
Embarcadero Conference
embarcadero

Padrões de Projeto em Delphi

[Guinther Pauli]

embarcadero

Embarcadero Conference

[Design Patterns]

- Uma premissa básica no desenvolvimento de software é a necessidade de evoluir
- Sistemas de software reais evoluem e tornam-se mais complexos ao longo do tempo
- Novos requisitos, novas tecnologias, difícil manutenção, o software se afasta do seu projeto original
- Evoluir para uma nova plataforma, nova arquitetura, um novo banco de dados, um novo engine de acesso a dados, um novo framework gráfico, nova versão do Delphi

[Design Patterns]

- Estima-se que de 50% a 70% do custo total de um sistema de software é gasto com sua manutenção
- O que torna um software difícil de manter e evoluir: código rígido, difícil de ler, complexidade, acoplamento entre units / classes, separação incorreta de responsabilidades, duplicação, falta de padrões, condicionais, componentes de terceiros etc.
- Lembre-se, você passará a maior parte da sua vida clicando em “File > Open” e não “File > New”

[Design Patterns]

- Para muitas empresas, senão a maioria delas, exceto startups, o foco principal não é criar software rapidamente, mas software que seja fácil de ser mantido ao longo do tempo, pois isso reduz custos a médio e longo prazo
- Aplicar padrões de projeto no contexto ágil evolutivo, entra como a ponte, a chave, o elo, a solução para se criar software com código sempre limpo e funcional, preparado para mudanças, que sempre existirão

[Design Patterns]

- Padrões de Projeto – Design Patterns - são soluções reutilizáveis usadas para resolver problemas comumente encontrados em sistemas orientados a objetos
- Usar padrões de projeto torna mais fácil reutilizar soluções de arquiteturas bem-sucedidas
- Maximizam a reutilização de código, facilitam a manutenção e evolução de sistemas de software
- Padrões ajudam a projetar sistemas de software mais bem preparados para mudanças

[Design Patterns]

- Uma boa prática é evoluir um código existente, tendo como resultado final dessa transformação um código em conformidade com um padrão de projeto
- Serve como uma alternativa ao chamado Grande Projeto Antecipado (GPA), ou *excesso de engenharia* – não se cria inicialmente um grande projeto, prevendo possibilidades de mudanças que possam ocorrer - complexidade desnecessária
- “Programadores não são videntes”
- Evita a *escassez de engenharia*, ou evoluir um sistema de software sem nunca aplicar um padrão - *débito de projeto,* ou *débito técnico*

[Design Patterns]

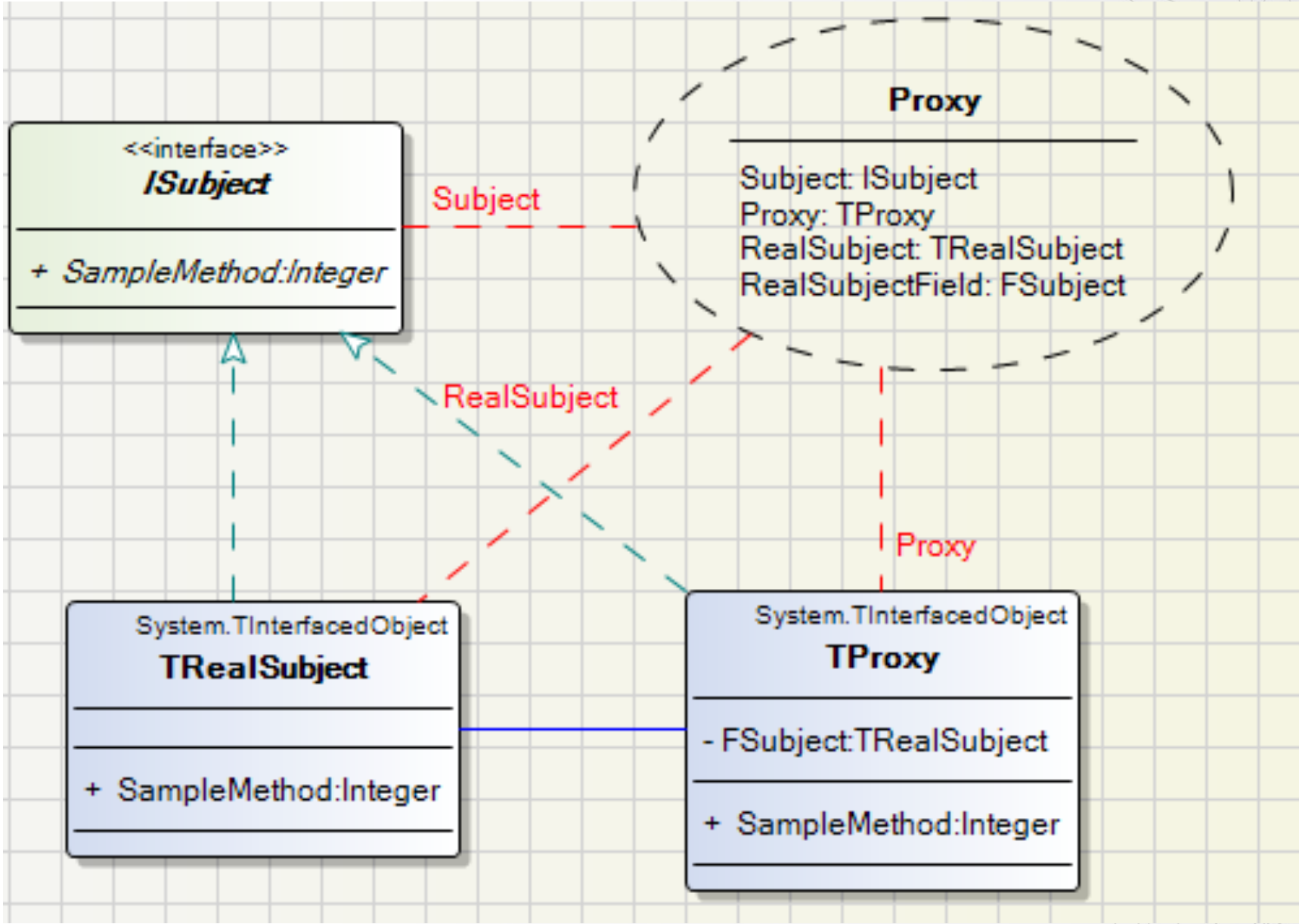
Exemplos práticos:

- Proxy
- Adapter
- Façade
- Strategy / State
- Chain Of Responsibility
- Observer
- Template Method

[Proxy]

- **Intenção**: Fornecer um substituto ou espaço reservado para outro objeto para controlar o acesso a ele.

[Proxy]


Proxy
Subject: ISubject
Proxy: TProxy
RealSubject: TRealSubject
RealSubjectField: FSubject
<<interface>>
ISubject
+ SampleMethod:Integer
Subject
RealSubject
Proxy
System.TInterfacedObject
TRealSubject
+ SampleMethod:Integer
System.TInterfacedObject
TProxy
- FSubject:TRealSubject
+ SampleMethod:Integer

[Adapter]

- **Intenção**: Converte a interface de uma classe em outra interface esperada pelos clientes. Permite que classes trabalhem em conjunto, pois de outra forma não poderiam devido a terem interfaces incompatíveis.

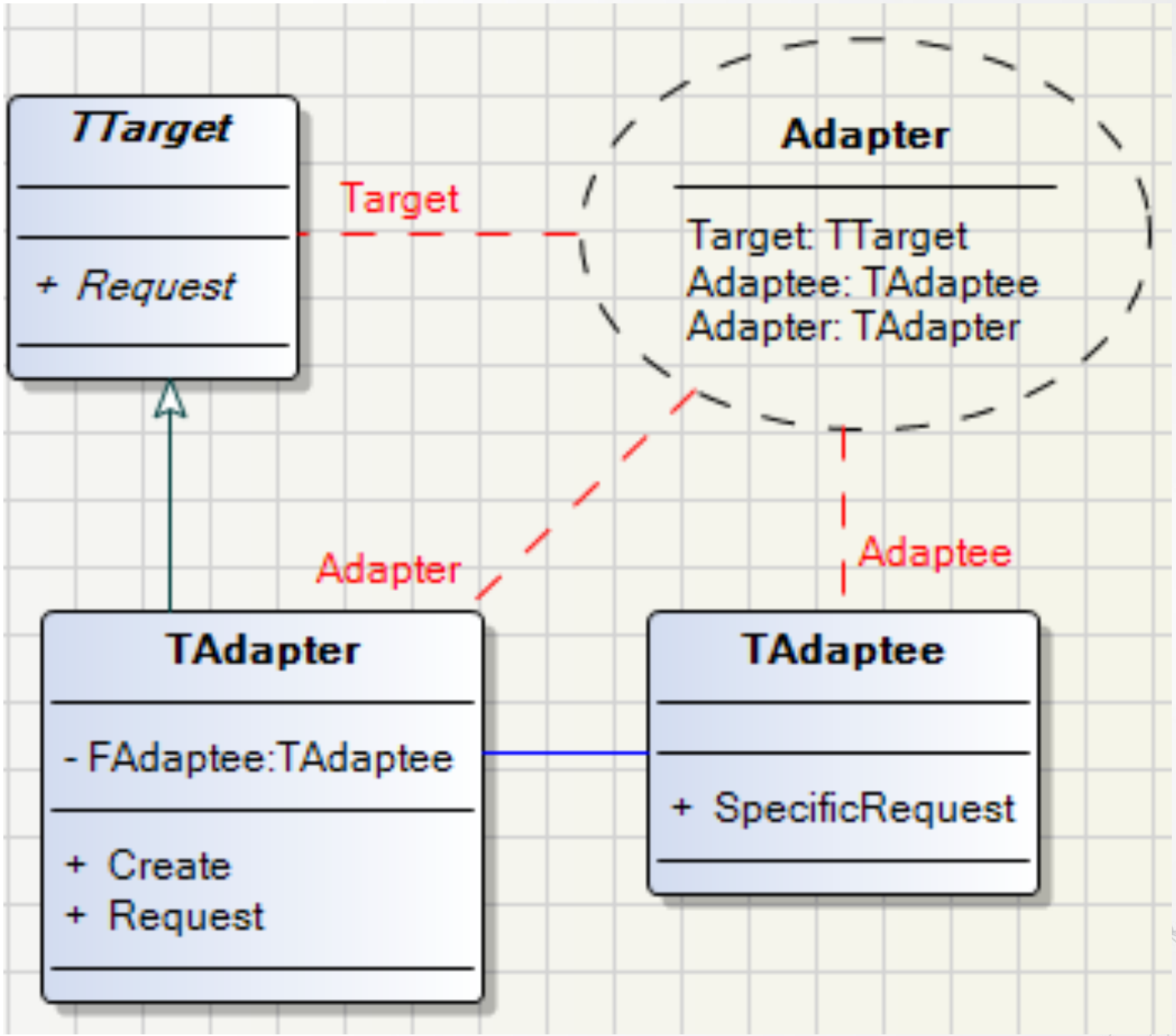
[Adapter]

Adaptee

Adapter

Target

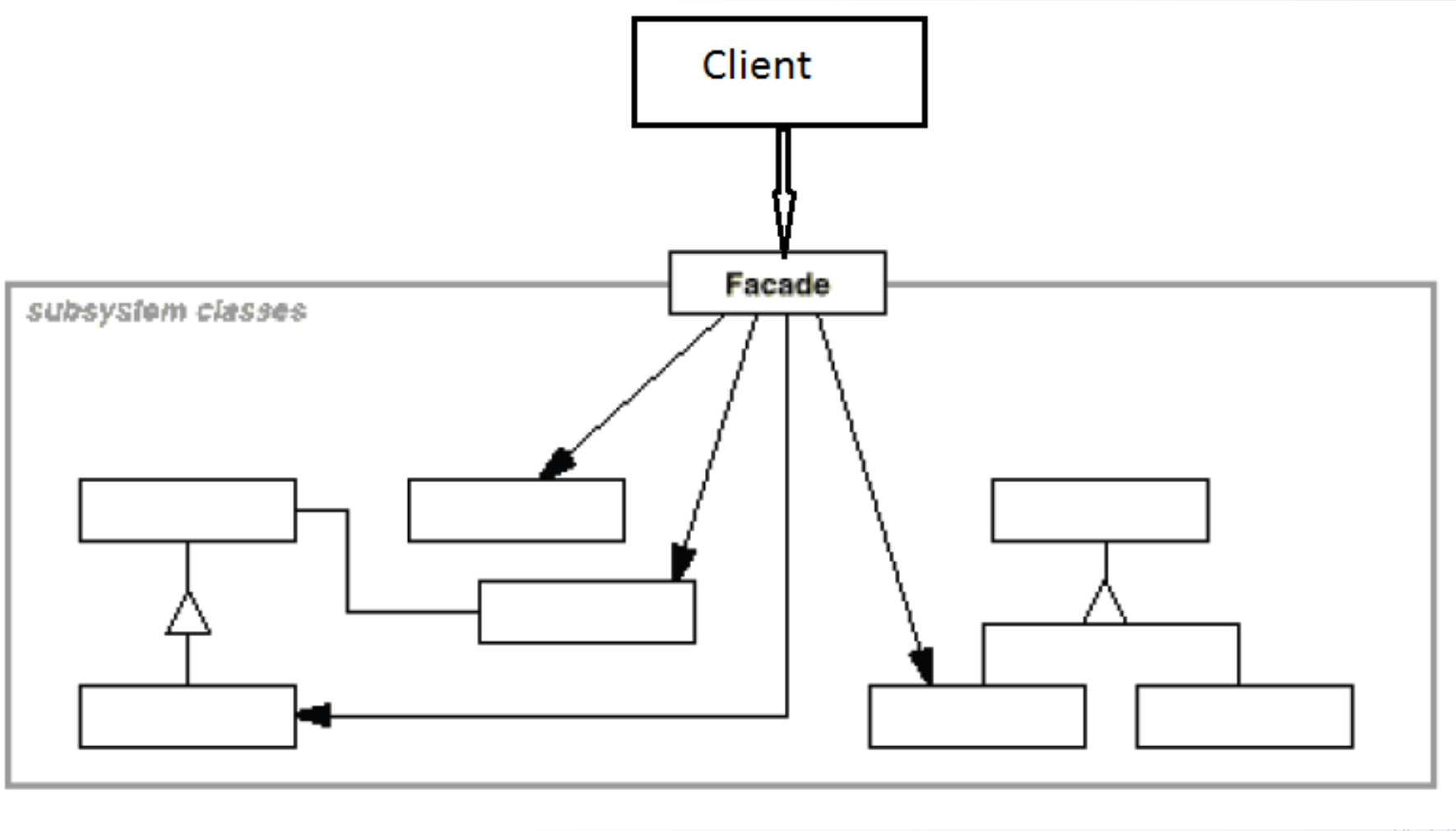
[Adapter]



