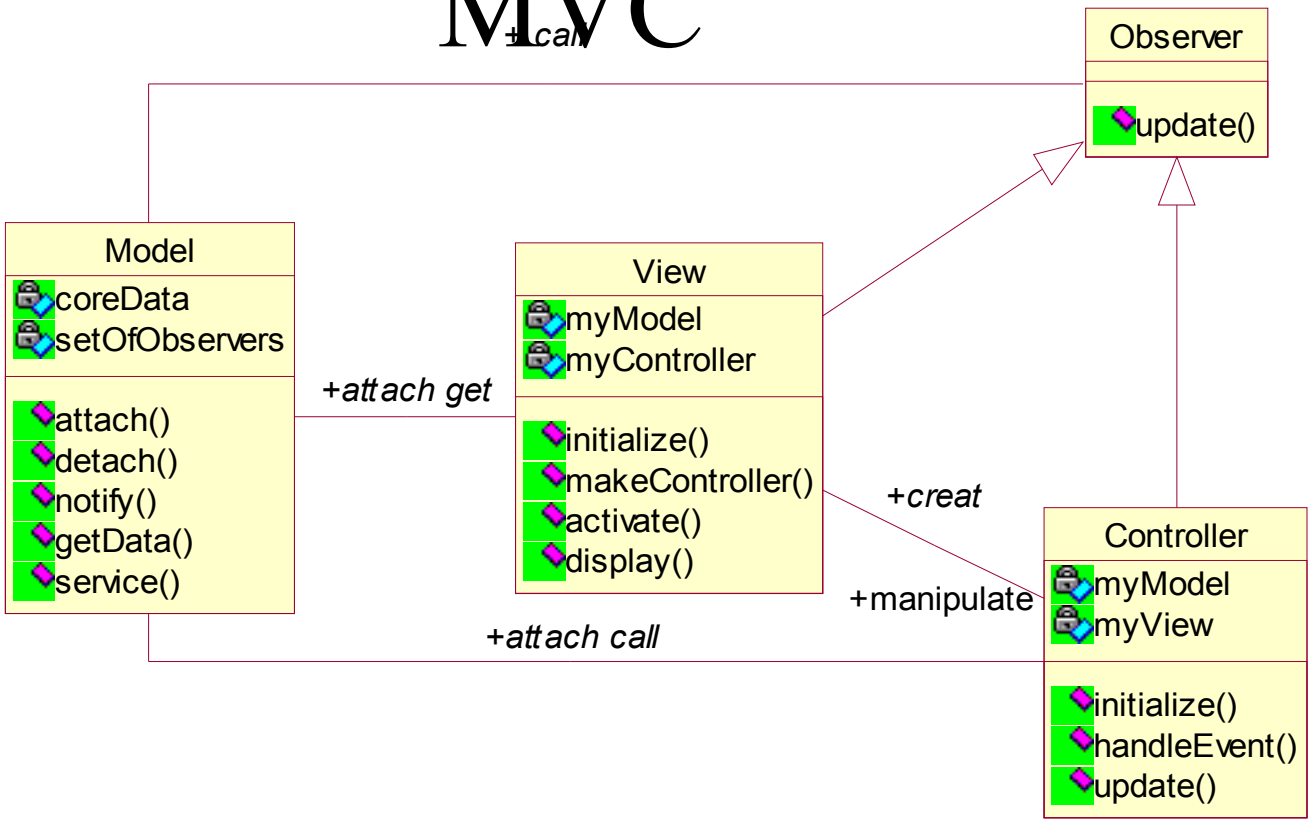
Padrões Arquiteturais

Silvia Regina Vergilio

Exemplo de Padrão Arquitetural: MVC



Model-View-Controller

- **Motivação**:

  - Separação de interesses.
  - Considere o uso deste padrão quando estiver desenvolvendo sistemas com interface de usuário.
  - Interfaces de usuário são propensas a mudanças. Por exemplo, quando as funcionalidades de uma aplicação são estendidas ou adaptadas, menus devem ser modificados.
  - O sistema pode ser executado em diferentes plataformas, com diferentes padrões de interface gráfica.
  - A interface de usuário deve estar o mais independente possível do *kernel* funcional da aplicação.

