Faculdade XPe

Capítulo 02

CATÁLOGO DE PADRÕES

Gladston Junio Aparecido

2024

Introdução ao Catálogo de Padrões

Arquitetura é uma das principais disciplinas da Engenharia de Software. Os produtos resultantes de seus processos contribuem de forma substancial para a garantia dos atributos de qualidade de um software. Além disso, a arquitetura envolve definições das partes elementares de um sistema e, portanto, espera-se que todas sejam fundamentadas em práticas de engenharia. Um arquiteto deve ser capaz de identificar características relevantes dos principais problemas que ocorrem durante o ciclo de vida de um software e propor soluções efetivas. Para auxiliar esse processo existem diversas propostas de padrões. Christopher Alexander define padrões como uma entidade que descreve um problema que ocorre repetidas vezes em um ambiente e apresenta uma essência de solução para esse problema, de tal forma que a solução possa ser replicada milhões de vezes sem nunca produzir resultados idênticos.

Padrões são utilizados como templates para solucionar problemas recorrentes que se manifestam ao longo do ciclo de vida de um software. Por se tratarem de propostas amplamente testadas e bem documentadas, os padrões contribuem para garantir a qualidade do produto. Além disso, os padrões colaboram para a redução dos riscos dos projetos e para o estabelecimento de um vocabulário comum entre os profissionais, sejam desenvolvedores, testers, arquitetos, etc.

Padrões na Arquitetura de Software têm sido estudados por décadas. Dado o grande número de propostas, eles são frequentemente organizados e divulgados em catálogos. Os catálogos concentram padrões com algum uso em comum, seja pelo tipo de problema ao qual se propõem a resolver ou por características das aplicações ao qual podem ser aplicados. A documentação dos padrões frequentemente possui os seguintes elementos:

- **Nome:** descreve de forma sucinta o padrão;
- **Problema:** define o problema que o padrão se propõe a resolver e quando o padrão deve ser utilizado.
- **Solução:** descreve como o padrão propõe resolver o problema.
  - A solução deve ser sempre apresentada de forma genérica para possibilitar que o padrão seja aplicado em diferentes domínios de problemas e de forma independente de tecnologia. É comum a solução descrever os elementos, os relacionamentos entre os elementos e suas responsabilidades.
- **Resultados e consequências:** descrevem os benefícios da adoção do padrão e suas consequências de forma a fundamentar a possível utilização ou não do padrão. Podem conter, por exemplo, impactos na flexibilidade, extensibilidade, portabilidade e manutenibilidade.

Existem diversos catálogos de padrões na literatura e alguns são enumerados na lista a seguir:

- **GoF:** um dos catálogos de padrões mais conhecidos e utilizados. Apresenta vinte e três padrões para solução de problemas frequentemente encontrados no desenho (projeto) de softwares orientados a objetos. Publicado no livro Design Patterns Elements of Reusable Object-Oriented Software de Gama et al.
- **AntiPatterns:** catálogo de padrões que usa uma abordagem diferente dos demais. Ele apresenta soluções ruins, que não devem ser utilizadas, para problemas de projeto e codificação de software. Tais soluções podem comprometer a compreensão, a evolução e a manutenção do código fonte. Publicado no livro AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis de Brown et al.
- **Pattern Oriented Software Architecture (POSA):** focado em padrões para desenvolvimento de sistemas de missão-crítica, muito utilizados no desenvolvimento de sistemas operacionais, servidores web, middlewares e softwares de plataforma. Publicado no livro Pattern-Oriented Software Architecture Volume 1: A System of Patterns de Buschmann et al.
- **Patterns of Enterprise Application Architecture (POEAA):** apresenta diversos padrões para “aplicações corporativas”, com foco nas linguagens .Net e Java. Publicado no livro Patterns of Enterprise Application Architecture de Martin Fowler.

