UNIVERSIDADE DE SÃO PAULO

INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO

Padrões e Frameworks de Software

(Notas Didáticas)

Editores:
José Carlos Maldonado
Rosana Teresinha Vaccare Braga
Fernão Stella Rodrigues Germano
Paulo Cesar Masiero

Sumário

1 – Padrões

A origem dos padrões de projeto deu-se com o trabalho feito pelo arquiteto Christopher Alexander no final dos anos 70. Ele escreveu dois livros: “A Pattern Language” [Alexander 77] e “A Timeless Way of Building” [Alexander 79] que, além de exemplificarem, descrevem seu método para documentação de padrões. O trabalho de Alexander, apesar de ser voltado para a arquitetura, possui uma fundamentação básica que pode ser abstraída para a área de software.

Os padrões permaneceram esquecidos por um tempo, até que ressurgiram na conferência sobre programação orientada a objetos (OOPSLA) de 1987, mais especificamente, no Workshop sobre Especificação e Projeto para Programação Orientada a Objetos [Beck 87]. Nesse trabalho, Beck e Cunningham falam sobre uma linguagem de padrões para projetar janelas em Smalltalk. A partir de então, muitos artigos, revistas e livros têm aparecido abordando padrões de software, que descrevem soluções para problemas que ocorrem freqüentemente no desenvolvimento de software e que podem ser reusadas por outros desenvolvedores. Um dos primeiros trabalhos estabelecendo padrões foi o de Coad [Coad 92], no qual são descritos sete padrões de análise. Nesse mesmo ano Coplien publicou um livro definindo inúmeros “idiomas”, que são padrões de programação específicos para a linguagem C++. Em 1993, Gamma e outros introduzem os primeiros de seus vinte e três padrões de projeto [Gamma 93], que seriam publicados em 1995 [Gamma 95]. Esses foram os trabalhos pioneiros na área de padrões, seguidos por muitos outros nos anos seguintes.

Um padrão descreve uma solução para um problema que ocorre com freqüência durante o desenvolvimento de software, podendo ser considerado como um par “problema/solução” [Buschmann 96]. Projetistas familiarizados com certos padrões podem aplicá-los imediatamente a problemas de projeto, sem ter que redescobri-los [Gamma 95]. Um padrão é um conjunto de informações instrutivas que possui um nome e que capta a estrutura essencial e o raciocínio de uma família de soluções comprovadamente bem sucedidas para um problema repetido que ocorre sob um determinado contexto e um conjunto de repercussões [Appleton 97].

Padrões de software podem se referir a diferentes níveis de abstração no desenvolvimento de sistemas orientados a objetos. Assim, existem padrões arquiteturais, em que o nível de abstração é bastante alto, padrões de análise, padrões de projeto, padrões de código, entre outros. As diversas categorias de padrões são discutidas a seguir. O uso de padrões proporciona um vocabulário comum para a comunicação entre projetistas, criando abstrações num nível superior ao de classes e garantindo uniformidade na estrutura do software [Gall 96]. Além disto, eles atuam como blocos construtivos a partir dos quais projetos mais complexos podem ser construídos [Gamma 95].

1.1 – Componentes de um padrão

Apesar de existirem diversos padrões de padrão, padrões têm sido descritos em diferentes formatos. Porém, existem componentes essenciais que devem ser claramente identificáveis ao se ler um padrão [Appleton 97]:

- **Name**: Todo padrão deve ter um nome significativo. Pode ser uma única palavra ou frase curta que se refira ao padrão e ao conhecimento ou estrutura descritos por ele. Se o padrão possuir mais do que um nome comumente usado ou reconhecível na literatura, subseções *"Aliases"* ou *"Also know as"* devem ser criadas.
- **Problem:** Estabelece o problema a ser resolvido pelo padrão, descreve a intenção e objetivos do padrão perante o contexto e forças específicas.
- **Context:** Pré-condições dentro das quais o problema e sua solução costumam ocorrer e para as quais a solução é desejável, o que reflete a aplicabilidade do padrão. Pode também ser considerado como a configuração inicial do sistema antes da aplicação do padrão.
- **Forces:** Descrição dos impactos, influências e restrições relevantes para o problema e de como eles interagem ou são conflitantes entre si e com os objetivos a alcançar. Um cenário concreto que serve como motivação para o padrão.
- **Solution:** Relacionamentos estáticos e regras dinâmicas descrevendo como obter o resultado desejado. Equivale a dar instruções que descrevem como o

problema é resolvido, podendo para isso utilizar texto, diagramas e figuras. São possíveis as seguintes subseções:

- "*Structure*", revelando a forma e organização estática do padrão;
- "*Participants*", descrevendo cada um desses componentes;
- "*Dynamics*", exibindo o comportamento dinâmico do padrão.
- "*Implementation*", mostrando detalhes de implementação do padrão; e
- "*Variants*", discutindo possíveis variações e especializações da solução.

- **Examples:** Uma ou mais aplicações do padrão que ilustram, num contexto inicial específico, como o padrão é aplicado e transforma aquele contexto em um contexto final.
- **Resulting context:** O estado ou configuração do sistema após a aplicação do padrão, podendo ter uma subseção "*Conseqüências*" (tanto boas quanto ruins). Descreve as pós-condições e efeitos colaterais do padrão.
- **Rationale:** Uma explicação das regras ou passos do padrão que explicam como e porque ele trata suas influências contrárias, definidas em 'Forces', para alcançar os objetivos, princípios e filosofia propostos. Isso nos diz realmente como o padrão funciona, porque funciona e porque ele é bom.
- **Related Patterns:** Os relacionamentos estáticos e dinâmicos desse padrão com outros dentro da mesma linguagem ou sistema de padrões. Padrões relacionados geralmente compartilham as mesmas influências. São possíveis os seguintes tipos de padrões relacionados:
  - padrões predecessores, cuja aplicação conduza a esse padrão;
  - padrões sucessores, que devem ser aplicados após esse;
  - padrões alternativos, que descrevem uma solução diferente para o mesmo problema mas diante de influências e restrições diferentes; e
  - padrões codependentes, que podem (ou devem) ser aplicados simultaneamente com esse padrão.
- **Known Uses:** Descreve ocorrências conhecidas do padrão e sua aplicação em sistemas existentes. Isso ajuda a validar o padrão, verificando se ele é realmente uma *solução provada para um problema recorrente.*

1.2 – Anti-padrões

Anti-padrões representam uma “lição aprendida”, ao contrário dos padrões, que representam a “melhor prática” [Appleton 97]. Os anti-padrões podem ser de dois tipos:

- Aqueles que descrevem uma solução ruim para um problema que resultou em uma situação ruim
- Aqueles que descrevem como escapar de uma situação ruim e como proceder para, a partir dela, atingir uma boa solução.

Os anti-padrões são necessários porque é tão importante ver e entender soluções ruims como soluções boas. Se por um lado é útil mostrar a presença de certos padrões em sistemas mal sucedidos, por outro lado é útil mostrar a ausência dos anti-padrões em sistemas bem sucedidos.

Segundo uma outra definição [CMG 98] um anti-padrão é:

- um padrão que diz como ir de um problema para uma solução ruim ou
- um padrão que diz como ir de uma solução ruim para um solução boa.

A primeira parece inútil, exceto se considerarmos o padrão como uma mensagem “Não faça isso”. O segundo é definitivamente um padrão positivo, no qual o problema é a solução ruim.

1.3 – Classificação de Padrões

Conforme já mencionado, padrões de software abrangem diferentes níveis de abstração, podendo portanto ser classificados em diversas categorias de modo a facilitar sua recuperação e uso. Porém, essa classificação não é rigorosa, podendo haver padrões que se encaixem em mais do que uma categoria. A seguir resumem-se algumas categorias importantes de padrões. Outras categorias de padrões são discutidas em [Coplien 95, Vlissides 96, Martin98].

- **Padrões de processo**: definem soluções para os problemas encontrados nos processos envolvidos na engenharia de software: desenvolvimento, controle de configuração, testes, etc.

- **Padrões arquiteturais**: expressam o esquema ou organização estrutural fundamental de sistemas de software ou hardware.
- **Padrões de padrão** (em inglês, *patterns on patterns*): são padrões descrevendo como um padrão deve ser escrito, ou seja, que padronizam a forma com que os padrões são apresentados aos seus usuários.
- **Padrões de análise:** descrevem soluções para problemas de análise de sistemas, embutindo conhecimento sobre o domínio de aplicação específico.
- **Padrões de projeto:** definem soluções para problemas de projeto de software.
- **Padrões de interface:** definem soluções para problemas comuns no projeto da interface de sistemas. É um caso particular dos padrões de projeto.
- **Padrões de programação:** descrevem soluções de programação particulares de uma determinada linguagem ou regras gerais de estilo de programação.
- **Padrões de Persistência:** descrevem soluções para problemas de armazenamento de informações em arquivos ou bancos de dados.
- **Padrões para Hipertexto:** descrevem soluções para problemas encontrados no projeto de hipertextos.
- **Padrões para Hipermídia:** descrevem soluções para problemas encontrados no desenvolvimento de aplicações hipermídia.

1.4 – Coletâneas de padrões

Conforme já mencionado, diversos autores têm proposto centenas de padrões nas mais diversas áreas de aplicação. Esses padrões são descobertos após a abstração de fatores comuns em diversos pares problema-solução, podendo-se situar em várias faixas de escala e abstração. Existem padrões de domínio específico, como por exemplo para sistemas multimídia educativos e também padrões independentes de domínio, como por exemplo padrões para projeto de interface. Há padrões que ajudam a estruturar um sistema de software em subsistemas e outros que ajudam a implementar aspectos de projeto particulares.

Olhando mais de perto para muitos padrões, percebe-se que um padrão resolve um problema, mas sua aplicação pode gerar outros problemas, que podem ser resolvidos por

outros padrões. Em geral não existem padrões isolados: um padrão pode depender de outro no qual esteja contido, ou das partes que ele contém, ou ser uma variação de outro, ou ser uma combinação de outros. De maneira geral, um padrão e seus variantes descrevem soluções para problemas muito similares, que variam em algumas das influências envolvidas.

Portanto, deve-se pensar em formas de agrupar os padrões existentes segundo algum critério, de forma a facilitar sua recuperação e reuso. Isso pode ser feito por meio de coleções de padrões, catálogos de padrões, sistemas de padrões ou linguagem de padrões.

