

Parte 1 – Introdução ao Desenvolvimento Baseado em Componentes

C.H: 33 horas

Dias: 15/2; 15/3 e 12/4/14



Prof. Sergio Akio Tanaka
sergio.tanaka@audare.com.br

Agenda

- Motivação
- Reutilização de Software
- Desenvolvimento Baseado em Componentes: Histórico, Definição, Componentes e Interfaces
- Qualidade de Componentes
- Modelagem de Componentes e utilizando a UML
- Tecnologias envolvidas: Orientação a Objetos, Arquitetura de Software, Padrões de Software
- Abordagens / Métodos existentes para DBC
- Arquitetura MDA (CIM, PIM e PSM)
- Linha de Produto de Software
- Introdução a Governança Corporativa
- Exercícios práticos

Referências Bibliográficas

- OMG. OBJECT MANAGEMENT GROUP. Disponível em:www.omg.org/mda. 2014.
- GAO, Jerry Zeyu et al. Testing and Quality Assurance for Component-based Software. Ed. Artech House. UK. 2003.
- CUMMINS, Fred A. Building the Agile Enterprise with SOA, BPM and MPM. MK. OMG. 2009.
- ATKINSON, Colin et al. Component-based Product Line Engineering with UML. Ed. Addison Wesley. UK. 2002.
- BUSCHMANN, F. et al. **A System of Patterns**. Pattern – Oriented Software Architecture. New York: Wiley, 1996.
- D'SOUZA, D.; WILLS, A. **Objects, Components and Frameworks with UML - The Catalysis Approach**. USA: Addison-Wesley, 1999.
- GIMENES, I. M. S.; TANAKA, S. A.; PALAZZO, J. M. O. **An Object Oriented Framework for Task Scheduling**. In: TOOLS EUROPE 2000, França, IEEE Computer Society Press.
- GARLAN, D., PERRY, D. E., Introduction to Special Issue on Software Architecture, **IEEE Transaction on Software Engineering**, New York, Apr. 1995.

Parte I - Fundamentos

Motivação

- .Necessidade de melhoria no processo de desenvolvimento de software: maior produtividade e menor custo;
- .Antes: Desenvolvimento de software em blocos monolíticos;
- .Solução: Uso de Técnicas de Reutilização de Software para criação de componentes interoperáveis;

Reutilização de Software

- **Definição:** “Utilização de produtos de software, construídos ao longo do processo de desenvolvimento, em uma situação diferente daquela para a qual foram originalmente produzidos” (Freeman, 1980);

Reutilização de Software

- **Benefícios:**
 - Qualidade
 - Produtividade
 - Redução de custos/ tempo
 - Flexibilidade
- **Dificuldades:**
 - Identificação, recuperação, compreensão
 - Modificação
 - Qualidade de componentes
 - Barreiras psicológicas, legais e econômicas
 - Necessidade de incentivos

Reutilização de Software

- **Gerência de Reuso:**
 - Planejamento
 - Criação de componentes
 - Gerência de componentes
 - Utilização de componentes

Desenvolvimento baseado em Componentes

- **Histórico**

01/02

- 1969, McIlRoy já vislumbrava uma indústria de componentes de software reutilizáveis;
- linguagens de interconexão de módulos, em 1976;
- abordagens até então se baseavam fortemente em código;

- **Impulso para DBC**

- Internet (Web Services);
 - **SOA**
 - **REST (Forma de implementação de SOA)**
 - Modelos de componentes: CORBA, DCOM, RMI, JEE,...
 - Processamento distribuído;
 - **Estilos Arquiteturais**

Desenvolvimento baseado em Componentes

- **Objetivos:**

02/02

- quebra de blocos monolíticos em componentes interoperáveis;
- componentes são construídos / empacotados com o objetivo de serem reutilizados em diferentes aplicações;
- um componente provê um conjunto de serviços acessíveis através de uma interface bem definida;
- **Dificuldades:**
 - O que é de fato um componente?
 - Que tecnologias estão envolvidas?
 - Como desenvolver componentes?

O que é um Componente?

- “**Um componente como um elemento arquitetural.** Este componente deve prover e estar em conformidade com um conjunto de interfaces”;
- “**Um componente como um elemento implementacional** acessados através de interfaces bem documentadas que podem ser descobertas em tempo de execução”;

O que é um Componente?

