

Engenharia de Softwares Escaláveis

Design Patterns e Domain-Driven Design com Java

Agenda

Etapas 6: Integração Entre Contextos Delimitados.

- Camada Anticorrupção - Parte 2.
- Q&A.



Camada Anticorrupção - Parte 2

Fachadas

É uma interface alternativa para um subsistema que simplifica o acesso do cliente e torna o subsistema mais fácil de usar.

Como sabemos exatamente quais funcionalidades do outro sistema queremos utilizar, podemos criar uma **fachada** que facilita e agiliza o acesso a essas funcionalidades e esconde o resto.

Deve ser escrita estritamente de acordo com o modelo do outro sistema, caso contrário, na melhor das hipóteses, espalhará a responsabilidade pela tradução em vários objetos e sobrecarregará a **fachada**. Na pior das hipóteses, acabará criando outro modelo, um que não pertence ao outro sistema ou ao seu próprio contexto delimitado.

A **fachada** pertence ao contexto delimitado do outro sistema. Apenas apresenta um rosto mais amigável e especializado para suas necessidades.

Adaptadores

Um **adaptador** é um wrapper que permite a um cliente usar um protocolo diferente daquele compreendido pelo implementador do comportamento.

Quando um cliente envia uma mensagem para um **adaptador**, ela é convertida em uma mensagem semanticamente equivalente e enviada ao “adaptado”.

A resposta é convertida e devolvida.

Para cada serviço que definimos, precisamos de um **adaptador** que suporte a sua interface e saiba fazer solicitações equivalentes do outro sistema ou de sua **fachada**.

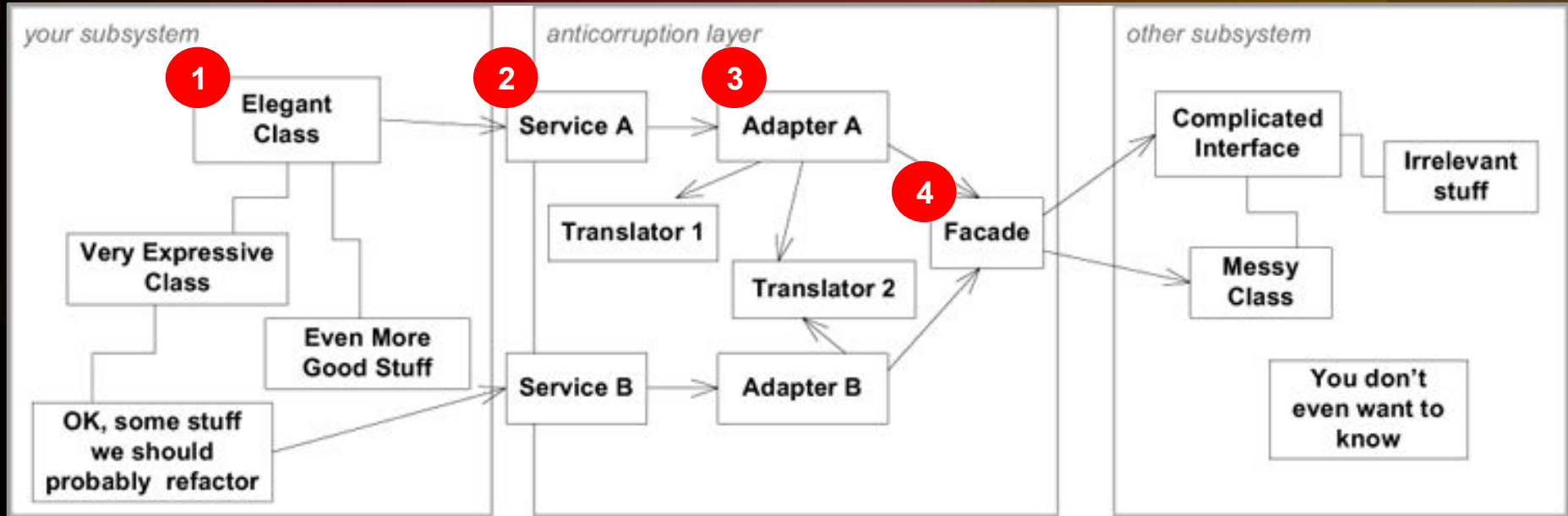
Tradutores

A função do **adaptador** é saber fazer uma solicitação.

A conversão real de objetos ou dados conceituais é uma tarefa distinta e complexa que pode ser colocada em seu próprio objeto, tornando ambos muito mais fáceis de entender.

Um **tradutor** pode ser um objeto leve que é instanciado quando necessário.

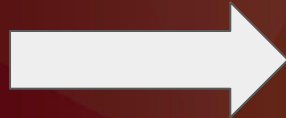
Não precisa de estado e não precisa ser distribuído, porque pertence aos **adaptadores** que os empregam.



Contexto A - ECommerce

Possui o modelo de Pedidos com entidades Pedido, ItemPedido, Cliente, etc.

O time usa DDD, linguagem ubíqua e regras de negócio claras.




Contexto B - Faturamento

É um sistema externo (ou legado dentro da empresa) que gera Notas Fiscais.