Este capítulo têm por objetivo fornecer os conceitos relacionados a esses quatro tipos de agrupamento de padrões, bem como exemplificá-los, deixando clara a diferença entre eles e a utilidade de cada um deles em particular.

1.4.1 – Coleções de padrões

Uma coleção de padrões é uma coletânea qualquer de padrões que não possuem nenhum vínculo entre si e, em geral, nenhuma padronização no formato de apresentação. Os padrões podem estar reunidos por terem sido apresentados em um mesmo congresso, por terem sido propostos pelo mesmo autor, ou por se referirem a um mesmo domínio, mas não possuem um relacionamento semântico significativo.

1.4.2 – Catálogos de padrões

Um catálogo de padrões é uma coleânea de padrões relacionados (talvez relacionados apenas fracamente ou informalmente). Em geral subdivide os padrões em um pequeno número de categorias abrangentes e pode incluir algumas referências cruzadas entre os padrões [Appleton 97].

Um catálogo de padrões pode oferecer um esquema de classificação e recuperação de seus padrões, já que eles estão subdivididos em categorias. Um catálogo de padrões adiciona uma certa quantidade de estrutura e organização a uma coleção de padrões, mas usualmente não vai além de mostrar apenas as estruturas e relações mais superficiais (se é que o faz) [Appleton 97].

1.4.3 – Sistemas de padrões

Um sistema de padrões é um conjunto coeso de padrões co-relacionados que trabalham juntos para apoiar a construção e evolução de arquiteturas completas. Além de ser organizado em grupos e subgrupos relacionados em múltiplos níveis de granularidade, um sistema de padrões descreve as diversas inter-relações entre os padrões e seus grupamentos e como eles podem ser combinados e compostos para resolver problemas mais complexos. Os padrões num sistema de padrões devem ser descritos num estilo consistente e uniforme e precisam cobrir uma base de problemas e soluções suficientemente abrangente para permitir a construção de parcelas significativas de arquiteturas completas [Appleton 97].

Um sistema de padrões interliga padrões individuais, descreve como os padrões que o constituem são conectados com outros padrões no sistema, como esses padrões podem ser implementados e como o desenvolvimento de software com padrões é apoiado. Um sistema de padrões é um veículo poderoso para expressar e construir arquiteturas de software [Buschmann 96].

Além da estrutura e organização oferecidas por um catálogo de padrões, num sistema de padrões é adicionada uma maior profundidade à estrutura, maior riqueza à interação dos padrões e maior uniformidade ao catálogo de padrões [Appleton 97].

1.4.4 – Linguagens de padrões

Uma linguagem de padrões é uma coleção estruturada de padrões que se apoiam uns nos outros para transformar requisitos e restrições numa arquitetura [Coplien 98].

Os padrões que constituem uma linguagem de padrões cobrem todos os aspectos importantes em um dado domínio. Pelo menos um padrão deve estar disponível para cada aspecto da construção e implementação de um sistema de software: não pode haver “vazios” ou “brancos”.

Uma linguagem de padrões é uma forma de subdividir um problema geral e sua solução complexa em um número de problemas relacionados e suas respectivas soluções. Cada padrão da linguagem resolve um problema específico no contexto comum compartilhado pela linguagem. É importante notar que cada padrão pode ser usado

separadamente ou com um certo número de padrões da linguagem. Isso significa que um padrão sozinho é considerado útil mesmo se a linguagem não for ser usada em sua plenitude.

Conforme já foi visto, a linguagem de padrões proposta por [Meszaros & Doble 98] possui uma sub-seção dedicada a padrões a serem seguidos por linguagens de padrões, dentre os quais padrões especificando que uma linguagem de padrões deve: dar ao leitor uma visão geral da ling. de padrões, fornecer uma tabela resumindo todos os padrões, deixar claro quando houver formas alternativas para resolver o mesmo problema, deixar clara a estruturação da linguagem, utilizar um mesmo exemplo em toda a linguagem para ilustrar cada padrão e oferecer um glossário de termos como parte da linguagem.

Parte das linguagens de padrões encontradas na literatura descrevem seus padrões utilizando a forma de Alexander e outra parte utiliza variações dos padrões de padrão.

1.4.5 – Síntese de trabalhos afins

Buschmann e outros [Buschmann 96] distinguem sistemas de padrões de linguagens de padrões, salientando que uma linguagem de padrões tem a obrigação de ser completa em um certo domínio restrito, isto é, tem que ter pelo menos um padrão disponível para cada aspecto da construção e implementação de sistemas de software. Não pode haver falhas ou omissões. Já um sistema de padrões pode abranger um domínio mais amplo, sem ter essa obrigação de ser completo. Sua definição de um sistema de padrões para arquitetura de software é: "uma coleção de padrões para arquitetura de software, juntamente com diretrizes para sua implementação, combinação e uso prático no desenvolvimento de software". Tal sistema deve dar apoio ao desenvolvimento de sistemas de alta qualidade, que satisfaçam tanto a requisitos funcionais quanto não-funcionais.

Os requisitos desejáveis para um sistema de padrões são os seguintes:

- Deve abranger uma base suficiente de padrões: deve haver padrões para especificação da arquitetura básica do sistema, padrões que ajudem ao refinamento dessa arquitetura básica e padrões que ajudem a implementar essa arquitetura de software em uma linguagem de programação específica

- Deve descrever todos os padrões de maneira uniforme: a forma de descrição deve captar tanto a essência do padrão quanto a definição precisa de seus detalhes e deve permitir a comparação do padrão com outros
- Deve expor os vários relacionamentos entre padrões, tais como: refinamentos, combinações, etc.
- Deve organizar os padrões que o constituem: o usuário deve poder encontrar rapidamente um padrão que resolva seu problema de projeto concreto e deve ser possível que o usuário explore soluções alternativas endereçadas por padrões diferentes
- Deve apoiar a construção de sistemas de software: deve haver diretrizes de como aplicar e implementar seus padrões constituintes
- Deve apoiar sua própria evolução: assim como a evolução tecnológica, um sistema de padrões deve evoluir também. Assim, deve ser permitida a modificação de padrões existentes, a melhoria de sua descrição, a adição de novos padrões e a retirada de padrões não mais utilizados

Riehle e Züllighoven [Riehle 95] apresentam uma linguagem de padrões chamada "A Pattern Language for Tool Construction and Integration Based on the Tools and Materials Metaphor". Essa linguagem foi desenvolvida para ajudar no aprendizado da "metáfora de ferramentas e materiais", segundo a qual o trabalho possui uma perspectiva específica: as pessoas tem competência e talento necessário para seu trabalho; não há necessidade de definir um fluxo de trabalho fixo, porque as pessoas saber o que querem e podem arcar com mudanças na situação de forma adequada e as pessoas devem poder decidir sozinhas como organizar seu trabalho e seu ambiente (inclusive o software que utilizam), de acordo com suas tarefas. A idéia principal dessa metáfora é que a maioria dos objetos de um domínio de aplicação devem pertencer ou a uma "ferramenta" ou a um "material". Ferramentas são a forma de operar em materiais, onde materiais são resultado do trabalho em um domínio.

Aarsten e outros [Aarsten 95] apresentam a G++, uma linguagem de padrões para manufatura integrada ao computador. O problema abordado por essa linguagem é o projeto de sistemas de informações concorrentes e provavelmente distribuídos, com aplicações na manufatura integrada ao computador. O objetivo do trabalho é aumentar o reuso de projeto

orientado a objetos desde o nível de componentes até o nível arquitetural, oferecendo um modelo conceitual para arquiteturas concorrentes e distribuídas.

Cunningham [Cunningham 95] apresenta uma linguagem de padrões chamada "*The CHECKS Pattern Language of Information Integrity*", que diz como fazer checagem nas entradas de dados para separar dados inválidos dos válidos e assegurar que o menor número possível de dados inválidos seja registrado. O objetivo é fazer essa checagem sem complicar os programas e comprometer a flexibilidade futura.. Em [Cunningham 96] ele apresenta a linguagem de padrões chamada "*EPISODES: A Pattern Language of Competitive Development*", que descreve uma forma de desenvolvimento de software apropriada para organizações "entreprenurial". Assume-se que os desenvolvedores dessa organização trabalham em equipes pequenas de pessoas brilhantes e altamente motivadas e que o tempo de mercado é altamente valorizado pela organização. Essa organização também espera liberar outras versões em um tempo razoável, isto é, ela espera ter sucesso e explorar esse sucesso pelo tempo que os clientes desejarem.

Kerth [Kerth 95] apresenta uma linguagem de padrões chamada "*Caterpillar's Fate: A Pattern Language for Transformation from Analysis to Design*". Ela é usada para apoiar a transformação dos documentos de análise em um projeto inicial de software. O nome da linguagem ("destino da lagarta") vem da analogia que é feita entre a transição da análise para o projeto com a lagarta que magicamente se transforma em uma borboleta. A linguagem tenta captar a sabedoria adquirida durante o desenvolvimento de soluções de projeto a partir de modelos de análise, documentando o que deve ser feito durante a transição.

Braga e outros [Braga 99] apresentam uma linguagem de padrões chamada "*A Pattern Language for Business Resource Management*", que é composta de padrões para guiar o desenvolvimento de sistemas para gerenciamento de recursos de negócios tais como: produtos, carros, serviços, etc. A linguagem é composta de quinze padrões

Algumas outras linguagens de padrões podem ser citadas, como a "*Evolving Frameworks – A Pattern Language for developing object-oriented frameworks*" [Roberts 98], a "*A Pattern Language for Pattern Writing*" [Meszaros 98], a "*A Pattern Language for Developing Form Style Windows*" [Bradac 98], a "*A Pattern Language of Transport Systems (Point and Route)"* [Zhao 98] e a "Patterns for System Testing..

1.4.6 – Exemplos

Coleções de padrões

[Vlissides 96] é uma coleção de padrões, no caso dos padrões apresentados no PLOP'95. A Tabela 1 mostra alguns deles. Estudando esses padrões de forma detalhada, observa-se que eles abrangem domínios e níveis de abstração variados, e não possuem relacionamento explícito. São padrões escritos por diversos autores diferentes e de maneira independente, isto é, um autor não tinha conhecimento do que os demais autores estavam escrevendo. O editor do livro agrupou os padrões que pertencem ao mesmo domínio ou fase do processo de desenvolvimento em capítulos, mas isso não é suficiente para que essa coleção de padrões possa ser considerada um catálogo de padrões.

