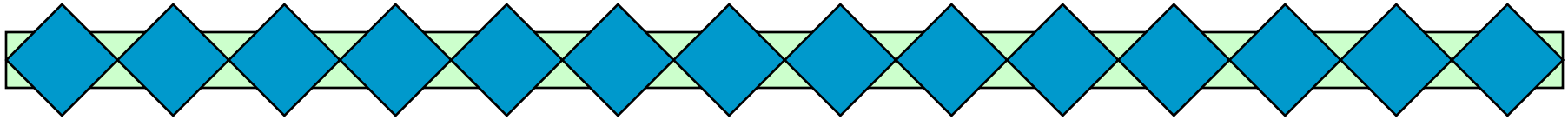
Reúso de Software



Profa. Dra. Rosana Teresinha Vaccare Braga

15 de abril de 2016

Laboratório de Engenharia de Software

Instituto de Ciências Matemáticas e de Computação

Organização

- Introdução
- Alguns Tipos de Reúso de Software
  - Componentes
  - Padrões
  - Linguagens de Padrões
  - Frameworks
  - Aspectos
  - Serviços
  - Linha de Produtos de Software
  - Model Driven Software Engineering
- Conclusões

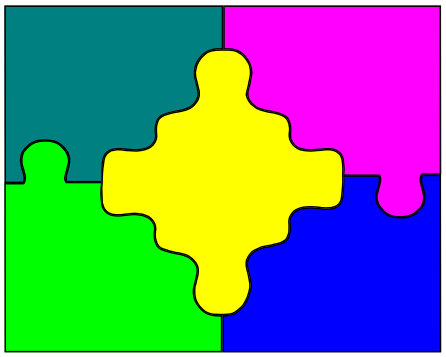
Introdução

- Qual é o futuro da Engenharia de Software?
  - Software que funciona!!!
  - Por que isso não ocorre?
    - Complexidade
    - Questões de negócios: cronograma, funcionalidade e qualidade: escolha dois!
    - (adaptado de: good, fast and cheap: pick any two)

Introdução

- Como mudar isso?
  - Melhores ferramentas de teste e depuração?
  - Atacar a complexidade?

Escrever menos código!!



Introdução

- Como escrever menos código???

Reúso de Software!!!

Introdução

- Reúso de Software:
  - Anos 70: módulos e sub-rotinas
  - Anos 80: classes e geradores de aplicação
  - Anos 90: análise de domínio, componentes, padrões, frameworks
  - Anos 00: aspectos, serviços, MDA
  - Anos 10: SaS, SoS, IoT...

Benefícios da reutilização

- Melhores índices de produtividade
- Produtos de melhor qualidade, mais confiáveis, consistentes e padronizados
- Redução dos custos e tempo envolvidos no desenvolvimento de software
- Maior flexibilidade na estrutura do software produzido, facilitando sua manutenção e evolução

Dificuldades

- Identificação, recuperação e modificação de artefatos reutilizáveis
- Qualidade de artefatos reutilizáveis
- Composição de aplicações a partir de componentes
- Aumento nos custos de manutenção

Dificuldades

- Falta de ferramentas de apoio
- Barreiras psicológicas: síndrome do *não foi inventado aqui*
- Barreiras legais e econômicas

Necessidade da criação de incentivos à reutilização

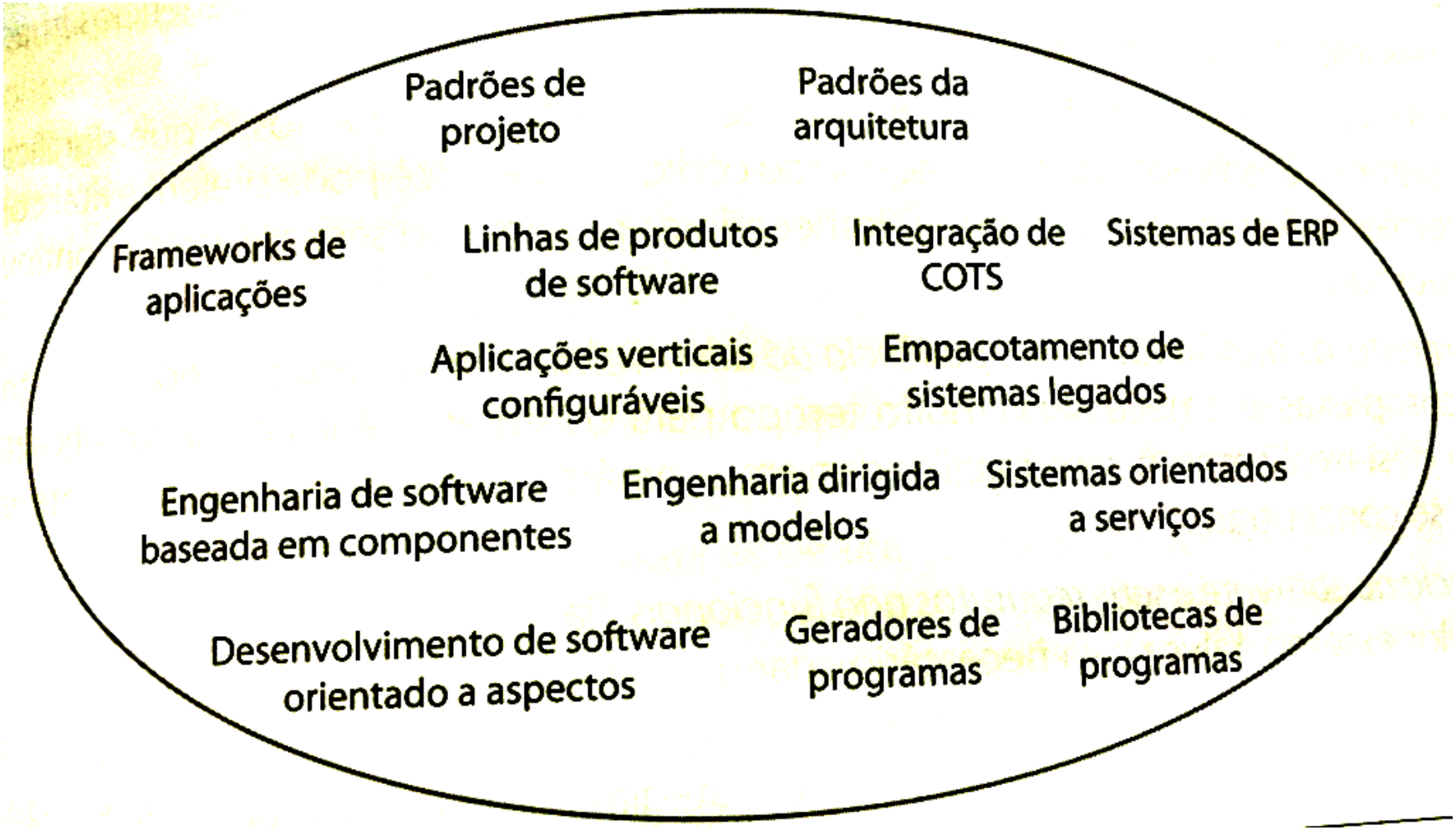
Reúso de Software

- Pode ocorrer em diversos níveis:
  - Código (componentes ou unidades de software)
  - Projeto Detalhado
  - Arquitetura
  - Modelos de Análise
  - “Idéias” ou soluções para problemas recorrentes
- Quanto mais alto o nível, geralmente mais altos são os ganhos.

Tipos de Reúso de Software

- Oportunístico (ad hoc) ou Planejado?
  - Exemplo de reúso oportunístico: cortar e colar.
  - Exemplo de reuso planejado: reúso de um framework desenvolvido previamente.
- A tecnologia determina o tipo de reúso? (Pense em componentes, por exemplo?)

Panorama de Reúso



Abordagens para apoiar reúso

Abordagem	Descrição
Padrões de arquitetura	Padrões de arquitetura de software que oferecem suporte a tipos comuns de sistemas de aplicação são usados como base de aplicações. São descritos nos capítulos 6, 13 e 20.
Padrões de projeto	Abstrações genéricas que ocorrem em todas as aplicações são representadas como padrões de projeto, mostrando os objetos abstratos e concretos e as interações. São descritos no Capítulo 7.
Desenvolvimento baseado em componentes	Sistemas são desenvolvidos através da integração de componentes (coleções de objetos) que atendem aos padrões de modelos e componentes. São descritos no Capítulo 17.
Framework de aplicações	Coleções de classes abstratas e concretas são adaptadas e estendidas para criar sistemas de aplicação.
Empacotamento de sistemas legados	Sistemas legados (veja o Capítulo 9) são 'empacotados' pela definição de um conjunto de interfaces e acesso a esses sistemas legados por meio dessas interfaces.
Sistemas orientados a serviços	Sistemas são desenvolvidos pela ligação de serviços compartilhados, que podem ser fornecidos externamente. São descritos no Capítulo 19.
Linhas de produtos de software	Um tipo de aplicação é generalizado em torno de uma arquitetura comum para que esta possa ser adaptada para diferentes clientes.

Abordagens para apoiar reúso

Reúso de produto COTS	Sistemas são desenvolvidos pela configuração e integração de sistemas de aplicação existentes.
Sistemas de ERP	Sistemas de grande porte que sintetizam a funcionalidade e as regras de negócios genéricos são configurados para uma organização.
Aplicações verticais configuráveis	Sistemas genéricos são projetados para poder ser configurados para as necessidades dos clientes de sistemas específicos.
Bibliotecas de programas	Bibliotecas de classe e funções que implementam abstrações comumente usadas são disponibilizadas para reúso.
Engenharia dirigida a modelos	O software é representado como modelos de domínio e modelos de implementação independentes. O código é gerado a partir desses modelos. São descritos no Capítulo 5.
Geradores de programas	Um sistema gerador incorpora o conhecimento de um tipo de aplicação, e é usado para gerar sistemas nesse domínio a partir de um modelo de sistema fornecido pelo usuário.
Desenvolvimento de software orientado a aspectos	Quando o programa é compilado, os componentes compartilhados são integrados em uma aplicação em diferentes locais. São descritos no Capítulo 21.

