Engenharia de Software

Arquitectura de Software
Padrões de Arquitectura de Software

Luís Morgado

Instituto Superior de Engenharia de Lisboa
Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

Padrões de Arquitectura

- Os *padrões de arquitectura* de software são soluções gerais, reutilizáveis, para problemas comuns no desenvolvimento de software
- São um exemplo de *heurísticas* de desenvolvimento
  - Meios de tornar o desenvolvimento mais expedito, neste caso com base em soluções conhecidas (padrões) e boas práticas que orientam o processo de desenvolvimento de um sistema
- Têm a forma de descrições ou modelos de como resolver um determinado tipo de problema, que podem ser utilizados em diferentes situações
- Consolidam boas práticas que foram organizadas e formalizadas para resolver problemas comuns ao desenvolver um sistema
- Permitem acelerar o processo de desenvolvimento ao proporcionar soluções já anteriormente testadas
- Permitem identificar por antecipação questões não imediatamente aparentes do problema a resolver

Padrões de Arquitectura

- No âmbito da Engenharia de Software
  - **Solução geral**, reproduzível, para um problema de ocorrência frequente
  - **Não é uma definição acabada** que possa ser directamente implementada
  - **É uma descrição (padrão) acerca de como resolver um problema** que pode ser utilizada em diferentes situações
- Utilização
  - Podem permitir **acelerar o processo de desenvolvimento** ao proporcionar soluções já anteriormente testadas
  - Podem permitir **identificar por antecipação questões não imediatamente aparentes** do problema a resolver
  - Disponibilizam **soluções de carácter geral** não comprometidas com um problema específico

Padrões de Arquitectura

- Elementos base
  - **Designação**
    - Para comunicação e documentação
  - **Problema**
    - Problema abordado
    - Define o contexto de aplicação
  - **Solução**
    - **Estrutura**
      - Partes e relações que constituem o padrão
    - **Comportamento**
      - Interacção entre as partes
  - **Consequências**
    - Consequências da utilização do padrão, nomeadamente, em termos de benefícios, desvantagens e alternativas

Padrões de Arquitectura

- Critérios de classificação
  - **Nível de arquitectura**
    - Mecanismos
      - Agregados de elementos
      - Funcionalidade local
    - Subsistemas
      - Agregados de mecanismos
      - Funcionalidade global
  - **Domínio de problema**
    - Padrões de criação
      - Relacionados com a abstracção do processo de instanciação de objectos
    - Padrões de estrutura
      - Relacionados com a forma como classes e objectos se relacionam em estruturas com finalidades específicas
    - Padrões de comportamento
      - Relacionados com aspectos algorítmicos e de atribuição de responsabilidades comportamentais

Padrões de Mecanismos

Exemplos

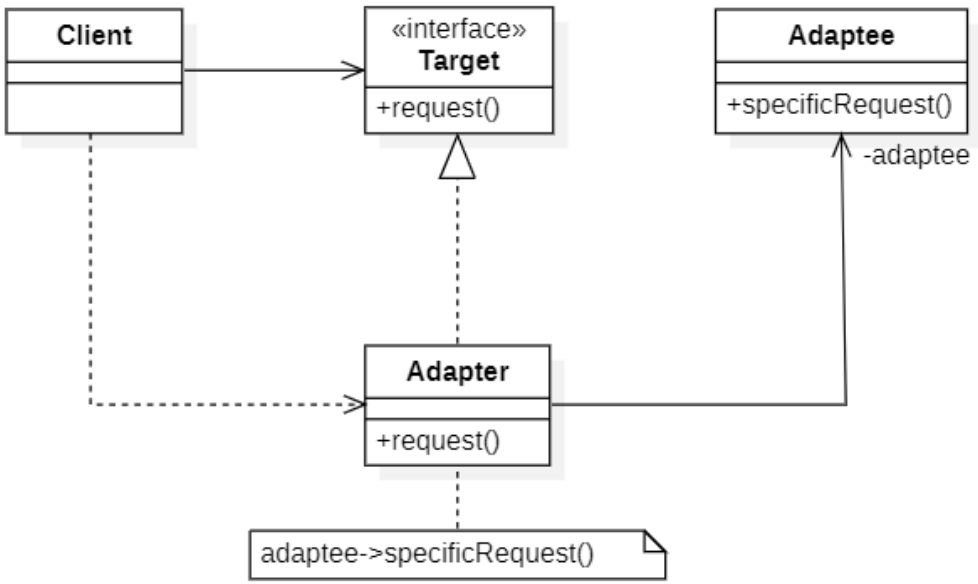
- Adaptador (*Adapter*)
  - Padrão estrutural
  - Converte a interface de uma classe noutra interface que os clientes esperam, permitindo que classes com interfaces incompatíveis possam ser utilizadas em conjunto
- Fachada (*Facade*)
  - Padrão estrutural
  - Fornece uma interface unificada para um conjunto de interfaces de um subsistema, definindo uma interface de maior abstracção que torna o subsistema mais fácil de utilizar
- Observador (*Observer*)
  - Padrão comportamental
  - Define uma dependência de notificação de um para muitos, entre objectos, de modo que quando um objecto muda de estado, todos os seus dependentes são notificados e actualizados
- Comando (*Command*)
  - Padrão comportamental
  - Define um pedido de realização de uma acção como um objeto encapsulado, permitindo assim parametrizar clientes com diferentes pedidos, independentemente da forma como são executados

Padrão Adaptador (*Adapter*)

- **Problema**
  - Necessidade de utilização de uma classe existente com uma interface que não corresponde ao que se pretende
- **Solução**
  - Criação de uma classe intermediária, o *adaptador*, que realiza a adaptação entre a classe que possui a interface incompatível (*adaptado*) e a classe que espera uma interface específica (*alvo*)
  - O adaptador implementa a interface de *alvo* e encapsula uma instância de *adaptado*
- **Consequências**
  - Reutilização de código: possibilita a utilização de classes já existentes mesmo com interfaces não compatíveis
  - Redução de acoplamento: possibilita a utilização de classes incompatíveis mesmo sem conhecer os seus detalhes
  - Integração de sistemas legados: especialmente útil quando se trabalha com sistemas legados, pois permite a integração de componentes antigos com novos componentes sem necessidade de modificar código existente

Padrão Adaptador (*Adapter*)