- **DDD Patterns:** discute padrões relevantes para equipes que utilizam a metodologia Domain Driven Design. Os detalhes de alguns padrões de DDD podem ser consultados no livro Patterns, Principles, and Practices of Domain-Driven Design de Scott Millett e Nick Tune.
- **SOA Patterns:** o livro Patterns: Service-Oriented Architecture and Web Services de Endrei et al. apresenta um catálogo com diversos padrões para solução de problemas frequentes na implementação de arquiteturas orientadas a serviços (SOA). Os padrões envolvem questões de projeto de arquitetura SOA, implementação de barramento de comunicação, descoberta de serviços, dentre outros.

Pattern of Enterprise Application Architecture

Motivado pelos desafios frequentemente encontrados no desenvolvimento de aplicações corporativas, Martin Fowler publicou o livro Patterns of Enterprise Application Architecture onde descreve cinquenta e um padrões de projeto. Os padrões descritos neste catálogo remetem a problemas que envolvem acesso a dados, concorrência, sistemas distribuídos, desenvolvimento web, dentre outros. Nesta seção serão discutidos alguns dos padrões de acesso a dados propostos pelo referido catálogo.

Padrões de Acesso a Dados

Na arquitetura de um software existem requisitos transversais que impactam todo o sistema como, por exemplo, segurança e persistência de dados. As decisões em relação a esses requisitos têm impacto considerável e são fatores determinantes para diversos atributos de qualidade e de sucesso dos projetos. Se tratando de persistência de dados, o arquiteto deve conhecer as principais abordagens para acesso às diferentes fontes de armazenamento de dados, como bancos de dados relacionais e NoSQL, arquivos, externos, streams, etc.

A Tabela 1 enumera os principais padrões de acesso a dados documentados por Martin Fowler.

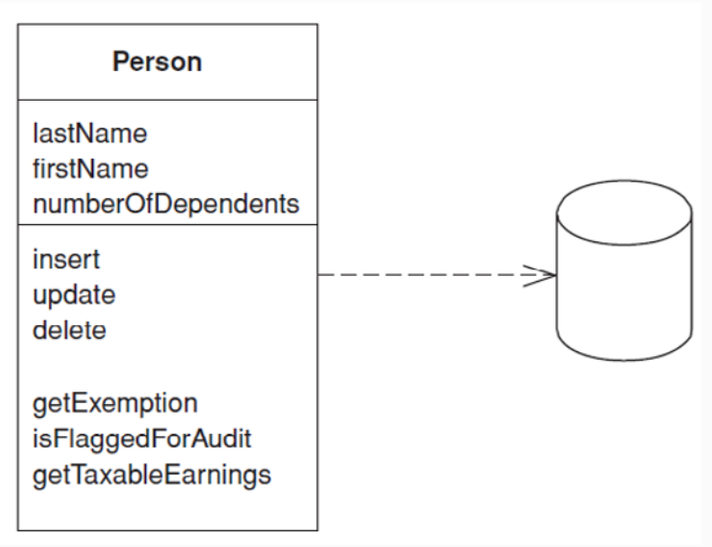
Tabela 1: Padrões de Acesso a Dados

Table Data Gateway	Identity Field
Row Data Gateway	Foreign Key Mapping
Active Record	Association Table Mapping
Data Mapper	Dependent Mapping
Unit of Work	Embedded Value
Lazy Load	Serialized LOB
Query Object	Single Table Inheritance
Record Set	Class Table Inheritance
Repository	Concrete Table Inheritance
Identity Map	Inheritance Mappers

A seguir vamos discutir brevemente alguns desses padrões.

- **Active Record:** padrão de acesso a dados recomendado para cenários em que as regras de persistência são simples. Consiste no encapsulamento, no mesmo objeto, de dados e comportamentos do negócio, assim com as regras de persistência. Sua estrutura é ilustrada Figura 1.