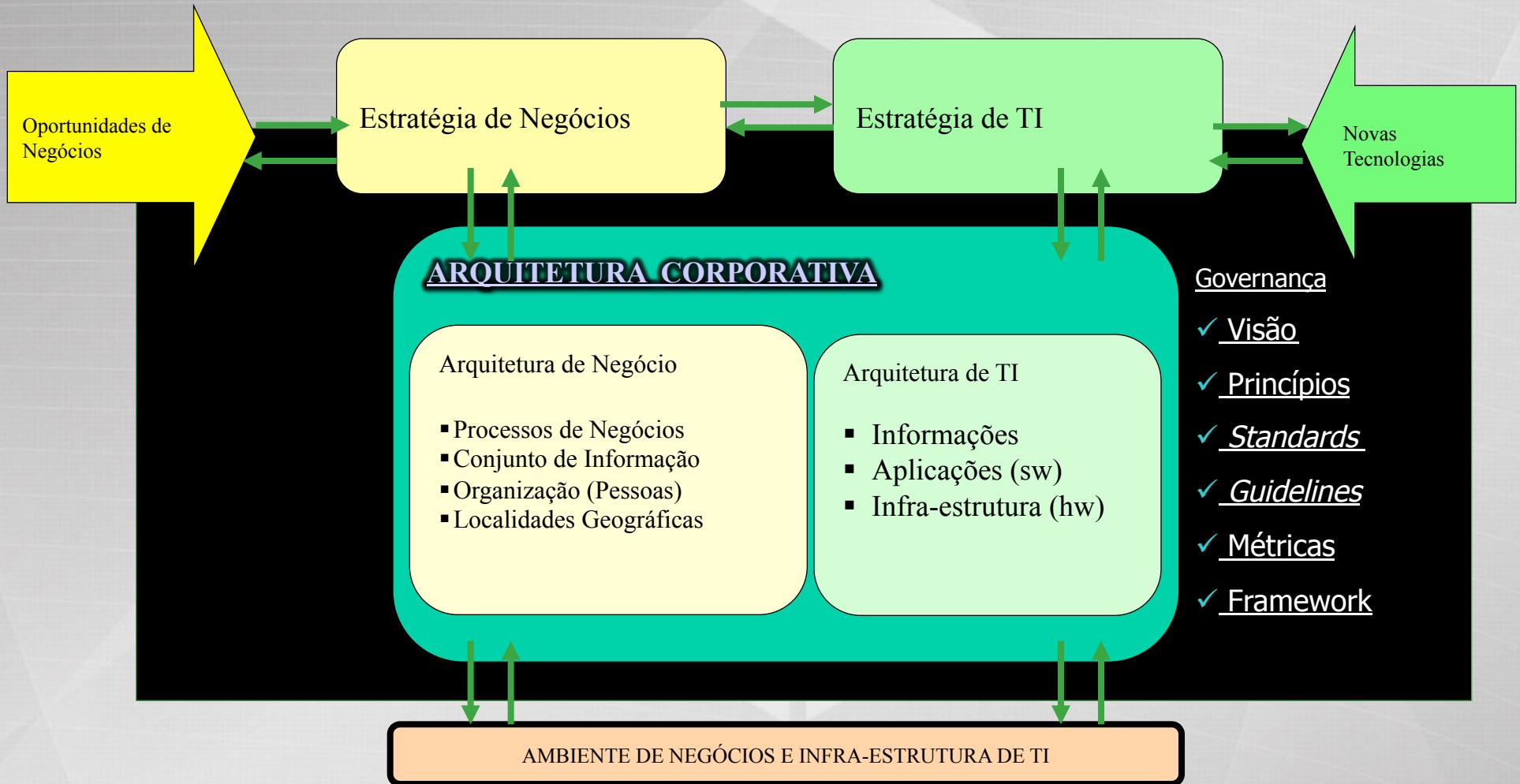
- **Características de consenso:**

- descrever ou realizar uma função específica;
- estar em conformidade e prover um conjunto de interfaces bem definidas;
- ter disponível uma documentação adequada;
- prover um grau de maturidade do componente;
- estar inserido no contexto de um modelo que guie a composição deste componente com outros (arquitetura de software);
- Abordagem conhecida como componentware

O que é um Componente?

- **Visão geral**
 - Um componente representa a **implementação** em software de um conceito autônomo ou **processo de negócio**;
- **Dois tipos de componentes**
 - componentes de negócio, que o domínio em si consegue reconhecer (ex. cliente, contas, etc);
 - componentes de infra-estrutura ou **componentes** de suporte aos componentes de negócio;

Posicionamento da Arquitetura Corporativa



Arquitetura Corporativa

- Uma composição de Arquiteturas...

Arquitetura Corporativa

Arquitetura de Negócio

**Arquitetura de
Informação
(ou dados)**

**Arquitetura de
Aplicação
(ou sistemas)**

**Arquitetura de Infra-estrutura
(ou tecnologia)**

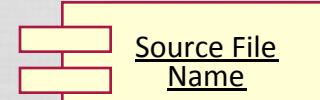


O que é um Componente?

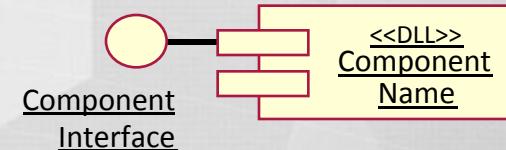
(1.x)

- Uma parte não-trivial, próximo da independência, e substituível de um sistema que possui funções claras que completam um contexto de uma arquitetura bem-definida.
- Um componente pode ser:

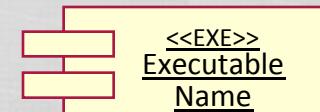
– Um componente de código fonte



– Um componente de tempo de execução



– Um componente executável

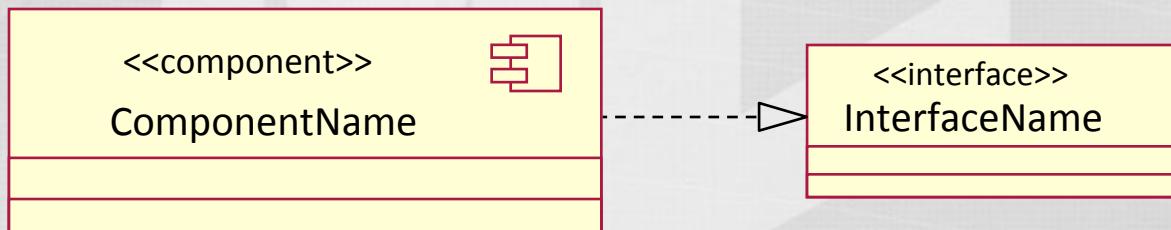


Substituível no Tempo de Execução

- Se um componente puder ser reimplementado em um sistema em execução, ele será chamado de "substituível no tempo de execução". Isso permite que seja feito upgrade do software sem perda de disponibilidade.

