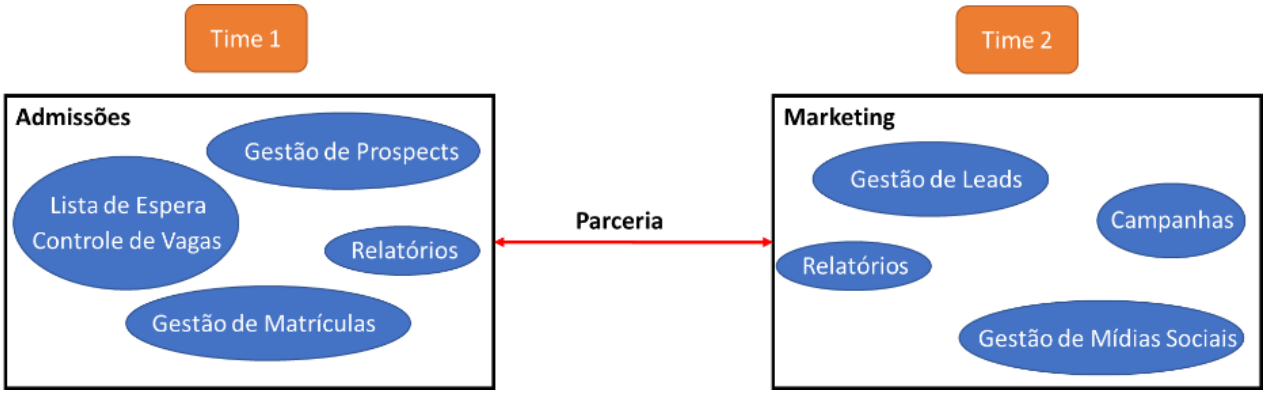

Figura 1 – Times trabalhando nos Contextos Delimitados em Formato de Parceria
Fonte: Elaborado pelo autor (2023)

Alguns pontos importantes que podemos notar nesse modelo é que a comunicação é muito frequente e mudanças serão normais, principalmente nos primeiros momentos do desenvolvimento enquanto se discute sobre a melhor solução.

Kernel Compartilhado

Outro caso de colaboração que teremos é o de Kernel compartilhado, ou seja, temos uma parte do modelo que é comum a todos os contextos envolvidos e deve seguir as definições de todos para que funcione. Em nosso modelo temos um caso deste, ao verificar que os dois contextos que estamos trabalhando têm em comum o CRM e os Relatórios, como na figura 2 – “Times trabalhando nos contextos que possuem um Kernel compartilhado”,

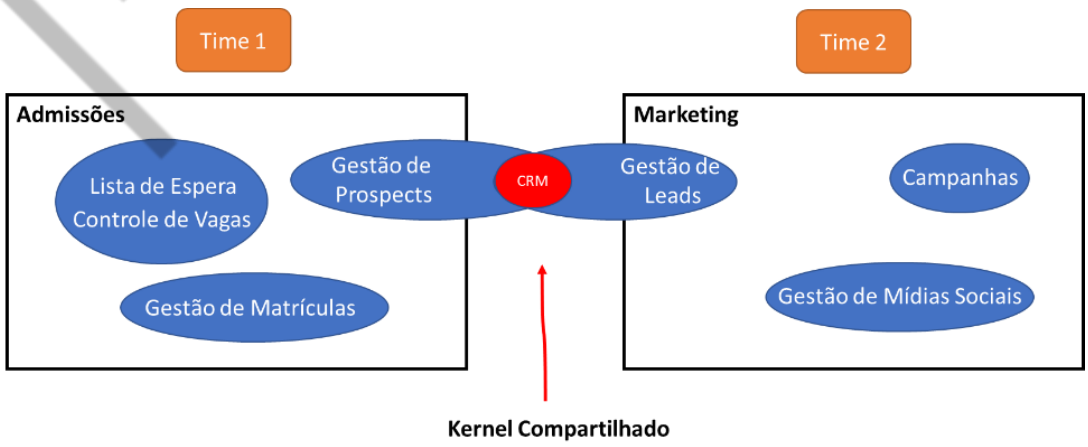

Figura 2 – Times trabalhando nos Contextos que possuem um Kernel Compartilhado
Fonte: Elaborado pelo autor (2023)

Esse é um caso em que a comunicação deve ser essencial, pois, teoricamente, viola todo o princípio dos contextos delimitados. Aqui as duas soluções serão utilizadas além dos limites propostos de seus contextos. Temos mais de um time trabalhando nas soluções e temos demandas distintas ocorrendo na mesma solução.

