Padrões de Projeto

Uma Introdução

Fernando Deschamps

Sistemas Industriais Inteligentes – S2i

Departamento de Automação e Sistemas – DAS

Universidade Federal de Santa Catarina – UFSC

Agenda

- Introdução:
  - O que são padrões de projeto?
  - Motivação de seu uso.
- Conceitos básicos:
  - Principais atributos.
  - Problemas que os padrões de projeto resolvem.
  - Como selecionar um padrão.
  - Como usar um padrão.
- Alguns padrões de projeto.
- Exemplos da estrutura de alguns dos padrões de projeto.
- Onde obter mais informações.

Introdução

- Em geral, em engenharia de software, dois são os principais temas tratados:
  - Metodologia para o desenvolvimento de sistemas.
  - Linguagem de modelagem para o projeto de software orientado a objetos.
- As dificuldades encontradas são decorrentes da falta de experiência de quem está aprendendo ambos os temas pela primeira vez ou da dificuldade de combinação de todos os elementos que fazem parte do projeto de um sistema complexo.
- "Estudos de casos" são uma fonte bastante rica para a solução de problemas de projeto, mesmo para projetistas experientes.

Algumas Definições...

O que é um padrão de projeto?

- Um padrão de projeto é uma estrutura recorrente no projeto de software orientado a objetos. Pelo fato de ser recorrente, vale a pena que seja documentada e estudada.

O que faz um padrão de projeto?

- Um padrão de projeto nomeia, abstrai e identifica os aspectos chave de uma estrutura de projeto comum para torná-la útil para a criação de um projeto orientado a objetos reutilizável.

Problemas de Projeto Solucionados por Padrões

- Procurando objetos apropriados.
- Determinando a granularidade dos objetos.
- Especificando interfaces dos objetos.
- Especificando implementações dos objetos.
  - Herança de classe versus herança de interface.
  - Programando para uma interface, não para uma implementação.
- Colocando os mecanismos de reutilização para funcionar.
  - Herança versus composição.
  - Delegação.
  - Herança versus tipos parametrizados.

Problemas de Projeto Solucionados por Padrões

- Relacionando estruturas de tempo de execução e de tempo de compilação.
- Projetando para mudanças.
  - Dependências de operações específicas, plataformas de software e hardware, de representações ou implementações de objetos e algorítmicas.
  - Acoplamento forte.
  - Incapacidade na alteração de classes.
- Programas de aplicação.
- Bibliotecas de classes (toolkits).
- Arcabouços de classes (frameworks).

Principais Atributos

- Os principais atributos de uma boa descrição de um padrão de projeto são:
  - Nome: referência que descreve de forma bastante sucinta o padrão.
  - Problema (motivação, intenção e objetivos, aplicabilidade): apresenta o contexto do padrão e quando ele pode ser usado.
  - Solução (estrutura, participantes, exemplo de código): descreve a solução e os elementos que a compõem.
  - Conseqüências e padrões relacionados: analisa os resultados, vantagens e desvantagens obtidos com a aplicação do padrão.

Como selecionar um padrão de projeto?

- Considere como os padrões de projeto solucionam problemas de projeto.
- Examine as seções de descrição do problema de cada padrão.
- Estude como os padrões se interrelacionam.
- Estude padrões de finalidades semelhantes.
- Examine uma causa de reformulação de projeto.
- **Considere o que deveria ser variável no seu projeto.**

Como usar um padrão de projeto?

- Leia o padrão por inteiro, uma vez, para obter uma visão geral.
- Estude as seções de descrição do problema e do padrão.
- **Olhe exemplos de código do padrão.**
- Escolha nomes para os participantes do padrão que tenham sentido no contexto da aplicação.
- Defina as classes.
- Defina nomes específicos da aplicação para as operações no padrão.
- Implemente as operações para suportar as responsabilidades e colaborações presentes.

Alguns Padrões de Projeto

- Em geral os padrões de projeto podem ser classificados em três diferentes tipos:
  - Padrões de criação: abstraem o processo de criação de objetos a partir da instanciação de classes.
  - Padrões estruturais: tratam da forma como classes e objetos estão organizados para a formação de estruturas maiores.
  - Padrões comportamentais: preocupam-se com algoritmos e a atribuição de responsabilidade entre objetos.
- Cada um desses tipos pode ser subclassificado em:
  - Padrões de classes: em geral estáticos, definidos em tempo de compilação.
  - Padrões de objetos: em geral dinâmicos, definidos em tempo de execução.

Padrões de Criação

- **Abstract Factory**: fornece uma interface para a criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.
  - Uso conhecido: transportabilidade entre diferentes bibliotecas de interfaces gráficas (Gnome e KDE).
- **Builder**: separa a construção (geralmente passo a passo) de um objeto complexo da sua representação, de modo que o mesmo processo de construção possa criar diferentes representações.
  - Uso conhecido: conversão de formatos de texto em editores.