O que é um Componente?

- Uma parte modular de um sistema que oculta o seu conteúdo e aparenta ser substituível dentro de seu ambiente
 - Define os seus comportamentos em termos de interfaces fornecidas e requeridas
 - Podem ser substituídas em tempo de design ou de execução por um componente que oferece a funcionalidade equivalente, baseado na compatibilidade de suas interfaces.

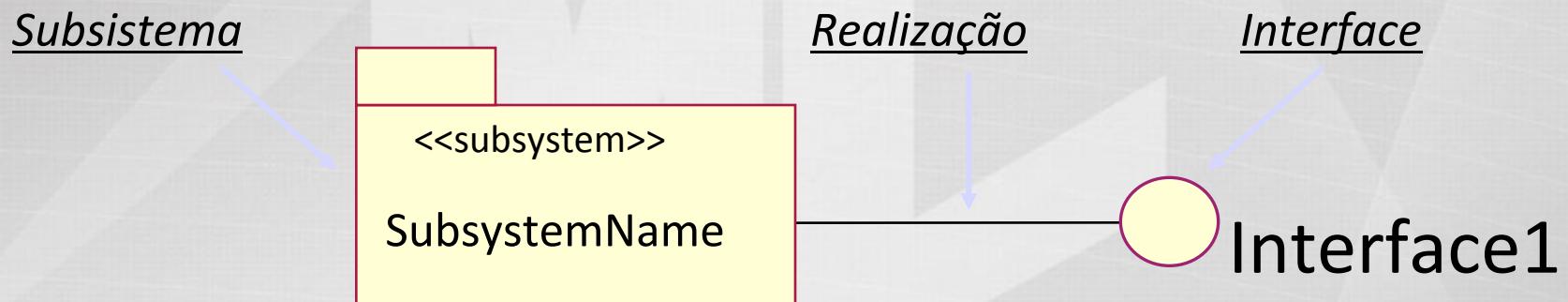


Princípios de OO: Encapsulamento e Modularidade

O que é um Subsistema?

(1.x)

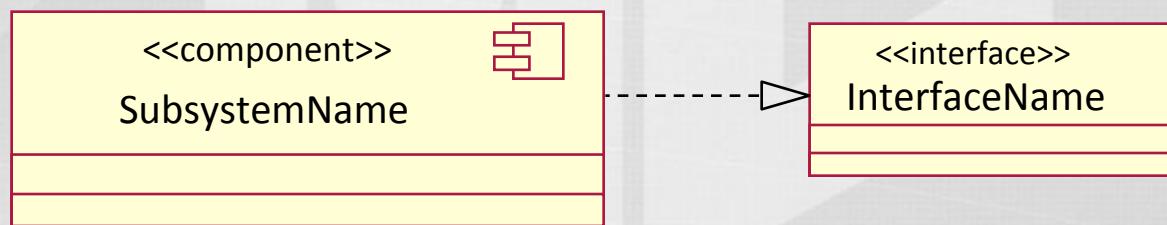
- Uma combinação de um pacote (que pode conter outros elementos de modelo) e uma classe (que possui o comportamento)
- Realiza um ou mais interfaces, o qual define o seu comportamento



Princípios de OO: Encapsulamento e Modularidade

O que é um Subsistema?