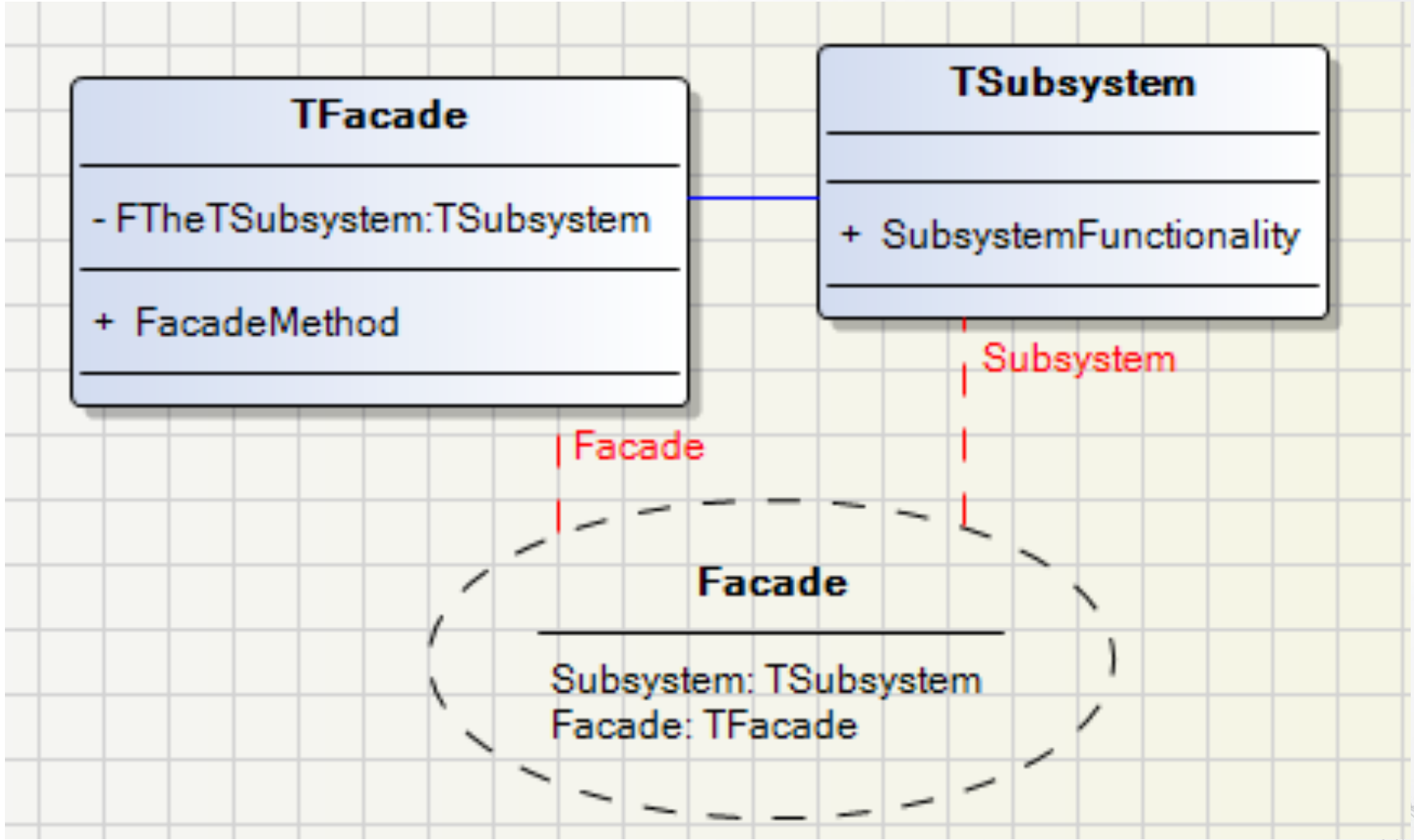
[Façade]