Organização

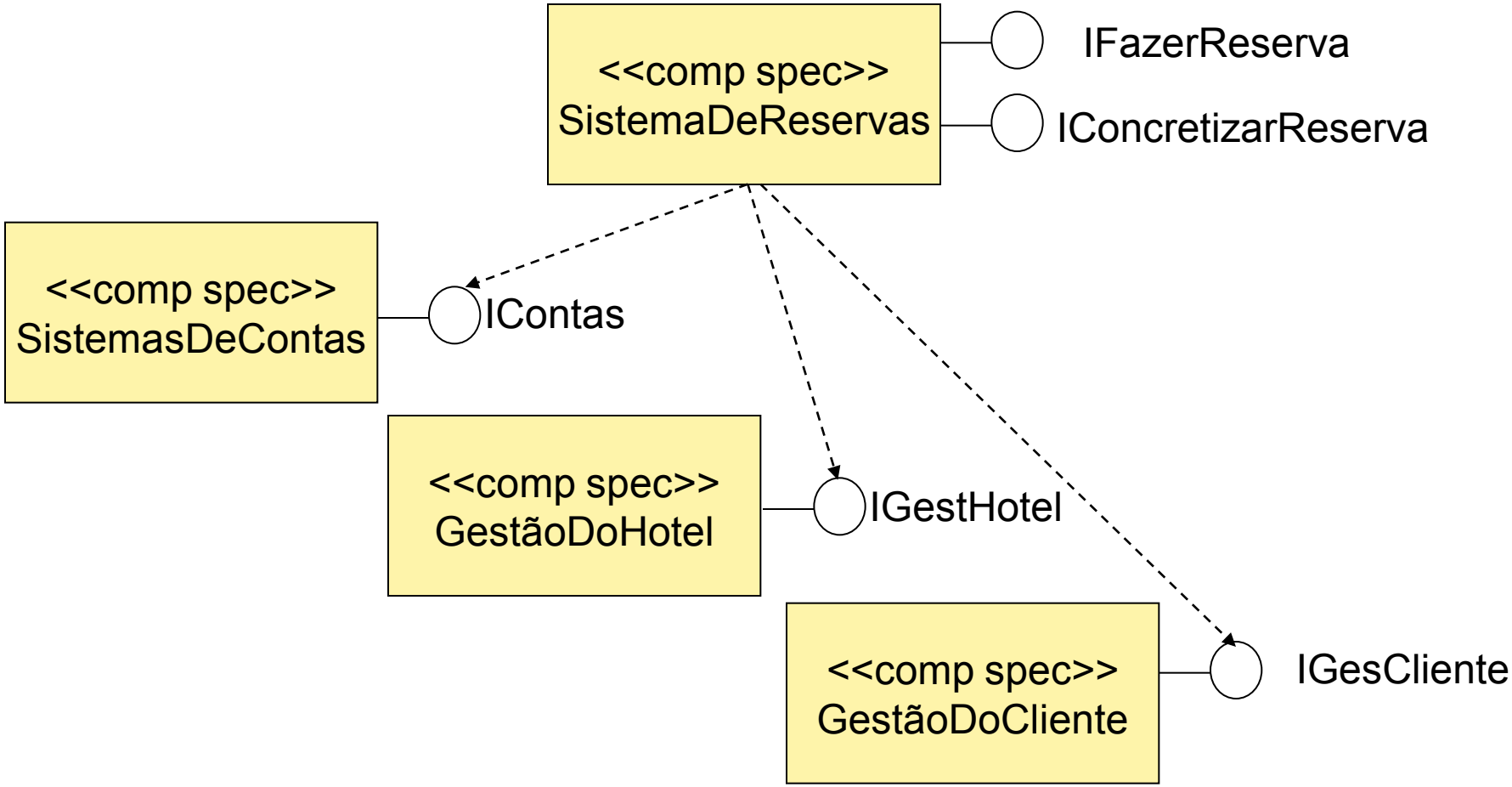
- Introdução
- Alguns Tipos de Reuso de Software
  - Componentes
  - Padrões
  - Linguagens de Padrões
  - Frameworks
  - Aspectos
  - Serviços
  - Linha de Produtos de Software
  - Model Driven Software Engineering
- Conclusões

O que é um Componente?

- **Componentes reutilizáveis são artefatos auto-contidos, claramente identificáveis, que descrevem e realizam uma função específica e têm interfaces claras, em conformidade com um dado modelo de arquitetura de software, documentação apropriada e um grau de reutilização definido**

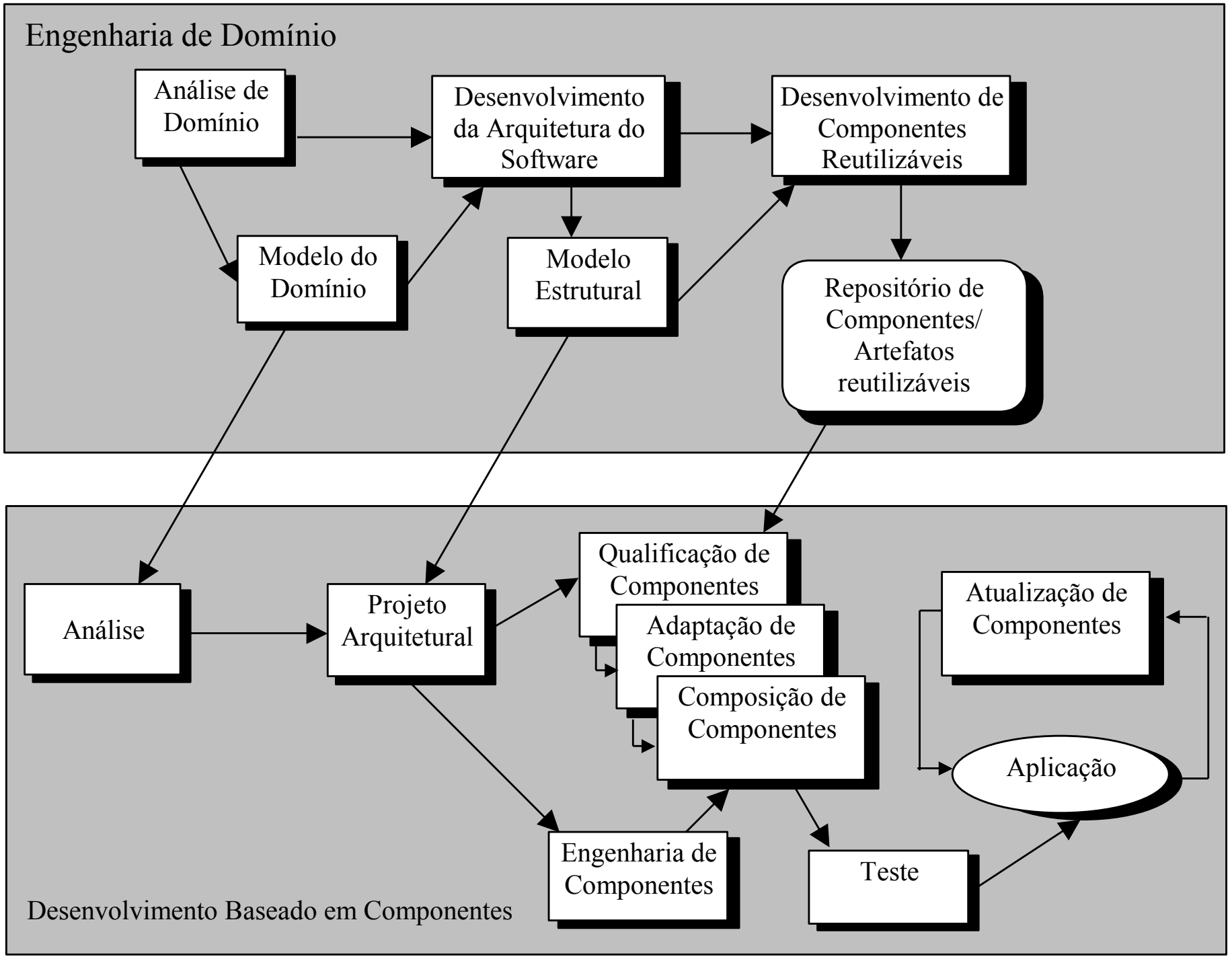
Definição elaborada por Werner e Braga (2000), com base na definição de Sametinger (1997), com algumas considerações em relação a arquitetura de software feitas por Krutchen (1998)

Notação UML para componentes (extensão de Cheesman & Daniels)



Desenvolvimento Baseado em Componentes (DBC)

- Metodologias para o DBC
  - UML Components
    - J. J. Cheesman and J. Daniels
  - Catalysis (http://www.iconcomp.com/catalysis)
    - D. D'Souza and A. A. Wills
  - KobrA
    - C. Atkinson et al.


Engenharia de Domínio
Análise de Domínio
Desenvolvimento da Arquitetura do Software
Desenvolvimento de Componentes Reutilizáveis
Modelo do Domínio
Modelo Estrutural
Repositório de Componentes/ Artefatos reutilizáveis
Análise
Projeto Arquitetural
Qualificação de Componentes
Adaptação de Componentes
Composição de Componentes
Atualização de Componentes
Aplicação
Engenharia de Componentes
Teste
Desenvolvimento Baseado em Componentes

Organização

- Introdução
- Alguns Tipos de Reuso de Software
  - Componentes
  - Padrões
  - Linguagens de Padrões
  - Frameworks
  - Aspectos
- Conclusões

Padrões de Software

Por que Padrões ?

- Desenvolvedores acumulam soluções para os problemas que resolvem com freqüência
- Essas soluções são difíceis de serem elaboradas e podem aumentar a produtividade, qualidade e uniformidade do software
- Como documentar essas soluções de forma que outros desenvolvedores, menos experientes, possam utilizá-las?

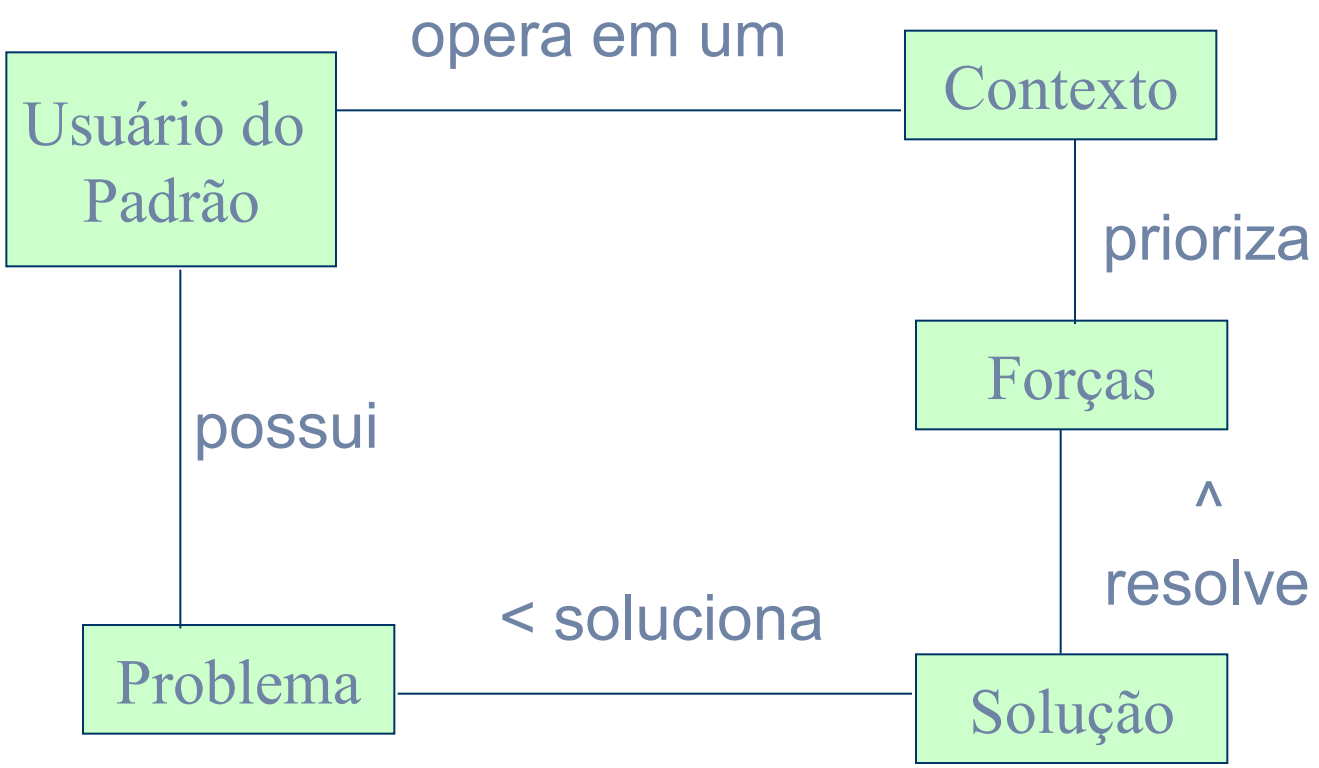
Vantagem de Padrões

- Aumento de produtividade
- Uniformidade na estrutura do software
- Aplicação imediata por outros desenvolvedores
- Redução da complexidade: blocos construtivos