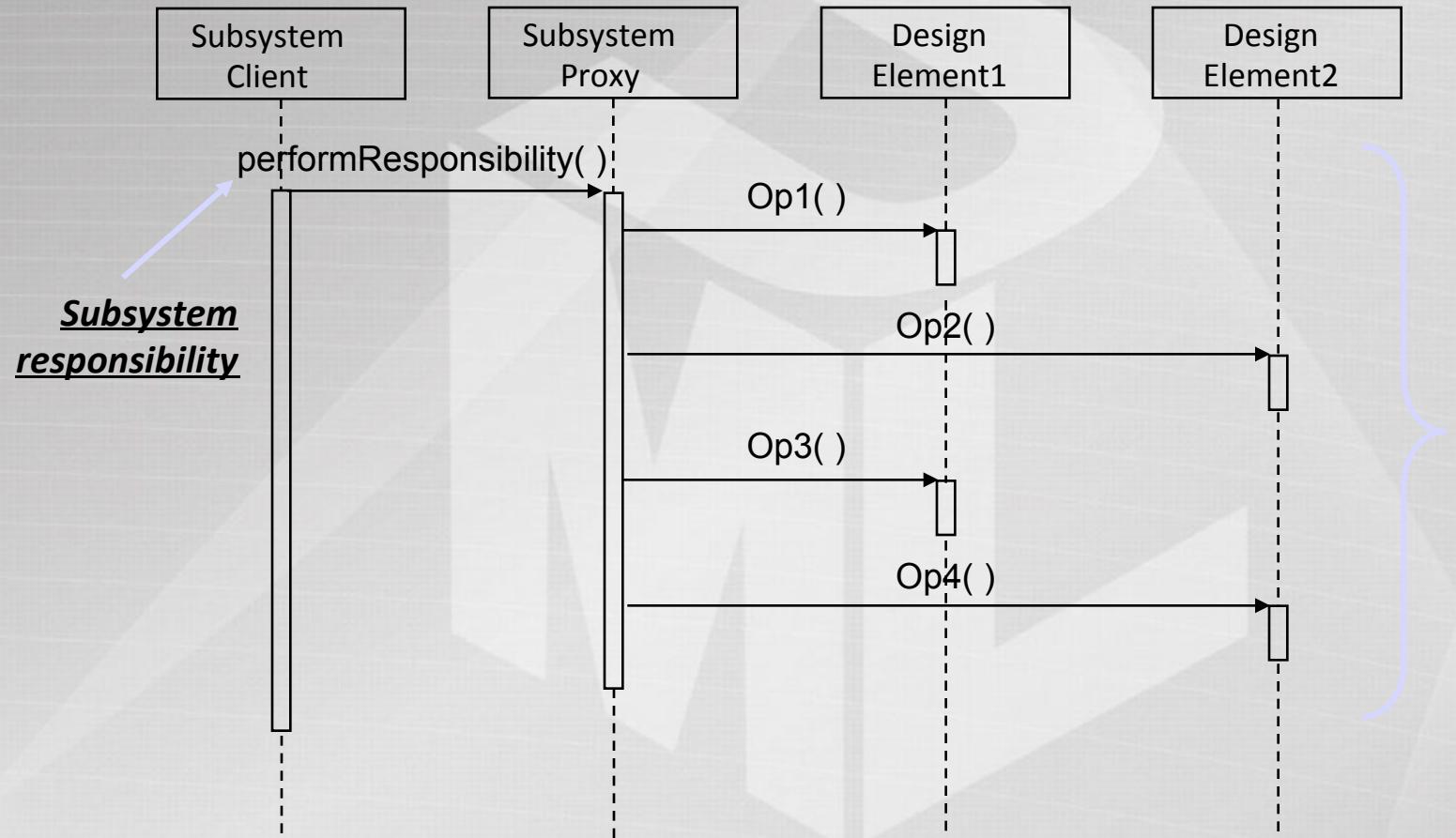
- Uma parte de um sistema que encapsula comportamentos, expõe um conjunto de interfaces e contém outros elementos de modelo.
 - Modelado como um componente



Princípios de OO: Encapsulamento e Modularidade

Convenção de Modelagem: Diagrama de Interação de Subsistemas

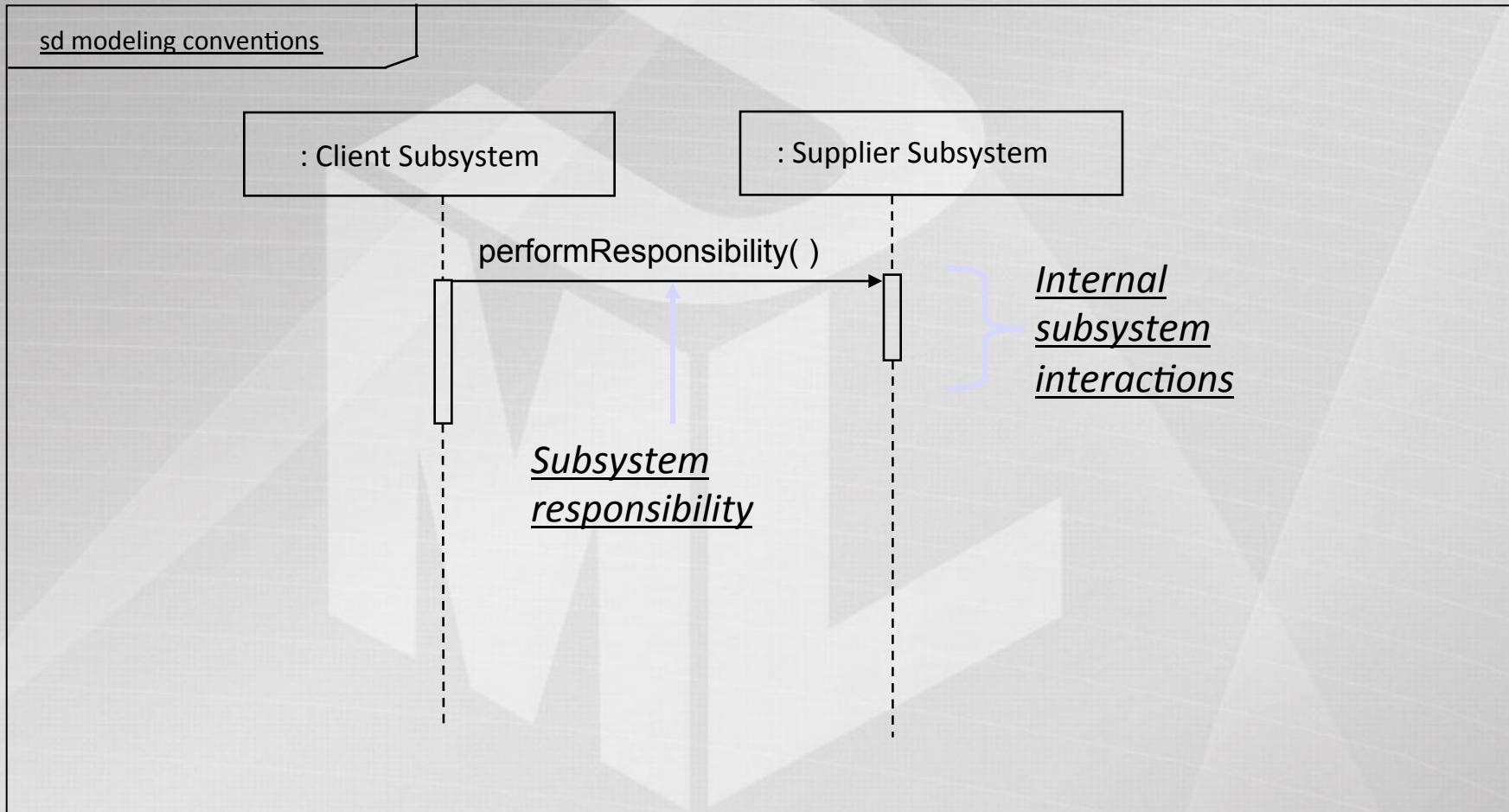
(1.X)