Model-View-Controller

- **O padrão propõe a divisão de uma aplicação em 3 áreas (ou tipos de componentes): modelo, controle e apresentação**
  - O ***modelo*** representa as classes do domínio do problema, sendo independente de uma forma específica de apresentação.
  - Encapsula os dados e funcionalidade do negócio. O modelo deve ser independente de representações de saída específicas e do tratamento das interações dos usuários com a aplicação.
  - A ***visão*** apresenta as informações do modelo ao usuário. Podem existir múltiplas visões de um mesmo modelo. Cada Visão tem um controlador.

Model-View-Controller

- **O padrão propõe a divisão de uma aplicação em 3 áreas (ou tipos de componentes): modelo, controle e apresentação**
  - Os ***controladores*** recebem a entrada, geralmente um evento (i.e. movimentos do mouse, ativação de botões, teclas etc.), que é traduzida em requisições de serviços ao modelo ou visão.
- Se o usuário altera o modelo através do controlador de uma visão, todas as outras visões dependentes destes dados devem refletir a mudança.
- As Visões devem refletir o estado atual do modelo.
- Padrão arquitetural utilizado no Smalltalk.

Model-View-Controller

- **Vantagens:** flexibilidade e reutilização.

- Aplica o padrão de projeto *Observer*.

- Pode aplicar o padrão de projeto *Composite* quando trabalha com objetos Visão complexos. Por exemplo, um Frame pode ser composto por painéis etc. O agrupamento de objetos é tratado como um objeto individual.

- Outros padrões de projeto podem ser aplicados.

Outros Exemplos de Padrões Arquiteturais

Nome do Padrão

Camadas

Contexto

Um sistema grande que requer decomposição.

Problema

Um sistema que deve resolver as questões em diferentes níveis de abstração. Por exemplo: as questões de controle de hardware, as questões de serviços comuns e as questões específicas de domínio. Seria extremamente indesejável escrever componentes verticais que lidem com essas questões em todos os níveis. Uma mesma questão deveria ser resolvida (possivelmente de maneira inconsistente) várias vezes em diferentes componentes.

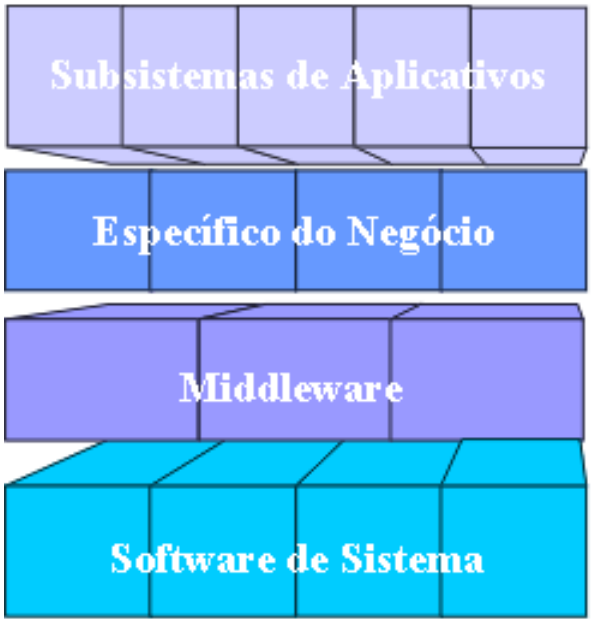
Força

- As partes do sistema devem ser substituíveis
- As alterações efetuadas nos componentes não devem ser irregulares
- Responsabilidades similares devem ser agrupadas juntas
- Tamanho dos componentes—componentes complexos talvez precisem ser decompostos

Solução

Estruture os sistemas em grupos de componentes que formem camadas umas sobre as outras. Faça com que as camadas superiores utilizem os serviços somente das camadas abaixo (nunca das camadas acima). Tente não usar serviços que não sejam os da camada diretamente abaixo (não pule camadas, a menos que as camadas intermediárias somente adicionem componentes de acesso).