Padrões de Software

- Definição: Padrões descrevem soluções para problemas que ocorrem com freqüência no desenvolvimento de software (Gamma 95)
- Existem diversos **formatos** para os padrões, mas todos eles possuem os seguintes elementos:
  - Problema
  - Contexto
  - Solução
  - Padrões relacionados

Relacionamento entre Elementos do Padrão (Meszaros & Doble)



Pattern 2: QUANTIFY THE RESOURCE

Context

You have identified a resource that your application deals with and its relevant qualities. An important issue to be considered now is the form of resource quantification. There are certain applications in which it is important to trace specific instances of a resource, because they are transacted individually. For example, a book in a library can have several copies, each lent to a different reader. Some applications deal with a certain quantity of the resource or with resource lots. In these applications, it is not necessary to know what particular instance of the resource was actually transacted. For example, a certain weight of steel is sold. In other applications, the resource is dealt with as a whole, as for example a car that goes to maintenance or a doctor that examines a patient.

Problem

How does the application quantify the business resource?

Forces

Knowing exactly what is the form of quantification adopted by the application is important during analysis. A wrong decision at this point may compromise future evolution.

. . .

Structure

There are four slightly different solutions for this problem, depending on the form of quantification. Figures 4 through 7 show the four QUANTIFY THE RESOURCE sub-patterns. When it is important to distinguish among resource instances, use INSTANTIABLE RESOURCE sub-pattern (Figure 4). When the resource is managed in a certain quantity, use the MEASURABLE RESOURCE sub-pattern (Figure 5).

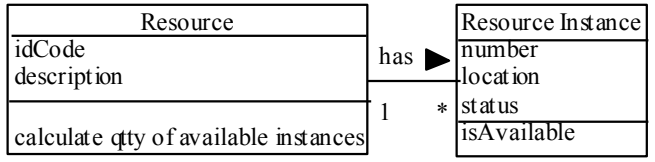


Figure 4: **INSTANTIABLE RESOURCE sub-pattern**

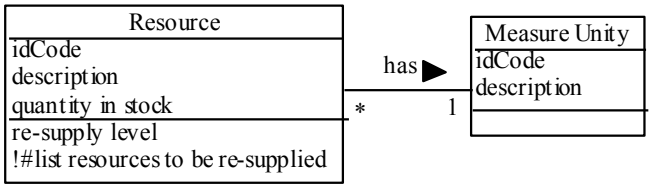


Figure 5: **MEASURABLE RESOURCE sub-pattern**

Participants

. . .

Example

. . .

Following patterns

After you Quantify the Resource, examine your application to verify whether it is important to know about the resources storage. If so, try to apply STORE THE RESOURCE (3). If not, proceed examining your application to verify which kind of resource transactions are done. If the application concerns resource location or rental, you should apply RENT THE RESOURCE (4). If the application concerns resource trading, i.e., resource purchase or sale, you should apply TRADE THE RESOURCE (6). If the application deals with resource repair, you should apply MAINTAIN THE RESOURCE (9).

Tipos de Padrão

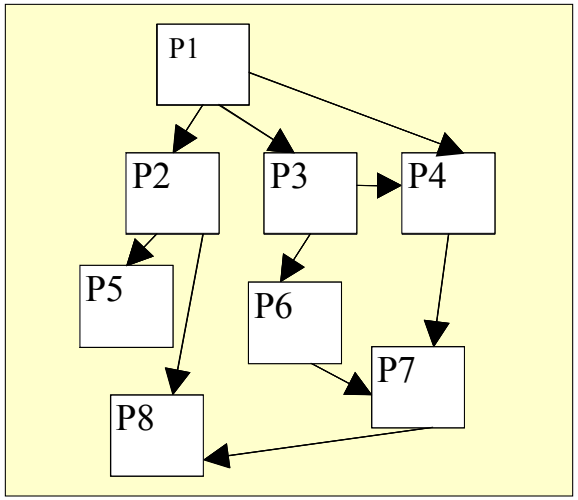
- Existem padrões em diversos níveis de abstração:
  - Padrões de Análise
  - Padrões de Projeto
  - Padrões de Interface
  - Padrões Arquiteturais
  - Padrões de Programação
  - Padrões de Processo
  - Padrões de Padrão

Padrões conhecidos

- Análise: Fowler 97, Coad 92/97
- Projeto: Gamma 95 (GoF)
- Arquiteturais: Buschman, 98 (POSA)
- Programação: Coplien 92

Linguagem de Padrões

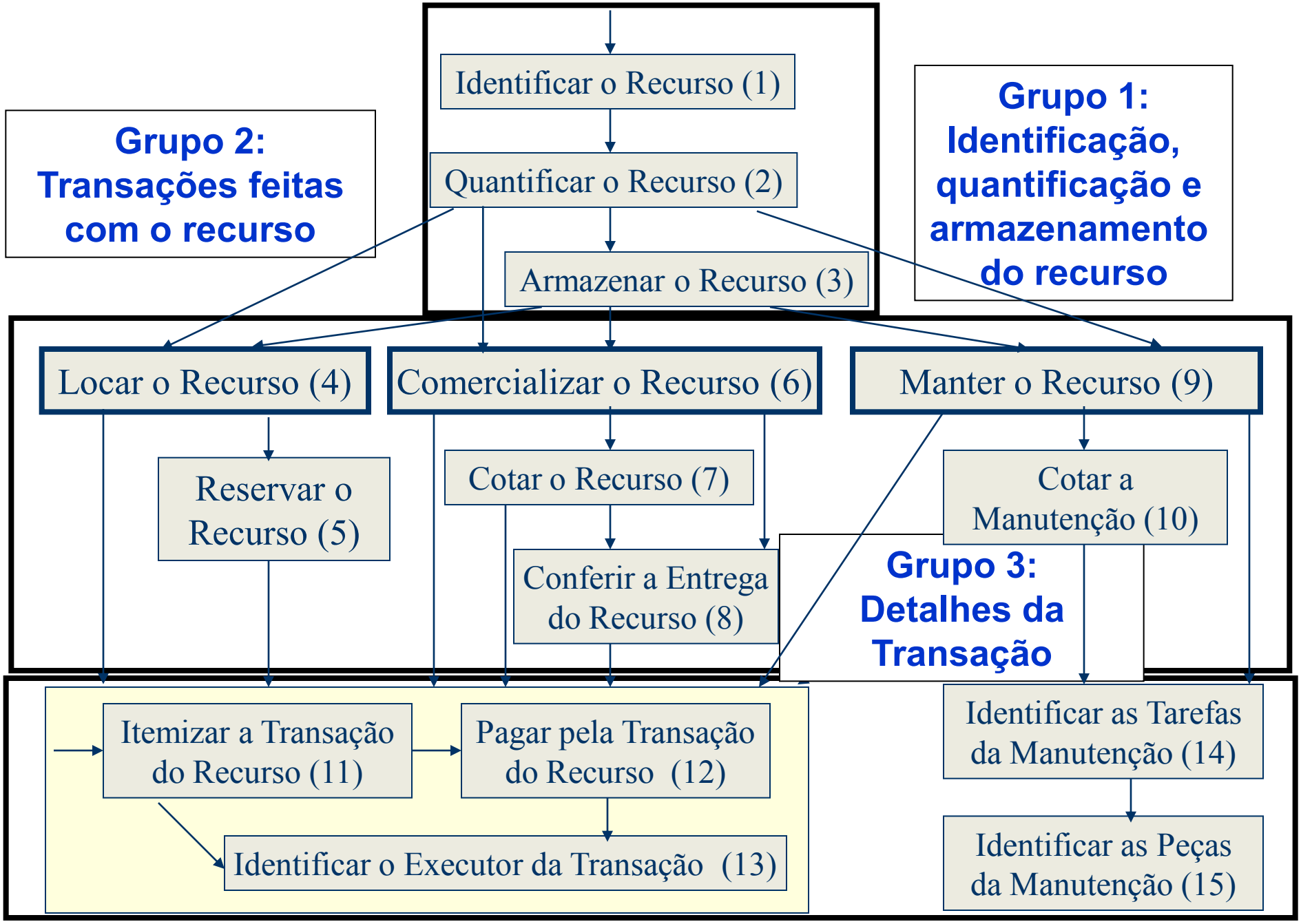
- Coleção estruturada de padrões que se apóiam uns nos outros para transformar requisitos e restrições numa arquitetura (Coplien, 98).