- **Intenção**: Fornecer uma interface unificada para um conjunto de interfaces em um subsistema. Facade define uma interface de nível mais elevado que faz o subsistema mais fácil de usar.

[Façade]



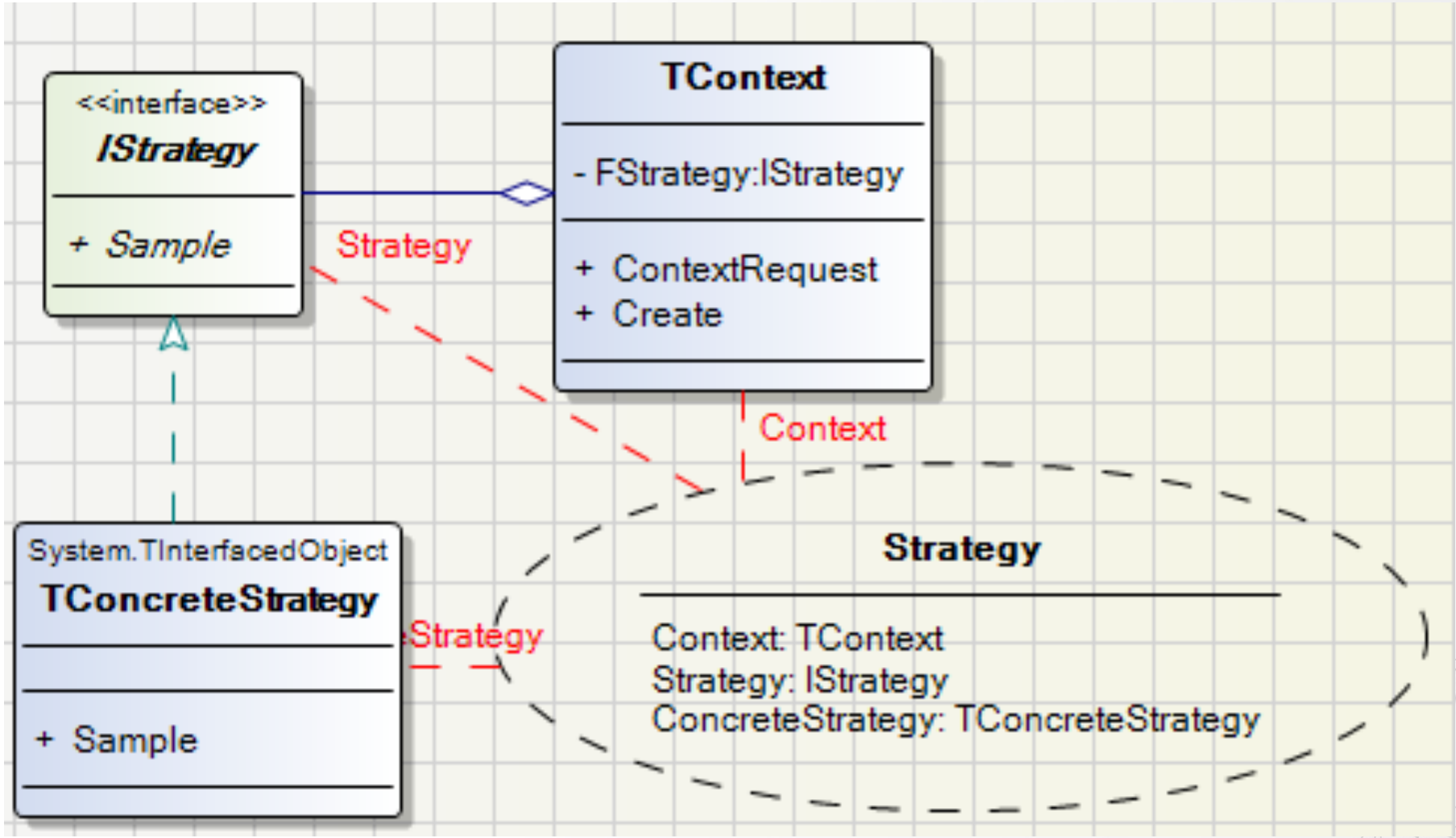
[Façade]