Estrutura



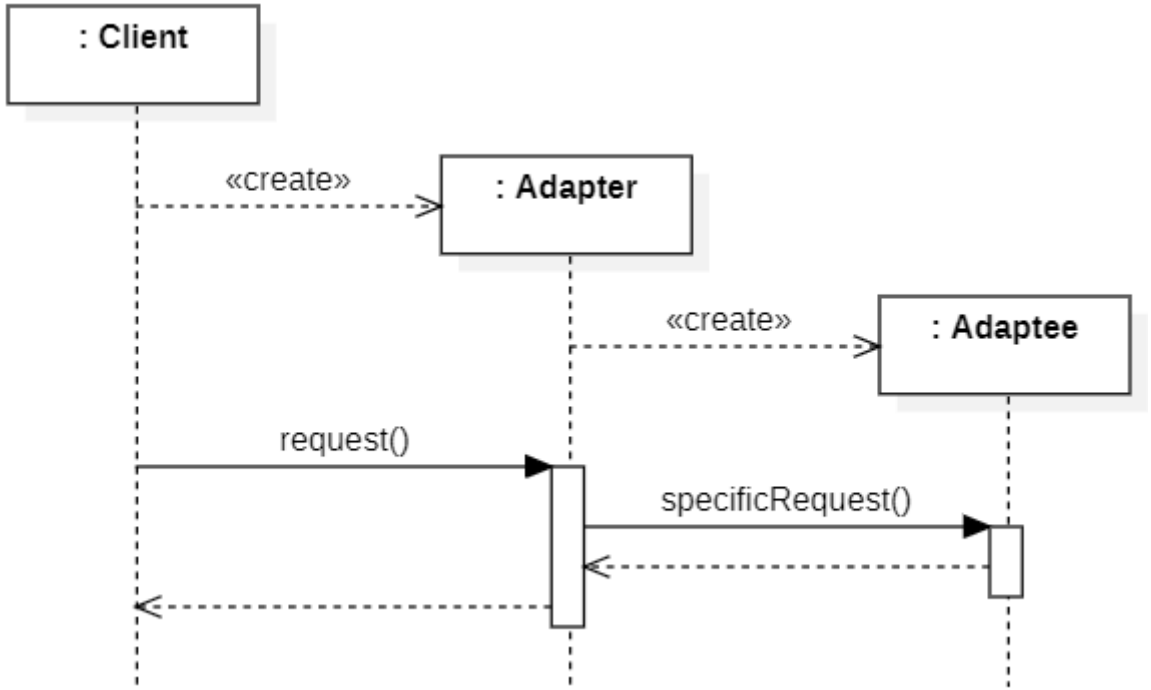
Participantes

- Cliente (*Client*)
  - Colabora com objectos que estão em conformidade com a interface alvo
- Alvo (*Target*)
  - Define a interface específica do domínio que o cliente utiliza
- Adaptado (*Adaptee*)
  - Define uma interface existente que precisa de ser adaptada
- Adaptador (*Adapter*)
  - Adapta a interface do adaptado à interface alvo

Padrão Adaptador (*Adapter*)

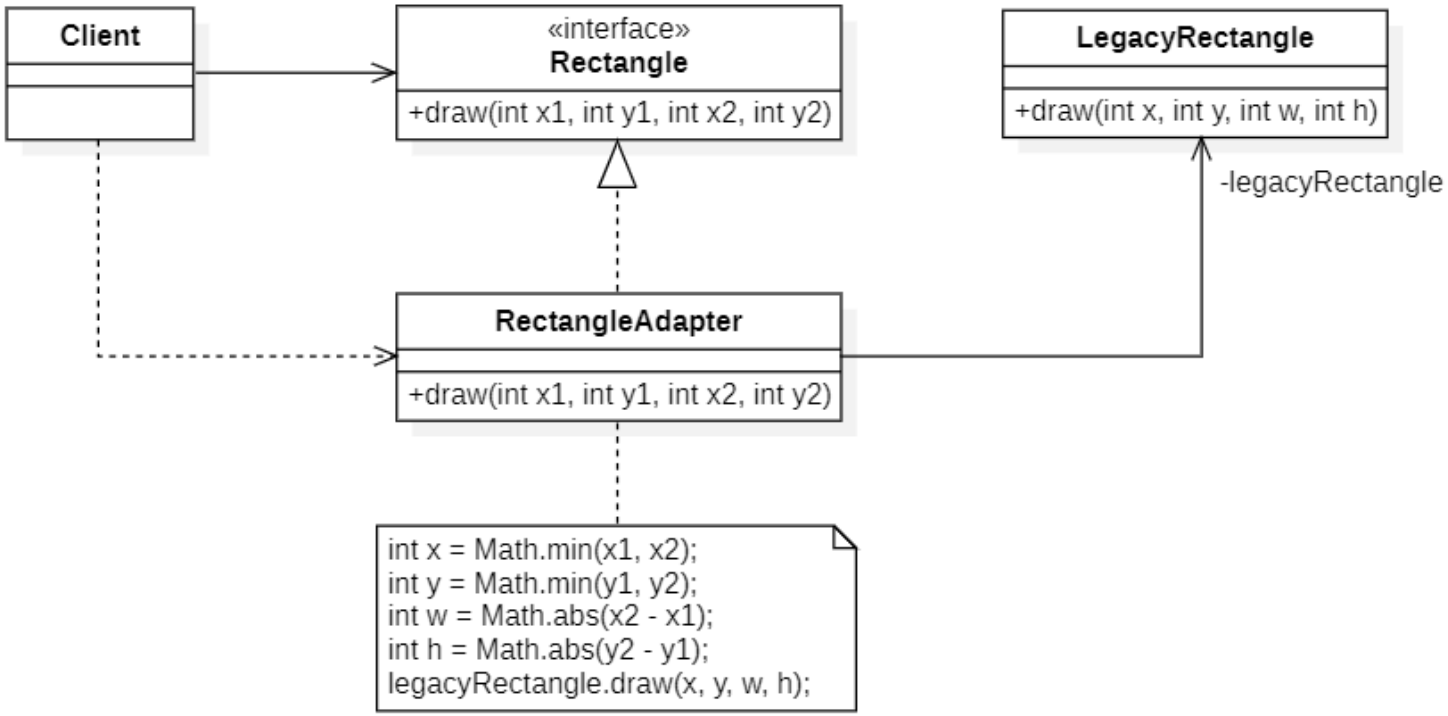
Comportamento

- Os clientes evocam operações sobre uma instância de adaptador
- O adaptador evoca as operações respectivas sobre o adaptado que as executa



Padrão Adaptador (*Adapter*)