Sugestão de estrutura para uma Linguagens de Padrões

- Intenção da Linguagem
  - Breve descrição do que a linguagem almeja
  - Semelhante a um resumo da linguagem
- Mapa da Linguagem
  - Um diagrama que mostra um exemplo de como os padrões apóiam-se uns nos outros e de como se relacionam
- Descrição da Linguagem
  - Uma descrição de como a linguagem de padrões é morfologicamente e funcionalmente completa
- Os Padrões que a compõem

GRN: Uma linguagem de Padrões para Gestão de Recursos de Negócios



Frameworks

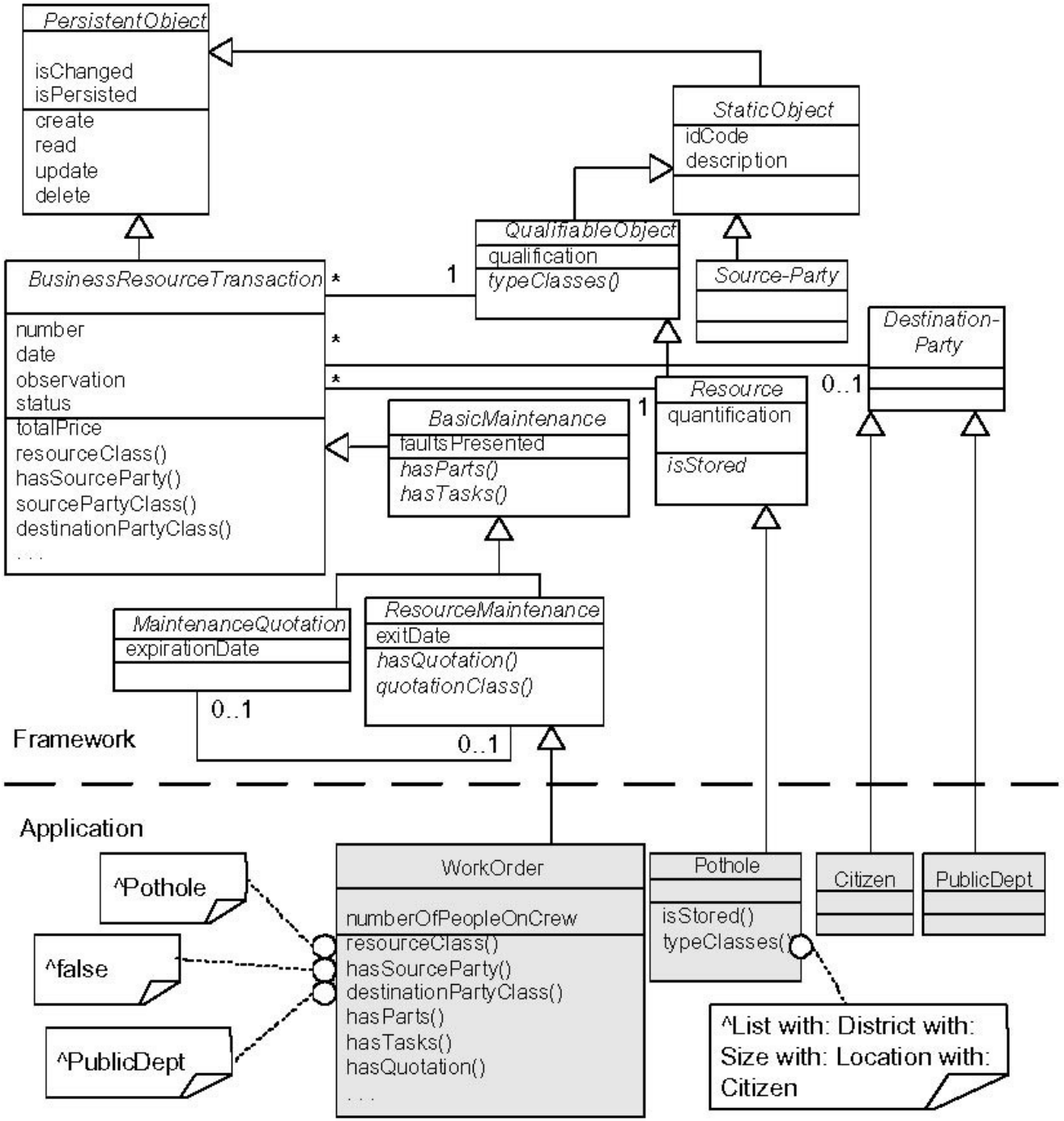
- Definições:

  – Aplicação semi-completa reutilizável que, quando especializada, produz aplicações personalizadas (Johnson & Foote, 1988) ou

  – Coleção de classes abstratas e concretas e a interface entre elas, representando o projeto de um sub-sistema (Pree, 1995)

Frameworks

- Hot-Spots
  - **Representam as partes do framework de aplicação que são específicas de sistemas individuais**
  - **São projetados para serem genéricos - podem ser adaptados às necessidades da aplicação**
- Frozen-Spots
  - **Definem a arquitetura geral de um sistema de software - seus componentes básicos e os relacionamentos entre eles**
  - **Permanecem fixos em todas as instanciações do framework de aplicação**


PersistentObject
isChanged
isPersisted
create
read
update
delete
StaticObject
idCode
description
QualifiableObject
qualification
typeClasses()
Source-Party
Destination-Party
BusinessResourceTransaction
number
date
observation
status
totalPrice
resourceClass()
hasSourceParty()
sourcePartyClass()
destinationPartyClass()
. . .
*
1
*
*
1
0..1
Resource
quantification
isStored
BasicMaintenance
faultsPresented
hasParts()
hasTasks()
MaintenanceQuotation
expirationDate
ResourceMaintenance
exitDate
hasQuotation()
quotationClass()
0..1
0..1
Framework
Application
^Pothole
^false
^PublicDept
WorkOrder
numberOfPeopleOnCrew
resourceClass()
hasSourceParty()
destinationPartyClass()
hasParts()
hasTasks()
hasQuotation()
. . .
Pothole
isStored()
typeClasses()
Citizen
PublicDept
^List with: District with: Size with: Location with: Citizen

Tipos de Frameworks

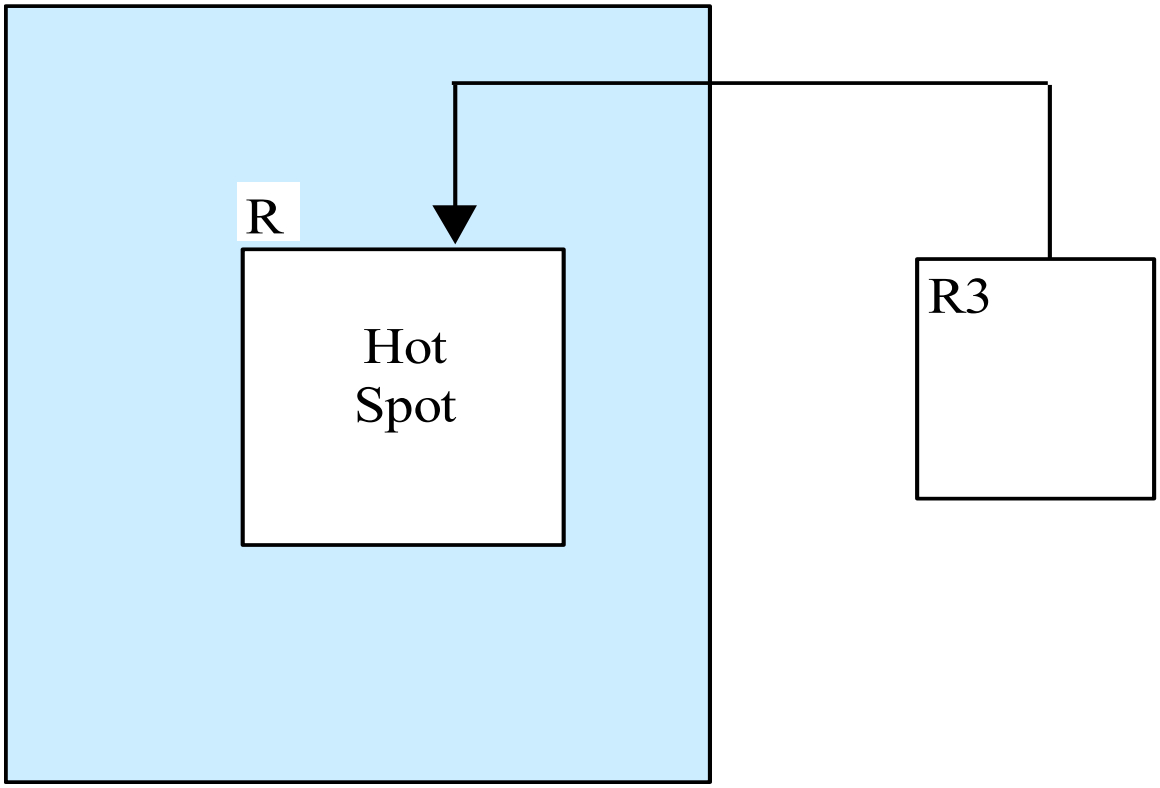
- Framework caixa branca:
  - reuso por herança e associação dinâmica
  - deve-se entender detalhes de como o framework funciona
- Framework caixa preta:
  - reuso por composição ou definição de interfaces para os componentes.
  - deve-se entender apenas a interface do cliente

Tipos de Frameworks

- Framework caixa cinza:
  - combinação do caixa branca e do caixa-preta
  - reuso por herança, associação dinâmica e definição de interfaces
  - Levantamento realizado por Yassin e Fayad em 1999: 55% dos frameworks caixa-cinza, 30% dos frameworks caixa-branca e 15% dos frameworks caixa-preta