Subsystem interface not shown

Diagramas de Interação de Subsistemas

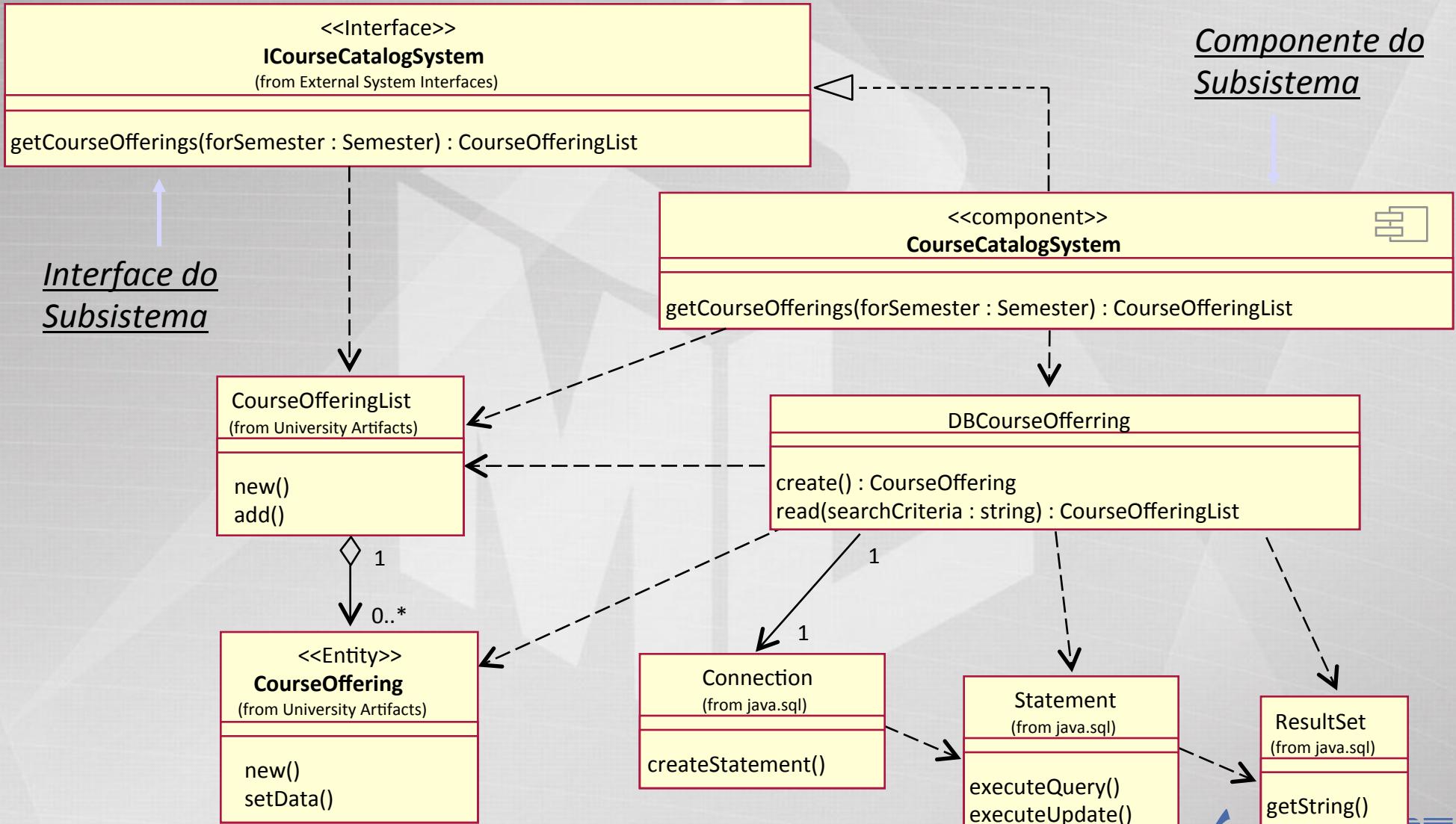
(2.0)