O grande desafio aqui é que qualquer mudança na solução afeta todos os contextos envolvidos. Sendo assim, qualquer mudança deverá ser testada para todos os cenários em todos os contextos.

A implementação de um Kernel compartilhado é desafiadora, pois envolve compartilhar o mesmo código com todos os times envolvidos. Por isso, esse tipo de modelo é desencorajado, mas, se for realmente necessário, deve ser analisado e coordenado com muita atenção, evitando assim perda de dados ou problemas de execução.

Há casos em que o uso de Kernel compartilhado é inevitável, como por exemplo em sistemas legados, que não foram construídos de forma a separar seus contextos (já escuto vozes gritando MONOLITOS!), e o uso de compartilhamento de Kernel é essencial enquanto se moderniza a solução.

Cliente-Fornecedor

Passamos agora para o segundo modelo de desenho que podemos encontrar, o de Cliente-Fornecedor, sendo:

- Fornecedor (Upstream): provê um serviço.
- Cliente (Downstream): consome um serviço.

Esse modelo ocorre quando os times que desenvolvem os contextos podem desenvolvê-los separadamente, porém há alguma dependência por conta de um serviço. Nesse caso, temos uma situação em que os times podem não se entender quanto à integração, tal como vimos no modelo de parceria, e agora os times têm poder de decidir como querem se integrar. Vamos ao nosso cenário, como você pode visualizar na figura 3 – “Cenário onde temos os contextos e times, bem como seus relacionamentos”.

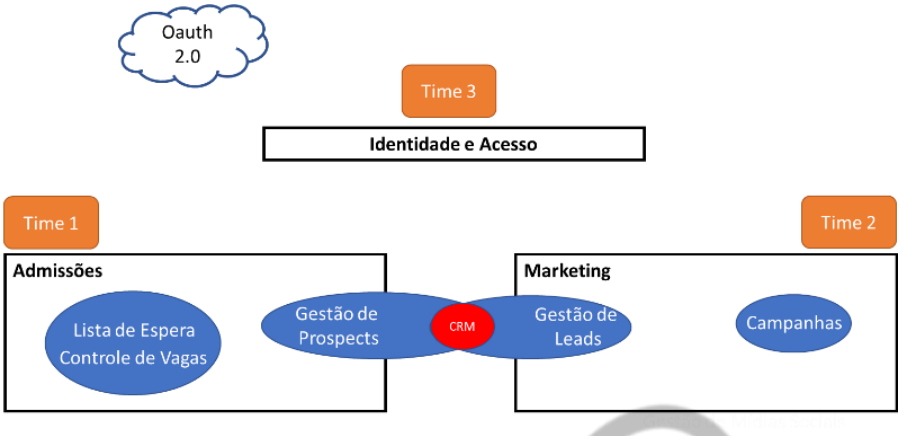

Figura 3 – Cenário onde temos os Contextos e Times, bem como seus relacionamentos
Fonte: Elaborado pelo autor (2023)

Como podemos observar na figura 3 – “Cenários onde temos os Contextos e Times, bem como seus relacionamentos”, temos 3 contextos, cada um com seu time de desenvolvimento, e já desenhamos o fluxo de serviços, ou seja, quem provê e quem consome.

Conformista

Vamos ao nosso caso: escolhemos utilizar um serviço terceiro que trabalha com OAuth 2.0, uma vez que já utilizamos serviços desse provedor, porém precisamos criar nossa camada de gestão de identidade e acesso, para controlar quem acessa o que em nosso ambiente.

Como o serviço de autenticação pertence a um fornecedor externo, e ele provê serviços a vários clientes globalmente, não temos como negociar para que ele se adeque às nossas necessidades. Sendo assim, quem “manda” na relação é o fornecedor, ou seja, temos que nos conformar em trabalhar com o padrão que nos é dado, e adequar nossa solução para utilizar esse padrão, tal como é representado na figura 4 – “Cenário onde o cliente se conforma com o que é passado pelo fornecedor”:

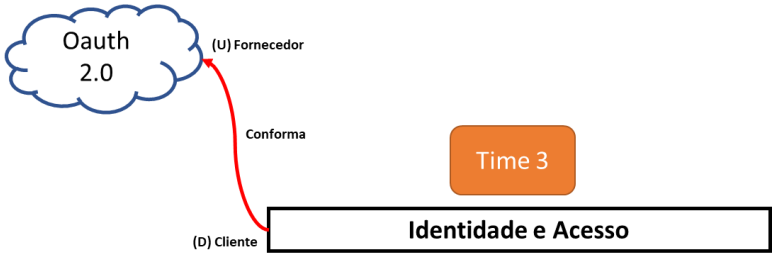

Figura 4 – Cenário onde o Cliente se Conforma com o que é passado pelo Fornecedor
Fonte: Elaborado pelo autor (2023)

Quando o padrão de serviço é determinado pelo fornecedor (U) e este não renuncia a seu protocolo, ou seja, ignora as demandas do cliente, este tem que “se conformar” e se adequar ao que o fornecedor o provê. Esse é o modelo conformista.

Camada Anticorrupção (Anti-Corruption Layer)

Agora que configuramos a integração do nosso contexto de identidade e acesso, este deve prover serviços para os contextos que estão “abaixo”, como demonstrado na figura 5 – “Cenário onde um contexto provê serviços aos demais contextos”:

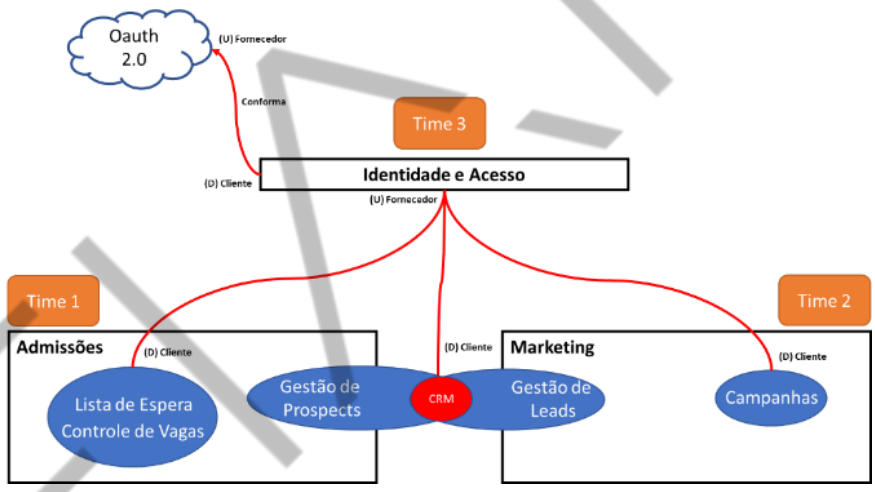

Figura 5 – Cenário onde um Contexto provê serviços aos demais contextos
Fonte: Elaborado pelo autor (2023)

Porém, aqui temos uma situação em que o CRM, que tem o Time 1 e Time 2 trabalhando, não aceita o padrão que vêm do Time 3, do contexto de identidade e acesso, e este não vai alterar o seu protocolo para atender a demanda do CRM, que também não vai alterar o seu protocolo em função do anterior, o que caracteriza em um não conformismo. Com base nisso, os Times 1 e 2 decidem que vão criar uma camada para abstrair o protocolo de identidade e acesso, e convertê-lo ao seu próprio modelo, mantendo assim a integridade da sua solução.

Essa camada se chama camada anticorrupção, ou anti-corruption layer (em inglês), e tem como sigla ACL, sendo assim, nosso modelo de integração ficaria

conforme a figura 6 – “Cenário onde um contexto interno cria uma camada anticorrupção":

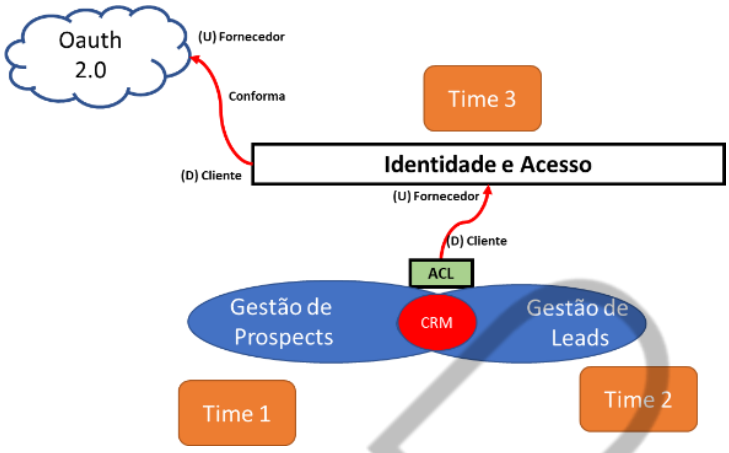

Figura 6 – Cenário onde um Contexto interno cria uma Camada Anticorrupção
Fonte: Elaborado pelo autor (2023)

E talvez você esteja se perguntando: quando é recomendável utilizar uma ACL?