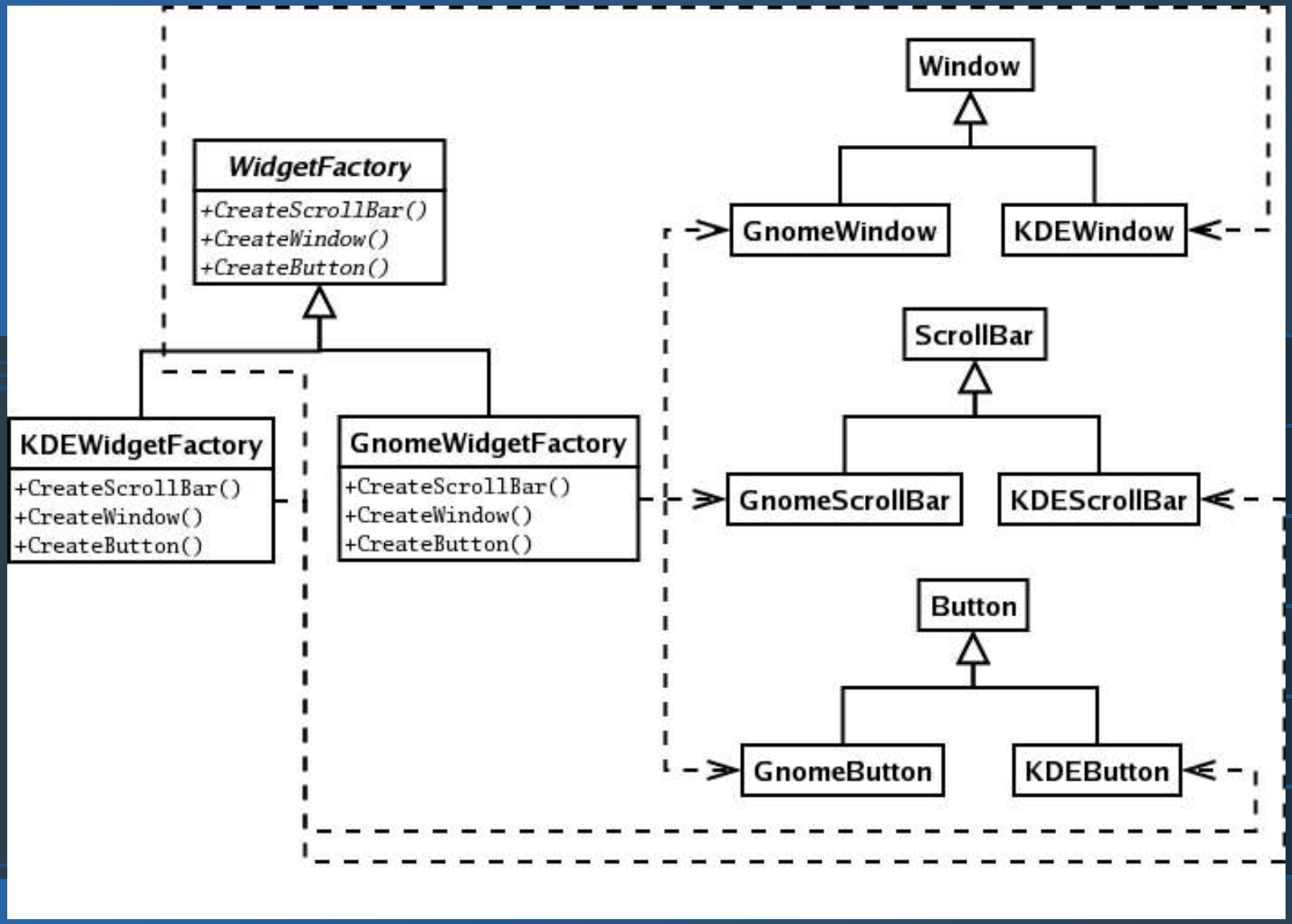
Padrões de Criação

- **Factory Method**: define uma interface para criar um objeto, mas deixa as subclasses decidirem qual classe será instanciada. Permite a uma classe postergar a instanciação de subclasses.
  - Uso conhecido: bastante usado em toolkits e frameworks para a instanciação de objetos.
- **Prototype**: especifica os tipos de objetos a serem criados usando uma instância prototípica e cria novos objetos copiando este protótipo.
  - Uso conhecido: depurador Etgdb.

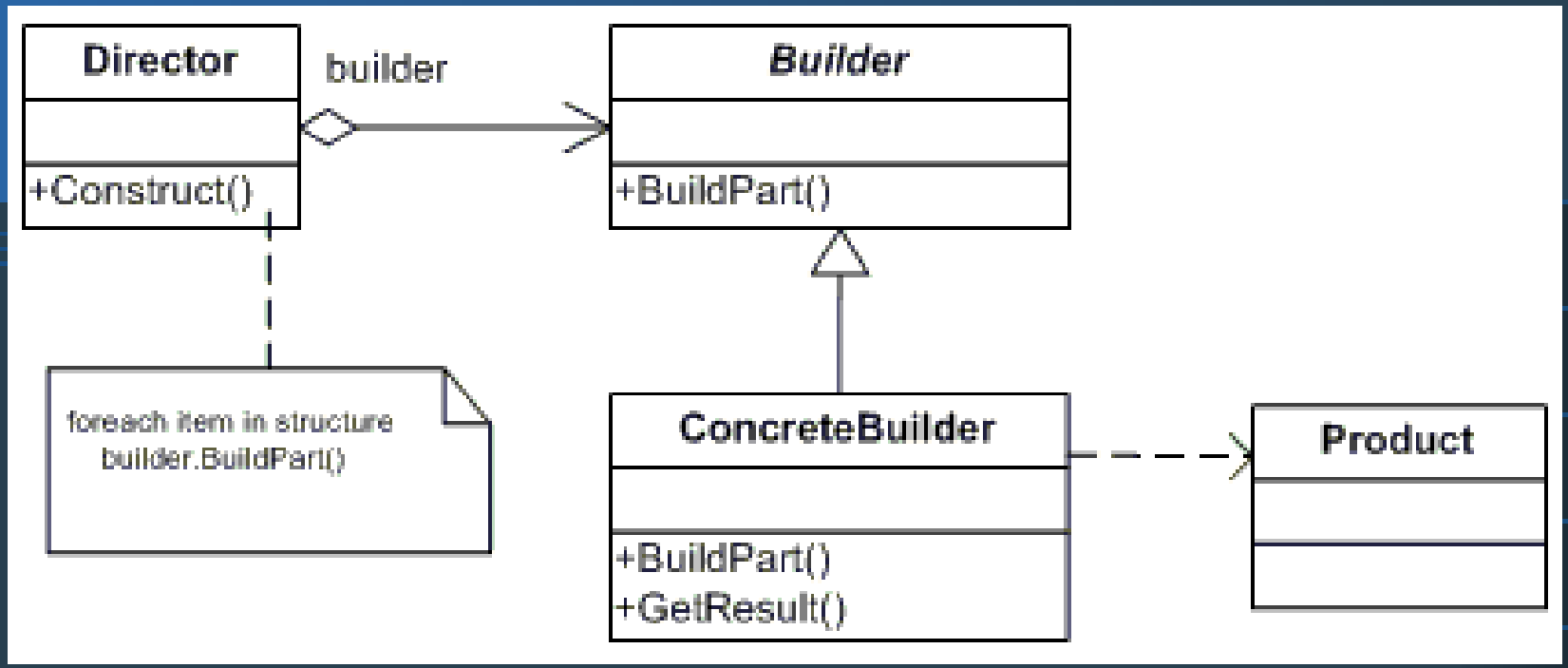
Padrões de Criação

- **Singleton**: garante que uma classe tenha somente uma instância e fornece um ponto global de acesso para ela.
  - Uso conhecido: programas que só podem ter uma instância sendo executada em um dado momento.

Exemplo: Abstract Factory



Exemplo: Builder



Padrões Estruturais

- **Adapter**: converte a interface de uma classe em outra interface esperada pelos clientes. Permite que certas classes trabalhem em conjunto, pois de outra forma seria impossível por causa de suas interfaces incompatíveis.
  - Uso conhecido: popularmente conhecido como **wrapper**, usado para adaptar a interface de classes.
- **Bridge**: separa uma abstração da sua implementação, de modo que as duas possam variar independentemente.
  - Uso conhecido: para evitar o vínculo entre abstração e implementação quando de mudanças na implementação em tempo de execução.