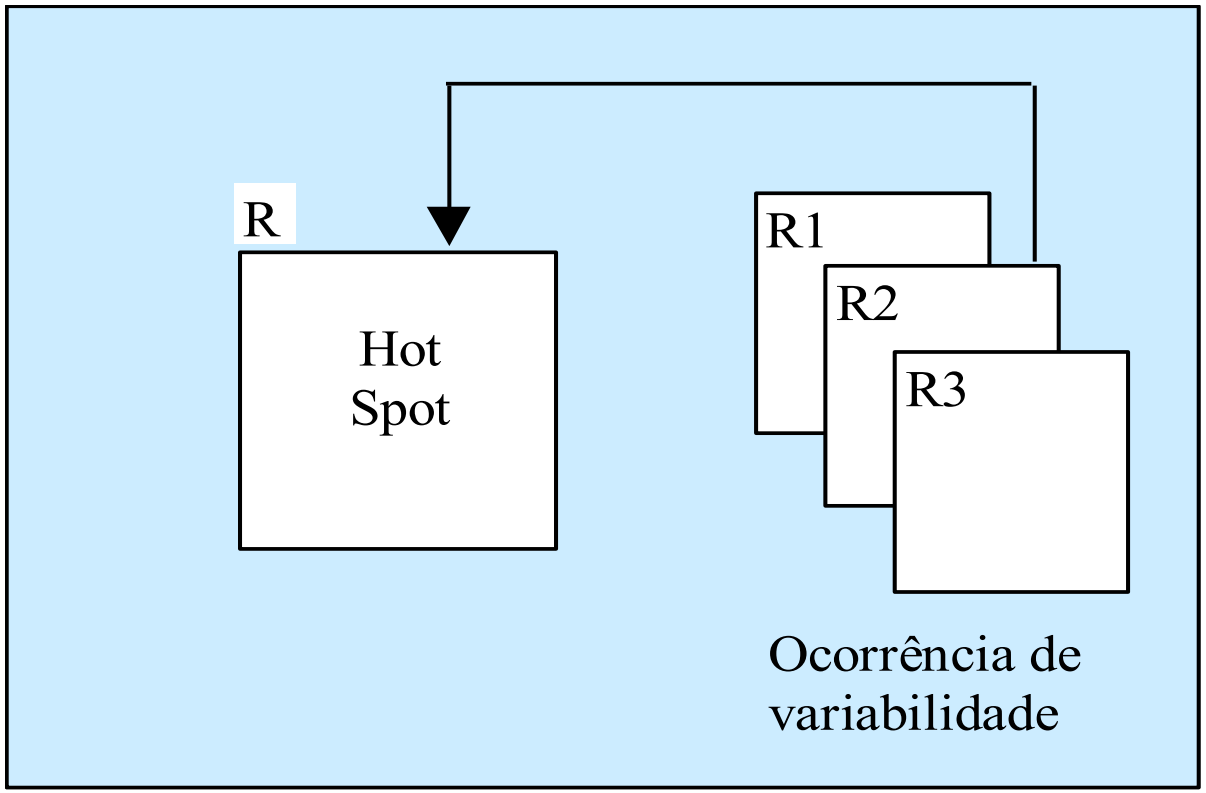
Tipos de Frameworks

Framework Caixa Branca



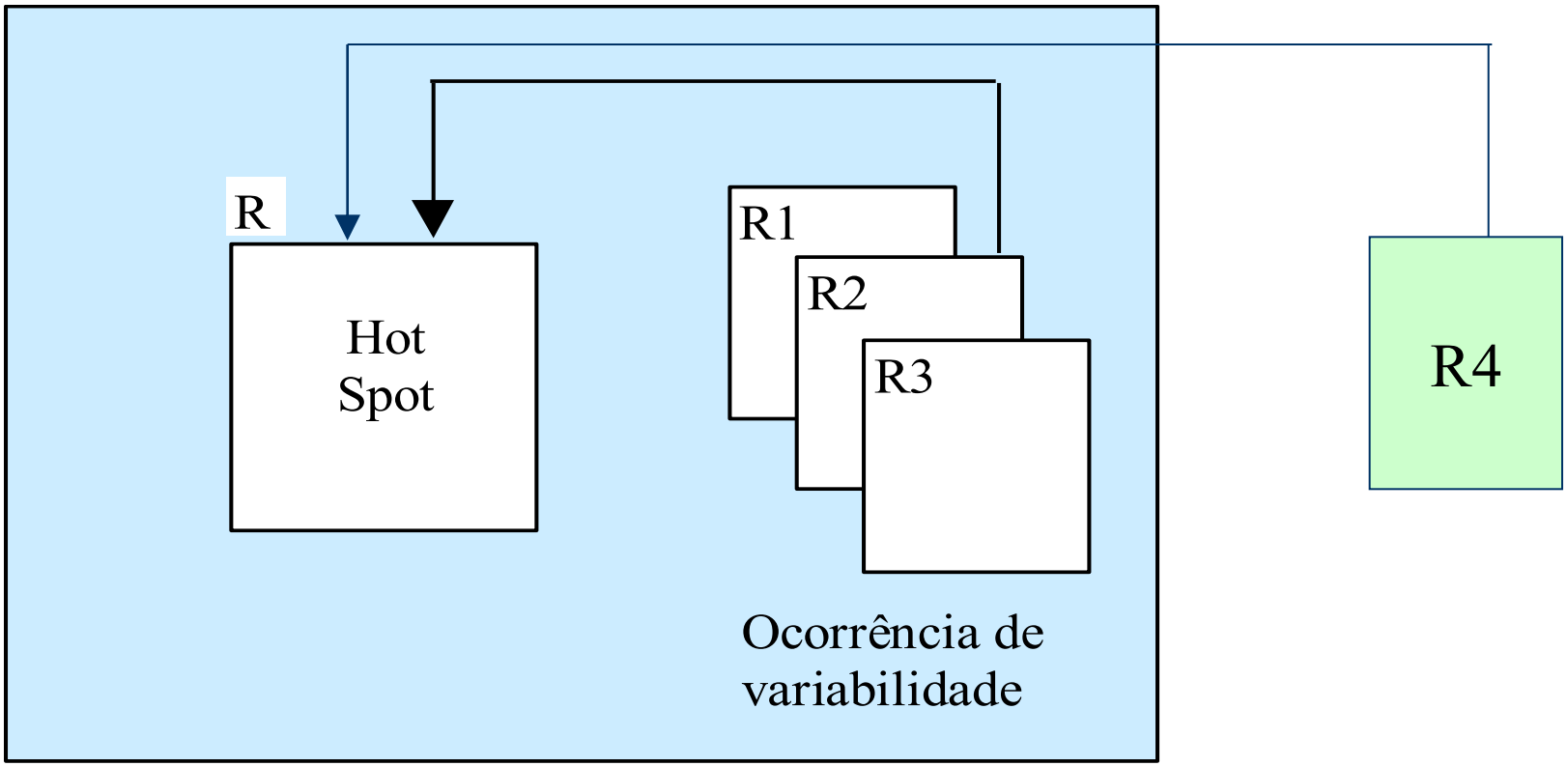
Tipos de Frameworks

Framework Caixa Preta



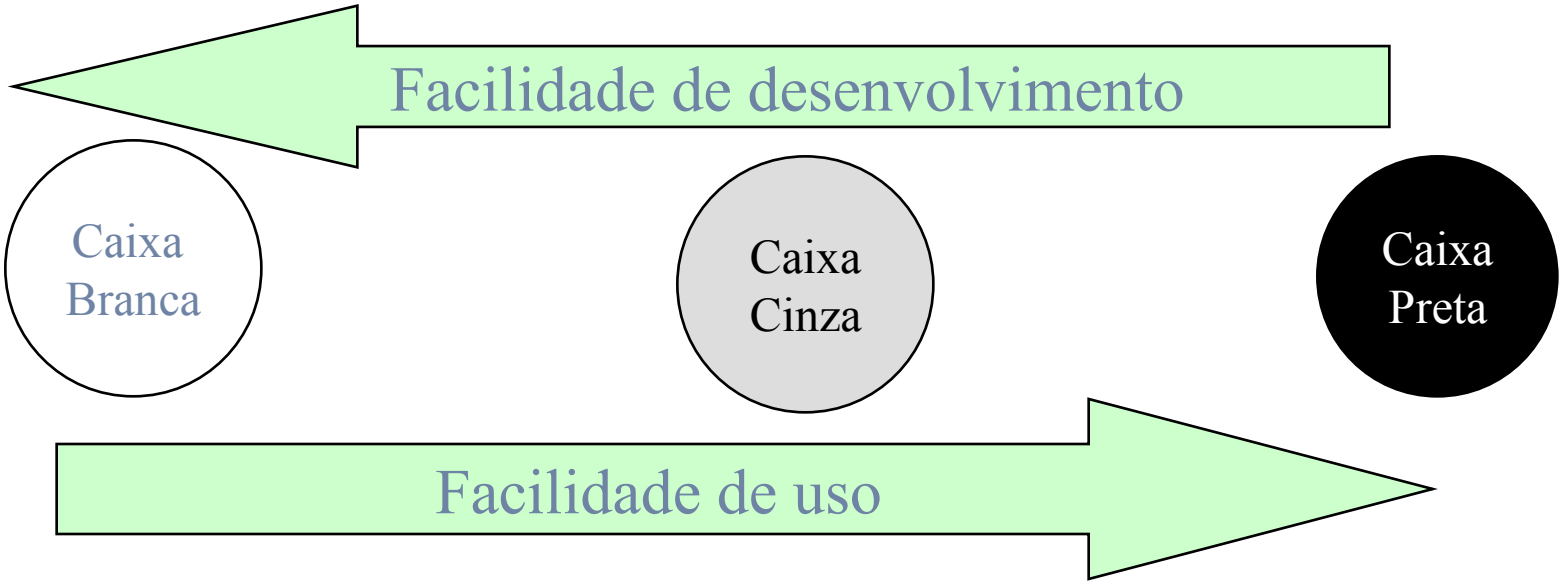
Conceitos Básicos

Framework Caixa Cinza



Tipos de framework

- framework caixa branca é mais fácil de projetar
- framework caixa preta é mais fácil de usar
- frameworks caixa-branca evoluem para se tornar mais caixa preta

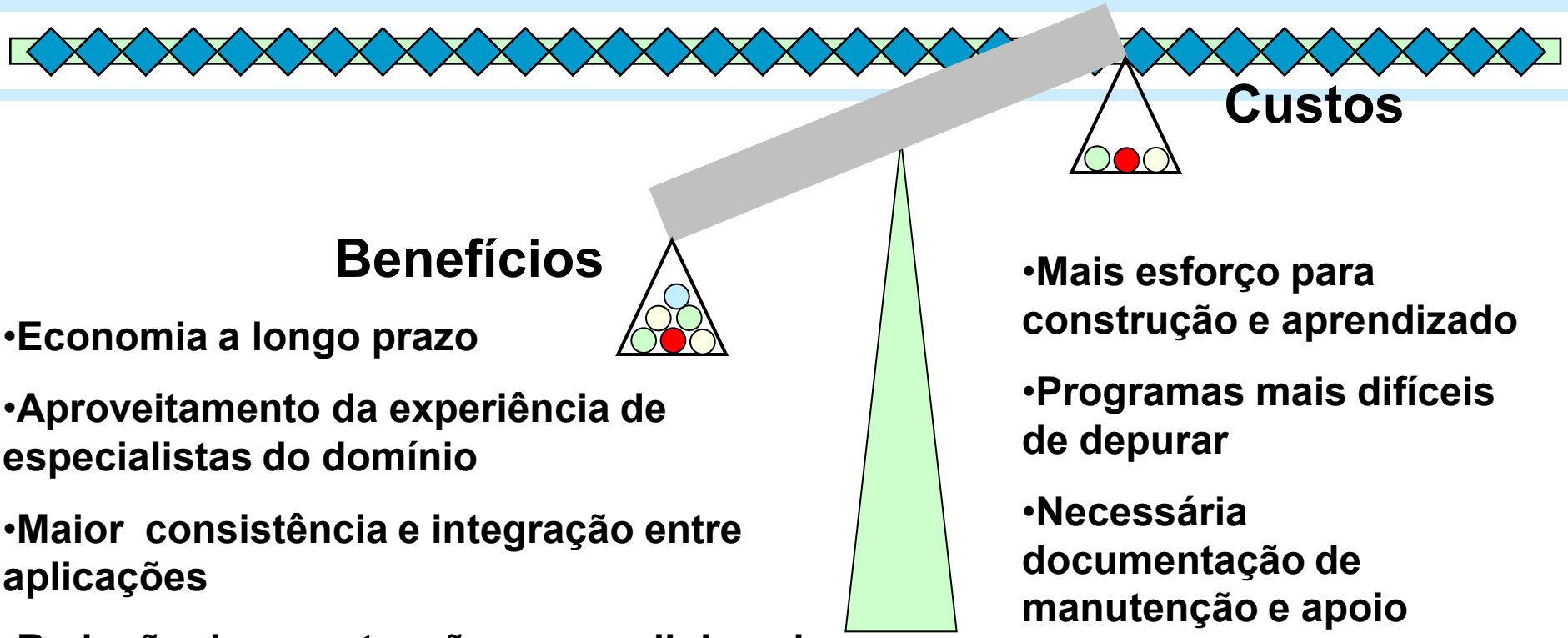


Frameworks

Inversão de Controle

- Biblioteca de subrotinas ou componentes
  - Programa do usuário chama código reusado
  - Usuário projeta estrutura do programa
- Framework
  - Código reusado chama programa do usuário
  - Estrutura do programa determinada principalmente pelo código reusado

Frameworks



Benefícios

- Economia a longo prazo
- Aproveitamento da experiência de especialistas do domínio
- Maior consistência e integração entre aplicações
- Redução da manutenção: menos linhas de código nas aplicações, reparos no framework se propagam pelas aplicações
- Melhora da produtividade: os programadores podem se concentrar nas características específicas de suas aplicações

Custos

- Mais esforço para construção e aprendizado
- Programas mais difíceis de depurar
- Necessária documentação de manutenção e apoio