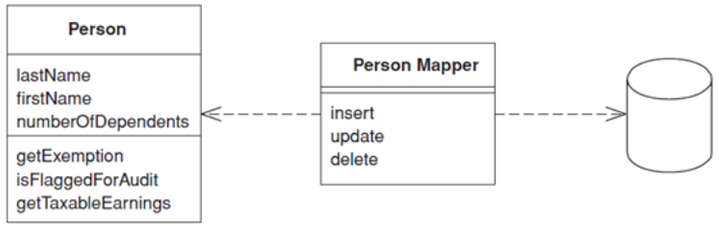
Figura 1 – Padrão Active Record



Fonte: FOWLER (2002).

- **Data Mapper:** visa criar uma camada de tradução (baseada em mapeamento) capaz de traduzir os conceitos, princípios e comandos da Orientação a Objetos em conceitos e comandos da fonte de persistência. Sua estrutura é ilustrada na Figura 2.

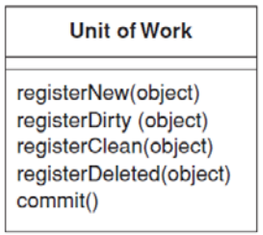
Figura 2 – Padrão Data Mapper



Fonte: FOWLER (2002).

- **Unit of Work:** disponibiliza uma estrutura capaz de armazenar, em memória, todos os objetos afetados por uma transação (de negócio e não transação de banco de dados) e, posteriormente, coordena a sensibilização da fonte de persistência com as mudanças ocorridas nos objetos. Sua estrutura é ilustrada na Figura 3.

Figura 3 – Padrão Unit of Work



Fonte: FOWLER (2002).

Esse padrão é muito utilizado em aplicações web por exemplo. Nesse cenário se cria uma unit of work para cada requisição HTTP de forma a centralizar nela as inclusões, alterações e exclusões necessárias para satisfazer o processo de negócio requisitado e, ao final da execução da requisição, caso não ocorram erros, a unit of work transfere o resultado para a fonte de persistência.

- **Repository:** um repositório funciona como uma coleção de objetos em memória que tem como objetivo atuar como uma camada intermediária de apoio. Nele se concentram as construções de queries e demais regras de acesso à fonte de persistência. Ao contrário do Data Mapper, nos repositórios os métodos específicos de acesso à fonte de persistência são públicos. O padrão repositório é recomendado para aplicações com múltiplos domínios ou domínios com queries complexas. A Figura 4 ilustra um código de exemplo de montagem de queries no padrão Repository.

Figura 4 – Padrão Repository

```
public class Person {
   public List dependents() {
      Repository repository = Registry.personRepository();
      Criteria criteria = new Criteria();
      criteria.equal(Person.BENEFACTOR, this);
      return repository.matching(criteria);
   }
}
```

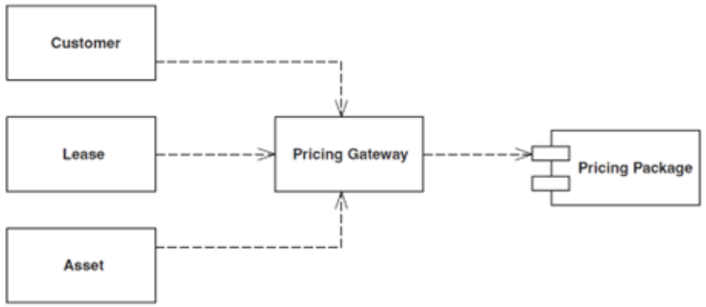
Fonte: FOWLER (2002).