[Strategy]

- **Intenção**: Definir uma família de algoritmos, encapsular cada um, e torná-los intercambiáveis. Permite que o algoritmo varie independentemente dos clientes que o utilizam.

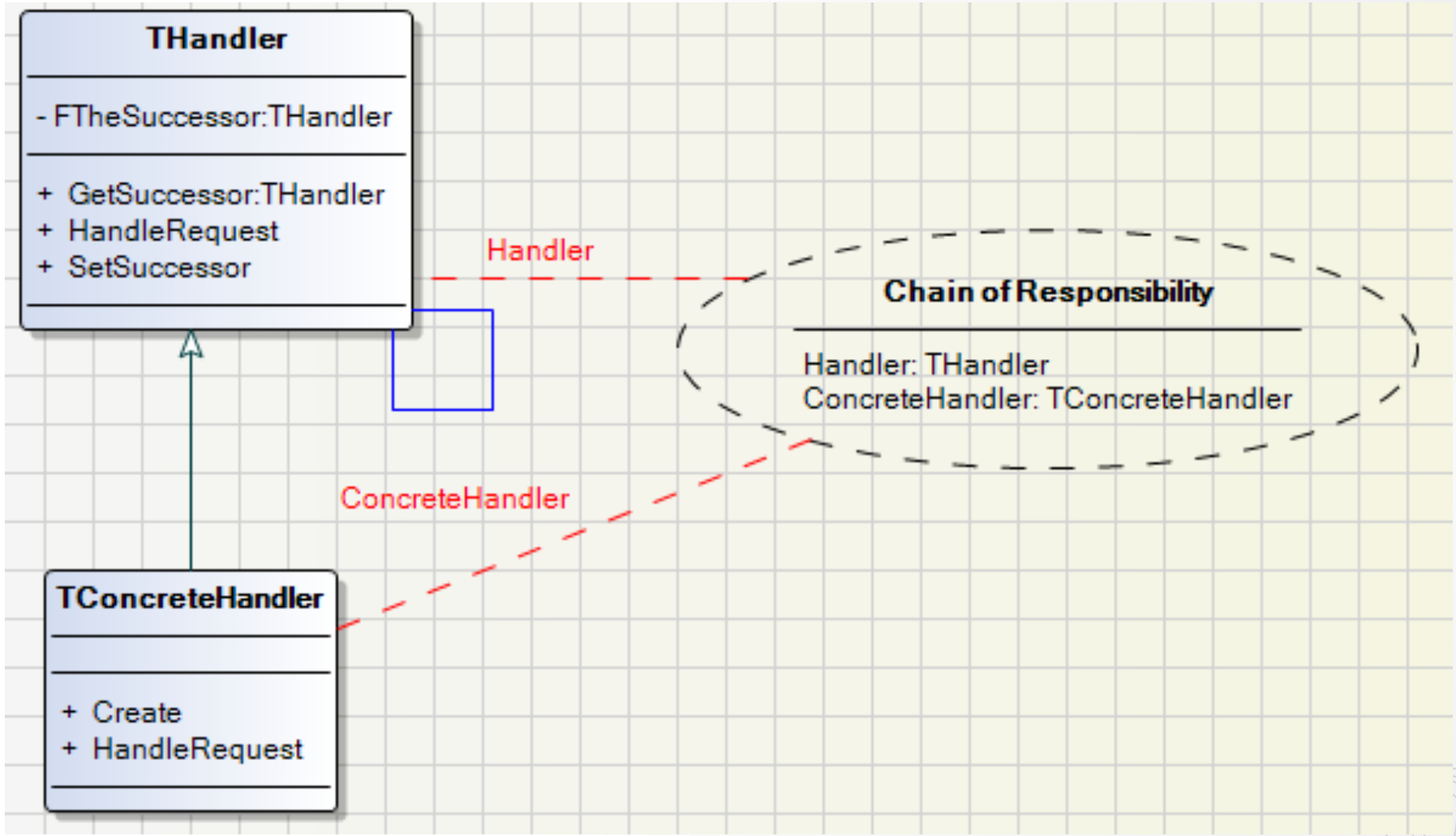
[Strategy]