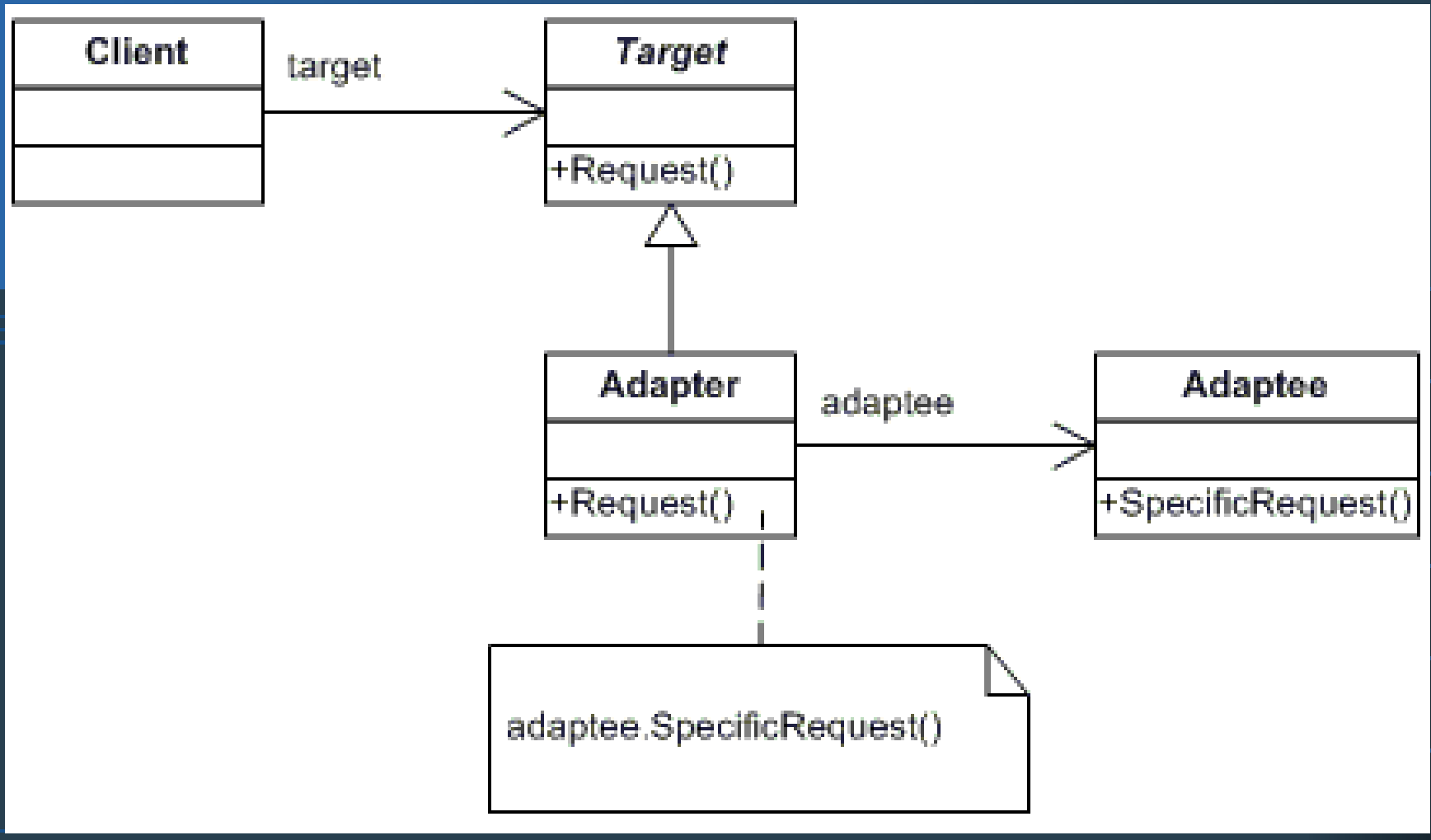
Padrões Estruturais

- **Composite**: compõe objetos em estrutura de árvore para representar hierarquias do tipo partes-todo. Permite que os clientes da estrutura tratem objetos individuais e composições de objetos de maneira uniforme.
  - Uso conhecido: para representar hierarquias partes-todo.
- **Decorator**: atribui responsabilidades adicionais a um objeto dinamicamente. Fornece uma alternativa flexível à utilização de subclasses para a extensão de funcionalidades.
  - Uso conhecido: para a atribuição de enfeites gráficos e outras funcionalidades acessórias a widgets.