Visão externa de uma interação de um subsistema

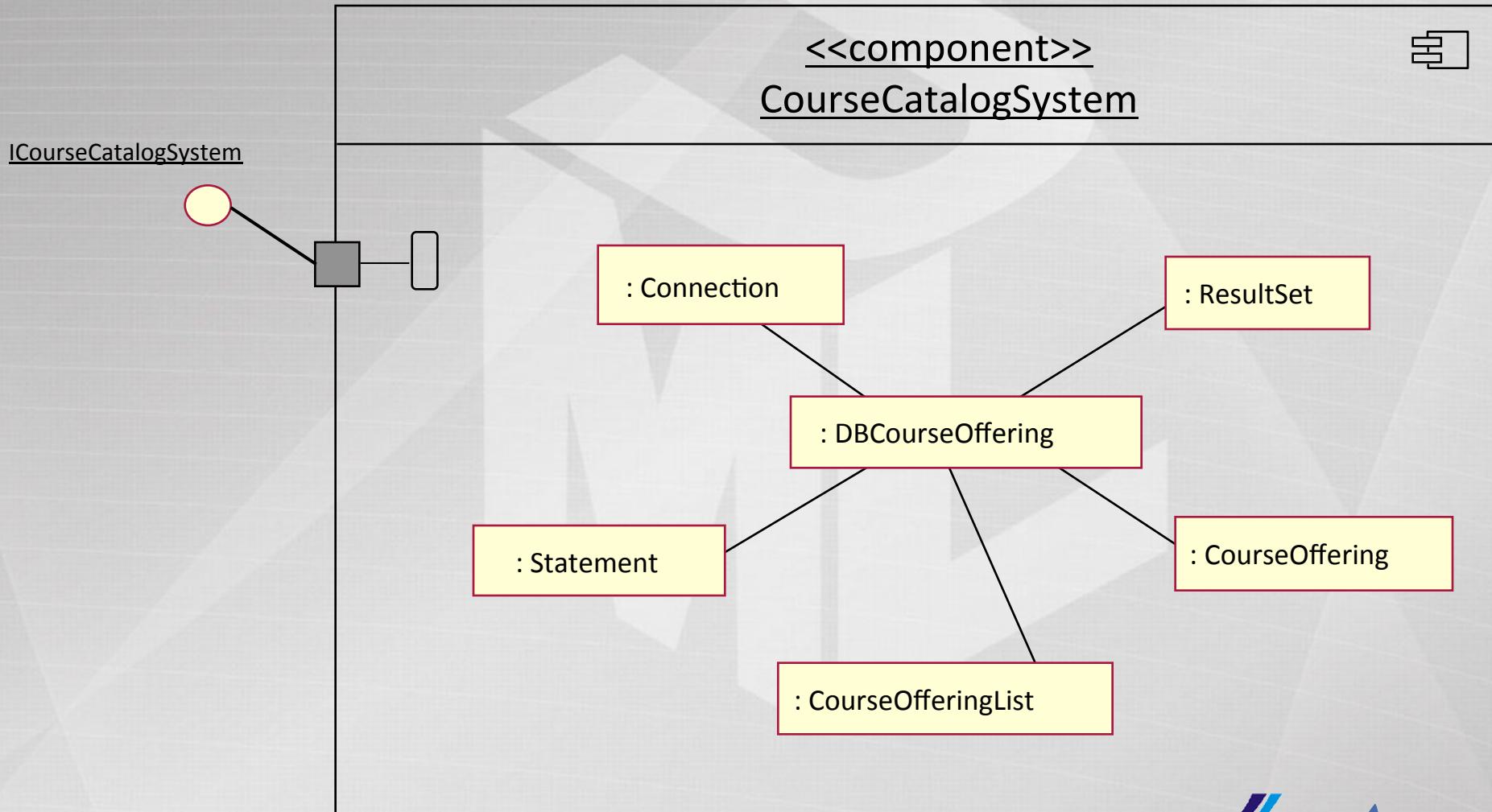
Exemplo: Elementos do Subsistema CourseCatalogSystem

(2.0)



Exemplo de Diagrama de Estrutura Composta

(2.0)



Tecnologias envolvidas

Orientação a Objetos e DBC

- a OO pura não apresenta todas as abstrações necessárias para o DBC
 - necessidade de se especificar o contexto onde um componente pode ser considerado uma peça de reposição (falta a definição de uma **arquitetura/framework** onde o componente se insere);
 - alguns mecanismos da OO dificultam a posterior reutilização do componente como “caixa preta” ;
 - herança...

Tecnologias envolvidas

Métodos OO e DBC

- Métodos OO foram desenvolvidos sem considerar explicitamente a reutilização de modelos e código;
- É preciso modificar o processo para enfatizar a reutilização de modelos já criados;
- Exemplos: Catalysis, RSEB (*Reuse-driven Software Engineering Business*), UML Components, Kobra.

Tecnologias envolvidas

Arquitetura de Software e DBC

- “É a estrutura dos componentes de um sistema, seus inter-relacionamentos, princípios e diretrizes que governam seu projeto e evolução ao longo do tempo” (Clements, 1996);
- Um componente é inseparável da arquitetura sobre a qual ele irá atuar;
- A interface do componente deve possuir as regras arquiteturais para o relacionamento entre componentes em uma dada arquitetura.