<<interface>>
IStrategy
+ Sample
TContext
- FStrategy:IStrategy
+ ContextRequest
+ Create
Strategy
Context
System.TInterfacedObject
TConcreteStrategy
+ Sample
Strategy
Strategy
Context: TContext
Strategy: IStrategy
ConcreteStrategy: TConcreteStrategy

[Chain of Responsibility]

- **Intenção**: Evitar o acoplamento do remetente de uma solicitação ao seu recepetor, ao dar a mais de um objeto a oportunidade de tratar a solicitação. Encadear os objetos receptores, passando a solicitação ao longo da cadeia até que um objeto a trate.

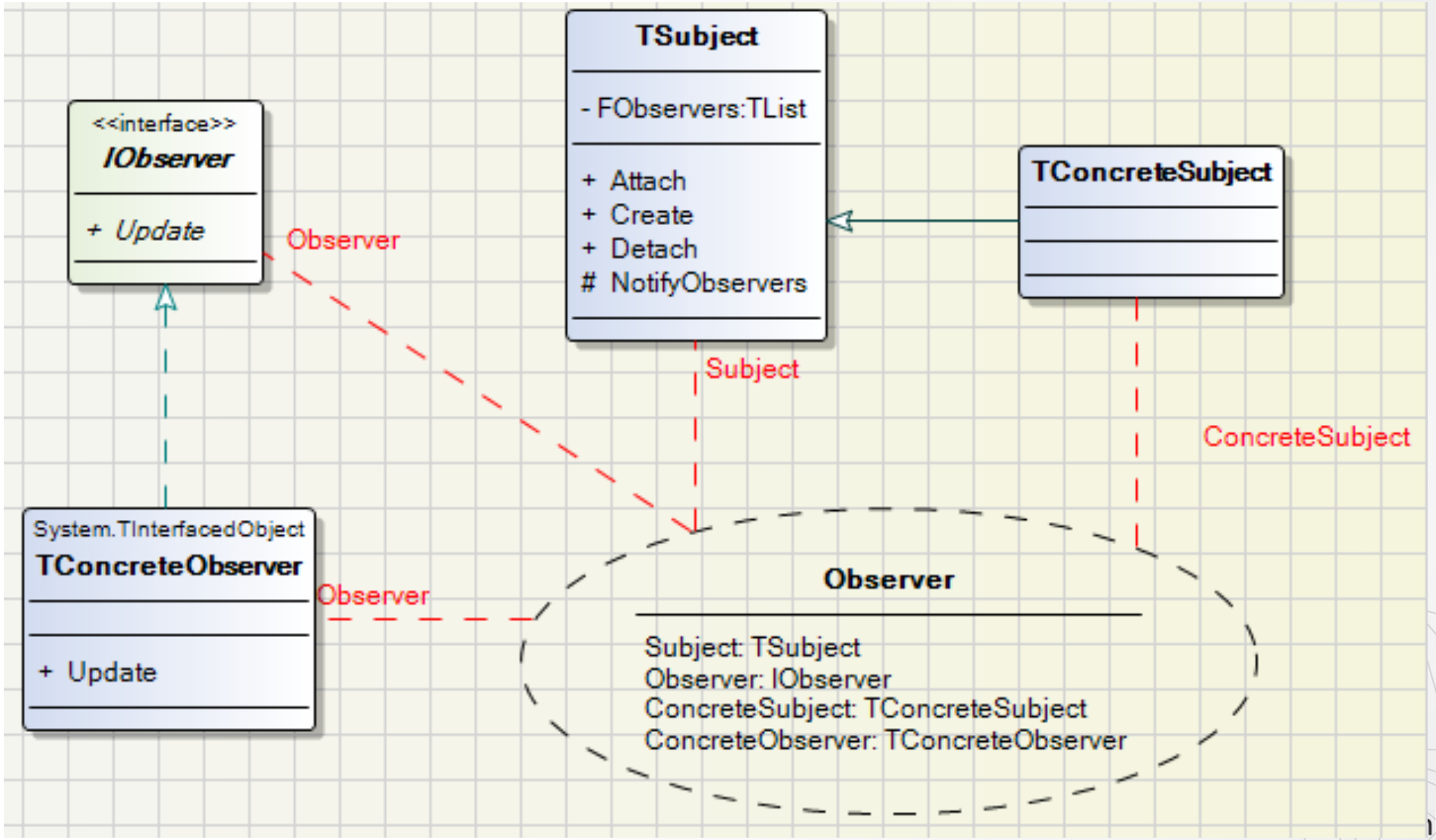
[Chain of Responsibility]



[Observer]

- **Intenção**: Define uma dependência um-para-muitos entre objetos de modo que quando um objeto muda de estado, todos os seus dependentes são notificados e atualizados automaticamente.