- - Quando o contexto Cliente (D) contém um subdomínio principal. Isso evita que se corrompa ou interfira na implementação da solução principal, isolando os códigos.
- - Quando o modelo do fornecedor (U) é ineficiente ou não serve totalmente ao cliente. Nesse caso, criamos a ACL para filtrar somente o que é necessário. Essa solução é muito usada em caso de integração com legados.
- - Quando o Fornecedor (U) muda constantemente seu protocolo, o que aumenta a complexidade da solução. Tendo uma ACL, a manutenção fica apenas nesse código.

Serviço de Host Aberto (Open-Host Service)

Nos casos anteriores falamos sempre de situações em que o fornecedor (U) tem o controle da solução e “força” o cliente a se adaptar à sua realidade, mas e quando o cliente (D) tem a preferência na integração?

Agora o fornecedor (U), provê uma camada de integração onde este expõe seus serviços para que o cliente (D) não precise se adequar, mas sim utilizar algo

padronizado que não muda com o tempo. Isso ajuda o fornecedor (U) a separar seus desenvolvimentos internos do que é publicado aos clientes (D), e permite a estes que se conectem usando um protocolo que não muda e está aberto. Este é o Serviço de Host Aberto ou Open-Host Service (OHS):

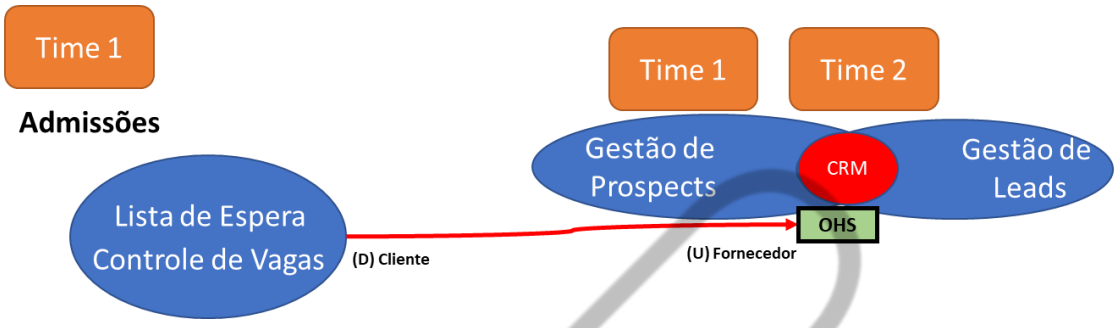

Figura 7 – Cenário onde um Contexto interno cria um Serviço de Host Aberto
Fonte: Elaborado pelo autor (2023)

Linguagem Publicada (Published Language)

O protocolo que tanto falamos anteriormente é a linguagem usada pelo cliente (D) ou pelo fornecedor (U). Aqui daremos um passo além do Serviço de Host Aberto (OHS), pois além de publicar o protocolo, o fornecedor (U) pode criar uma abstração a mais, que é a linguagem utilizada pelo cliente (D), como exemplificado na figura 8 – "Cenário em que, além do serviço de host aberto, também temos as linguagens do cliente implementadas":