Desenvolvimento de Software Orientado a Aspectos

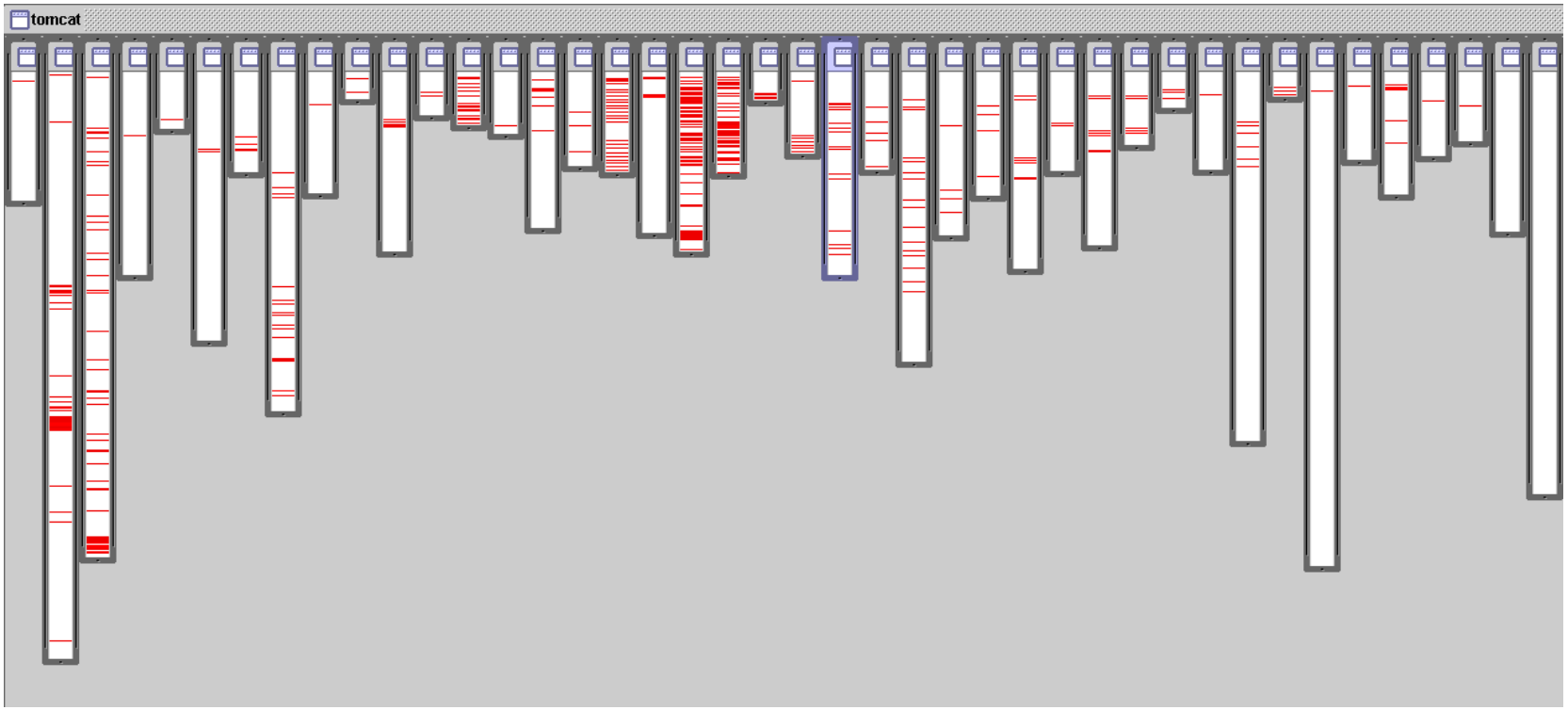
- Existem interesses com os quais um sistema deve lidar que não são modularizáveis segundo os paradigmas de desenvolvimento de software dominantes.
- São interesses ortogonais, porque se aplicam simultaneamente a várias partes do sistema.
- Exemplos: a persistência, a distribuição, o paralelismo, a concorrência, a segurança e controle de acesso.

Desenvolvimento de Software Orientado a Aspectos

- Como conseqüência da impossibilidade de modularização, normalmente tem-se como resultado um emaranhamento ou entrelaçamento de código, ou seja, o código referente à responsabilidade principal da classe está misturado com o código referente a uma ou mais responsabilidades ortogonais
- A programação orientada a aspectos (POA) apóia a separação de interesses no desenvolvimento de sistemas

Exemplo de Interesse Ortogonal

Código para Logging no Tomcat



- Code tangled (entrelaçamento de código)
- Code spread (espalhamento de código)

ApplicationSession

StandardSession

SessionInterceptor

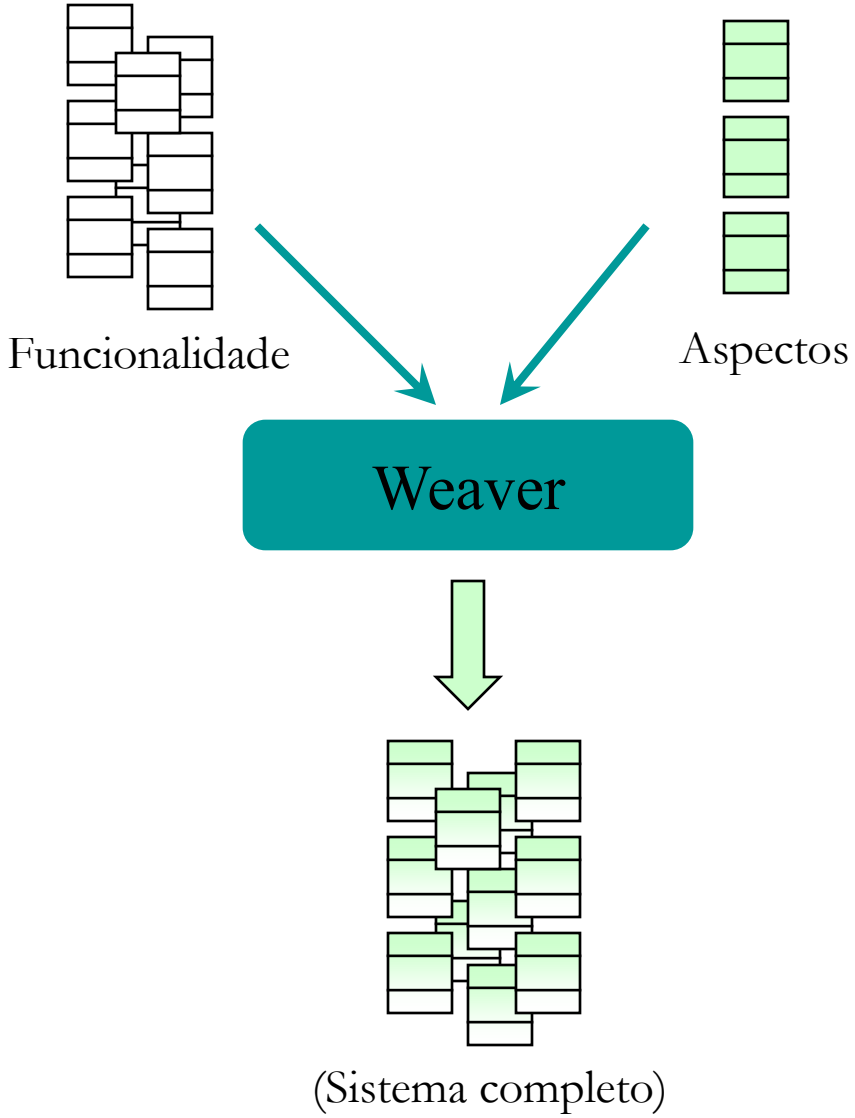
StandardManager

StandardSessionManager

ServerSession

ServerSessionManager

Composição (Weaving)



Principais Conceitos no AspectJ

- AspectJ implementa as abstrações da POA
  - Aspectos
    - Pointcuts (conjuntos de junção/conjunto de pontos de junção)
      - Join points (pontos de junção)
    - Advices (adendos)

Designadores de Pontos de Junção

- São especificados por meio de "designadores de conjuntos de junção", por exemplo:

  - call → chamadas a métodos
  - execution → execução de métodos
  - set → escrita de atributos
  - get → leitura de atributos

Pontos de Junção

- Exemplo
  - `call (void Point.setX(int))`

  – Intercepta chamadas a um método denominado setX(), que contenha um parâmetro do tipo int e cujo tipo de retorno é void

  – Só realiza o entrecorte para métodos da classe Point

Pontos de Junção

- Podem ser compostos utilizando filtros de composição semântica e/ou wildcards

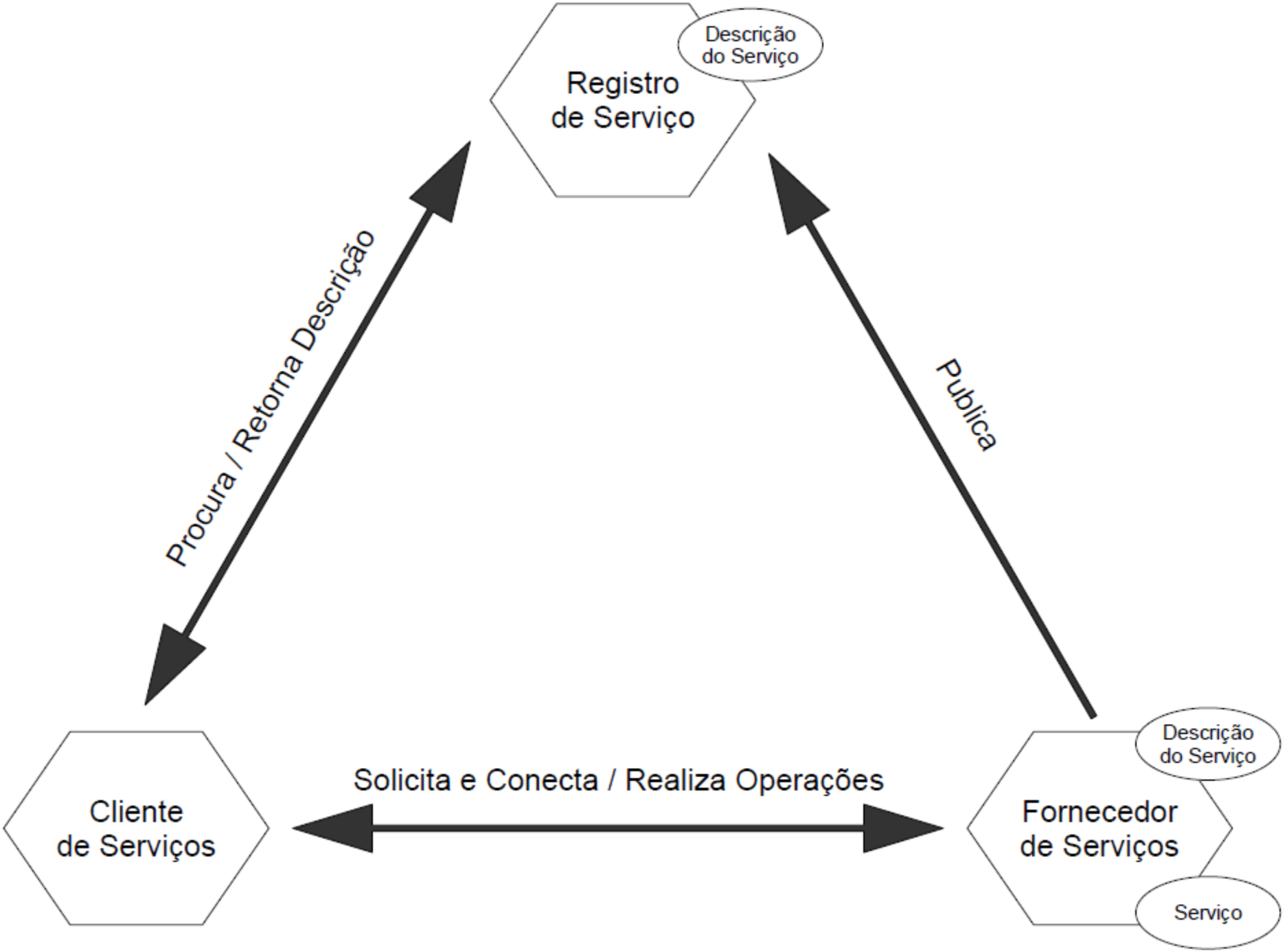
```
– call (Point.setX(int) || Point.setY(int))
– execution (void Figure+.make*(..))
– execution (public * Figure.*(..))
– set (int Figure.point);
```