Tecnologias envolvidas

Arquitetura de Software e DBC

- Uso de uma arquitetura que disponibilize serviços para:
 - permitir a busca de quais componentes estão realmente conectados à infraestrutura;
 - garantia da entrega de mensagens entre componentes;
 - gerenciar as interações, consistindo de múltiplas interações entre os componentes;
 - permitir a comunicação segura entre os componentes.

Ex. ORB

Tecnologias envolvidas

Padrões e DBC

- Estruturas que enfatizam boas práticas em modelagem em diversos níveis de abstração:
 - de projeto: design patterns de Gamma;
 - arquitetural: padrões arquiteturais de Buschmann;
 - de análise: padrões de análise de Martin Fowler;
- Atuam como planos ou estruturas a serem seguidas;
- Dispostos em conjunto de maneira organizada e estruturada (catálogos ou sistemas de padrões);

Tecnologias envolvidas

Uso de padrões em DBC

- necessidade de se utilizar boas práticas de modelagem para o desenvolvimento dos componentes e o relacionamento entre eles;
- a utilização nos diversos níveis de abstração, pode auxiliar no processo;
- servem para documentar componentes;

Tecnologias envolvidas

Engenharia de Software e DBC

- **mudança de paradigma de desenvolvimento** de aplicações centralizadas para o desenvolvimento de aplicações distribuídas, múltiplas-camada, sobre internet ou intranet;
- ênfase na **reutilização de componentes** já desenvolvidos;
- **composição de aplicações** a partir de partes bem identificáveis;
- ênfase no desenvolvimento de componentes para uma família de aplicações.

DBC – Tópicos de Pesquisa

- Como efetivamente armazenar e encontrar componentes adequados?
- Como facilitar o entendimento de componentes?
- Como garantir a qualidade dos componentes?

Repositório de Componentes

- Base preparada para o armazenamento e recuperação de componentes;
- Recuperação efetiva: informações adicionais relativas ao componente;
- Tipos de repositório:
 - Repositórios locais;
 - Repositórios específicos a um domínio.

Busca de Componentes

A chance que um desenvolvedor tem de reutilizar um dado componente, ao invés de desenvolver um novo, depende da disponibilidade do componente em um repositório, de forma que este possa ser facilmente localizado e recuperado.

- **Repositórios ativos**
- **Uso de mecanismos de busca inteligentes**

Compreensão de Componentes

- Necessidade de uma documentação adequada
- Possibilidade de utilização de recursos multimídia, navegação, adaptação de acordo com o perfil do usuário

Qualidade de Componentes

- Ainda não há um consenso na literatura;
- Aspectos não funcionais de um componente, tais como segurança, desempenho e confiabilidade (Han, 2000);
- Propriedades para a avaliação da qualidade de um componente (Sametinger, 1997):
 - uso de guidelines;
 - realização de testes, baseados em algum padrão de teste;
 - clareza conceitual, definição precisa para o grau de acoplamento (nível de conexão entre os componentes) e coesão (Nível de dependência).

Qualidade de Componentes

- **Níveis de certificação** de componentes (Sametinger, 1997)
- **Nível 1:** O componente é descrito por palavras-chave e é armazenado para recuperação automática. Nenhum teste é realizado e o nível de completude (completo) é desconhecido (caixa preta).
- **Nível 2:** O componente de código é compilado. Métricas são definidas e utilizadas para uma linguagem em particular
- **Nível 3:** São realizados testes, dados e resultados dos testes são adicionados aos componentes
- **Nível 4:** Um manual de reutilização é adicionado

Padrões de Software

- Os padrões permitem que soluções de software previamente testadas sejam reutilizadas, permitindo o desenvolvimento de aplicações flexíveis, elegantes e reutilizáveis. Eles capturam a experiência dos projetistas, permitindo que a mesma seja repassada a outros, aumentando o grau de reutilização de uma aplicação e atuando como uma solução para um problema que ocorre repetidamente em um determinado domínio [BUS 96].

Padrões de Software – Segundo Gamma

01/04

- 1. nome do padrão e classificação:** o nome do padrão expressa a essência do padrão sucintamente. Um bom nome é vital, porque ele se tornará parte do vocabulário de seu projeto;
- 2. intenção:** uma curta declaração responde as seguintes questões. O que faz o padrão do projeto ? Qual é a fundamentação e a intenção ? Qual é a característica específica do projeto ou problema endereçado ?;
- 3. também conhecido como:** outros nomes bem conhecidos para o padrão, se houver;

Padrões de Software – Segundo Gamma

02/04

4. motivação: um cenário que ilustra um problema de projeto e como as classes e estruturas de objetos no padrão resolvem o problema. O cenário ajuda a entender as descrições mais abstratas do padrão que segue;

5. aplicabilidade: quais são as situações nas quais o padrão do projeto pode ser aplicado? Quais são os exemplos de projetos inferiores que o padrão pode endereçar? Como podemos reconhecer essas situações?;

Padrões de Software – Segundo Gamma

03/04

- 6. estrutura:** uma representação gráfica de classes no padrão usando uma notação baseado em técnicas de orientação a objeto. Diagramas de interação para ilustrar seqüências de pedido e colaboração entre objetos também podem ser utilizadas;
- 7. participantes:** as classes e/ou objetos participantes no padrão e responsabilidades;
- 8. colaborações:** como o participante colabora para realizar suas responsabilidades;
- 9. consequências:** como o padrão suporta seus objetivos? Quais são os pontos de equilíbrio e resultados do uso do padrão? Qual o aspecto da estrutura de um sistema que pode variar independentemente?;

Padrões de Software – Segundo Gamma

04/04

10. implementação: quais os riscos, sugestões ou técnicas que deveriam estar explícitas quando implementar o padrão? Existem tópicos específicos dependentes de linguagem;

11. exemplos de código: fragmentos de código que ilustram como pode implementar o padrão em C++ ou Smalltalk;

12. usos conhecidos: exemplo do padrão encontrado em sistemas reais, incluídos pelo menos dois exemplos de diferentes domínios.