Exemplo



Subsistemas de Aplicativos distintos que compõem um aplicativo – contêm o software que agrega valor desenvolvido pela organização.

Específico do Negócio – contém um número de sistemas reutilizáveis específicos do tipo de negócio.

Middleware - oferece subsistemas para plataforma e classes de utilitários – serviços independentes para computação de objetos distintos em ambientes heterogêneos e outros.

Software do Sistema - contém o software para a infra-estrutura real como, por exemplo, sistemas operacionais, interfaces para hardware específico, drivers de dispositivos etc.

- ***Broker:*** para aplicações distribuídas, onde uma aplicação pode acessar serviços de outras aplicações simplesmente pelo envio de mensagens a objetos mediadores, sem se preocupar com questões específicas relacionadas à comunicação entre processos, como a sua localização.

Outros Exemplos de Padrões Arquiteturais

- ***Presentation-Abstraction-Control (PAC):*** define uma estrutura para sistemas na forma de uma hierarquia de agentes cooperativos. Adequado para sistemas interativos, onde cada agente é responsável por um aspecto específico da funcionalidade da aplicação e é composto por três componentes: apresentação, abstração e controle.

- ***Microkernel:*** propõe a separação de um conjunto de funcionalidades mínimas das funcionalidades estendidas e partes específicas de clientes. O encapsulamento dos serviços fundamentais da aplicação é realizado no componente "*microkernel*". As funcionalidades estendidas e específicas devem ser distribuídas entre os componentes restantes da arquitetura.

Referências

- Dewayne E. Perry and Alexander L. Wolf. Foundations for the study of software architecture. SIGSOFT Software Engineering Notes, 17(4):408211;52, October 1992.
- David Garlan and Mary Shaw. An introduction to software architecture. Technical report, Pittsburgh, PA, USA, 1994.
- David Garlan and Mary Shaw. An introduction to software architecture. Technical report CMU-CS-94-166, Carnegie Mellon University, Pittsburgh, PA 15213-3890, January 1994.
- Len Bass, Paul Clements, and Rick Kazman. SoftwareArchitecture in Practice. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.
- Len Bass, Paul Clements, and Rick Kazman. Software Architecture in Practice. Addison-Wesley Professional, 2 edition, April 2003.
- Frank Buschmann, Régine Meunier, Hans Rohnert, Peter Sommerlad e Michael Stahl 1996. *Pattern-Oriented Software Architecture - A System of Patterns*, Nova York, NY: John Wiley and Sons, Inc.
- Antônio Mendes. *Arquitetura de Software*. Campus-Elsevier, 2002, (ISBN 853521013X)
- http://www.sei.cmu.edu/architecture/

Frameworks

Definição

- “Um conjunto de classes cooperantes que constrõem um projeto reutilizável para uma classe de software específica” (JOHNSON, 1997)

- “Um framework é uma coleção de classes abstratas e concretas e de uma interface entre elas, e um projeto para um subsistema” (WIRFS-BROCK E JOHNSON, 1990)

- “Um framework é um software parcialmente completo (subsistema) que pretende ser instanciado” (BUSCHMANN ET AL. 1996)

Classificação de *Frameworks* quanto ao escopo

- ***Frameworks* de Infra-estrutura**: *frameworks* que apóiam a infra-estrutura de qualquer tipo de sistema (ex: SISOP, interfaces de usuário, persistência de objetos, de comunicação e de processamento de linguagens).