Exemplo

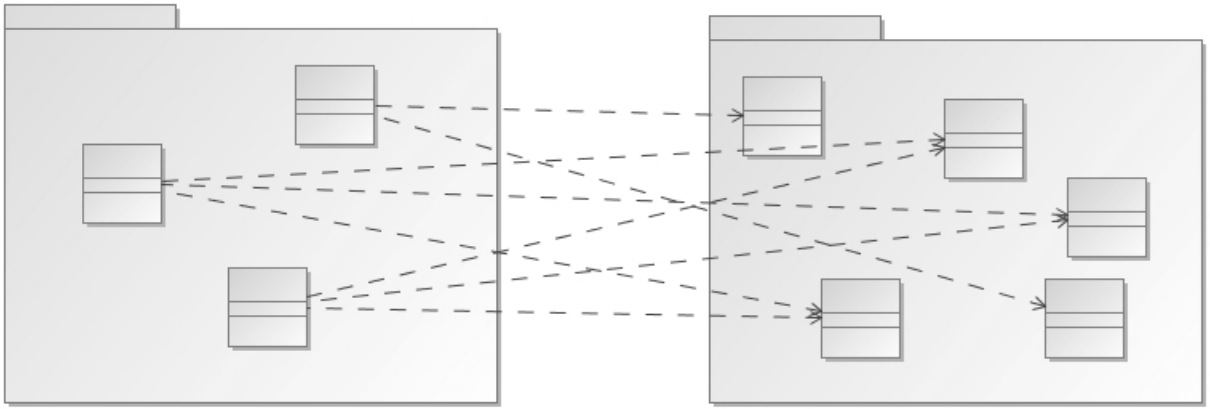


Padrão Fachada (*Facade*)

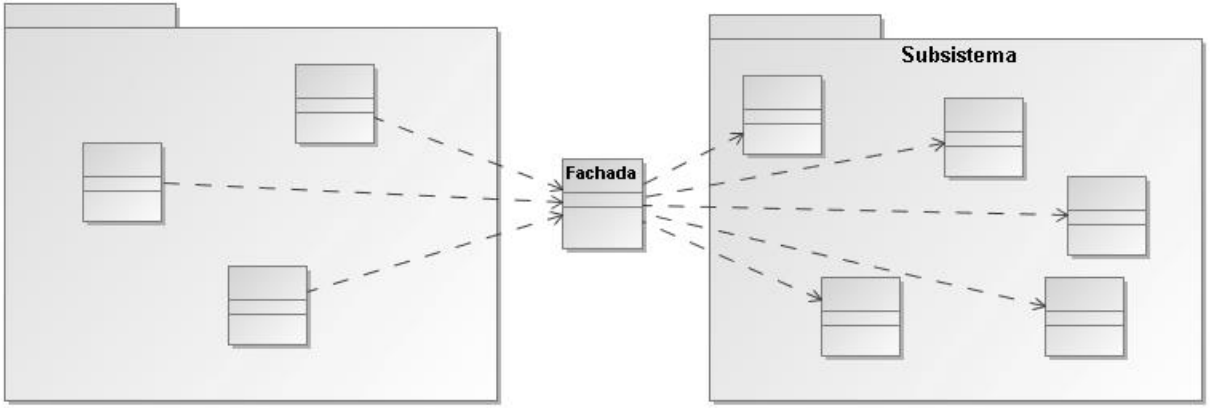
- **Problema**
  - Dificuldade de interacção com interfaces múltiplas e complexas
- **Solução**
  - Disponibilizar uma interface uniforme a partir de um conjunto de interfaces internas de um sistema
- **Consequências**
  - Encapsular um subsistema complexo com base num objecto mais simples
  - Permite reduzir o esforço de aprendizagem para utilização do subsistema
  - Promove a redução do nível de acoplamento
  - Pode limitar a flexibilidade de utilização do sistema

Redução de Acoplamento

Exemplo de arquitectura com elevado acoplamento devido à exposição das partes internas de um subsistema



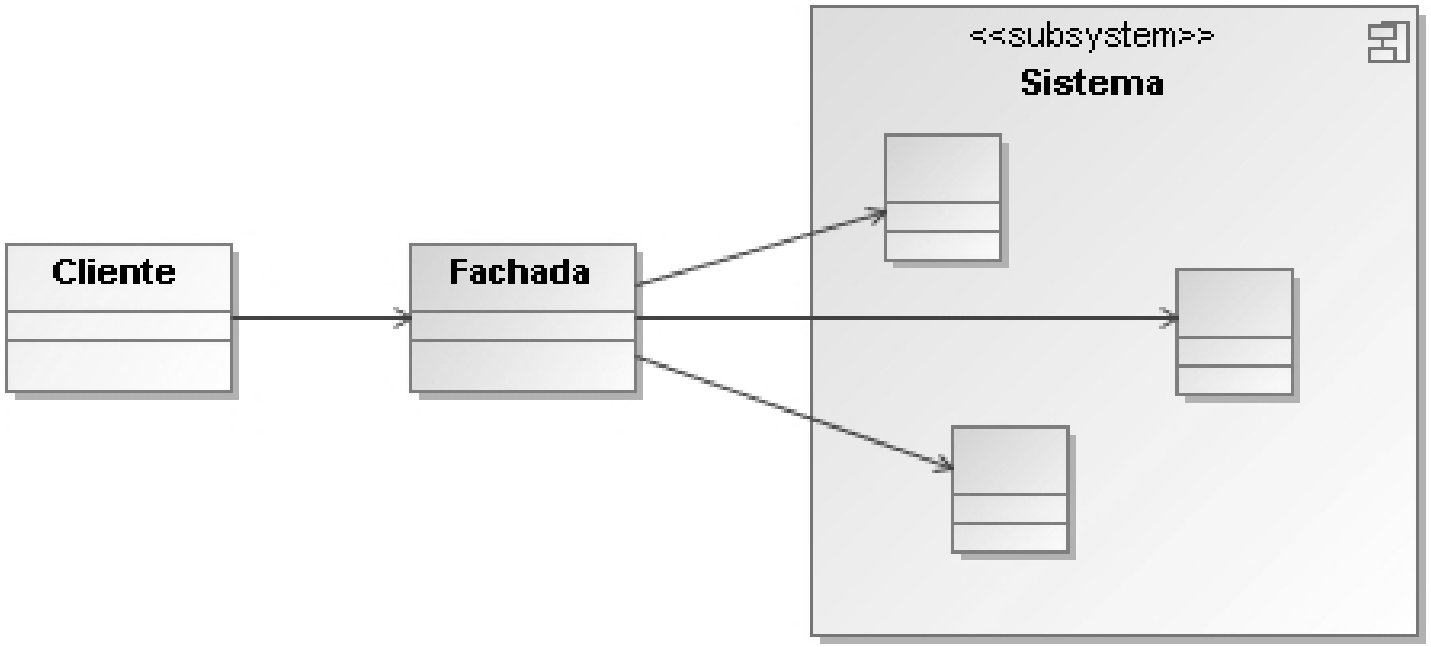
Como melhorar a arquitectura?
Encapsular e abstrair o acesso às partes internas através de uma fachada



Padrão Fachada (*Facade*)