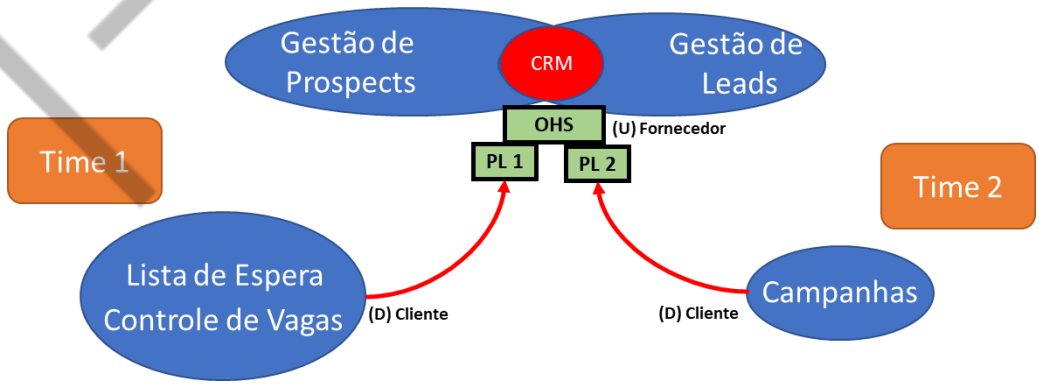

Figura 8 – Cenário em que, além do Serviço de Host Aberto, também temos as linguagens do Cliente implementadas
Fonte: Elaborado pelo autor (2023)

Dessa forma, além do fornecedor (U) conseguir desenvolver sua solução internamente sem afetar seus clientes, ele mantém diversas versões personalizadas

de linguagem, uma para cada cliente, mantendo seu protocolo aberto, e não impactando o cliente (D).

Alguns dos mercados que utilizam muito essa tecnologia são:

- Mercado financeiro, com bancos, adquirências (Stone, Cielo, Rede, entre outras), cartões de Crédito etc.
- Mercado logístico, transportes (integração de documentos de transporte), controle de estoque (EDI para monitoramento de estoques e automação de pedidos de compra).
- Mercado de manufatura (discreta e de processos), integração entre diversos fornecedores e clientes, padronização de protocolos etc.

Caminhos separados

Existe um modelo que é o de times simplesmente não se integrarem. Isso se dá quando os times não têm possiblidade de comunicação, o custo de integração é muito maior que o de criar uma solução interna, e não tem sentido em expor o serviço a outros contextos.

Vamos ao nosso caso: os times 1 e 2 decidem que seus contextos são muito distintos e resolvem partir sua solução de relatórios, uma vez que a complexidade de estrutura deixaria a solução cara demais para se integrar, e as necessidades das suas áreas são muito distintas quanto ao que esperam de suas soluções. Com isso a estrutura ficaria como demonstrado na figura 9 – “Cenário em que o contexto de relatório é duplicado, um em cada contexto”:

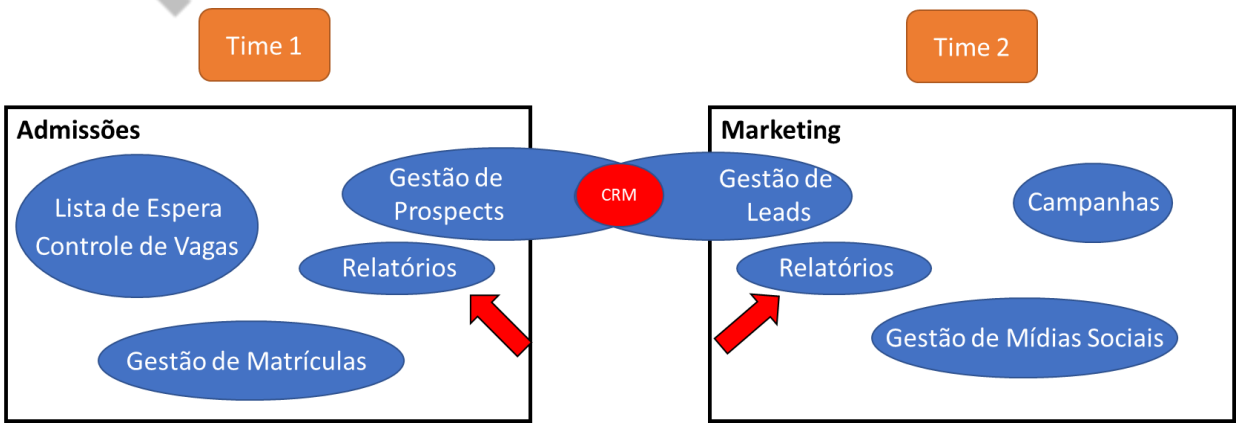

Figura 9 – Cenário em que o Contexto de Relatório é duplicado, um em cada Contexto

Fonte: Elaborado pelo autor (2023)

Outras situações em que podemos encontrar isso:

- Sistemas de Autenticação.
- Sistemas de Log.