Tabela 1 – Coleção de Padrões [Vlissides 96]

Capítulo	**Padrão**	**Autor**
2- Padrões de propósitos gerais	Command Processor	Peter Sommerlad
	Shopper	Jim Doble
3- Padrões de propósitos específicos	A Structural Pattern for Designing Transparent Layered Services	Aamod Sane e Roy Campbell
	Patterns for Processing Satellite Telemetry with Distributed Teams	Stephen Berczuk
	A Pattern Language for Object-RDBMS Integration	Kyle Brown e Bruce Whitenack
	Transactions and Accounts	Ralph Johnson
6- Padrões de Exposição	Patterns for Classroom Education	Dana Anthony
	A Pattern Language for the Preparation of Software Demonstrations	Todd Coram
	A Pattern Language for na Essay-Based Web Site	Robert Orenstein

Catálogos de padrões

[Gamma 95] é um catálogo de padrões de projeto, pois possui mais estrutura e organização, exibindo também relações entre os padrões. A Tabela 2 exibe os padrões desse catálogo. Deve-se lembrar que os critérios para classificação dos padrões são dois: escopo e propósito.

Tabela 2 – Catálogo de Padrões [Gamma 95]

		Propósito		
		De Criação	**Estrutural**	**Comportamental**
Escopo	**Classe**	Factory Method	Adapter	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Flyweygth Proxy	Chain of Responsability Command Iterator Mediator Memento Observer State Strategy Visitor

No critério “escopo” decide-se se o padrão atua sobre uma classe ou sobre objetos da classe. Padrões da classe lidam com os relacionamentos entre classes e suas subclasses, por meio do uso de herança, sendo portanto estabelecidos de forma estática (em tempo de compilação). Padrões do objeto lidam com relacionamentos entre objetos, que podem ser modificados durante a execução e são mais dinâmicos. A maioria dos padrões utiliza herança de alguma forma. Portanto os únicos padrões classificados na categoria “classe” são os que se concentram em relacionamentos de classes.

No critério “propósito” existem padrões de criação, padrões estruturais e padrões comportamentais. Padrões de criação concentram-se no processo de criação de objetos. Padrões estruturais lidam com a composição de classes ou objetos. Padrões comportamentais caracterizam as formas pelas quais classes ou objetos interagem e distribuem responsabilidades.

Gamma e outros propõem também uma outra forma de organizar seus padrões de projeto, mostrada na Figura 1. Nessa figura os padrões estão relacionados de acordo com a maneira com que cada um referencia os outros na seção “Related Patterns”.

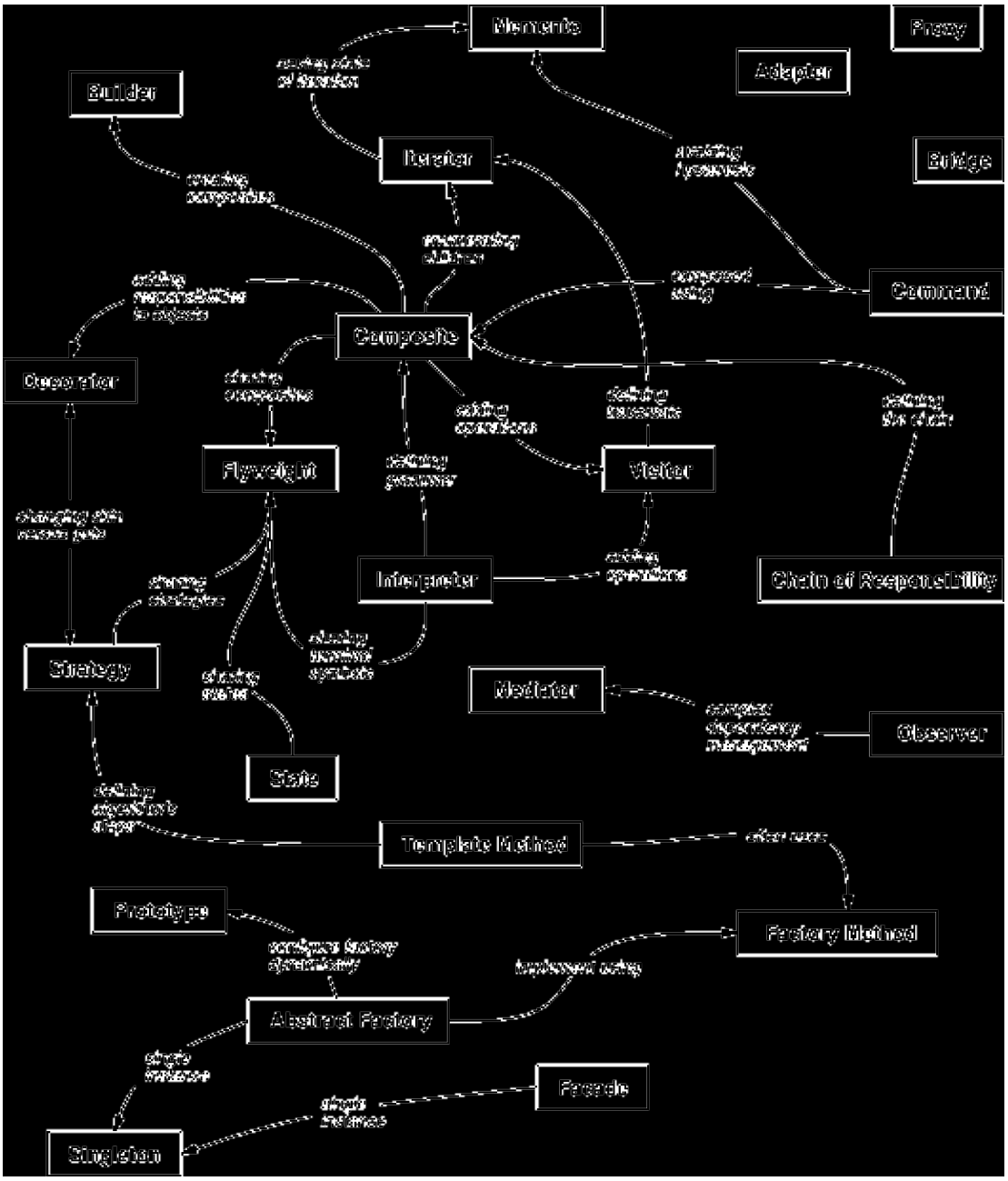

Figura 1 – Relacionamento entre padrões [Gamma 95]

Sistemas de padrões

[Buschmann 96] é um exemplo de um sistema de padrões, já que dá maior destaque à estrutura e à interação entre os padrões e os apresenta com maior uniformidade. A Tabela 3 mostra os padrões desse sistema, no qual são usados dois critérios para classificação: categorias de padrões e categorias de problemas. As categorias de padrões propostas por Buschmann e outros são:

- **padrões arquiteturais**, que podem ser usados no início do projeto de alto nível, quando se faz a especificação da estrutura fundamental de uma aplicação;
- **padrões de projeto**, que são aplicados no final do projeto, quando se refina e se estende a arquitetura fundamental de um sistema de software e
- **idiomas**, que são usados na fase de implementação para transformar a arquitetura de software em um programa escrito numa linguagem específica

Abstraindo os problemas específicos que ocorrem durante o desenvolvimento de um software podem ser definidas categorias de problemas, cada uma das quais envolvendo diversos problemas relacionados. Essas categorias de problemas correspondem diretamente a situações concretas de projeto. As categorias de problemas propostas por Buschamnn e outros para agrupar os padrões que fazem parte do seu sistema de padrões são:

- **Da lama para a estrutura** (em inglês, "*From Mud to Structure*"): padrões que apoiam a decomposição adequada de uma tarefa do sistema em sub-tarefas que cooperam entre si;
- **Sistemas Distribuídos** (em inglês, "*Distributed Systems"*): padrões que fornecem infra-estrutura para sistemas que possuem componentes localizados em processadores diferentes ou em diversos sub-sistemas e componentes;
- **Sistemas Interativos** (em inglês, "*Interactive Systems"*): padrões que ajudam a estruturar sistemas com interface homem-máquina;
- **Sistemas Adaptáveis** (em inglês, "*Adaptable Systems"*): padrões que fornecem infra-estruturas para a extensão e adaptação de aplicações em resposta a requisitos de evolução e mudança funcional;
- **Decomposição Estrutural** (em inglês "*Structural Decomposition"*): padrões que apoiam a decomposição adequada de sub-sistemas e componentes complexos em partes cooperativas;

- **Organização do Trabalho** (em inglês, *"Organization of Work"*): padrões que definem como componentes colaboram para oferecer um serviço complexo;
- **Controle de Acesso** (em inglês, *"Access Control"*): padrões que guardam e controlam o acesso a serviços e componentes;
- **Gerenciamento** (em inglês, *"Management"*): padrões para lidar com coleções homogêneas de objetos, serviços e componentes como um todo;
- **Comunicação** (em inglês, *"Communication"*): padrões que ajudam a organizar a comunicação entre componentes e
- **Manuseio de Recursos** (em inglês, *"Resource Handling"*): padrões que ajudam a gerenciar componentes e objetos compartilhados

Tabela 3 – Sistema de Padrões [Buschmann 96]

	Padrões arquiteturais	**Padrões de projeto**	**Idiomas**
Da lama para a estrutura	Layers Pipes and Filters Blackboard		
Sistemas Distribuídos	Brokers Pipes and Filters Microkernel		
Sistemas Interativos	MVC PAC		
Sistemas adaptáveis	Microkernel Reflection		
Decomposição da estrutura		Whole-Part	
Organização do trabalho		Master-Slave	
Controle de Acesso		Proxy	
Gerenciamento		Command Processor View Handler	
Comunicação		Publisher-Subscriber Forwarder-Receiver Cliente-Dispatcher-Server	
Manuseio de Recursos			Counted Pointer

Linguagem de padrões

[Meszaros 96] é uma linguagem de padrões para melhorar a capacidade e confiabilidade de sistemas reativos. Esses padrões são geralmente encontrados em sistemas

de telecomunicações, embora possam também ser aplicados a quaisquer sistemas reativos com características de pico de capacidade. Por sistemas reativos entende-se aqueles sistemas cuja função principal é a de responder a requisições de usuários de fora do sistema. Esses usuários podem ser pessoas ou máquinas. Exemplos de sistemas reativos são: sistemas computacionais de tempo compartilhado, centrais telefônicas de comutação, servidores, processamento de transações "on-line" (como redes bancárias ATM, etc.). Todos esses sistemas precisam de alta capacidade a um custo razoável e alta confiabilidade quando à frente de cargas extremas que podem levar à sobrecarga do sistema.