O modelo é diferente, usa termos contábeis, objetos anêmicos e um formato de API pouco amigável.

Queremos que o **ECommerce** envie os dados de Pedido para o **Faturamento**, mas sem poluir o modelo de domínio com detalhes do sistema legado.




ECommerce

```
class Pedido {  
    private String numero;  
    private Cliente cliente;  
    private List<ItemPedido> itens;  
    // getters...  
}
```

```
class Cliente {  
    private String nome;  
    private String cpf;  
    // getters...  
}
```

```
class ServicoDeFaturamento {  
    private FaturamentoAdapter adapter;  
  
    public ServicoDeFaturamento(FaturamentoAdapter adapter) {  
        this.adapter = adapter;  
    }  
  
    public NotaFiscal gerarNotaFiscal(Pedido pedido) {  
        return adapter.enviarPedidoParaFaturamento(pedido);  
    }  
}
```

Fachada

A white, multi-pointed starburst shape with the word "ECommerce" written in black text inside it.

ECommerce

Adaptador

```
class FaturamentoAdapter {  
    private SistemaLegadoFaturamento api;  
  
    public FaturamentoAdapter(SistemaLegadoFaturamento api) {  
        this.api = api;  
    }  
  
    public NotaFiscal enviarPedidoParaFaturamento(Pedido pedido) {  
        InvoiceRequestDTO dto = PedidoTranslator.toInvoiceDTO(pedido);  
        InvoiceResponseDTO response = api.emitirNotaFiscal(dto);  
        return PedidoTranslator.toNotaFiscal(response);  
    }  
}
```

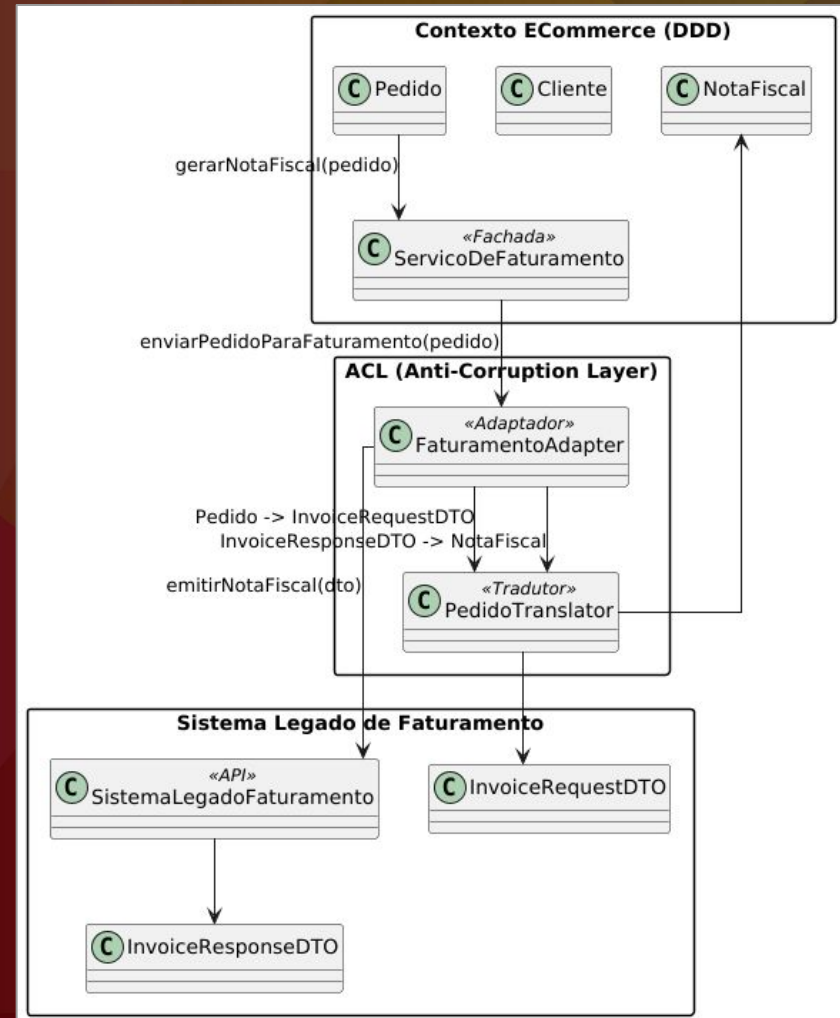
```
class PedidoTranslator {  
    public static InvoiceRequestDTO toInvoiceDTO(Pedido pedido) {  
        InvoiceRequestDTO dto = new InvoiceRequestDTO();  
        dto.setOrderNumber(pedido.getNumero());  
        dto.setCustomerName(pedido.getCliente().getNome());  
        dto.setCustomerTaxId(pedido.getCliente().getCpf());  
        // mapear itens etc.  
        return dto;  
    }  
  
    public static NotaFiscal toNotaFiscal(InvoiceResponseDTO response) {  
        return new NotaFiscal(response.getInvoiceNumber(), response.getTotalValue());  
    }  
}
```

```
class SistemaLegadoFaturamento {  
    public InvoiceResponseDTO emitirNotaFiscal(InvoiceRequestDTO dto) {  
        // Chamada SOAP/REST/XML para sistema legado  
        return new InvoiceResponseDTO("NF-12345", 199.90);  
    }  
}  
  
class InvoiceRequestDTO {  
    private String orderNumber;  
    private String customerName;  
    private String customerTaxId;  
}  
  
class InvoiceResponseDTO {  
    private String invoiceNumber;  
    private double totalValue;  
}  
  
class NotaFiscal {  
    private String numero;  
    private double valor;  
    public NotaFiscal(String numero, double valor) {  
        this.numero = numero;  
        this.valor = valor;  
    }  
}
```