Grande Bola de Lama

Vaughn Vernon (2013), em seu livro “Implementando Domain-Driven Design”, cita a Grande Bola de Lama (Big Ball of Mud), um modelo que é encontrado em sistemas muito grandes, em que os contextos não são bem limitados e são inconsistentes. Nesse caso específico, as recomendações são:

- Desenhe uma linha de limite em torno e chame de Grande Bola de Lama.
- Não tente aplicar nenhum método sofisticado de modelagem.
- Cuidado, pois essa Bola de Lama pode contaminar outros contextos.

Mapa de Contexto

Agora que criamos as integrações considerando todos os modelos, podemos finalmente desenhar o nosso mapa de contexto, que é a materialização visual dos nossos contextos delimitados e como estes se integram. Com isso, temos uma visão estratégica do todo, como é possível analisar na figura 10 – “Mapa de contexto demonstrando todas as integrações e contextos”:

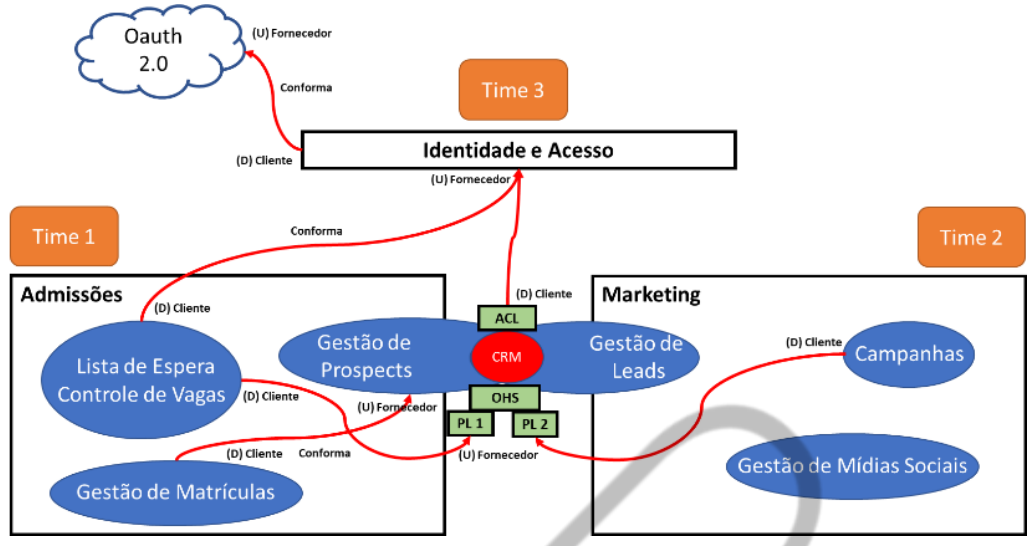

Figura 10 – Mapa de contexto, demonstrando todas as integrações e contextos
Fonte: Elaborado pelo autor (2023)

O Mapa de contexto não é algo estático, é um documento vivo, que deve ser atualizado constantemente. Afinal, como falamos anteriormente, é o mapa que reflete como os contextos são integrados e garantem que os subdomínios façam sentido.

O QUE VOCÊ VIU NESTA AULA?

Nesta aula, vimos que após definirmos os contextos delimitados, ainda temos muitos modelos para nos organizarmos durante o desenvolvimento, seja entrando em acordo com outro time que compartilhe o contexto conosco, seja nas parcerias que todos cedem ao que precisa ser feito, ou em modelos, tais como o conformista, onde o cliente se conforma com o que é proposto pelo fornecedor.

Lembre-se de que estamos disponíveis no Discord para bater um papo sobre o conteúdo, tirar dúvidas, fazer networking e muito mais! Nossos docentes, seus colegas e a pessoa de comunity management estão esperando por você!

REFERÊNCIAS

COCKBURN, A. **Writing Effective Use Cases**. [s.l.]. Addison Wesley, 2001.

EVANS, E. Domain-Driven Design, Tackling Complexity in the heart of Software. Boston: Pearson Education, 2003.

FOWLER, M. **Patterns of Enterprise Application Architecture**. [s.l.]. Addison-Wesley, 2002.

KHONONOV, V. **Learning Domain-Driven Design**. Sabastopol: O'Reilly Media, 2021.

MILLET, S. TUNE, N. **Patterns, Principles and Practicves of Domain-Driven Design**. Indianapolis: Wrox, 2015.

VERNON, V. **Implementing Domain-Driven Design**. Boston:. Pearson Education, 2013.

PALAVRAS-CHAVE

Contextos delimitados. Cliente-Fornecedor. Mapa de contexto.