Essa linguagem de padrões propõe padrões para lidar com a necessidade de sistemas de alta capacidade e sobrevivência diante de condições de sobrecarga do sistema. A Figura 2 ilustra, de forma gráfica, os padrões que constituem a linguagem e seus relacionamentos, enquanto a Tabela 4 lista os padrões e seus respectivos propósitos.

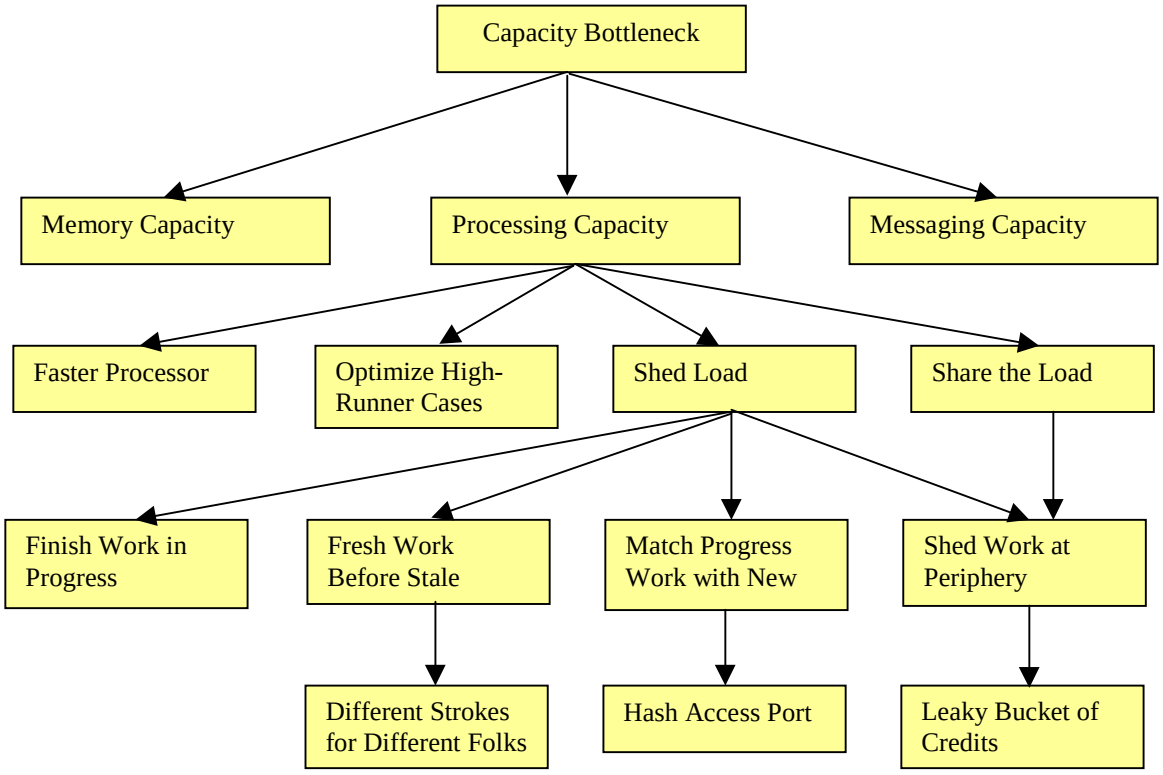

Figura 2 – Estrutura da Linguagem de Padrões apresentada graficamente

Para ilustrar a similaridade de um padrão de uma linguagem em relação a um padrão convencional, descreve-se a seguir o padrão "Finish Work in Progress". Deve-se observar que, como não existe um formato padrão para escrever os padrões de uma

linguagem, os autores têm usado o formato que acreditam ser mais apropriado para documentar os padrões que ele tem em mãos. No caso desse exemplo, Meszaros utilizou o formato "Nome/ Alias/ Problem/ Context/ Forces/ Solution/ Related Patterns". Observe na descrição do padrão que referências a outros padrões da linguagem são colocadas em sublinhado. Isso garante que o usuário da linguagem fique consciente do inter-relacionamento entre seus diversos padrões.

Tabela 4 – Padrões da Linguagem e propósitos

Padrão	Propósito
Capacity Bottleneck	Quando há um gargalo na capacidade do sistema, como proceder.
Memory Capacity	Quando a causa do gargalo na capacidade do sistema é a capacidade de memória, como proceder.
Processing Capacity	Quando a causa do gargalo na capacidade do sistema é a capacidade de processamento, como proceder.
Messaging Capacity	Quando a causa do gargalo na capacidade do sistema é a dificuldade de enviar mensagens entre processadores a tempo, como proceder.
Faster Processor	Quando um processador mais rápido é necessário para aumentar a capacidade de processamento do sistema, como proceder.
Optimize High-Runner Cases	Quando a otimização dos casos de maior demanda é necessária para aumentar a capacidade de processamento do sistema, como proceder.
Shed Load	Quando o descarte de parte da demanda é necessário para evitar sobrecarga da capacidade de processamento do sistema, como proceder.
Share the Load	Quando o atendimento da demanda deve ser compartilhado por mais de um processador para aumentar a capacidade de processamento do sistema, como proceder.
Finish Work in Progress	Quando o término do atendimento de trabalho em curso é necessário para evitar sobrecarga da capacidade de processamento do sistema,, como proceder.
Fresh Work before Stale	Quando o atendimento prioritário de certos clientes é necessário para evitar sobrecarga da capacidade de processamento do sistema, como proceder.
Match Progress work with new	Quando é necessário evitar o desperdício dos recursos já empregados no atendimento de certos clientes, causado por sua desistência de obter atendimento, como proceder.
Shed Work at Periphery	Quando é necessário o descarte da demanda que está além da capacidade de processamento do sistema, como proceder para detectar o quanto antes o que deverá ser descartado.
Different Strokes for Different Folks	Quando é necessário o estabelecimento de prioridades para o atendimento de certos clientes, como proceder para garantir que esse atendimento seja de boa qualidade.
Hash Access Port	Quando o estabelecimento de funções Hash na porta de entrada de atendimento prioritário é necessário para estabelecer as prioridades, como proceder.
Leaky Bucket of Credits	Quando reservatórios com vazamento são necessários para armazenar os créditos enviados aos processadores periféricos por processadores sobrecarregados que todavia ainda possuem capacidade de processamento disponível, como proceder

Padrão 5 – Finish Work in Progress (Termine o Trabalho em Andamento)

Também conhecido como: Termine o que você começou

Problema: Quais requisições devem ser aceitas e quais devem ser rejeitadas de forma a melhorar o "*throughput*" do sistema

Contexto: Você está trabalhando com um sistema reativo ou orientado a transações, no qual as requisições do usuário dependem de certa forma umas das outras. Uma requisição pode se apoiar em uma requisição prévia, fornecer informação adicional em resposta (ou em antecipação) a outra requisição, ou cancelar ou desfazer uma requisição prévia.

Influências: Entender as interdependências entre requisições é crucial para separar trabalho de forma produtiva. Rejeitar o trabalho errado pode negar investimento prévio considerável, ou pior ainda, resultar em "*deadlock*" por "pendurar" algumas transações juntamente com todos os recursos que elas estejam usando. Falhar na identificação do trabalho que pode ser rejeitado inibe a melhoria da capacidade do sistema pela aplicação do padrão Shed Load.

Solução: Requisições que são continuações de trabalho em andamento devem ser reconhecidas como tal e devem ser priorizadas em relação à requisições totalmente novas. Claramente categorize todas as requisições em categorias "Novo" e "Em Andamento", e assegure que todas as requisições do tipo "Em Andamento" sejam processadas antes das do tipo "Novo". A única exceção a essa regra é quando o tempo decorrido entre as requisições iniciais e as seguintes é fixo e previsível (por exemplo, t segundos). Nessa situação, bloquear completamente todas as requisições novas resultará em uma "queda de carga" t segundos depois que o sistema começar a separar o trabalho novo. Isso cria uma oscilação entre uma condição de sobrecarga e "subcarga"; o resultado é uma capacidade média reduzida. Isso pode ser contornado admitindo uma pequena quantidade de trabalho novo durante o período de sobrecarga para reduzir a profundidade da queda.

Padrões Relacionados: Esse padrão apoia a aplicação do padrão Shed Load.

1.5 – Regras para derivação de novos padrões

Algumas regras para derivação de novos padrões ("Pattern mining") são sugeridas por Buschmann e outros [Buschmann 96]. São elas:

1. Encontre pelo menos três exemplos nos quais um problema é resolvido efetivamente usando a mesma solução. .....
2. Extraia a solução, o problema e as influências
3. Declare a solução como candidata a padrão
4. Execute um “writer’s workshop” para melhorar a descrição do candidato e compartilhá-lo com outros
5. Aplique o candidato a padrão em um outro projeto de desenvolvimento de software
6. Declare o candidato a padrão como padrão se sua aplicação for bem sucedida. Caso contrário tente procurar uma solução melhor.

1.6 – Desenvolvimento de sistemas usando padrões

Segundo Buschmann e outros [Buschmann 96], padrões não definem um novo método para desenvolvimento de software que substitua os já existentes. Eles apenas complementam os métodos de análise e projeto gerais e independentes do problema, p.ex, Booch, OMT, Shlaer/Mellor, etc., com diretrizes para resolver problemas específicos e concretos. Em [Buschmann 96] sugerem-se os seguintes passos para desenvolver um sistema usando padrões de software:

1. Utilize seu método preferido para o processo de desenvolvimento de software em cada fase do desenvolvimento.
2. Utilize um sistema de padrões adequado para guiar o projeto e implementação de soluções para problemas específicos, isto é, sempre que encontrar um padrão que resolva um problema de projeto presente no sistema, utilize os passos de implementação associados a esse padrão. Se esses se referirem a outros padrões, aplique-os recursivamente.
3. Se o sistema de padrões não incluir um padrão para seu problema de projeto, tente encontrar um padrão em outras fontes conhecidas.
4. Se nenhum padrão estiver disponível, aplique as diretrizes de análise e projeto do método que você está usando. Essas diretrizes fornecem pelo menos algum apoio útil para resolver o problema de projeto em mãos.