Padrões para Sistemas Distribuídos

No ambiente corporativo atual, dificilmente uma aplicação tende a ser desenvolvida de forma isolada. Além disso, há décadas as empresas têm seguido a tendência de reaproveitar ao máximo suas aplicações legadas. Diante disso, os padrões que propõem soluções para o desenvolvimento de sistemas distribuídos constituem uma ferramenta importante para o trabalho dos arquitetos de softwares. Abaixo segue uma breve descrição de alguns dos padrões propostos por Martin Fowler que atuam na solução de problemas que ocorrem no desenvolvimento de sistemas distribuídos:

- **Gateway:** O padrão Gateway propõe a criação de um objeto para encapsular o acesso a um recurso externo (bancos de dados, API, web services, etc.). Conforme ilustrado na Figura 5, o gateway tem o papel de remover o acoplamento entre os objetos da aplicação e a dependência externa, o que remove a necessidade de uso dos conceitos da fonte externa (técnicos e funcionais) em diversas partes do sistema. O gateway deve ser mantido o mais simples possível e, como recomenda o autor, gerado dinamicamente sempre que possível.

Figura 5 – Padrão Gateway

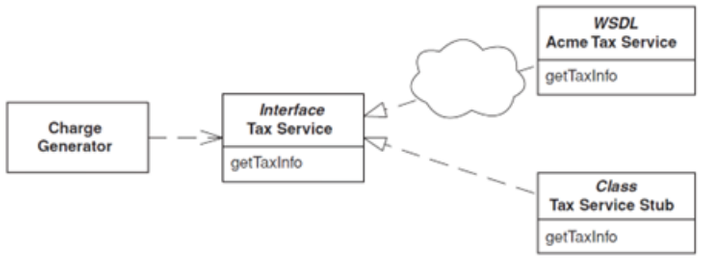


Fonte: FOWLER (2002).

Este padrão se assemelha aos padrões Adapter e ao Facade do catálogo de padrões de projeto do GoF. Porém, embora a implementação seja semelhante, seu propósito é distinto.

- **Service Hub:** A dinâmica do cotidiano no desenvolvimento de sistemas exige cada vez mais agilidade na execução das atividades. Diante disso, quando as dependências externas criadas com a integração de aplicações e o uso de fontes externas de dados apresentam lentidão ou erros em sua execução, o desempenho dos diversos testes executados ao longo do ciclo de vida de um software pode ser comprometido.

Figura 6 – Padrão Service Hub



Fonte: FOWLER (2002).

Conforme ilustrado em Figura 6, o padrão Service Hub visa remover as dependências para um serviço externo problemático durante os testes, por meio do uso de um objeto substituto, denominado stub. Neste caso, o stub simula o serviço real, provendo dados obtidos em memória e/ou de forma determinística, possibilitando a construção e execução dos casos de teste necessários e não comprometendo o tempo de execução dos testes por causas alheias à aplicação testada.

- **Registry:** Uma característica muito desejada em sistemas distribuídos é a transparência de localização (o cliente é capaz de enviar uma solicitação sem saber previamente a localização do servidor), entretanto, muitos middlewares não fornecem tal capacidade de forma nativa.
  O padrão Registry pode ser utilizado para adicionar essa característica a um sistema. Ele propõe a criação de um objeto de domínio público para a aplicação que tem o papel de encontrar objetos e serviços distribuídos. Com ele, um cliente pode obter a localização de um recurso de forma dinâmica e sempre atualizada, conforme ilustrado na Figura 7.

Figura 7 – Padrão Registry

Fonte: FOWLER (2002).

- **Remote Facade:** Na orientação por objetos, a comunicação ocorre por meio de troca de mensagens entre objetos que residem na memória do computador e, do ponto de vista de performance. Essa comunicação, por si só, tende a não comprometer o desempenho das aplicações. Entretanto, em sistemas distribuídos a comunicação entre dois objetos remotos é diretamente afetada pelo desempenho do canal de comunicação que, via de regra, é consideravelmente mais lento.
  Dessa forma, quando um objeto precisa consumir vários métodos de outro objeto remoto em sequência, o tempo de processamento é severamente comprometido. O padrão Remote Face sugere o agrupamento das diferentes chamadas (granulares) em uma única chamada, endereçada para um outro objeto denominado Remote Facade.
  Este objeto reside próximo ao objeto remoto consumido e, ao ser acessado, realiza as diversas chamadas granulares sem a degradação ocasionada pela latência e outras características do canal de comunicação.