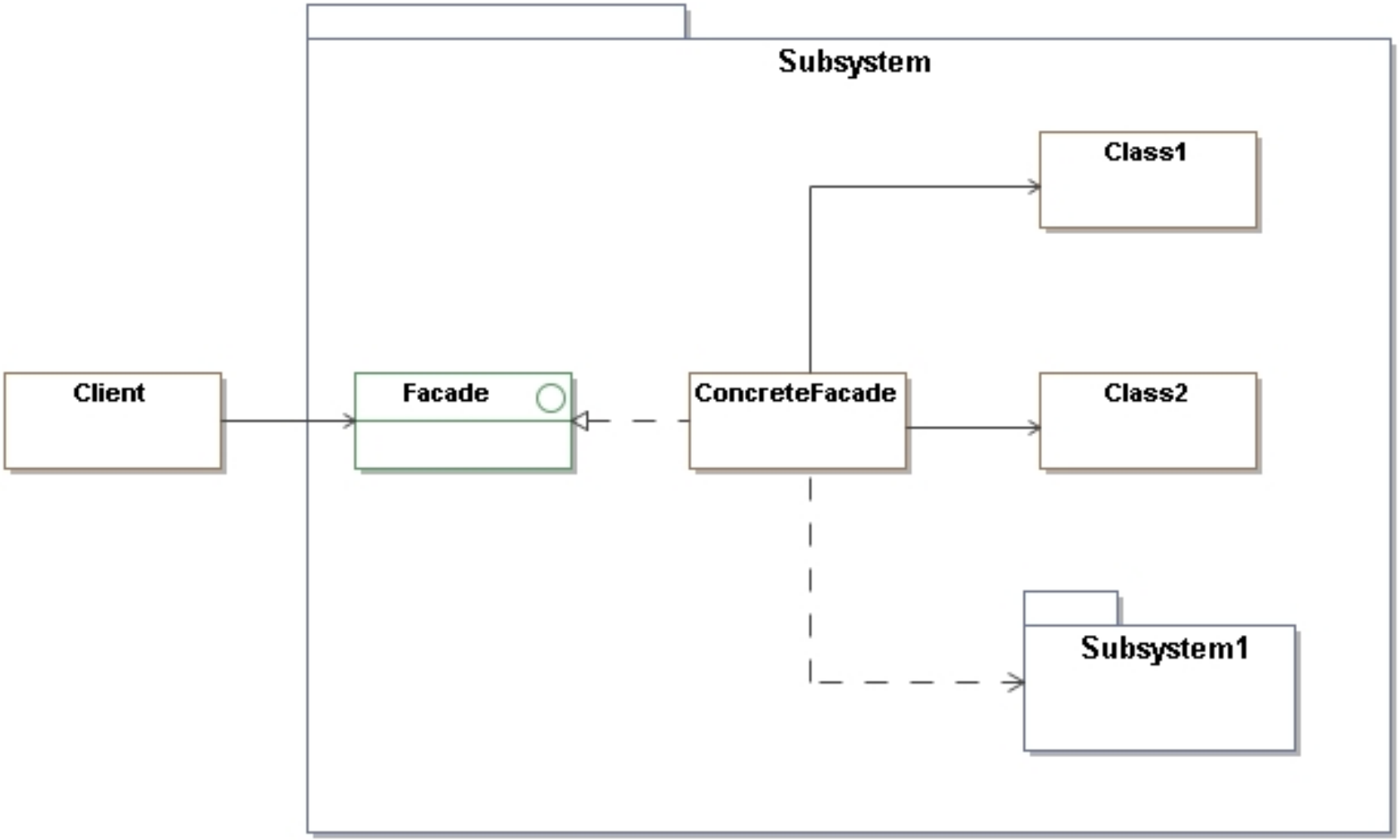
Aplicação de princípios de arquitectura

- Princípio do encapsulamento
- Princípio da minimização de acoplamento



Padrão Fachada (*Facade*)

Estrutura

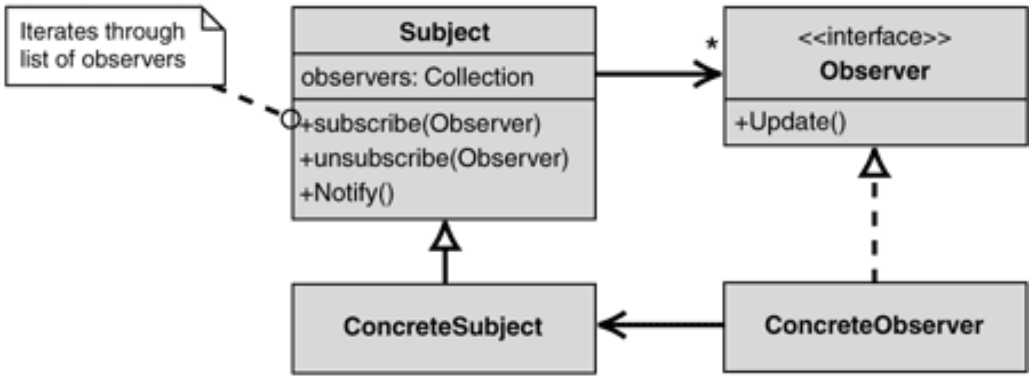


Padrão Observador (*Observer*)

- **Problema**
  - Num sistema, os diferentes objectos devem colaborar no sentido de atingir os objectivos globais desse sistema
  - Essa colaboração implica frequentemente que alguns objectos necessitem de informar outros objectos acerca de alterações no seu estado interno
  - Como pode um objecto notificar alterações do seu estado interno a outros objectos sem ter uma dependência desses objectos definida a priori?
- **Solução**
  - Manter informação actualizada dinamicamente, acerca dos objectos que possam estar interessados em ser notificados das alterações de estado
- **Consequências**
  - Redução do acoplamento: um sujeito de observação não precisa de conhecer os detalhes específicos de cada observador, o que permite reduzir a dependência entre as partes envolvidas
  - Facilidade de evolução e manutenção: o sujeito e os observadores estão desacoplados, pelo que alterações numa parte não tem implicações noutras partes, promovendo a flexibilidade e facilidade de evolução e manutenção do código
  - **Desvantagem**: possibilidade de ciclos de actualização encadeados gerando bloqueios de execução

Padrão Observador (*Observer*)