Segundo Buschmann, essa abordagem simples evita que se criem outros métodos de projeto. Ela combina a experiência de desenvolvimento de software captada pelos métodos de análise e projeto com as soluções específicas para problemas de projeto descritas pelos padrões.

Os autores destas notas didáticas discordam quanto a essa afirmação, por acharem que está claro que a abordagem proposta cria um novo método de desenvolvimento de sistemas, diferente dos já existentes.

1.7 – Exercícios

1) Discuta a evolução de uma coleção de padrões para se tornar um catálogo, um sistema e uma linguagem de padrões.
2) Quais as dificuldades de recuperação de um padrão nas diversas formas de agrupamento dos mesmos?
3) Apresente mais um critério de classificação que poderia ser útil na recuperação de padrões.
4) Em que você concorda/discorda em relação à proposta para derivação de novos padrões? Por que?
5) Com relação à seção 1.6, você concorda com Buschmann ou com os autores destas notas didáticas? Por que?

2 – Frameworks

2.1 – Introdução

Após a popularização da linguagem Smalltalk, no início da década de 80, paralelamente às bibliotecas de classes, começaram a ser construídos frameworks de aplicação, que acrescentam às bibliotecas de classes os relacionamentos e interação entre as diversas classes que compõem o framework. Com os frameworks, reutilizam-se não somente as linhas de código, como também o projeto abstrato envolvendo o domínio de aplicação.

Framework é definido por Coad como um esqueleto de classes, objetos e relacionamentos agrupados para construir aplicações específicas [Coad 92] e por Johnson como um conjunto de classes abstratas e concretas que provê uma infra-estrutura genérica de soluções para um conjunto de problemas [Johnson 88]. Essas classes podem fazer parte de uma biblioteca de classes ou podem ser específicas da aplicação. Frameworks possibilitam reutilizar não só componentes isolados, como também toda a arquitetura de um domínio específico. O Capítulo 2 explica em detalhes os conceitos associados a frameworks, exemplificando diversos deles.

Diversos *frameworks* têm sido desenvolvidos nas duas últimas décadas, visando o reuso de software e conseqüentemente a melhoria da produtividade, qualidade e manutenibilidade. Neste capítulo são definidos os *frameworks de software*, bem como diversos conceitos básicos necessários para entender sua finalidade. A seguir são dados exemplos de alguns frameworks existentes e o framework *Model-View-Controller* é visto em maiores detalhes.

2.2 – Conceitos & Terminologia

2.2.1 – Definições

Um “Framework” é o projeto de um conjunto de objetos que colaboram entre si para execução de um conjunto de responsabilidades. Um framework reusa análise, projeto e código. Ele reusa análise porque descreve os tipos de objetos importantes e como um

problema maior pode ser dividido em problemas menores. Ele reusa projeto porque contém algoritmos abstratos e descreve a interface que o programador deve implementar e as restrições a serem satisfeitas pela implementação. Ele reusa código porque torna mais fácil desenvolver uma biblioteca de componentes compatíveis e porque a implementação de novos componentes pode herdar grande parte de seu código das super-classes abstratas. Apesar de todos os tipos de reuso serem importantes, o reuso de análise e de projeto são os que mais compensam a longo prazo [Johnson 91].

2.2.2 – Inversão de controle

Uma característica importante dos frameworks é que os métodos definidos pelo usuário para especializá-lo são chamados de dentro do próprio framework, ao invés de serem chamados do código de aplicação do usuário [Johnson 88]. O framework geralmente faz o papel de programa principal, coordenando e sequenciando as atividades da aplicação. Essa inversão de controle dá força ao framework para servir de esqueletos extensíveis. Os métodos fornecidos pelo usuário especializam os algoritmos genéricos definidos no framework para uma aplicação específica.

A Figura 3 mostra algumas diferenças básicas entre bibliotecas e frameworks.

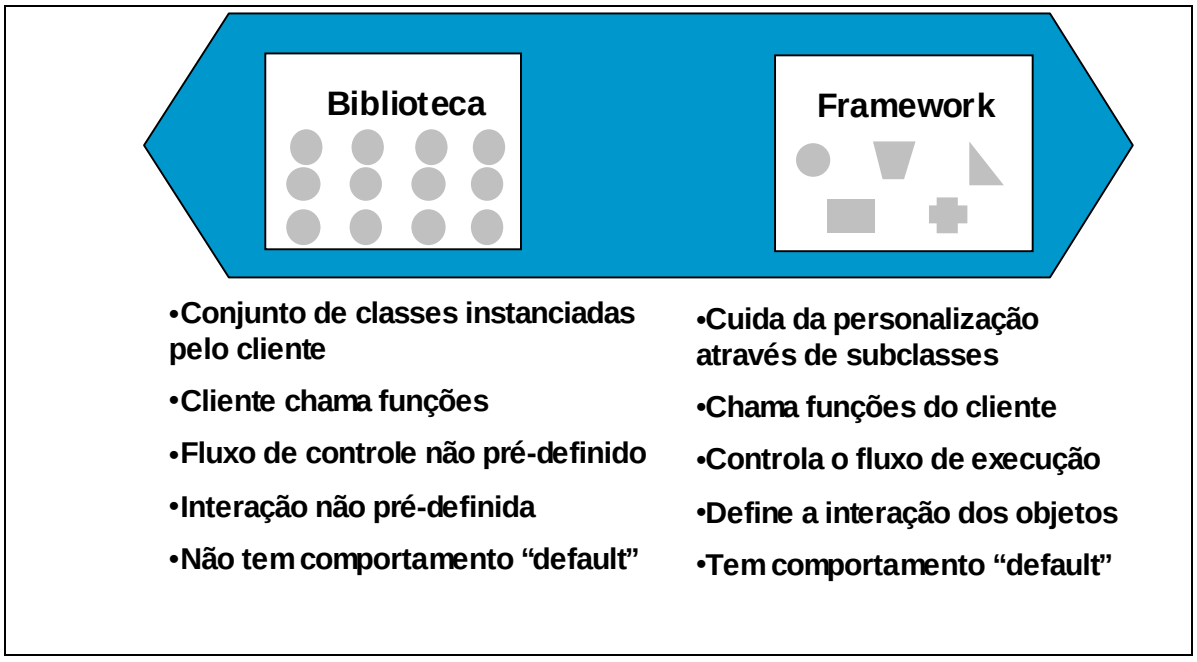

Figura 3 – Frameworks X Bibliotecas

2.2.3 – Frameworks Caixa Branca X Caixa Preta

O comportamento específico de um framework de aplicação é geralmente definido adicionando-se métodos às subclasses de uma ou mais de suas classes.

Existem dois tipos de frameworks: caixa branca e caixa preta [Johnson 88]. No framework caixa branca o reuso é provido por herança, ou seja, o usuário deve criar subclasses das classes abstratas contidas no framework para criar aplicações específicas. Para tal, ele deve entender detalhes de como o framework funciona para poder usá-lo. Já no framework caixa preta o reuso é por composição, ou seja, o usuário combina diversas classes concretas existentes no framework para obter a aplicação desejada. Assim, ele deve entender apenas a interface para poder usá-lo.

Um aspecto variável de um domínio de aplicação é chamado de "ponto de especialização" (em inglês, "*Hot spot*") [Buschman 96]. Diferentes aplicações dentro de um mesmo domínio são distinguidas por um ou mais *hot spots.* Eles representam as partes do framework de aplicação que são específicas de sistemas individuais. Os *hot-spots* são projetados para serem genéricos – podem ser adaptados às necessidades da aplicação.

"Pontos fixos" (em inglês, "*Frozen-spots"*) definem a arquitetura geral de um sistema de software – seus componentes básicos e os relacionamentos entre eles. Os *frozen-spots* permanecem fixos em todas as instanciações do framework de aplicação.

A Figura 4 ilustra um framework caixa branca com um único *hot-spot* R [Schmid 97]. Para utilização do framework é necessário fornecer a implementação referente a esse *hot-spot*, que no caso da Figura 4 é R3.

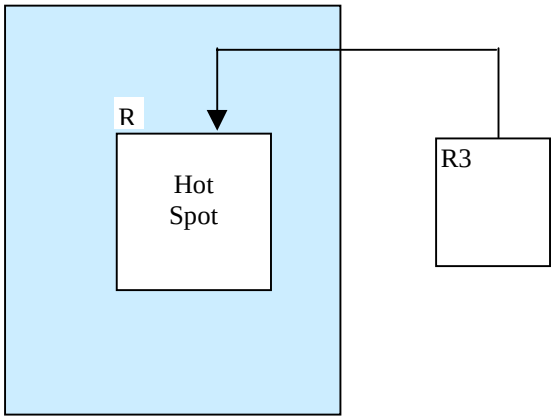

Figura 4 - Framework Caixa-branca ("White-box")

A Figura 5 ilustra um framework caixa-preta, também com um único hot-spot R. Nesse framework, existem três alternativas (R1, R2 e R3) para implementação da responsabilidade R. O usuário deve escolher uma delas para obter sua aplicação específica. Note que R1, R2 e R3 fazem parte do framework e são as únicas alternativas possíveis de implementação do *hot-spot*. Já no caso da Figura 4 R3 não fazia parte do framework, mas, em compensação, qualquer alternativa de implementação do *hot-spot* seria possível.

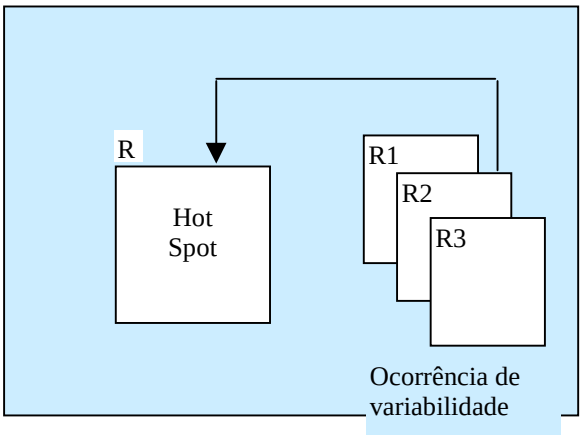

Figura 5 - Framework Caixa-preta (“Black-box”)

Resumindo, um framework caixa branca é mais fácil de projetar, pois não há necessidade de prever todas as alternativas de implementação possíveis. Já o framework caixa preta é mais difícil de projetar por haver a necessidade de fazer essa previsão. Por outro lado, o framework caixa preta é mais fácil de usar, pois basta escolher a implementação desejada, enquanto no caixa branca é necessário fazer essas implementações completas.