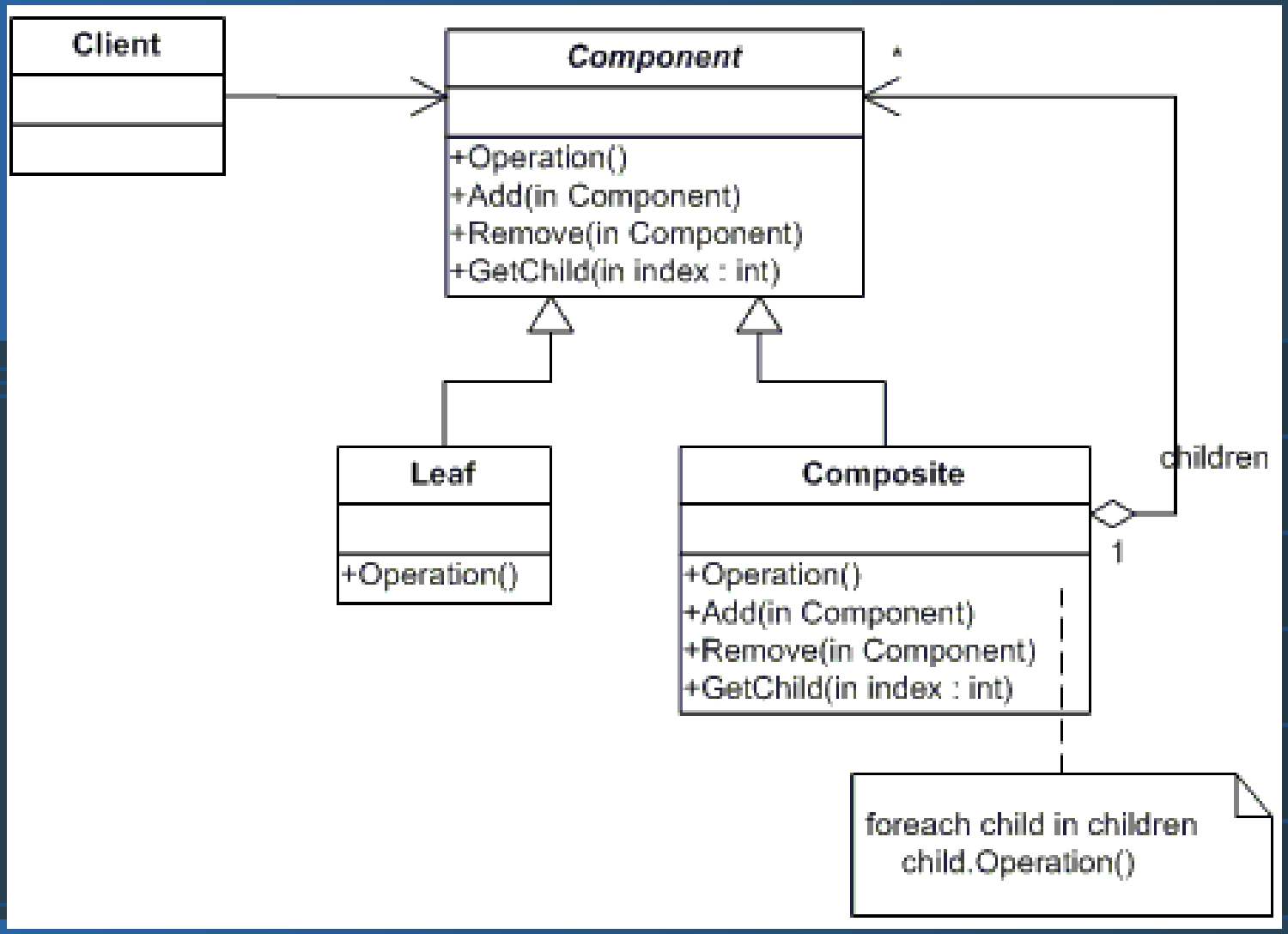
Padrões Estruturais

- **Façade**: fornece uma interface unificada para um conjunto de interfaces em um subsistema. Define uma interface de nível mais alto que torna o subsistema mais fácil de usar.
  - Uso conhecido: interface única em sistemas complexos.
- **Flyweight**: usa compartilhamento para suportar grandes quantidades de objetos, de granularidade fina, de maneira eficiente.
  - Uso conhecido: sistemas com grande número de objetos.
- **Proxy**: fornece um objeto representante ou um marcador de outro objeto para controlar o acesso ao mesmo.
  - Uso conhecido: algumas implementações de CORBA.