Estrutura



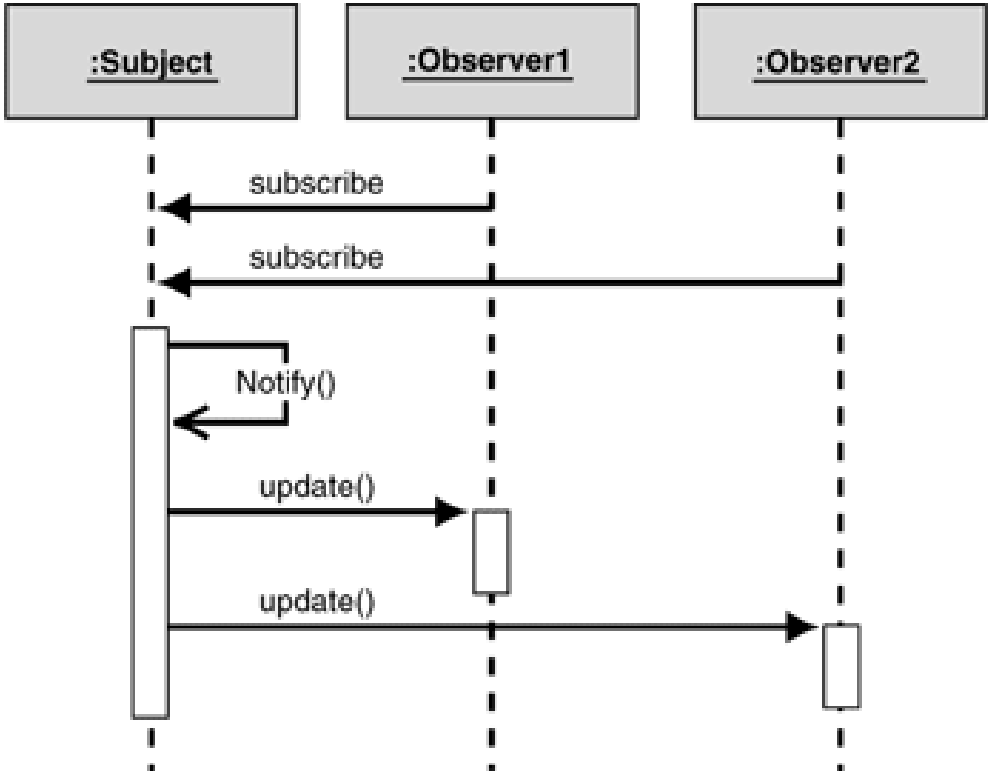
.NET Architecture Center (MSDN)
Enterprise Solution Patterns Using Microsoft .NET

Participantes

- Sujeito (*Subject*)
  - Conhece os observadores, disponibilizando a funcionalidade de registo e cancelamento de subscrições de notificação
- Sujeito concreto (*ConcreteSubject*)
  - Mantém estado e notifica os observadores em caso de alteração de estado
- Observador (*Observer*)
  - Define uma interface de notificação de alterações do sujeito
- Observador concreto (*ConcreteObserver*)
  - Mantém estado que deve ser consistente com o estado do sujeito, implementando a interface de notificação de alterações do sujeito

Padrão Observador (*Observer*)

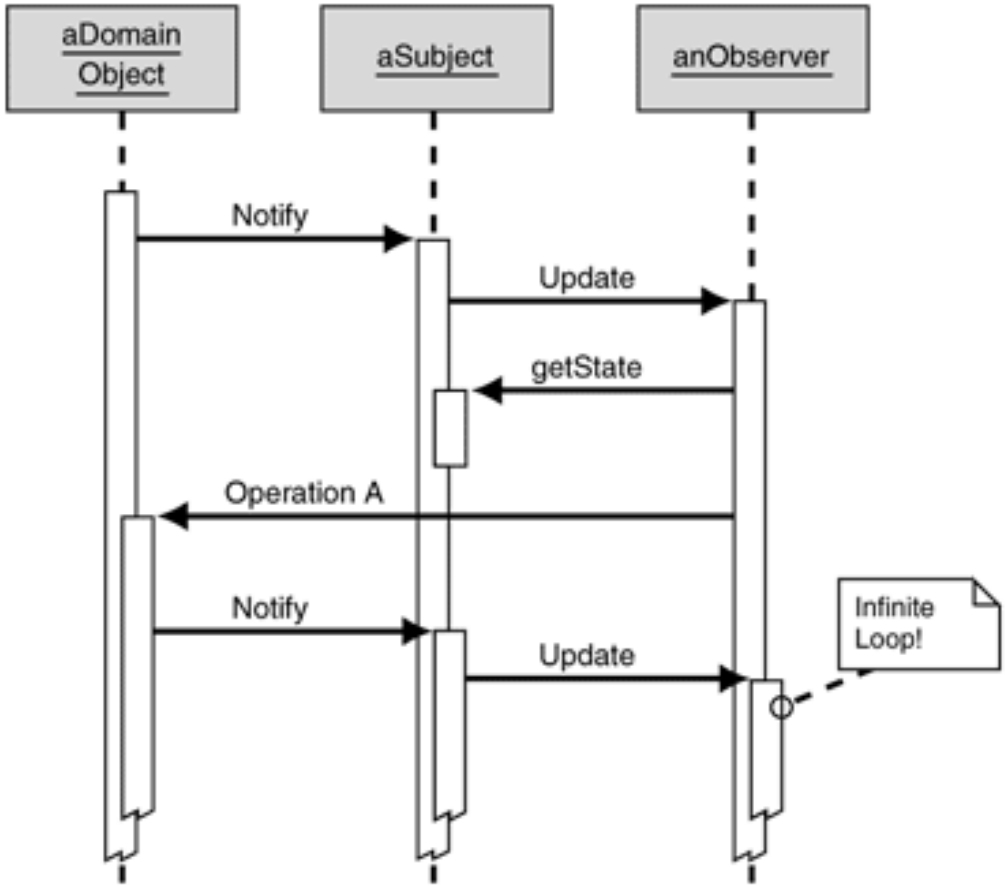
Comportamento



.NET Architecture Center (MSDN)
Enterprise Solution Patterns Using Microsoft .NET

Padrão Observador (*Observer*)

Problema



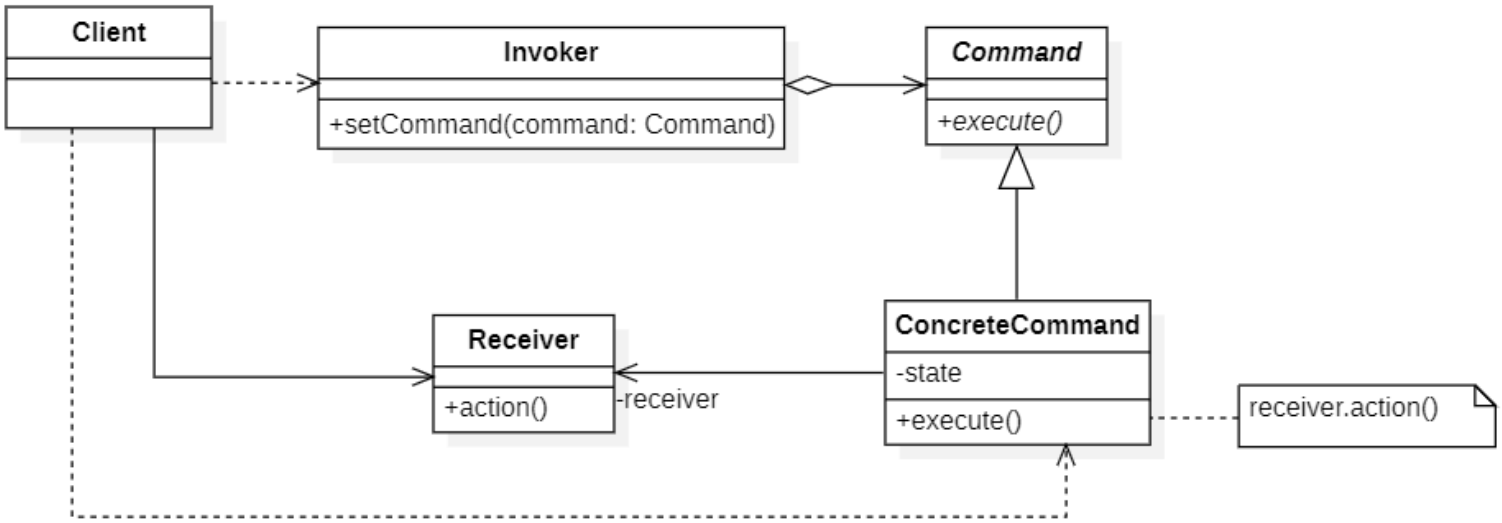
Risco: Possibilidade de ciclos de actualização encadeados