Frameworks caixa branca podem evoluir para se tornar cada vez mais caixa preta [Johnson 97]. Isso pode ser conseguido de forma gradativa, implementando-se várias alternativas que depois são aproveitadas na instanciação do framework. Ao mesmo tempo não se fecha totalmente o framework, permitindo ao usuário continuar usando-o como caixa branca. Após um certo tempo, estarão disponíveis diversas alternativas e então pode-se decidir tornar o framework caixa preta de fato. A medida em que o framework vai se tornando mais caixa preta, diminui o número de objetos criados, embora aumente a complexidade das suas interconexões. Além disso, o “scripting” fica mais importante, por

permitir a especificação das classes que farão parte da aplicação e sua interconexão. Pode-se também construir ferramentas de apoio que ajudem na especificação da aplicação. Essas ferramentas têm como objetivo facilitar a instanciação do framework, por meio do uso de componentes visuais que sejam mais fáceis de usar do que os comandos do “script”.

2.3 – Classificação de Frameworks

Os frameworks são classificados em três grupos [Fayad 97]: Frameworks de infra-estrutura do sistema, frameworks de integração de “middleware” e frameworks de aplicação empresarial.

Os frameworks de infra-estrutura do sistema simplificam o desenvolvimento da infra-estrutura de sistemas portáveis e eficientes, como por exemplo os sistemas operacionais, sistemas de comunicação, interfaces com o usuário e ferramentas de processamento de linguagem. Em geral são usados internamente em uma organização de software e não são vendidos a clientes diretamente.

Os frameworks de integração de “middleware” são usados, em geral, para integrar aplicações e componentes distribuídos. Eles são projetados para melhorar a habilidade de desenvolvedores em modularizar, reutilizar e estender sua infra-estrutura de software para funcionar “seamlessly” em um ambiente distribuído. Exemplos dessa classe de framework são o “Object Request Broker” (ORB), “middleware” orientado a mensagens e bases de dados transacionais.

Os frameworks de aplicação empresarial estão voltados a domínios de aplicação mais amplos e são a pedra fundamental para atividades de negócios das empresas, como por exemplo sistemas de telecomunicações, aviação, manufatura e engenharia financeira. Frameworks dessa classe são mais caros para desenvolver ou comprar, mas podem dar um retorno substancial do investimento, já que permitem o desenvolvimento de aplicações e produtos diretamente.

2.4 – Síntese de trabalhos afins

Diversos trabalhos na literatura têm dado enfoque à construção e uso de frameworks.

O framework de aplicação OOHDM-Java, cujo domínio é composto pelas aplicações hipermídia modeladas com o método OOHDM e disponibilizadas na WWW, considera uma separação entre os dados da aplicação, os objetos navegacionais e os objetos de interface, resultando em uma arquitetura mais flexível. Ele pode ser classificado como um framework caixa-branca, pelo fato de ser necessário que o programador entenda o seu projeto e a sua implementação, até um determinado grau de detalhe, para poder utilizá-lo. No entanto, o framework é composto internamente por alguns componentes considerados caixas-pretas, já que estes requerem apenas o conhecimento da sua interface externa para serem utilizados". O artigo traz detalhes de implementação e outros ligados ao OOHDM, que são mais difíceis de entender se o leitor não domina Java ou o método OOHDM. O interessante é que a implementação do framework foi bem elaborada e consistente com a fase de projeto. Usa várias tecnologias atuais e realmente enfatiza a importância de fazer a modelagem antes de implementar uma aplicação baseada nesse framework.

HotDraw [Johnson 92] é um framework gráfico bidimensional para editores de desenho estruturado, escrito no VisualWorks Smalltalk. Ele tem sido usado para criar diversos editores diferentes, desde ferramentas CASE até “HyperCard clone”. Pode-se facilmente criar novas figuras e ferramentas especiais de manipulação para seus desenhos. Ao contrário de muitos editores de desenhos, os desenhos do HotDraw podem ser animados. A Figura 6 mostra um exemplo do editor *default* editando algumas figuras.

2.5 – Exemplos

A maioria dos frameworks existentes é para domínios técnicos tais como interfaces com o usuário ou distribuição, como por exemplo os frameworks MacApp, específico para aplicações Macintosh, o Lisa Toolkit, o Smalltalk Model View Controller, o Interviews, etc. A maioria dos frameworks para aplicações específicas não é de domínio público. Exemplos são o ET++, ACE, Microsoft Foundation Classes (MFC) e DCOM, JavaSoft’s RMI e implementações do OMG’s CORBA.

O Model-View-Controller (MVC) [Krasner 88] foi o primeiro framework largamente utilizado. Ele foi inicialmente implementado em Smalltalk-80, mas atualmente conta com implementações em todas as versões existentes de Smalltalk, como o VisualWorks, VisualAge, Dolphin, Squeak, etc. O MVC é usado pelo Smalltalk como

interface com o usuário, tendo mostrado a adequação da orientação a objetos para implementação de interfaces gráficas com o usuário.

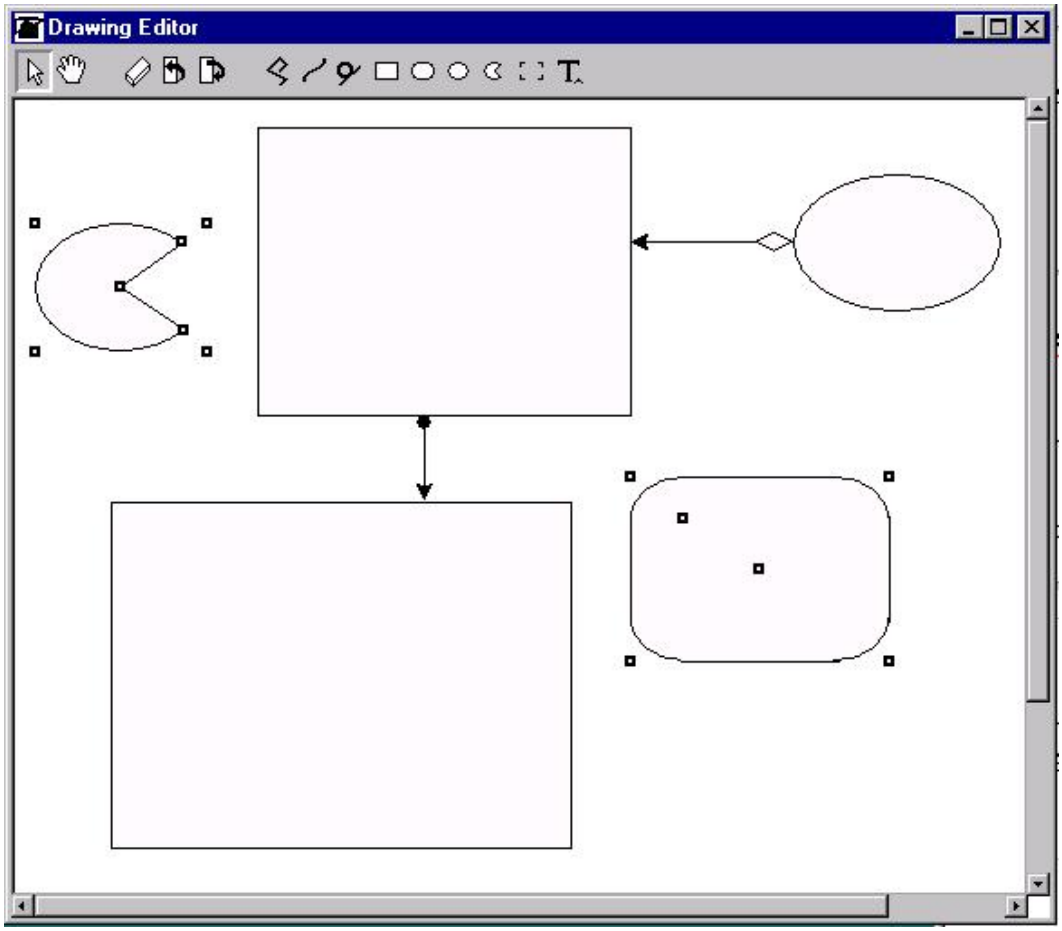

Figura 6 – Editor *default* do HotDraw

2.6 – Exercícios

1) Compare frameworks caixa branca e caixa preta.
2) Apresente um exemplo de um possível *hot-spot* em um framework qualquer.
3) Discuta o impacto da evolução de um framework quanto à manutenção de aplicações dele derivadas
4) Discuta