- ***Frameworks* de Integração**: estes *frameworks* são projetados para suportar a modularização, o reúso e a integração de aplicações que apresentam componentes distribuídos. (ex: *middleware* para sistemas distribuídos)

- ***Frameworks* de Aplicação**: são dirigidos a um domínio específico de aplicações, ou seja, a uma família de aplicações em uma determinada área.

Frameworks: Características

- **Aspectos variáveis - *hot-spots***
- **Aspectos invariáveis *frozen-spots***
- ***Hot-spot* - uma parte do *framework* onde uma adaptação pode ser feita**
- ***Frozen-spot* – uma parte do *framework* que não foi projetada para adaptação.**
- **Exemplo de Hot-spots: Classes Abstratas, métodos abstratos, métodos hook, etc.**
- **Exemplo de Frozen-spots: Classes Concretas, métodos template, etc.**

Frameworks: Características

- Template Method (padrão de projeto do Gamma):
  - Assim como as Classes e métodos abstratos, este padrão está no cerne do projeto de um Framework.
  - A idéia é definir um método gabarito (o Template Method) em uma superclasse, definindo o esqueleto de um algoritmo com suas partes variantes e invariantes.
  - O Template Method invoca outros métodos, alguns dos quais são operações que podem ser redefinidas em subclasses.
  - Assim, as subclasses podem redefinir os métodos que variam, de forma a acrescentar o seu próprio comportamento específico nos pontos de variação.

Frozen-Spots e *Hot-Spots*

- Identificar e projetar os *hot-spots* em um *framework* é uma das principais dificuldades para desenvolver projetos reutilizáveis.

- Um *framework* para ter um alto grau de qualidade deve ter bons *hot-spots* para permitir futuras adaptações.

- *Frozen-spots* definem a arquitetura geral de um sistema, ou seja, seus componentes básicos e os relacionamentos entre eles.

Classificação de *Frameworks* quanto à Adaptação

- **caixa branca (*white box*):** baseiam-se no conceito de herança e ligação dinâmica que permite uma sub-classe reutilizar a interface e a implementação de sua super-classe.

- **caixa preta (*black box*):** estão baseados no conceito de composição de objetos onde estes não revelam detalhes internos de sua implementação, tendo-se somente acesso à interface do mesmo.

- **caixa cinza (*gray box*):** permite adaptação tanto por herança e ligação dinâmica, quanto por composição de componentes.

Dificuldades em *Frameworks*

- **Desenvolvimento de *frameworks***:
  - *determinar partes variáveis e semelhantes numa família de aplicações;*
  - limitar a porção de código necessária para completar a aplicação, a qual deve ser pequena;
  - testes e liberação para uso de um framework;
  - evolução do framework;
  - custo e esforço de desenvolvimento.

Dificuldades em *Frameworks*

- **Utilização de *frameworks***:
  - verificação da aplicabilidade de um framework como solução ao problema em questão;
  - estimativa de esforço na compreensão e uso do framework;
  - confiabilidade.

Referências Bibliográficas:

- SHAW, M., GARLAN, D., 1996, “Software Architecture: perspectives on an emerging discipline”, 1 ed, Nova Jersey, Prentice-Hall: 1996.

- SEI, 1994, http://www.sei.cmu.edu/ata/ata_init.html.
  PENEDO, M. H., RIDDLE, W., 1993, “Process-sensitive SEE Architecture (PSEEA) – Workshop Summary”, Software Engineering Notes, ACM SIGSOFT, vol. 18, nº 3, July, pp.A78 – A94.

- APPLETON, Brad. Patterns and Software: Essential Concepts and Terminology. Disponível na internet em: http://www.enteract.com/~bradapp/docs/patterns-intro.html.

- BUSCHMANN, F. et al. Pattern-Oriented Software Architecture: a system of patterns. John Wiley & Sons, England, 1996. 467p.

- FAYAD, Mohamed et al. Building application frameworks: object-oriented foundations of framework design. John Wiley & Sons, 1999. 664p.