Padrão Comando (*Command*)

- **Problema**
  - Necessidade de realizar pedidos de execução de acções a realizar por outros objectos, independentemente da forma como são executados
- **Solução**
  - Definir um pedido de realização de uma acção como um objeto encapsulado, permitindo assim parametrizar clientes com diferentes pedidos, independentemente da forma como são executados
- **Consequências**
  - Redução de acoplamento, por desacoplamento entre emissor e recetor
  - Suporta o registo e sequenciação de pedidos
  - Permite a implementação de operações de desfazer (*undo*) / refazer (*redo*)
  - Proporciona flexibilidade e extensibilidade na definição de novos comandos
  - Facilita a reutilização de objectos de comando
  - Simplifica o código, separando a lógica de execução de um comando da lógica de criação e invocação do mesmo

Padrão Comando (*Command*)

Estrutura



Participantes

Comando (*Command*)
- Declara a interface para execução de uma acção

Comando concreto (*ConcreteCommand*)
- Define a ligação entre receptor e comando concreto
- Implementa a execução da acção por activação da acção correspondente no receptor

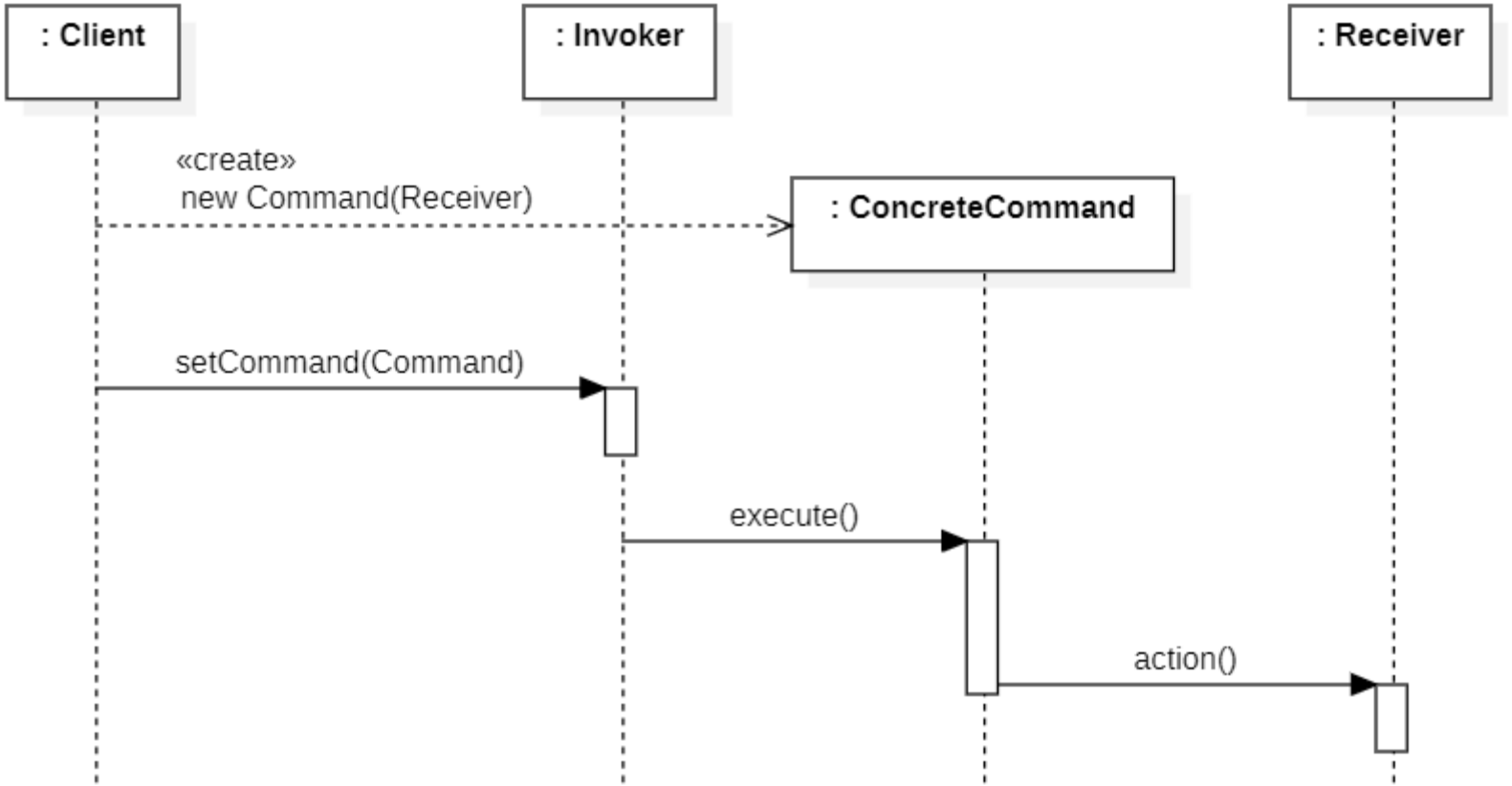
Cliente (*Client*)
- Cria o comando concreto e define o receptor