Serviços

- SOA é um estilo arquitetural para a construção de aplicações de software que utilizam serviços disponíveis em uma rede.
- Um serviço é a implementação de uma funcionalidade de negócio bem definida, que pode ser utilizado por clientes em diferentes aplicações.
- Um serviço possui uma interface pública
  - Web services X SOA.
  - WSs utilizam linguagens e protocolos baseados XML.

Web Services

- Identificado por um *Uniform Resource Identifier* (URI)
- Interface descrita em *Web Services Description Language* (WSDL)
- Interações entre máquinas definidas pela troca de mensagens transportadas via requisições *Hypertext Transfer Protocol* (HTTP)
- Requisições empacotadas via *Simple Object Access Protocol* (SOAP) ou *Representational State Transfer* (REST)


Registro
de Serviço
Descrição
do Serviço
Procura / Retorna Descrição
Publica
Cliente
de Serviços
Solicita e Conecta / Realiza Operações
Fornecedor
de Serviços
Descrição
do Serviço
Serviço

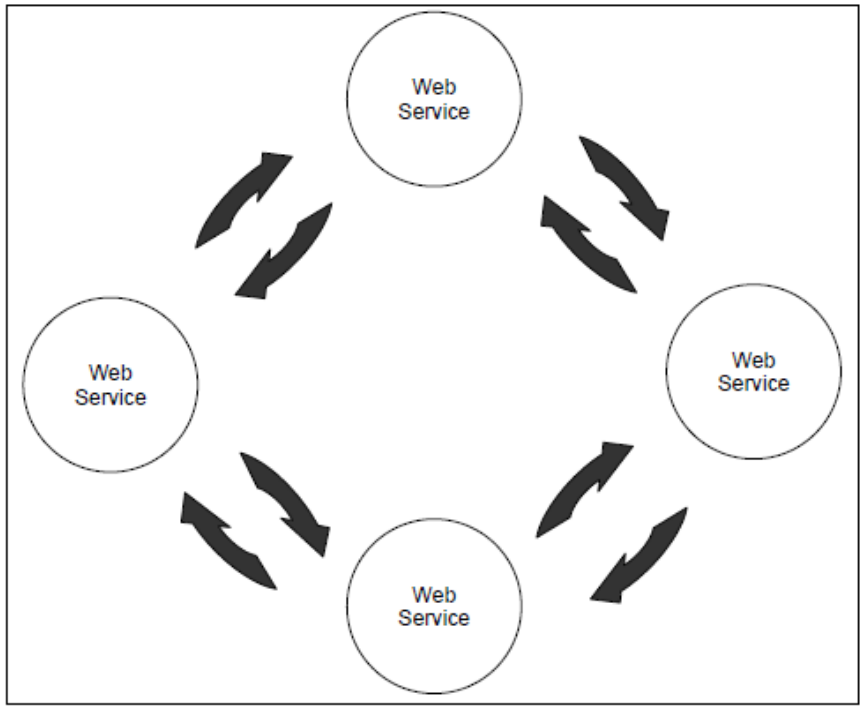


Figura: *Coreografia*

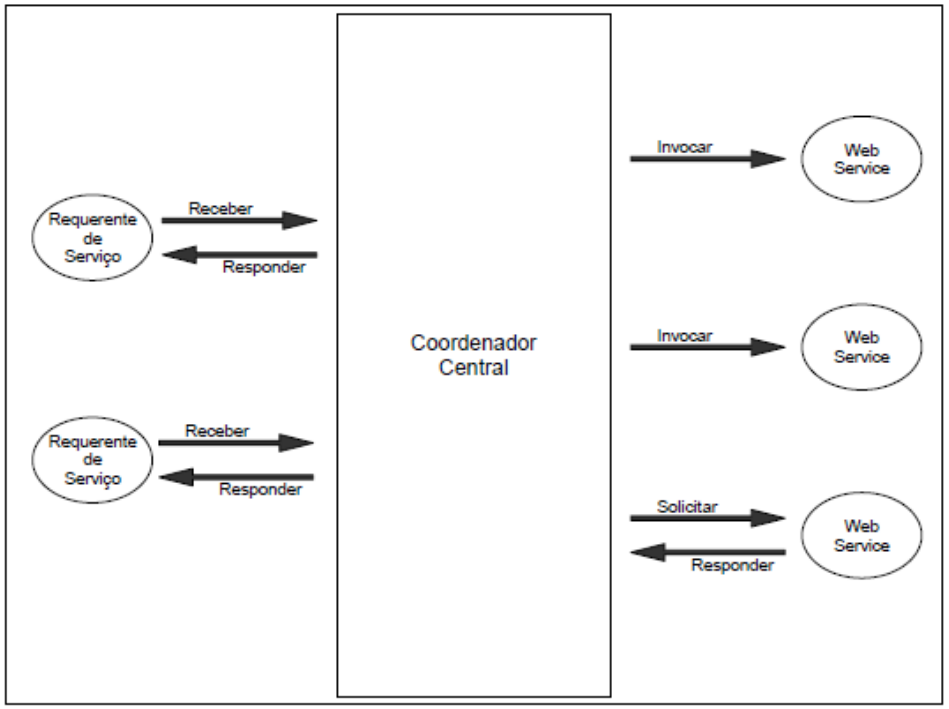


Figura: *Orquestração*

Linha de Produtos de Software (LPS)

- Técnica amplamente utilizada na indústria automotiva, aeronáutica e eletrônica

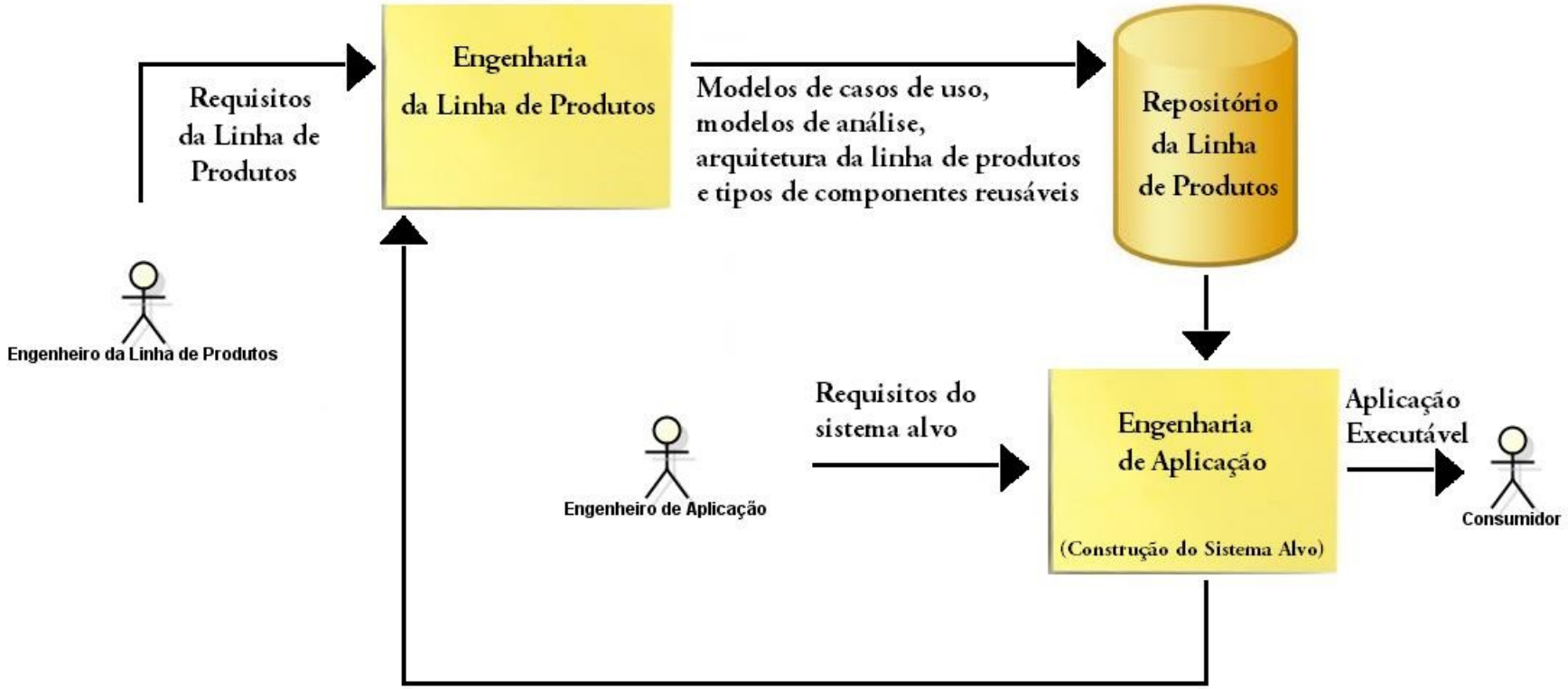
Linha de Produtos de Software (LPS)

- Consiste de um conjunto de sistemas de software que compartilham características comuns e gerenciadas e que satisfazem a uma necessidade específica de um segmento particular de mercado, sendo desenvolvidas a partir de um conjunto comum de ativos centrais, de forma sistemática (Clements e Northrop, 2001).