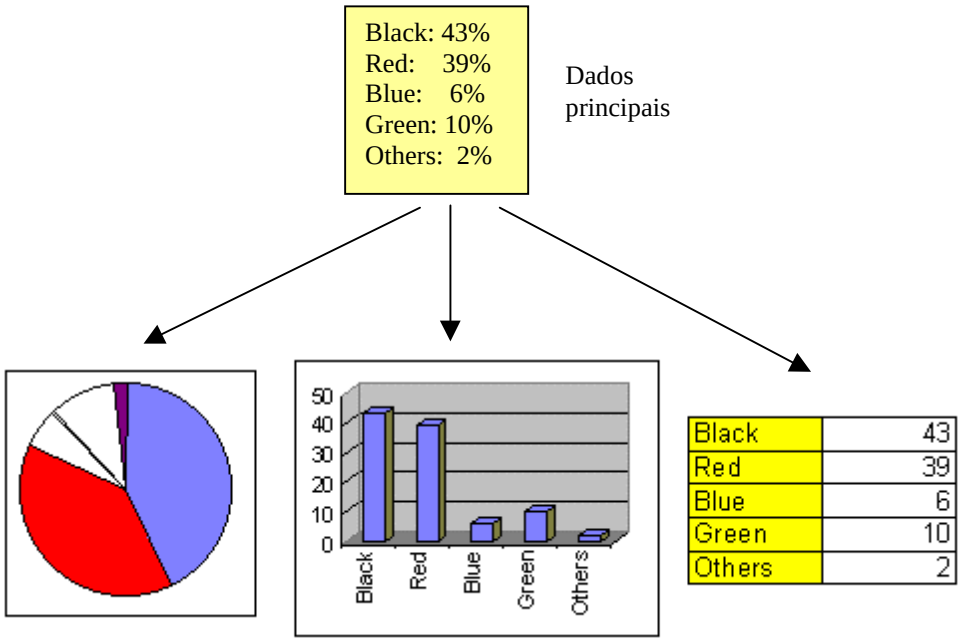

Figura 7 – Exemplo de modelo com várias visões

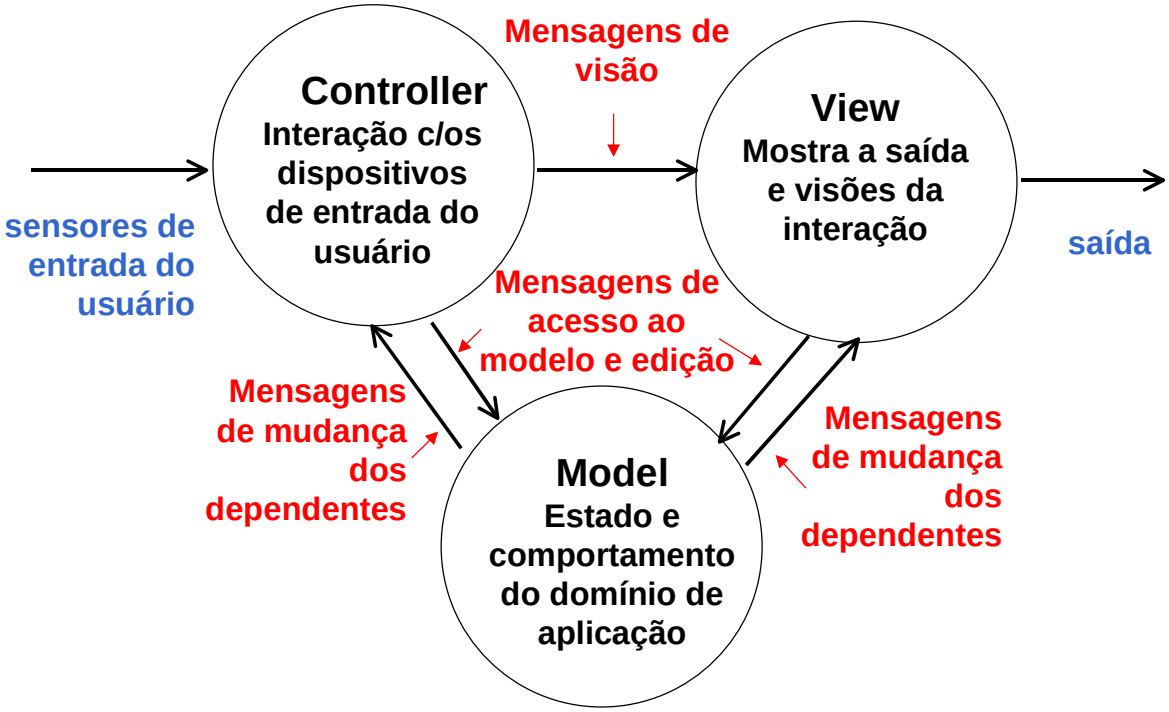

Figura 8 – Estrutura do Model-View-Controller

2.7 – Conclusões

Padrões documentam uma parte repetida de um projeto orientado a objetos, permitindo seu entendimento e aplicação em um contexto particular. Eles fornecem ao projeto um vocabulário comum aos desenvolvedores, facilitando a comunicação entre projetistas. Além disso, eles constituem uma base de experiência para construção de software reutilizável. Pode-se pensar em padrões como blocos construtivos a partir dos quais projetos mais complexos podem ser construídos

Padrões expõem conhecimento sobre a construção de software que foi ganho por especialistas em muitos anos. Portanto, todo trabalho sobre padrões deveria esforçar-se para colocar esse recurso precioso amplamente disponível [Buschmann 96]: “Todo desenvolvedor de software deveria ser capaz de usar padrões de forma efetiva. Quando isso for conseguido, seremos capazes de comemorar a inteligência humana que os padrões refletem, tanto em cada padrão individual quanto em todos os padrões na sua plenitude”.

Tanto padrões quanto frameworks são valiosos instrumentos de reuso. Além disso, os padrões são preciosos para a documentação dos frameworks.

Com frameworks bem projetados fica muito mais fácil fazer extensões, fatorar a funcionalidade comum, promover a interoperabilidade e melhorar a manutenção e confiabilidade de software [Taligent 93].

Frameworks fornecem a infra-estrutura e diretrizes arquiteturais de um sistema de software. A maior parte da funcionalidade já existe no framework, reduzindo codificação, teste e depuração. O exemplo de código fornecido guia os desenvolvedores no uso da tecnologia de objetos.

A Figura 9 ilustra os benefícios obtidos versus os custos de construção de frameworks [Taligent 93, 94].

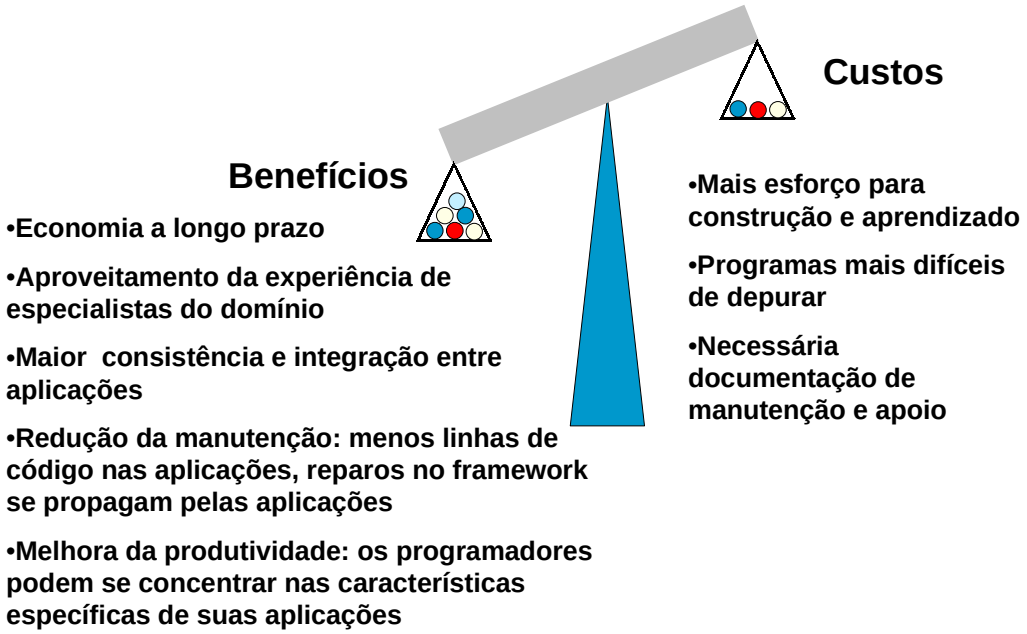

Figura 9 – Custo X Benefício de Frameworks

3 – Comparação entre as diversas formas de reuso

Fazendo uma comparação entre padrões e frameworks, Johnson [Johnson 97] diz que padrões são mais abstratos do que frameworks, porque um framework é um software executável, enquanto um padrão representa conhecimento e experiência sobre software (pode-se dizer que frameworks têm natureza física enquanto que padrões têm natureza lógica). Os padrões são menores do que frameworks. Em geral padrões são compostos por 2 ou 3 classes, enquanto os frameworks envolvem um número bem maior de classes, podendo englobar diversos padrões. Os padrões são menos especializados do que frameworks, já que os frameworks geralmente são desenvolvidos para um domínio de aplicação específico e os padrões (ou muitos deles) podem ser usados em diversos tipos de aplicação. Apesar de haver padrões mais especializados, eles não são capazes de ditar a arquitetura de uma aplicação, ao contrário dos frameworks.

Comparando frameworks com bibliotecas de programação, nas bibliotecas os componentes não estão inter-relacionados, enquanto em frameworks os componentes cooperam entre si. Em uma biblioteca de programação o desenvolvedor é responsável pela chamada de componentes e fluxo de controle do programa enquanto em um framework há a

inversão de papéis: o programa principal é reutilizado e o desenvolvedor decide quais componentes são chamados e deriva novos componentes. O framework determina a estrutura geral e o fluxo de controle do programa.

Comparando frameworks com geradores de aplicação, pelo menos duas semelhanças são encontradas:

- ambos envolvem análise de domínio para sua construção, na qual diferentes aplicações dentro do mesmo domínio são estudadas e posteriormente generalizadas
- e ambos resultam em código que integrará uma aplicação específica, embora com o uso de um framework novas classes possam ser criadas e nele embutidas para uso futuro.

Existem, porém, diversas diferenças entre eles:

- nos geradores de aplicação é necessário fornecer uma especificação do sistema em alto nível para obter o produto final, enquanto no framework são apenas informados os pontos variáveis, com possível criação de novas classes, para a instanciação do framework para uma aplicação específica.
- frameworks implicam o uso da orientação a objetos, já que sua definição envolve classes, enquanto geradores de aplicação são independentes do paradigma de desenvolvimento.
- nos frameworks o código é herdado enquanto em geradores de aplicação o código é gerado. Assim, grande parte do código de um framework é incorporado ao produto final, enquanto grande parte do código do gerador de aplicação existe apenas para fornecer mecanismos de geração do código do produto final.

Referências:

[Alexander 77] Christopher Alexander et. al., *A Pattern Language*, Oxford University Press, New York, 1977.

[Alexander 79] Christopher Alexander, *The Timeless Way of Building*, Oxford University Press, New York, 1979.

[Appeton 97] Appleton, Brad. *Patterns and Software: Essential Concepts and Terminology*, disponível na WWW na URL: http://www.enteract.com/~bradappdocpatterns-intro.html.

[Beck 87] Beck, Kent; Cunningham, Ward. *Using Pattern Languages for Object-Oriented Programs*, Technical Report nº CR-87-43, 1987, disponível na WWW na URL: http://c2.com/doc/oopsla87.html

[Bosch 97] Bosch, Jan. *Design Patterns as Language Constructs*, disponível na WWW na URL: http://st-www.cs.uiuc.edu/users/patterns/papers, 1997.

[Boyd 98] Boyd, L. *Business Patterns of Association Objects*. In: "Martin, R.C.; Riehle, D.; Buschmann, F. *Pattern Languages of Program Design 3*. Reading, MA, Addison-Wesley, 1998", p. 395-408.