Faturamento

1. O domínio de **ECommerce** pede `ServicoDeFaturamento.gerarNotaFiscal(pedido)`.
2. A **Fachada** recebe o pedido e aciona o **Adaptador**.
3. O **Adaptador** usa os **Tradutores** para converter `Pedido` → `InvoiceRequestDTO`.
4. O **Adaptador** chama o Sistema de **Faturamento**.
5. A resposta é traduzida de volta para `NotaFiscal` (objeto do domínio de **ECommerce**).
6. O domínio continua usando apenas seus próprios conceitos (`Pedido`, `NotaFiscal`), protegido da linguagem do outro sistema.



Benefícios da ACL

- **Isolamento**: protege os sistemas de mudanças na integração.
- **Abstração**: oferece uma interface unificada para interação com os sistemas.
- **Flexibilidade**: permite lidar com diferentes tipos de dados e regras de conversão.
- **Reusabilidade**: a lógica de adaptação pode ser reutilizada em outras integrações.

A **Camada Anticorrupção** é uma ferramenta valiosa para integrar sistemas com modelos de dados díspares, promovendo flexibilidade, modularidade e facilidade de manutenção na sua arquitetura DDD.



Q&A

Qual é a relação de DDD com Arquitetura em Camadas e Arquitetura de Software?

DDD é um tipo de Arquitetura de Software?

Aspecto	Domain-Driven Design (DDD)	Arquitetura em Camadas	Arquitetura de Software
O que é?	Conjunto de princípios e práticas para modelar sistemas a partir do domínio de negócio .	Um estilo arquitetural que organiza o sistema em camadas (UI, aplicação, domínio, infraestrutura).	A estrutura global do sistema e suas decisões técnicas de organização .
Foco	Capturar e expressar o conhecimento do domínio de forma clara e compartilhada.	Garantir separação de responsabilidades entre partes do sistema.	Garantir qualidades técnicas (desempenho, escalabilidade, manutenibilidade, etc.).
Prescrição	Não prescreve arquitetura ou camadas específicas.	Define uma forma específica de organizar código em camadas.	Define princípios e estilos de alto nível (padrões, estilos, comunicação entre componentes).
Qual é a Pergunta?	“Como representar fielmente as regras de negócio?”	“Em qual camada do sistema coloco essa lógica?”	“Qual estilo arquitetural vamos adotar para o sistema?”
Compatibilidade	Pode ser usado em várias arquiteturas: camadas, hexagonal, clean, event-driven, CQRS etc.	Pode ser usada junto com DDD, mas não é obrigatória.	Pode adotar DDD como abordagem para organizar o domínio .

Qual padrão de integração nunca deve ser usado para um subdomínio principal?

- A. Kernel compartilhado.
- B. Serviço de hospedagem aberta.
- C. Camada anticorrupção.
- D. Caminhos separados.

D Duplicar a funcionalidade de um subdomínio central em múltiplos contextos delimitados frustrará a possibilidade de evoluí-lo no futuro, pois todas as mudanças (frequentes) terão que ser coordenadas entre os dois contextos delimitados.

Qual subdomínio downstream tem mais probabilidade de implementar uma camada anticorrupção?

- A. Subdomínio principal.
- B. Subdomínio de suporte.
- C. Subdomínio genérico.

A O objetivo do padrão da camada anticorrupção é proteger o modelo downstream. É especialmente relevante se o downstream for um subdomínio principal.

Qual subdomínio upstream tem maior probabilidade de implementar um serviço de hospedagem aberta? Selecione todas as opções aplicáveis.

- A. Subdomínio principal.
- B. Subdomínio de suporte.
- C. Subdomínio genérico.

A O objetivo do padrão de serviço de host aberto é proteger os consumidores de mudanças no contexto delimitado upstream. Isso é especialmente relevante se o upstream for um subdomínio principal, pois eles são altamente voláteis.

Qual padrão de integração, em certo sentido, viola os limites de propriedade de contextos delimitados?

- A. Parceria.
- B. Kernel compartilhado.
- C. Caminhos separados.
- D. Nenhum padrão de integração deve quebrar os limites de propriedade dos contextos delimitados.

B O padrão de kernel compartilhado permite que várias equipes evoluam juntos uma parte de um modelo pertencente a um contexto delimitado. Como esse padrão é a exceção à regra e não a regra em si, ele deve ser usado com moderação e seu uso deve ser justificado.