Evocador *(Invoker*)
- Solicita ao comando a execução da acção

Receptor (*Receiver*)
- Tem capacidade de executar a acção

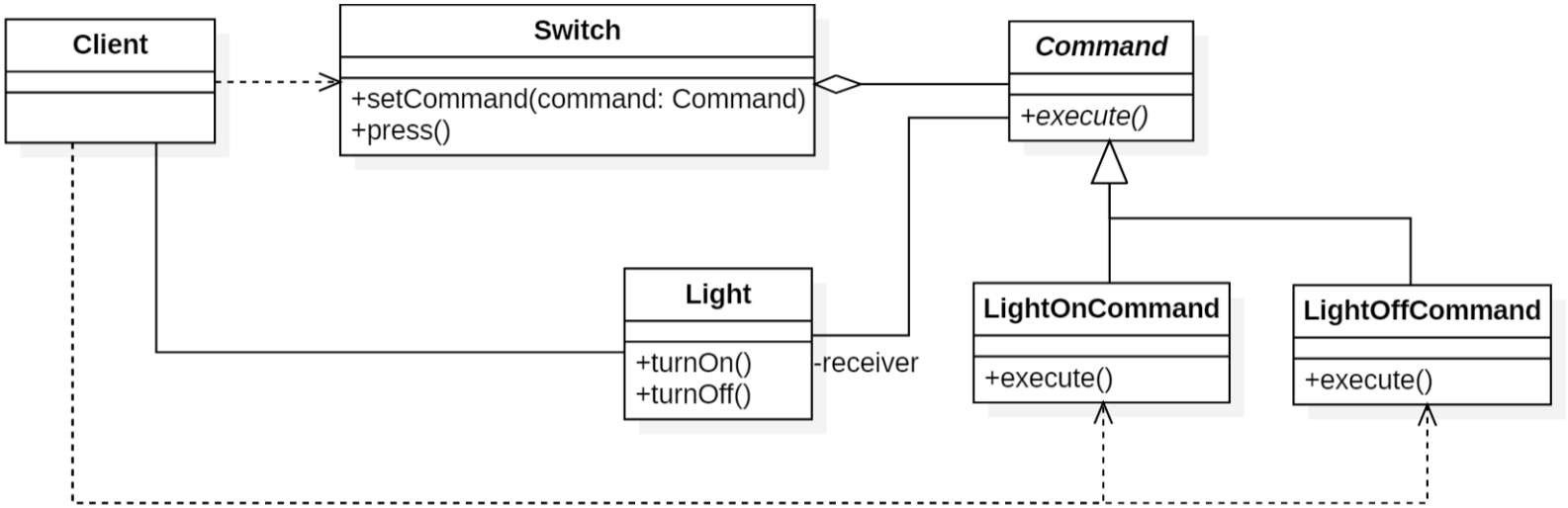
Padrão Comando (*Command*)

Comportamento



Padrão Comando (*Command*)

Exemplo



Padrões de Subsistemas

Exemplos

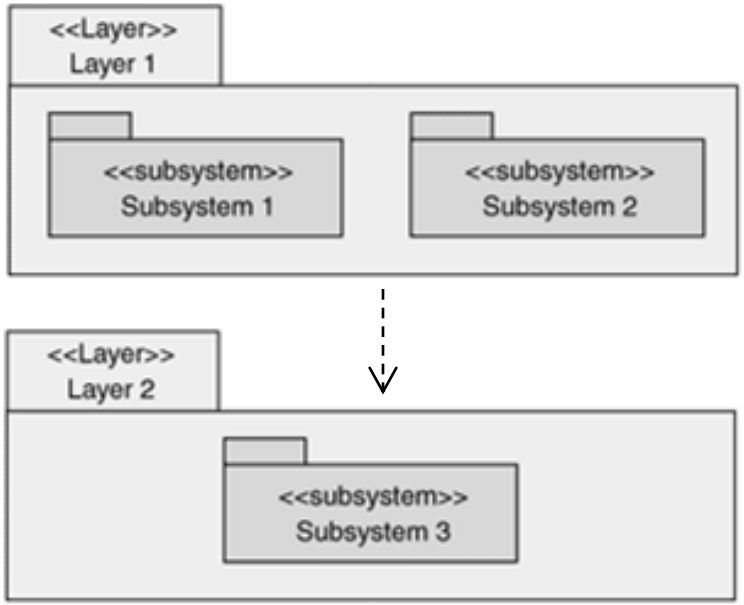
- Padrão Camadas (*Layers*)
  - Organizar um sistema num conjunto de camadas, em que cada camada apresenta um elevado nível de coesão, dependendo apenas das camadas subjacentes
- Arquitectura aplicacional de 3 camadas (*3 Layer Architecture*)
  - Organizar uma aplicação em 3 camadas com responsabilidades específicas nas seguintes vertentes de funcionalidade: interacção com o utilizador, lógica do domínio do problema, persistência de dados
- Arquitectura aplicacional multicamada (*Multilayer Architecture*)
  - Organizar uma aplicação em múltiplas camadas com responsabilidades específicas numa perspectiva de integração de serviços

Padrão Camadas (*Layers*)

- **Problema**
  - Como desenvolver um sistema grande e complexo, garantindo requisitos operacionais como facilidade de manutenção, facilidade de extensão e reutilização, escalabilidade, robustez e segurança, entre outros
- **Solução**
  - Organizar o sistema num conjunto de camadas, em que cada camada é coesa e definida num nível de abstração específico, com um acoplamento fraco com as camadas subjacentes
- **Consequências**
  - Possibilita uma adequada gestão da complexidade, por modularização, encapsulamento e abstracção
  - Proporciona flexibilidade e extensibilidade no desenvolvimento do sistema
  - Facilita a reutilização devido à modularização dos subsistemas

Padrão Camadas (*Layers*)

Estrutura



.NET Architecture Center (MSDN)
Enterprise Solution Patterns Using Microsoft .NET

Participantes

Camada (*Layer*)