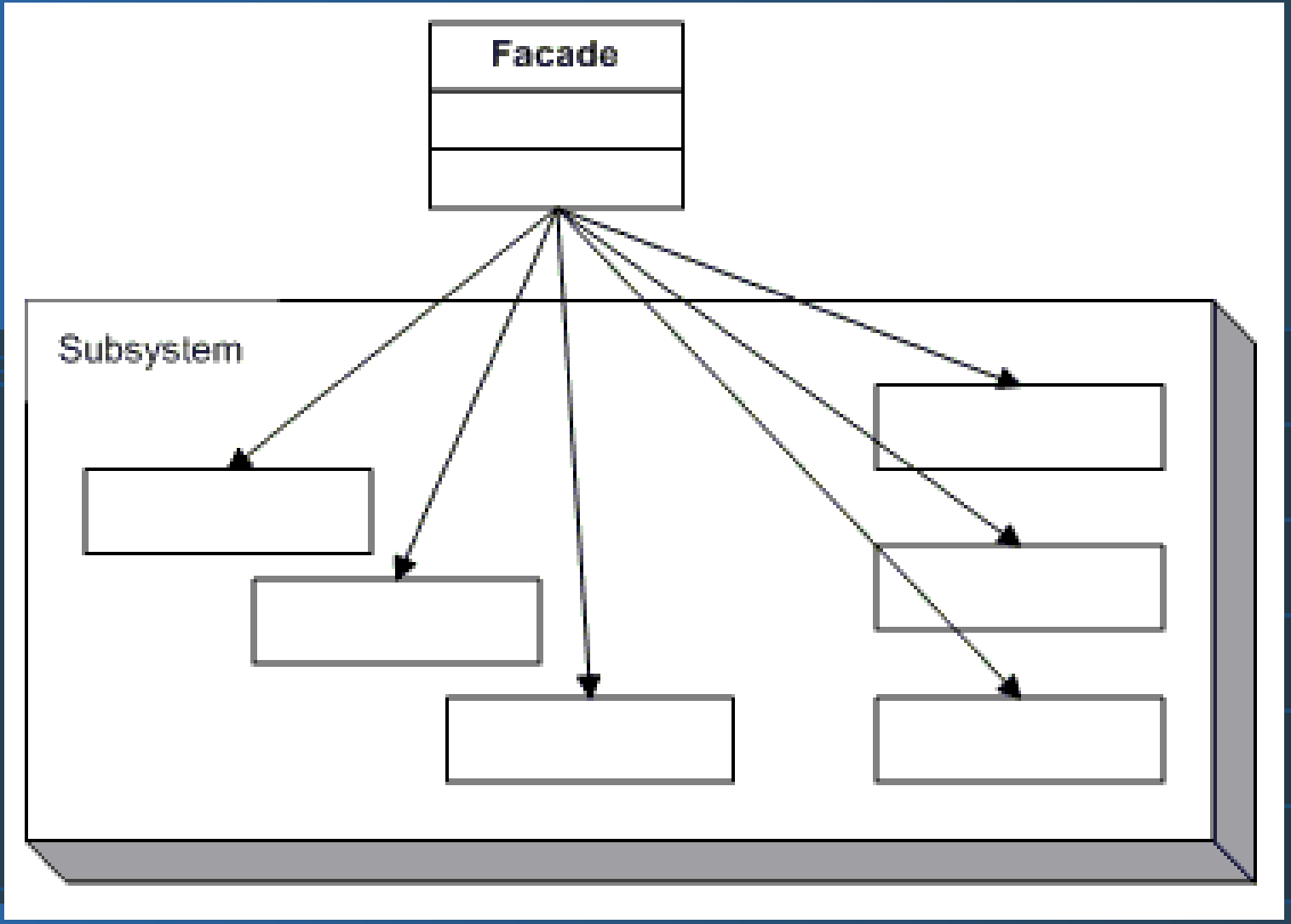
Exemplo: Adapter



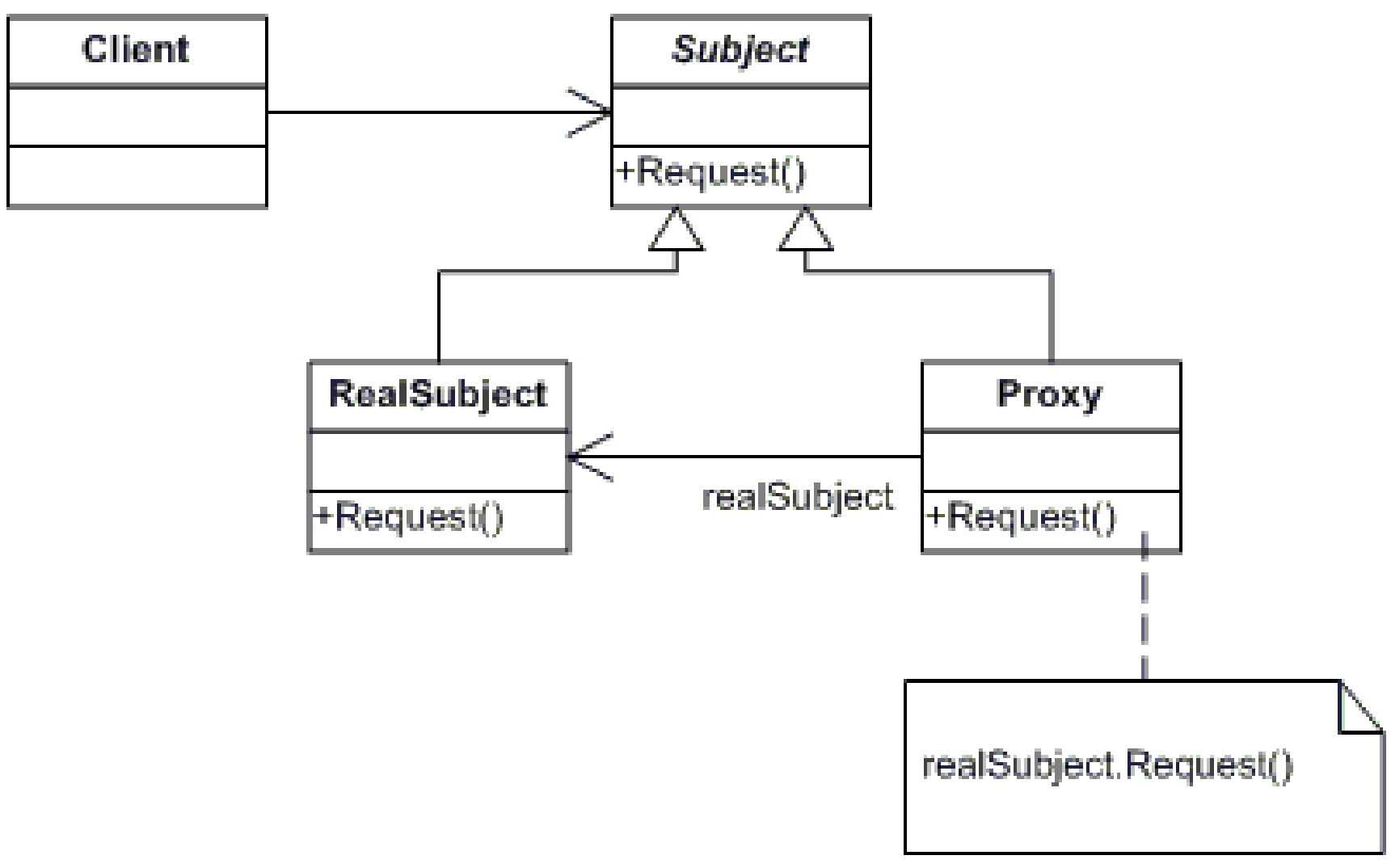
Exemplo: Composite



Exemplo: Façade



Exemplo: Proxy



Padrões Comportamentais

- **Chain of Responsibility**: evita o acoplamento entre o remetente de uma solicitação e o destinatário da solicitação, dando a mais de um objeto a chance de tratar a solicitação. Encadeia os objetos receptores e passa a solicitação ao longo da cadeia até que um objeto a trate.
  - Uso conhecido: tratamento de eventos de usuário.
- **Command**: encapsula uma solicitação como um objeto, permitindo a parametrização de clientes com diferentes solicitações, o enfileiramento e o registro de solicitações e o suporte a operações que possam ser, por exemplo, desfeitas.
  - Uso conhecido: suporte a “desfazer”.

Padrões Comportamentais

- **Interpreter**: dada uma linguagem, define uma representação para sua gramática juntamente com um interpretador que usa a representação para interpretar sentenças nesta linguagem.
  - Uso conhecido: interpretar uma linguagem com um árvore sintática abstrata, como em compiladores de linguagens orientadas a objetos.
- **Iterator**: fornece uma maneira de acessar seqüencialmente os elementos de um objeto agregado sem expor sua representação subjacente.
  - Uso conhecido: C++ Standard Template Library.

Padrões Comportamentais

- **Mediator**: define um objeto que encapsula a interação entre um conjunto de objetos. Promove o acoplamento fraco ao evitar que os objetos se refiram explicitamente uns aos outros, permitindo a variação das interações independentemente.
  - Uso conhecido: arquitetura de aplicações de Smalltalk.
- **Memento**: sem violar a encapsulação, captura e externaliza um estado interno de um objeto, de modo que o mesmo possa posteriormente ser restaurado para este estado.
  - Uso conhecido: armazenar um instantâneo do estado do objeto sem romper sua encapsulação.

Padrões Comportamentais

- **Observer**: define uma dependência um-para-muitos entre objetos, de modo que, quando um objeto muda de estado, todos os seus dependentes são automaticamente notificados e atualizados.
  - Uso conhecido: propagação de mudanças e atualizações com acoplamento fraco entre os objetos.
- **State**: permite que um objeto altere seu comportamento quando seu estado interno muda. O objeto parecerá ter mudado sua classe.
  - Uso conhecido: em algumas implementações da pilha TCP/IP.