Referências

BROWN, William J. et al. AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. 1. ed. Wiley, 2008. 336 p.

BUSCHMANN, Frank; SOMERLAD, Peter. Pattern-Oriented Software Architecture, a System of Patterns: Volume 1. 1. ed. Wiley, 1996. 476 p.

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. Sistemas Distribuídos: conceitos e projeto. 4. ed. Porto Alegre: Bookman, 2007. ISBN: 9788560031498.

FOWLER, Martin. Patterns of Enterprise Application Architecture. 1. ed. Addison-Wesley Professional, 2002. 560 p.

FOWLER, Martin. Refactoring: Improving the Design of Existing Code. 1. ed. Addison-Wesley Professional, 2018. 448 p.

FREEMAN, Eric. Head First - Design Patterns. 1. ed. O'Reilly, 2004. 638 p.

GAMMA, Erich et al. Design Patterns: Elements of Reusable Object-Oriented Software. 1. ed. Addison-Wesley Professional, 1994. 416 p.

HALL, Gary M. Adaptive Code via C#: Agile coding with design patterns and SOLID principles. 1. ed. Microsoft Press, 2014. 401 p.

HOHPE, Gregor. The Software Architect Elevator: Redefining the Architect's Role in the Digital Enterprise. 1. ed. O'Reilly, 2020. 350 p.

HOHPE, Gregor; WOOLF, Bobby. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. 1. ed. Addison-Wesley Professional, 2003. 736 p.

INGENO, Joseph. Software Architect's Handbook: Become a successful software architect by implementing effective architecture concepts. 1. ed. Packt Publishing, 2018. 594 p.

JOSHI, Bipin. Beginning SOLID Principles and Design Patterns for ASP.NET Developers. 1. ed. Apress, 2016. 420 p.

KEELING, Michael. Design It!: From Programmer to Software Architect. 1. ed. O'Reilly, 2017. 354 p.

Referências

KRAFGIZ, Dirk; BANKE, Kalr; SLAMA, Dirk. Enterprise SOA: Service-Oriented Architecture Best Practices. 1. ed. Editora Prentice Hall, 2004. 408 p.

MALVEAU, Raphael C. et al. AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. 1. ed. Wiley, 1998. 336 p.

MALVEAU, Raphael C.; MOWBRAY Thomas J. Software Architect Bootcamp. 1. ed. Prentice Hall, 2000. 352 p.

MARTIN, Robert C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. 1. ed. Pearson, 2017. 432 p.

MIKOWSKI, Michael; POWELL, Josh. Single Page Web Applications: JavaScript end-to-end. 1. ed. Greenwich:Manning Publications, 2013. 432 p.

MILLET, Scott; TUNE, Nick. Patterns, Principles and Practices of Domain-Driven Design. 1. ed. Wrox, 2014. 800 p.

MONSON-HAEFEL, Richard. 97 Things Every Software Architect Should Know: Collective Wisdom from the Expert. 1. ed. O'Reilly, 2009. 220 p.

PAI, Praseed; XAVIER, Shine. .NET Design Patterns. 1. ed. Packt Publishing, 2017. 314 p.

SHALLOWAY, Alan; TROTT, James R. Design Patterns Explained: A New Perspective on Object Oriented Design. 2. ed. Addison-Wesley Professional, 2014. 480 p.

SHYAM, Seshadri; BRAD, Green. AngularJS: Up and Running: Enhanced Productivity with Structured Web Apps Paperback. 1. ed. Sebastopol:O'Reilly Media, 2014. 275 p.

TANENBAUM, Andrew S.; STEEN, Maarten Van. Sistemas Distribuídos: princípios e paradigmas. 2. ed. Pearson/Prentice-Hall, 2007. ISBN: 9788576051428.

Faculdade XPe

xpeducacao.com.br