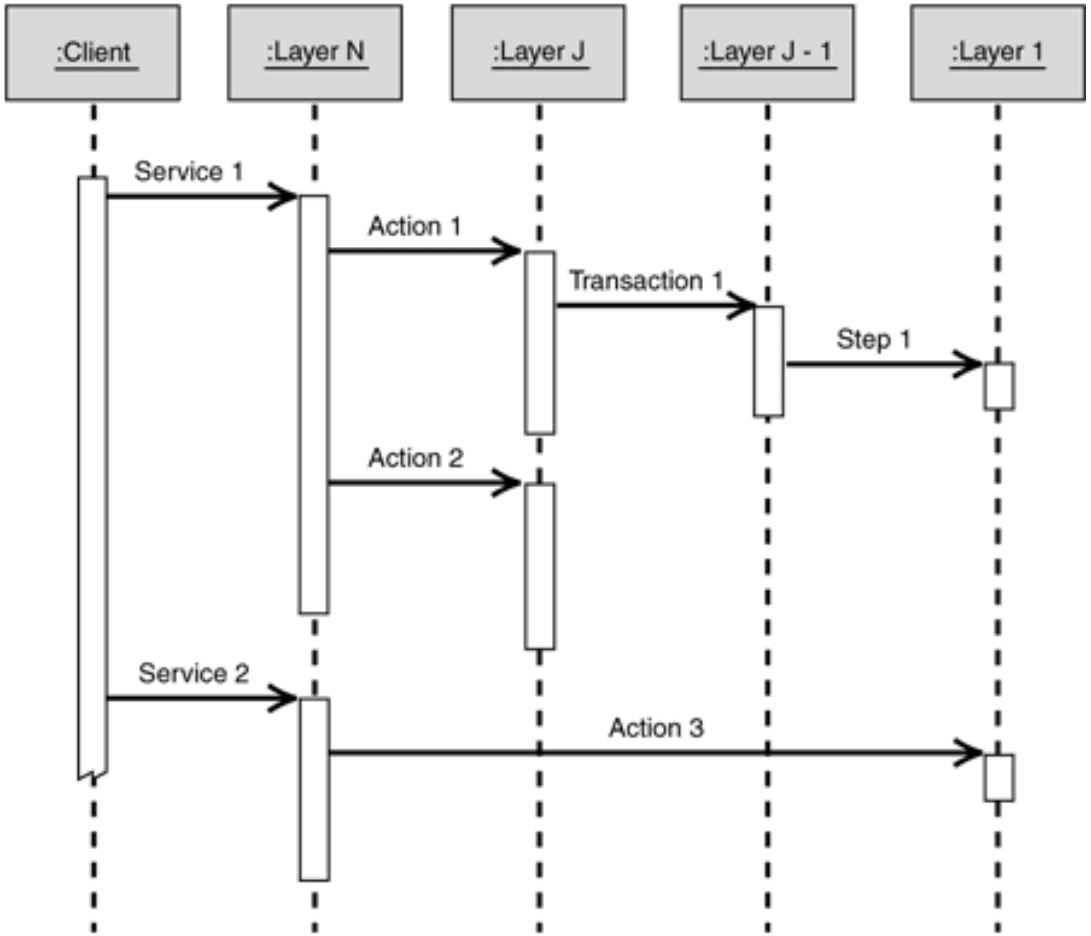
- Modulariza e encapsula um conjunto de subsistemas relacionados, de nível de abstracção idêntico, de forma coesa

Subsistema (*Subsystem*)

- Modulariza e encapsula um conjunto de mecanismos com funcionalidades relacionadas, de forma coesa

Padrão Camadas (*Layers*)

Comportamento



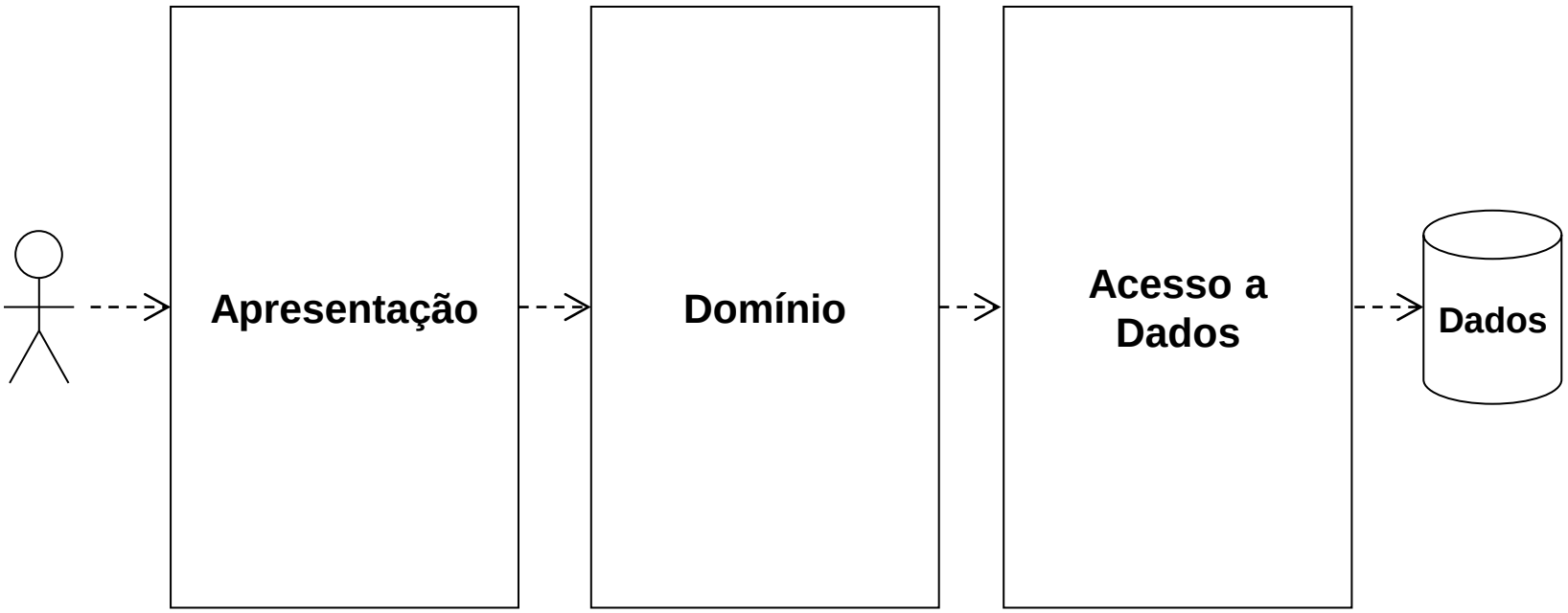
.NET Architecture Center (MSDN)
Enterprise Solution Patterns Using Microsoft .NET

Padrões de Organização de Subsistemas

Arquitectura Aplicacional de 3 Camadas

Tendo por base o padrão *Camadas*, é possível definir uma arquitectura aplicacional com base em três camadas principais

- Apresentação (*Presentation*): responsável pela interacção com o utilizador
- Domínio (*Business Logic*): responsável pela lógica do domínio do problema
- Acesso a Dados (*Data Access*): responsável pelo acesso e persistência de dados



Bibliografia

[Pressman, 2003]

R. Pressman, *Software Engineering: a Practitioner's Approach*, McGraw-Hill, 2003.

[Gamma et al., 1995]

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[Shaw & Garlan, 1996]

M. Shaw, D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, 1996.

[Vernon, 2013]

V. Vernon, *Implementing Domain Driven Design*, Addison-Wesley, 2013.

[Parnas, 1972]

D. Parnas, *On the Criteria to Be Used in Decomposing Systems into Modules*, Communications of the ACM 15-12, 1968.

[Kruchten, 1995]

F. Kruchten, *Architectural Blueprints - The "4+1" View Model of Software Architecture*, IEEE Software, 12-6, 1995.

[Burbeck, 1992]

S. Burbeck; *Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)*, http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html,1992

[Booch, 2004]

G. Booch, *Software Architecture*, IBM, 2004.