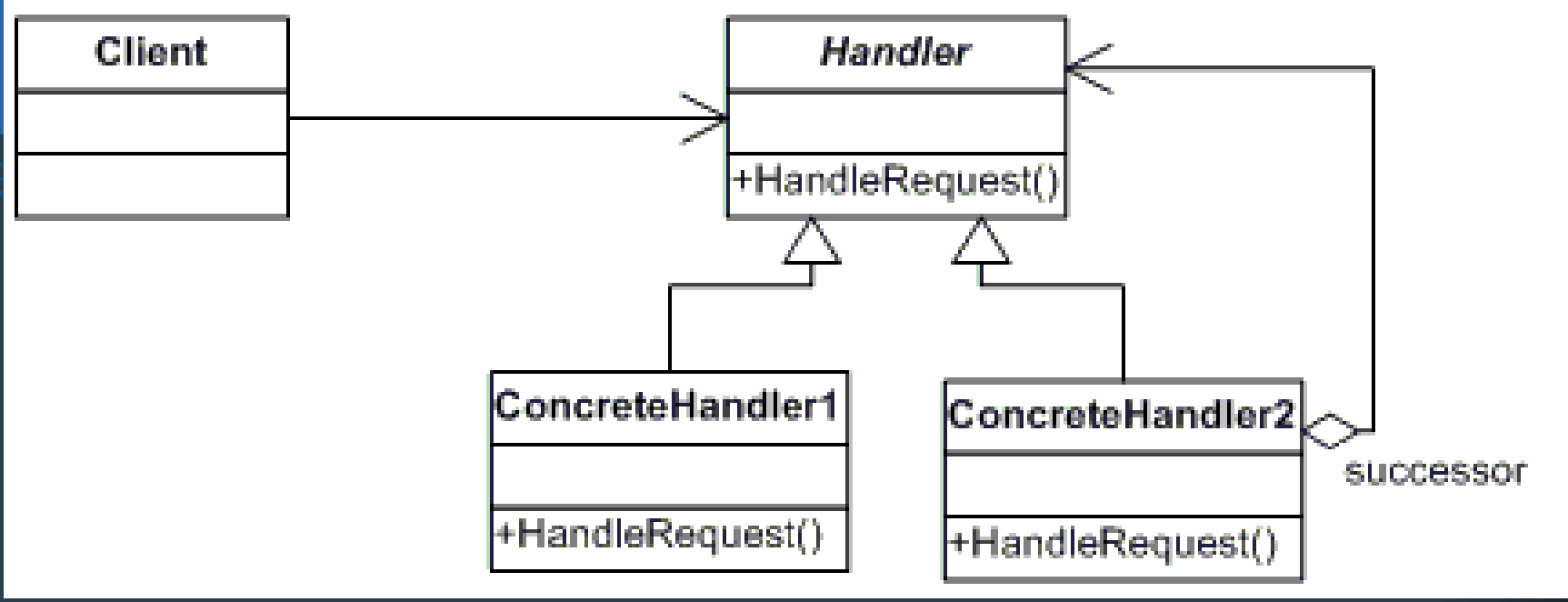
Padrões Comportamentais

- **Strategy**: define uma família de algoritmos, encapsula cada um deles e os faz intercambiáveis. Permite que o algoritmo varie independentemente dos clientes que o utilizam.
  - Uso conhecido: sistemas de otimização.
- **Template Method**: define o esqueleto de um algoritmo em uma operação, postergando a definição de alguns passos para subclasses. Permite que as subclasses redefinam certos passos de um algoritmo sem mudar sua estrutura.
  - Uso conhecido: arquiteturas Application/Document/View.

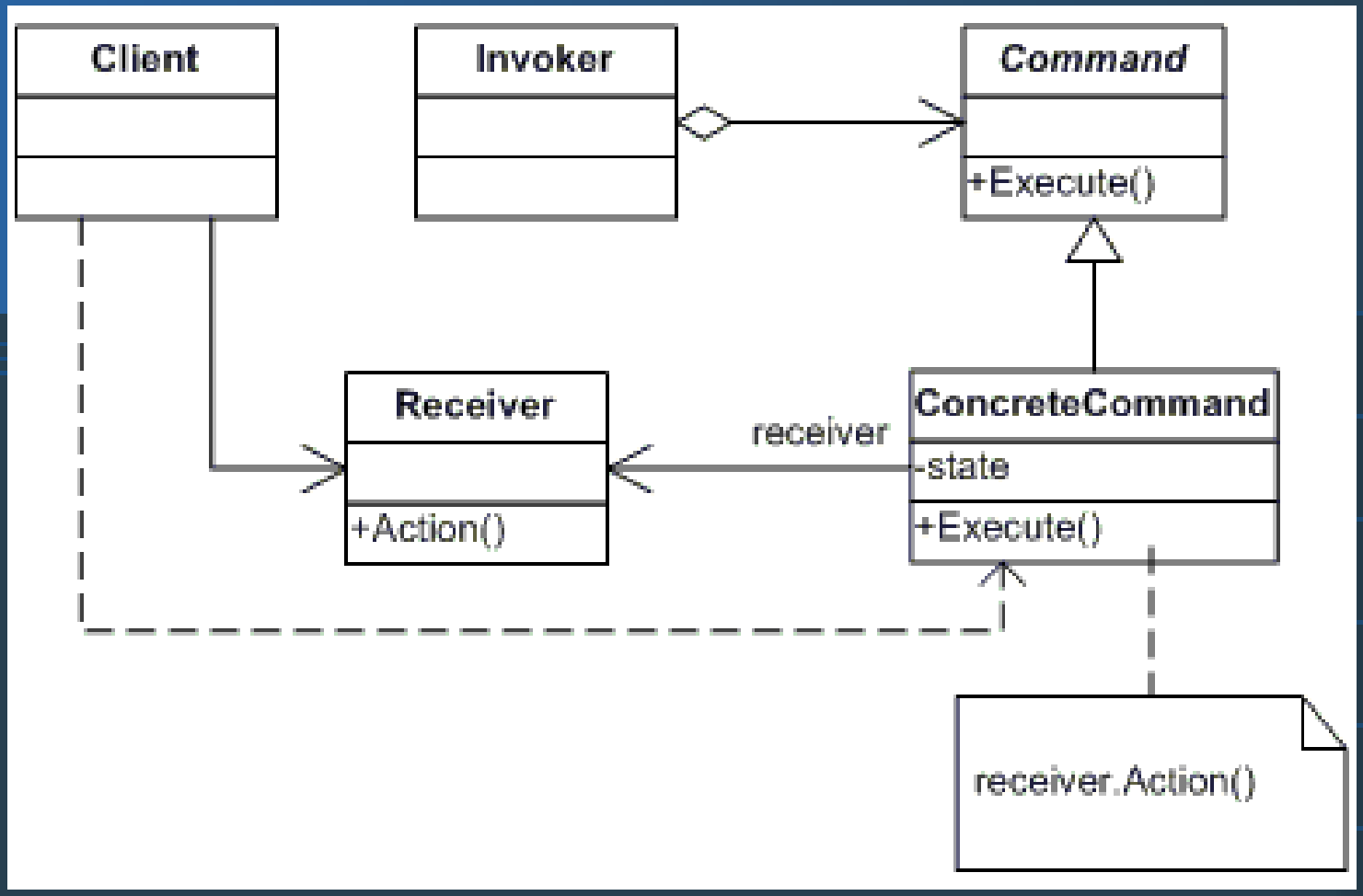
Padrões Comportamentais

- **Visitor**: representa uma operação a ser executada sobre os elementos de uma estrutura de objetos. Permite a definição de uma nova operação sem mudar as classes dos elementos sobre os quais opera.
  - Uso conhecido: para executar uma série de operações sobre objetos que possuem interfaces diferentes, sem poluir a interface dos mesmos.