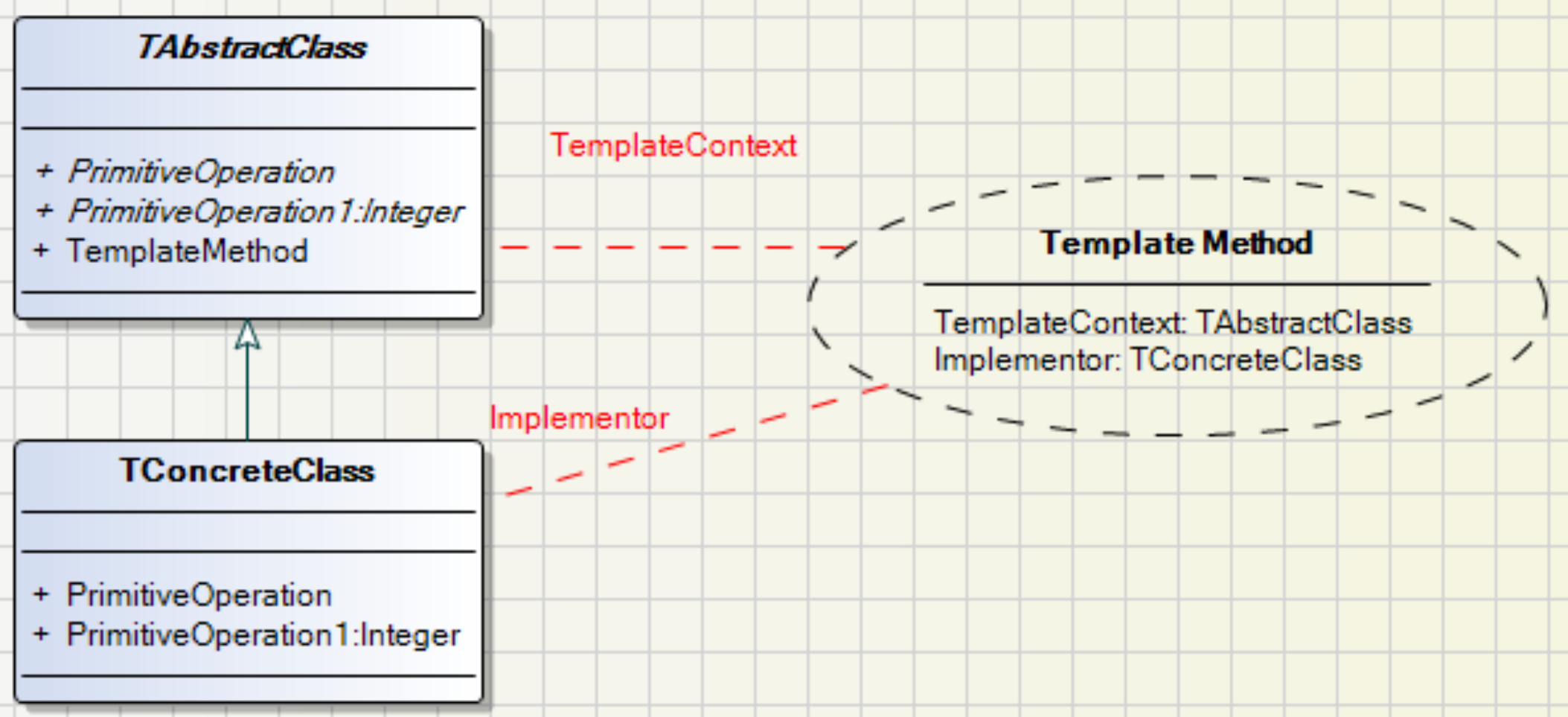
[Observer]



[Template Method]

- **Intenção**: Define o esqueleto de um algoritmo em uma operação, adiando alguns passos para subclasses. Permite que subclasses redefinam determinadas etapas de um algoritmo sem alterar a estrutura do algoritmo.

[Template Method]



@ guintherpauli@gmail.com

/guintherpauli

/guintherpauli


Embarcadero Conference
embarcadero