Linha de Produtos de Software (LPS)

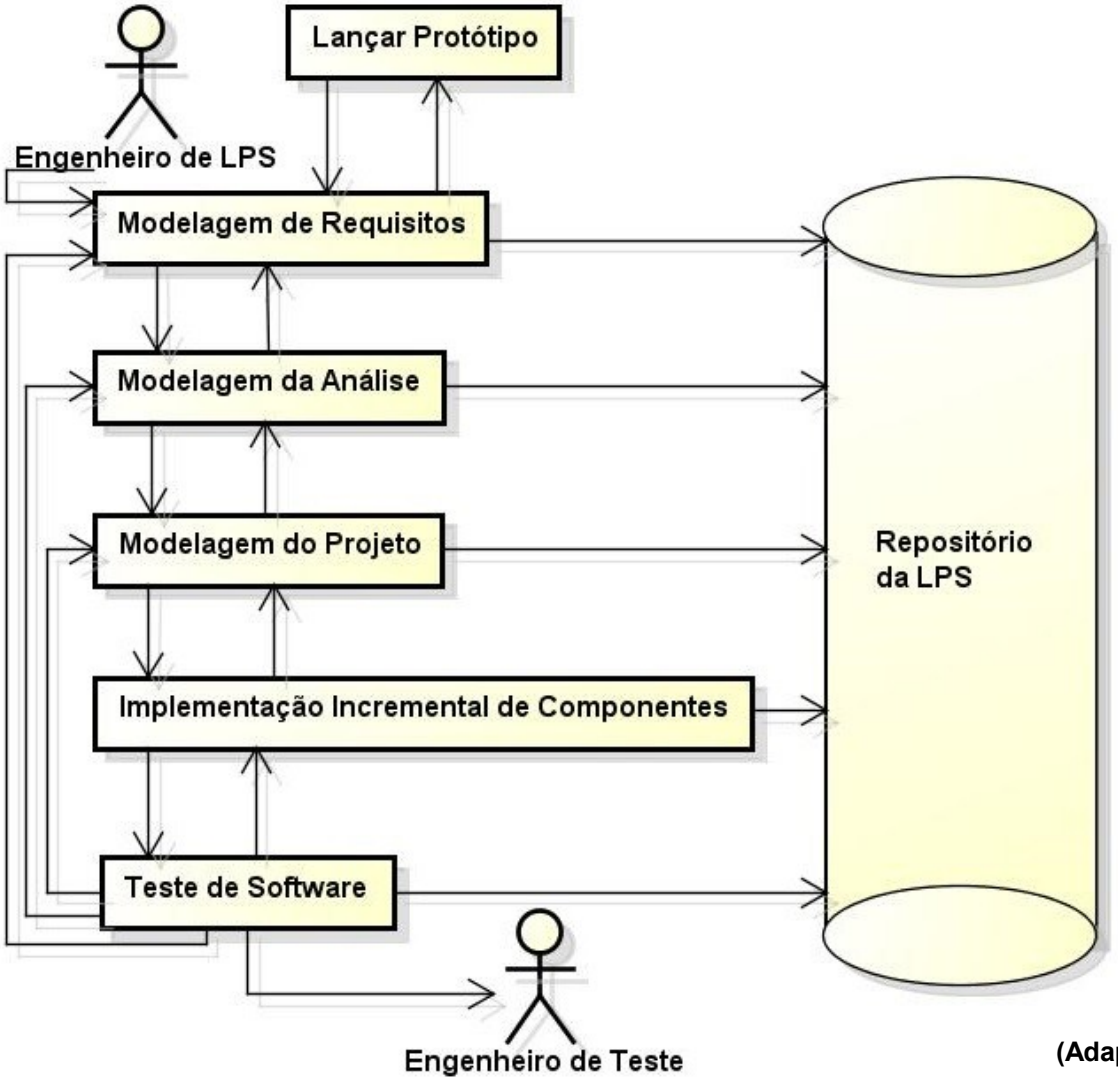
- Tem como objetivo o desenvolvimento em larga escala.
  - A engenharia de domínio produz artefatos usados pelos membros da LPS.
  - A engenharia de aplicação permite a configuração e montagem dos membros da linha a partir dos artefatos gerados na engenharia de domínio.

Product Line UML-based Software Engineering (**PLUS**)



(Adaptada de GOMAA, 2006)

Evolutionary Software Product Line Engineering Process (ESPLEP)

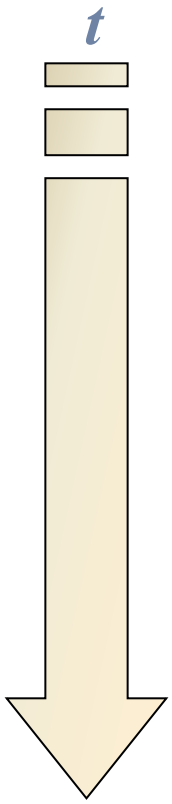


(Adaptada de GOMAA, 2006)

60

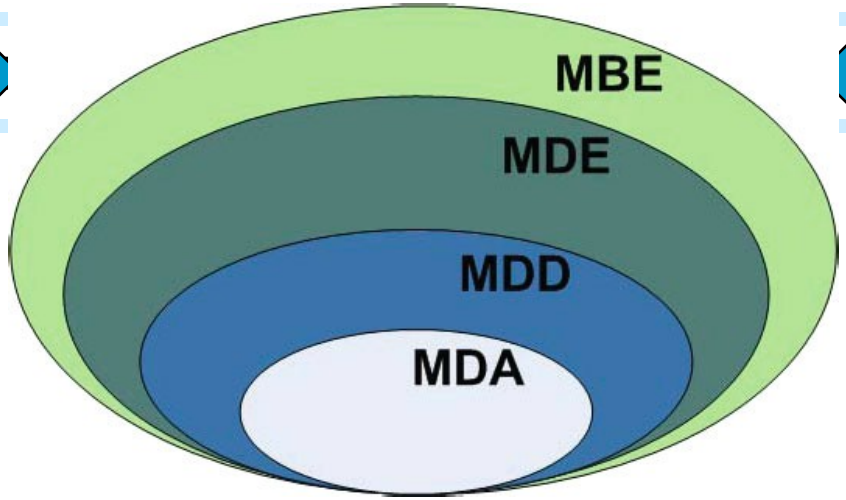
Model Driven Sofware Engineering

Application area of modeling



- ***Models as drafts***
  - Communication of ideas and alternatives
  - Objective: modeling per se
- ***Models as guidelines***
  - Design decisions are documented
  - Objective: instructions for implementation
- ***Models as programs***
  - Applications are generated automatically
  - Objective: models are source code and vice versa

The MD* Jungle of Acronyms

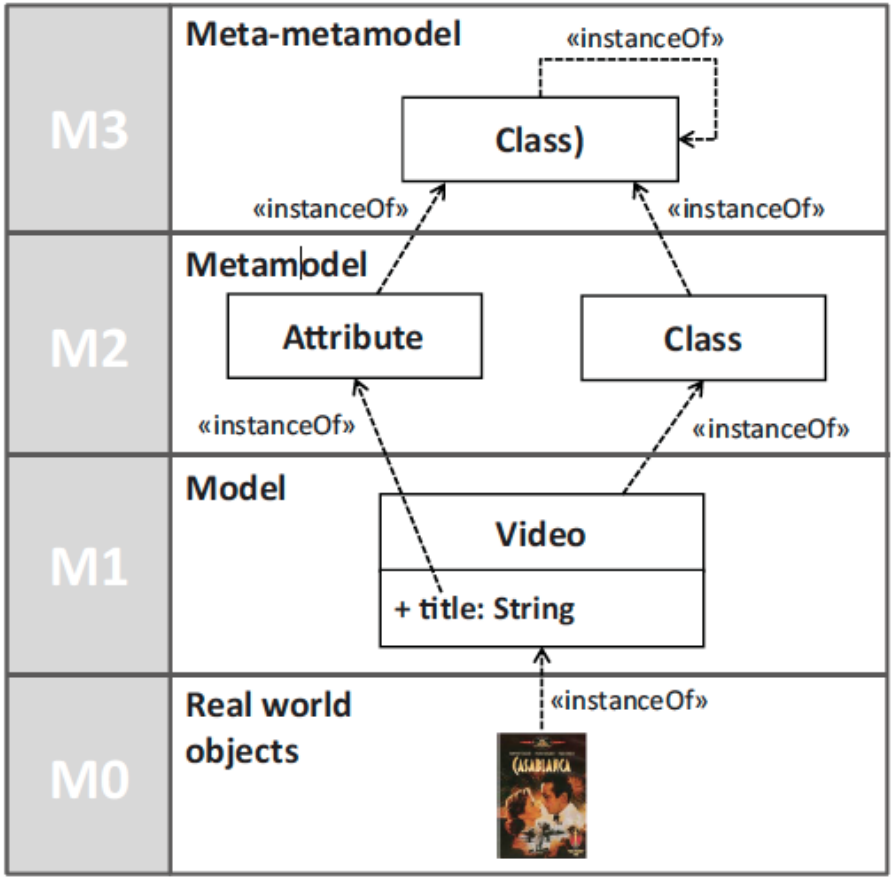


- **Model-Driven Development (MDD)** is a development paradigm that uses models as the primary artifact of the development process.
- **Model-driven Architecture (MDA)** is the particular vision of MDD proposed by the Object Management Group (OMG)
- **Model-Driven Engineering (MDE)** is a superset of MDD because it goes beyond of the pure development
- **Model-Based Engineering** (or “model-based development”) (**MBE**) is a softer version of ME, where models do not “drive” the process.

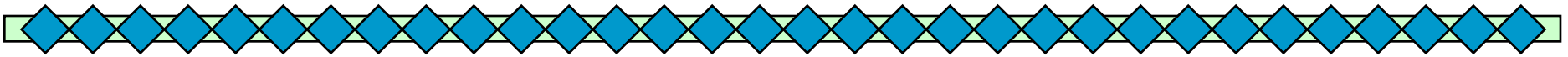
Metamodeling

- To represent the models themselves as “instances” of some more abstract models.
- **Metamodel** = yet another abstraction, highlighting properties of the model itself

- Metamodels can be used for:
  - defining new languages
  - defining new properties or features of existing information (metadata)



Model Transformations



- Transforming items
- MDSE provides appropriate languages for defining model transformation rules
- Rules can be written manually from scratch by a developer, or can be defined as a refined specification of an existing one.
- Alternatively, transformations themselves can be produced automatically out of some higher level mapping rules between models
  - defining a mapping between elements of a model to elements to another one (**model mapping or model weaving**)
  - automating the generation of the actual transformation rules through a system that receives as input the two model definitions and the mapping
- Transformations themselves can be seen as models!!

Conclusões

- Várias técnicas de reúso têm surgido e têm tido sucesso ultimamente, contribuindo para eliminar, ou pelo menos amenizar, a dificuldade de reúso
- Alunos e professores do Labes – ICMC – USP têm colaborado com pesquisas sobre reuso utilizando padrões, frameworks, componentes, serviços, aspectos, MDSE, SoS e LPS.

Contato

- Pesquisadores do ICMC ligados à pesquisa sobre reuso
  - Profs. Rosana Braga, Paulo Masiero e José Carlos Maldonado, Elisa Nakagawa e Ellen Francine
  - Alunos de doutorado
  - Alunos de mestrado e IC

Contato

- Rosana T. Vaccare Braga
  - ICMC-Universidade de São Paulo
  - São Carlos – SP – Labes (Laboratório de Engenharia de Software)
  - rtvb@icmc.usp.br
  - http://www.icmc.usp.br/~rtvb