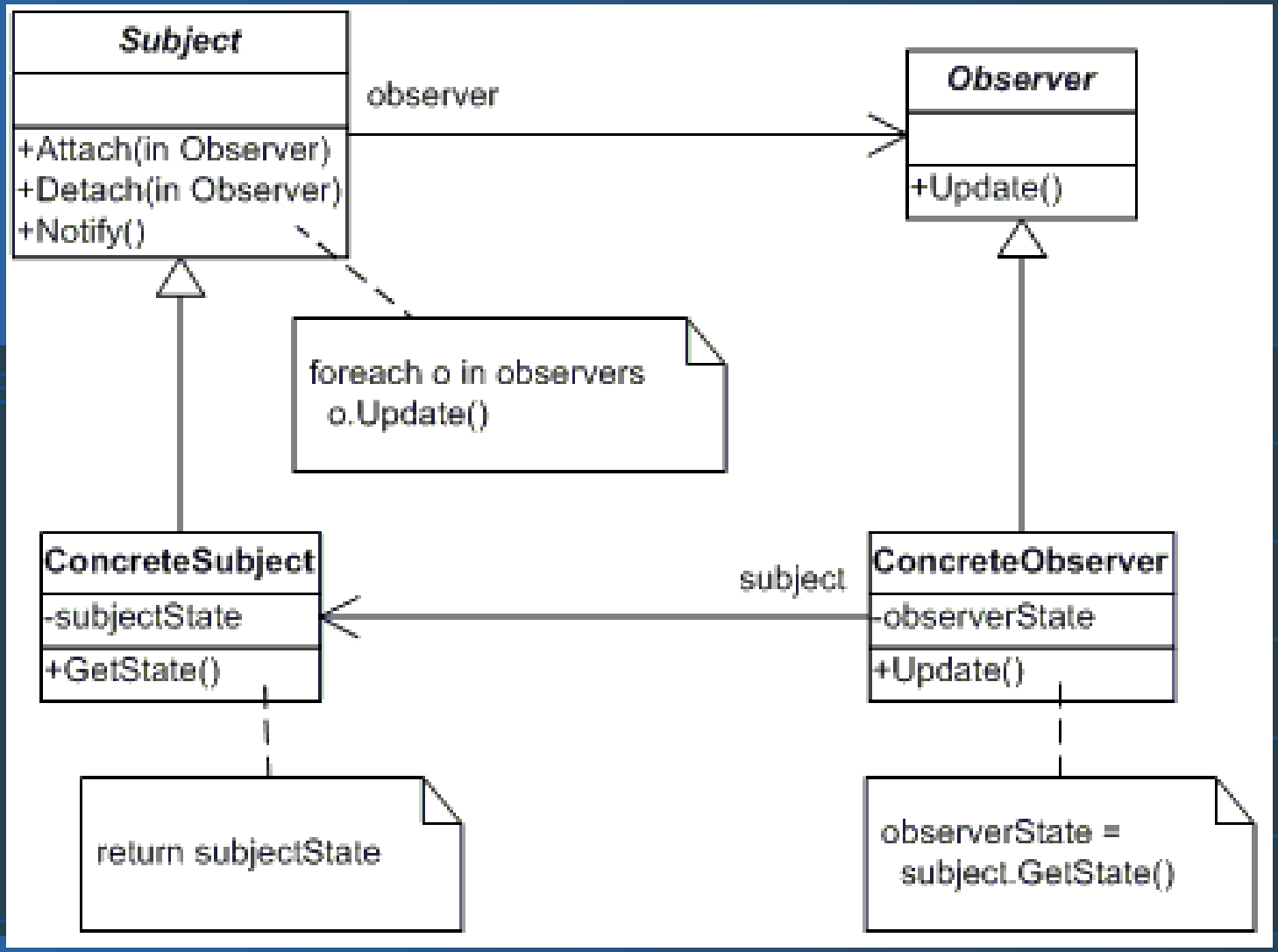
Exemplo: Chain of Responsibility



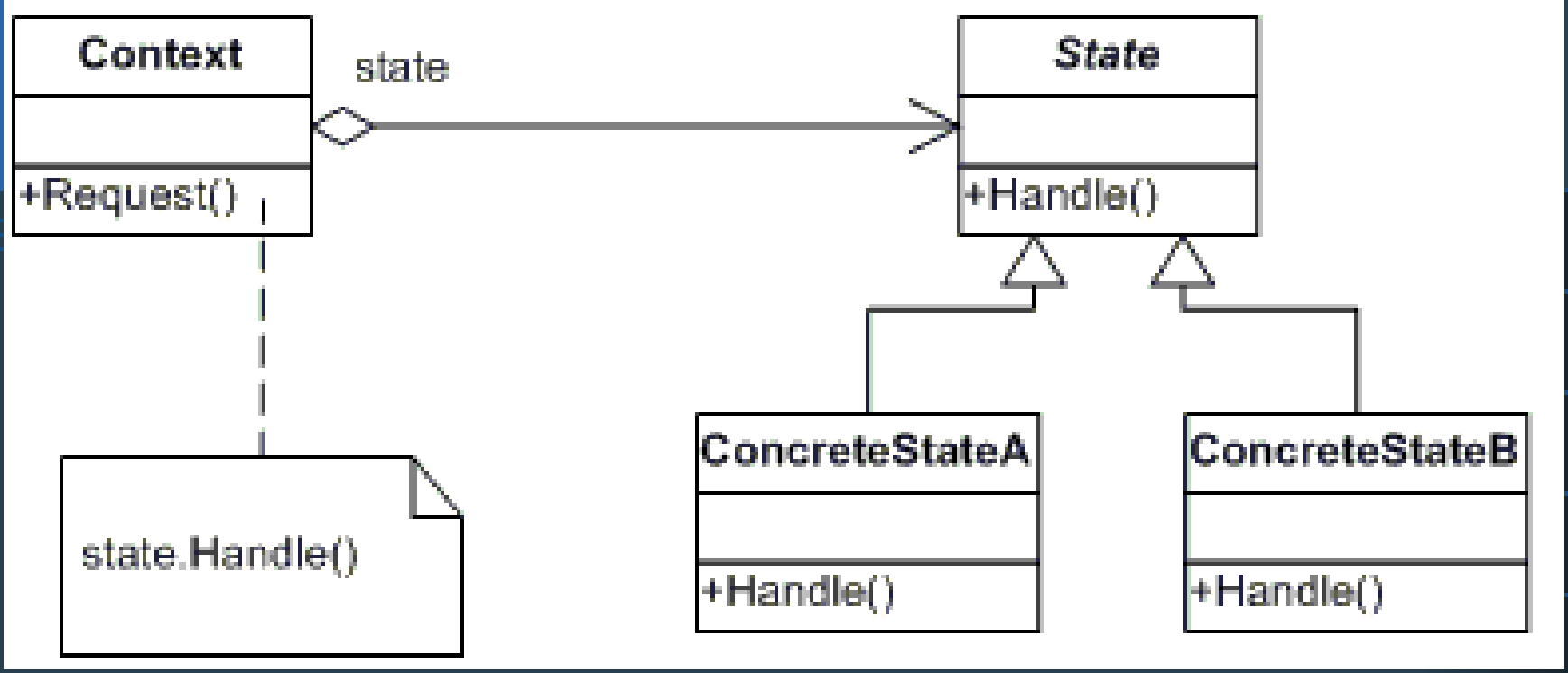
Exemplo: Command



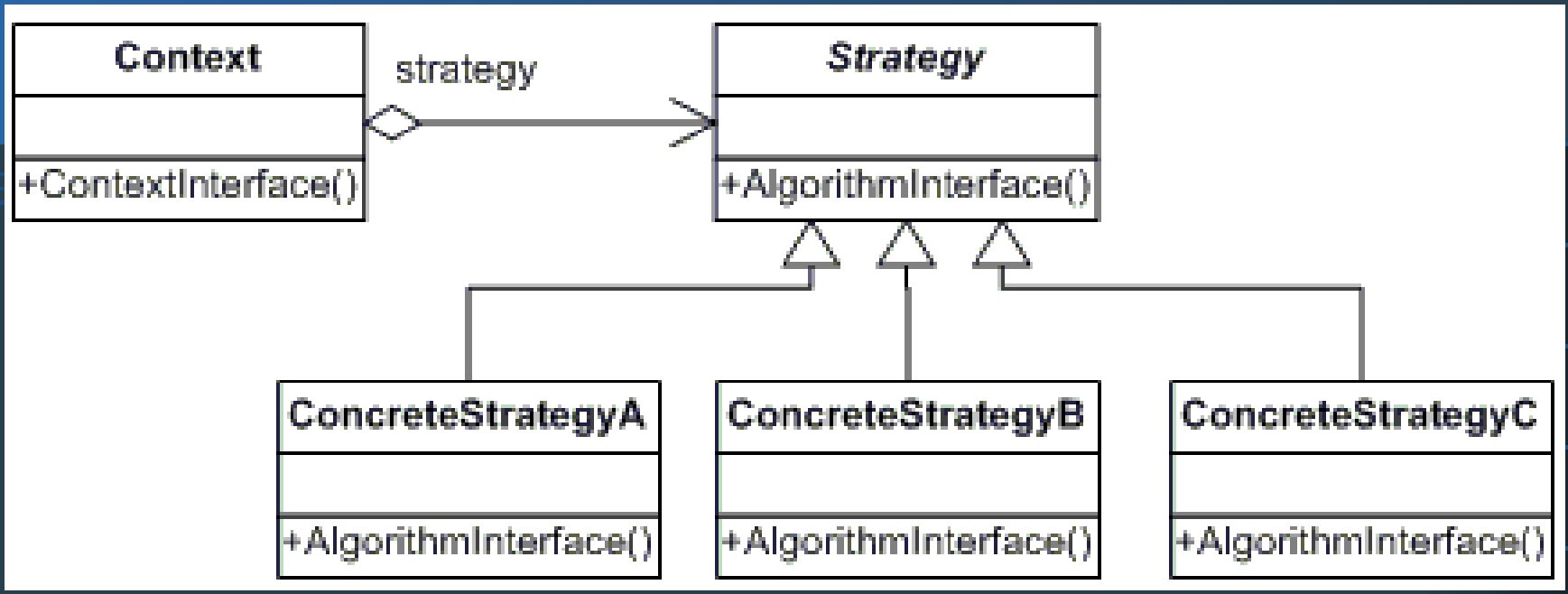
Exemplo: Observer



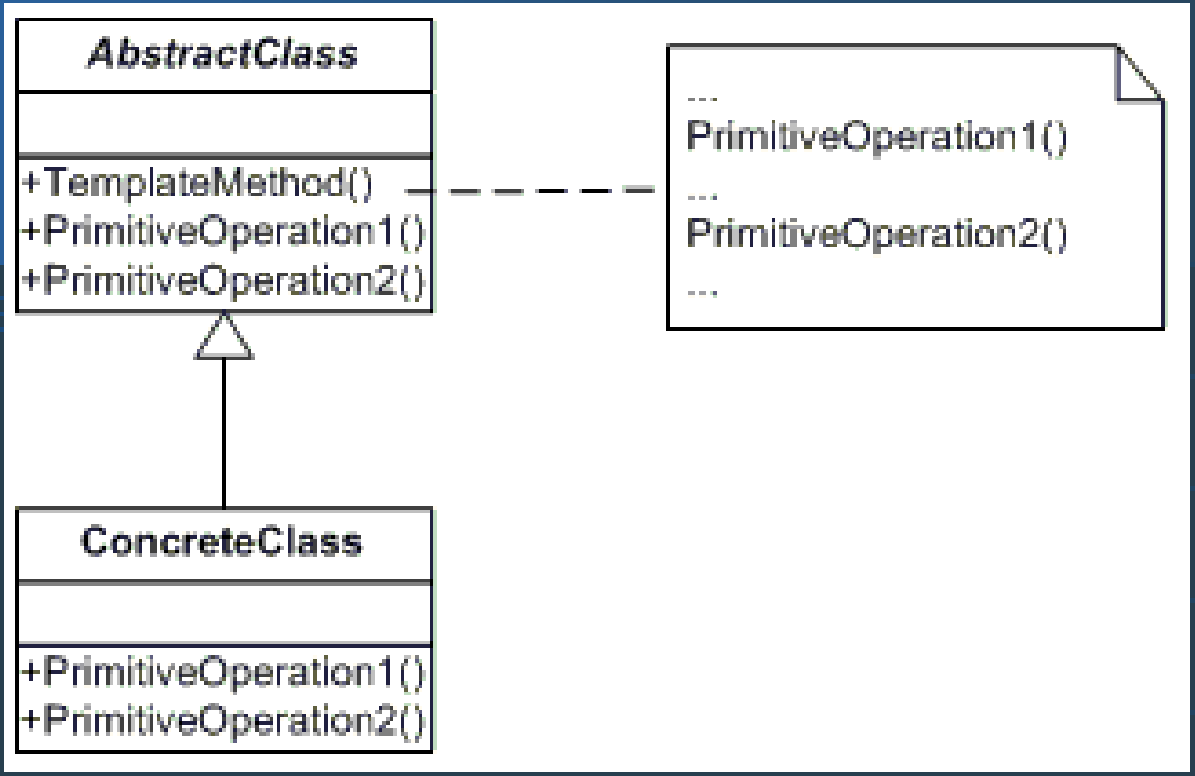
Exemplo: State



Exemplo: Strategy



Exemplo: Template Method



Outras Referências

- Livro base: **Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos** - “The Gang of Four”.
- Design Patterns Tutorial:
  http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/patterns/objectives.html
- Design Patterns, Pattern Languages and Frameworks:
  http://www.cs.wustl.edu/~schmidt/patterns.html
- A Learning Guide to Design Patterns:
  http://www.industriallogic.com/papers/learning.html
- Data & Object Factory's Design Patterns Page:
  http://www.dofactory.com/patterns/Patterns.aspx
- Overview of Design Patterns:
  http://www.mindspring.com/~mgrand/pattern_synopses.htm