[Braga 98a] Braga, Rosana T. V.; Germano, Fernão S.R.; Masiero, Paulo C. *A Family of Patterns for Business Resource Management*. In Proceedings of the 5th Annual Conference on Pattern Languages of Programs (PLOP'98), Monticello, Illinois, Technical Report #WUCS-98-25, Washington University in St. Louis, Missouri, EUA, agosto de 1998, disponível na WWW na URL: http://jerry.cs.uiuc.edu/plop/plopd4-submissions/ plopd4-submissions.html.

[Braga 98b] Braga, Rosana T.V.; Germano, Fernão S.R.; Masiero, Paulo C. *Experimentos para implementação do padrão Type-Object em linguagem Delphi*. São Carlos, USP, 1998. (Documento de Trabalho, ICMSC-USP, São Carlos, SP)

[Budinsky 96] Budinsky, F.J, et al. *Automatic code generation from design patterns.* IBM Systems Journal, V. 35, nº 2, p. 151-171, 1996.

[Buschmann 96] Buschmann, F. et at. *A System of Patterns*, Wiley, 1996.

[Chikofsky 90] Chikofsky, E.J.; Cross, J.H. *Reverse Engineering and Design Recovery: A Taxonomy.* IEEE Trans. Software, pp. 13-17, Janeiro 1990.

[CMG 98] Component Management Group (CMG). *Anti-patterns*, disponível na WWW na URL: http://160.79.202.73/Resource/AntiPatterns/index.html.

[Coad 91] Coad, Peter/ Yourdon, Edward. *Object-Oriented Analysis*, 2ª edição, Yourdon Press, 1991.

[Coad 92] Coad, Peter. *Object-Oriented Patterns.* Communications of the ACM, V. 35, nº9, p. 152-159, setembro 1992.

[Coad 95] Coad, P.; North, D.; Mayfield, M. *Object Models: Strategies, Patterns and Applications*, Yourdon Press, 1995.

[Cook 94] Cook, Steve; Daniel, John. *Designing Object Systems.* Prentice Hall, 1994.

[Coleman 94] Coleman, D. et al. *Object Oriented Development - the Fusion Method*. Prentice Hall, 1994.

[CMG 98] Component Management Group (CMG). *Anti-patterns*, disponível na WWW na URL: http://160.79.202.73/Resource/AntiPatterns/index.html.

[Coplien 92] Coplien, J.O. *Advanced C++ Programming Styles and Idioms*. Reading-MA, Addison-Wesley, 1992.

[Coplien 95] Coplien, J.; Schmidt, D. (eds.) *Pattern Languages of Program Design*, Reading-MA, Addison-Wesley, 1995.

[Coplien 98] James O. Coplien. Software Design Patterns: Common Questions and Answers. In Linda Rising, editor, The Patterns Handbook: Techniques, Strategies, and Applications, p. 311-320. Cambridge University Press, New York, January 1998.

[Cunningham 95] Cunningham,Ward. *The CHECKS Pattern Language of Information Integrity*. In "Coplien, J.; Schmidt, D. (eds.) *Pattern Languages of Program Design*, Reading-MA, Addison-Wesley, 1995", p. 145-155.

[Cunningham 96] Cunningham,Ward. *EPISODES: A Pattern Language of Competitive Development.* In: Vlissides, J.; Coplien, J.; Kerth, N (eds.). *Pattern Languages of Program Design 2.* Reading-MA; Addison-Wesley, 1996", p. 371-388.

[Eriksson 98] Eriksson, H.; Penker, M. *UML Toolkit.* New York, Wiley Computer Publishing, 1998.

[Fayad 97] Fayad, M.E.; Schmidt, D.C. (eds) *Object-Oriented Application Frameworks*. Communications of the ACM, V. 40, nº 10, p. 32-38, 1997.

[Fowler 97] Fowler, M. *Analysis Patterns.* Menlo-Park-CA, Addison-Wesley, 1997.

[Gall 96] Gall, Harald C., Klösh, René R.; Mittermeir, Roland T. *Application Patterns in Re-Engineering: Identifying and Using Reusable Concepts.* Proceedings of the 6th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, Julho 1996, p. 1099-1106.

[Gamma 93] Gamma, E.; Helm, R.; Johnson,R.; Vlissides, J. *Design Patterns - Abstraction and Reuse of Object-Oriented Design*. LNCS, nº 707, p. 406-431, julho de 1993.

[Gamma 95] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns - Elements of Reusable Object-Oriented Software*. Reading-MA, Addison-Wesley, 1995.

[Graham 94] Graham, Ian. *Object Oriented Methods*, 2ª edição, Addison-Wesley, 1994.

[Johnson 88] Johnson, Ralph E.; Foote B. *Designing Reusable Classes*. Journal of Object Oriented Programming – JOOP, 1(2):22-35, Junho/Julho 1988.

[Johnson 91] Johnson, Ralph E.; Russo, Vincent. *Reusing Object-Oriented Designs*. Relatório Técnico da Universidade de Illinois, UIUCDCS 91-1696, 1991.

[Johnson 97] Johnson, Ralph E. *CS497 Lectures – Lecture 12, 13, 14 e 17*, disponível na WWW na URL: http://st-www.cs.uiuc.edu/users/johnson/cs497 /notes98/online-course.html

[Johnson 97a] Johnson, Ralph. E. *Frameworks = (Components + Patterns).* Communications of the ACM, V. 40, nº 10, p. 39-42, 1997.

[Johnson 98] Johnson, Ralph E.; Woolf, B. *Type Object.* In: "Martin, R.C.; Riehle, D.; Buschmann, F. *Pattern Languages of Program Design 3.* Reading-MA, Addison-Wesley, 1998", p. 47-65.

[Kerth 97] Kerth, Norman L.; Cunningham, Ward. *Using Patterns to Improve Our Architectural Vision*, IEEE Software, pp. 53-59, janeiro, 1997.

[Kramer 96] Krämer, Christian; Prechelt, Lutz. *Design Recovery by Automated Search for Structural Design Patterns in Object-Oriented Software.* Proceeding of the 3rd Working Conference on Reverse Engineering (WCRE), Monterey-CA, EUA, 1996, p. 208-215.

[Krasner 88] Krasner, E. K.; Pope, S.T. *A Cookbook for using the Model-View-Controller User Interface Paradigm in Smalltalk-80*. Journal of Object Oriented Programming, p. 26-49, Agosto-Setembro 1988.

[Martin 98] Martin, R.C.; Riehle, D.; Buschmann, F. (eds.) *Pattern Languages of Program Design 3*, Reading-MA, Addison-Wesley, 1998.

[Masiero 93] Masiero, P.C.; Meira, C.A. A. *Development and Instantiation of a Generic Application Generator*. Journal of Systems and Software, Vol. 23, pp. 27-37, 1993.

[Masiero 98] Masiero, P.C.; Germano, F.S; Maldonado, J.C. *Object and System Life Cycles Revisited: Their Use in Object-Oriented Analysis and Design Methods.* Proceedings of the 3rd CaiSE/IFIP8.1 (International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design), Pisa, Italy, pp. O1-O12, Junho 1998.

[Meira 91] MEIRA, C.A.A. *Sobre Geradores de Aplicação*, Dissertação de Mestrado apresentada ao Instituto de Ciências Matemáticas de São Carlos, USP, setembro de 1991.

[Meszaros 95] Meszaros, Gerard. *A Pattern Language for Improving the Capacity of Reactive Systems.* In: "Coplien, J.; Schmidt, D. (eds.) *Pattern Languages of Program Design*, Reading-MA, Addison-Wesley, 1995", p. 575-591.

[Pree 95] Pree, Wolfgang. *Design Patterns for Object-Oriented Software Development*. Reading-MA, Addison-Wesley, 1995.

[Pressman 95] Pressman, Roger S. *Engenharia de Software*, Makron Books, 1995.

[Riehle 95] Riehle, Dirk; Züllighoven, Heinz. *A Pattern Language for Tool Construction and Integration Based on the Tools and Materials Metaphor.* In: "Coplien, J.; Schmidt, D. (eds.) *Pattern Languages of Program Design*, Reading-MA, Addison-Wesley, 1995", p. 9-42.

[Roberts 98] Roberts, Don; Johnson, Ralph E. *Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks,* In "Martin, R. C.; Riehle, D. and Buschmann, F. Pattern Languages of Program Design 3", Addison-Wesley, pp. 471-486, 1998, disponível na WWW na URL: http://st-www.cs.uiuc.edu/users/droberts/evolve.html

[Schmidt 96] Schmidt, Douglas C.; Fayad, Mohamed; Johnson, Ralph E. (guest editors). *Software Patterns*. Communications of the ACM, V. 39, nº10, pp. 36-39, outubro 1996.

[Schmid 97] Schmid, H. A. *Systematic Framework Design by generalization.* Communications of the ACM, V. 40, nº 10, p. 48-51, 1997.

[Taligent 93] Taligent Inc. *Leveraging Object-Oriented Frameworks*. A Taligent White Paper, 1993, disponível na WWW na URL:

http://www.ibm.com/java/education/ooleveraging.

[Taligent 94] Taligent Inc. *Building Object-Oriented Frameworks*. A Taligent White Paper, 1994, disponível na WWW na URL: http://www.ibm.com/java/education/oobuilding/index.html.

[Turine 98] Turine, Marcelo A. S. *Fundamentos, Conceitos e Aplicações do Paradigma de Orientação a Objetos.* Apresentação didática, agosto de 1998, disponível na WWW na URL: http://nt-labes.icmsc.sc.usp.br/cursos/sce220/02_98/aulas.html

[Vlissides 95] Vlissides, John. *Reverse Architecture*, Position Paper for Software Architectures Seminar, Schloss Daghstuhl, Germany, agosto 1995, disponível na WWW na URL: http://st-www.cs.uiuc.edu/users/patterns/papers/,.

[Vlissides 96] Vlissides, J.; Coplien, J.; Kerth, N (eds.). *Pattern Languages of Program Design 2*. Reading-MA; Addison-Wesley, 1996.

[Yoder 98] Yoder, J.W.; Johnson, R.E.; Wilson, Q.D. *Connecting Business Objects to Relational Databases*. Proceedings of the 5th Conference on the Pattern Languages of Programs, Monticello-IL, EUA, agosto de 1998. (Relatório Técnico nº WUCS-98-25 da Universidade de Washington), disponível na WWW na URL: http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/