13. relato do padrão: qual padrão do projeto estão intimamente relacionados com este? Quais são as diferenças importantes? Com quais outros padrões este deveria ser usado?

Design Patterns descrito em [GAM95]

Contexto	Criação	Estrutura	Comportamento
Classe	<i>Factory Method</i>	<i>Adapter</i> (classe)	<i>Interpreter</i> <i>Template Method</i>
Objeto	<i>Abstract Factory</i> <i>Builder</i> <i>Prototype</i> <i>Singleton</i>	<i>Adapter</i> (objeto) <i>Bridge</i> <i>Composite</i> <i>Decorator</i> <i>Facade</i> <i>Flyweight</i> <i>Proxy</i>	<i>Chain of Responsibility</i> <i>Command</i> <i>Iterator</i> <i>Mediator</i> <i>Memento</i> <i>Observer</i> <i>State</i> <i>Strategy</i> <i>Visitor</i>

Construção de *Frameworks*

Frameworks - Definição

01/02

- Um *framework* orientado a objeto é frequentemente caracterizado como um conjunto de classes concretas e abstratas que colaboram entre si para fornecer o esqueleto de uma implementação para uma aplicação. Catalysis [D'SO 99] propõe um conceito mais abrangente de *frameworks*.
- *Frameworks* são representados por uma forma genérica de pacotes chamada *framework* de modelo (*Model Framework*) ou pacotes de modelos (*template package*).

Frameworks - Definição

02/02

- *Frameworks* são projetados em mais alto nível, definindo a arquitetura genérica de uma aplicação para ser importada com substituições e inclusões, gerando assim aplicações específicas.

Biblioteca de Classes X Frameworks OO

Bibliotecas de Classes	Frameworks
conjunto de classes instanciadas pelos desenvolvedores	fornecem adaptações para subclasses, ou seja classes que podem ser estendidas através da herança
acesso a funções pré-definidas	estende funções pré-definidas
nenhum fluxo de controle pré-definido	fluxo de controle de execução onde é definido o interrelacionamento das classes e as regras de como deve ser a sequência de execução e suas dependências
nenhuma interação pré-definida	define interações entre objetos (interrelacionamento entre as classes)
nenhum comportamento padrão	fornecer comportamento padrão

Categorias de Framework Horizontal e Framework Vertical

- Segundo [ROG 97] existem duas categorias de *frameworks*:
 - **framework horizontal**: é mais abrangente pois pode ser usado por mais tipos de aplicações. ToolKits GUI são exemplos de *frameworks* horizontais, pois são *frameworks* que podem ser usados para construir interfaces de usuários para uma grande faixa de aplicações.
 - **framework vertical**: são mais específicos para um domínio particular. Ele não é usualmente utilizado em domínios diferentes. Por exemplo, um *framework* para executar análise estatística de dados econômicos é específico para aplicações financeiras.

Modelos de Frameworks

- **MVC** (Modelo-Visão-Controlador);
- **MFC** (Microsoft Foundation Class 1994);
- **ET++**[WEI88];
- **INTERVIEWS**[VLI92];
- **Frameworks de modelos** [D' SO99]

Abordagens / Métodos para desenvolvimento de frameworks

- Projeto dirigido por “*Exemplo*” [JOH96];
- Projeto dirigido por “*Hot Spot*” [PRE95];
- Método “*Taligent*” [TAL96];
- Abordagem “*Catalysis*” [D’S99].

Projeto dirigido por “*Exemplo*” [JOH96]

- O processo apontado como ideal para o desenvolvimento de *frameworks*, segundo o projeto dirigido por exemplo, consiste das seguintes etapas:

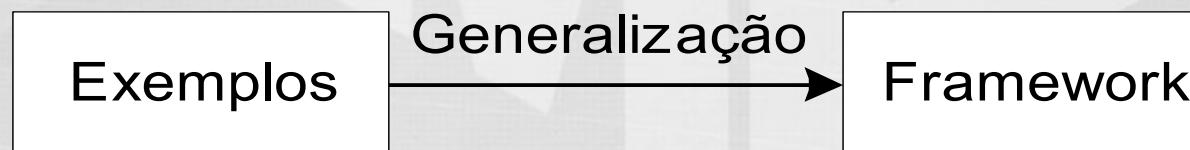
1. Analisar o domínio do problema

- analisar as abstrações já conhecidas;
- coletar exemplos de programas que poderiam ser construídos a partir do *framework* (mínimo 4 ou 5).

2. Generalização: projetar uma abstração que cubra os exemplos.

3. Testar o framework utilizando-o para resolver os exemplos

- cada exemplo é uma aplicação separada;
- realizar o teste significa escrever código.



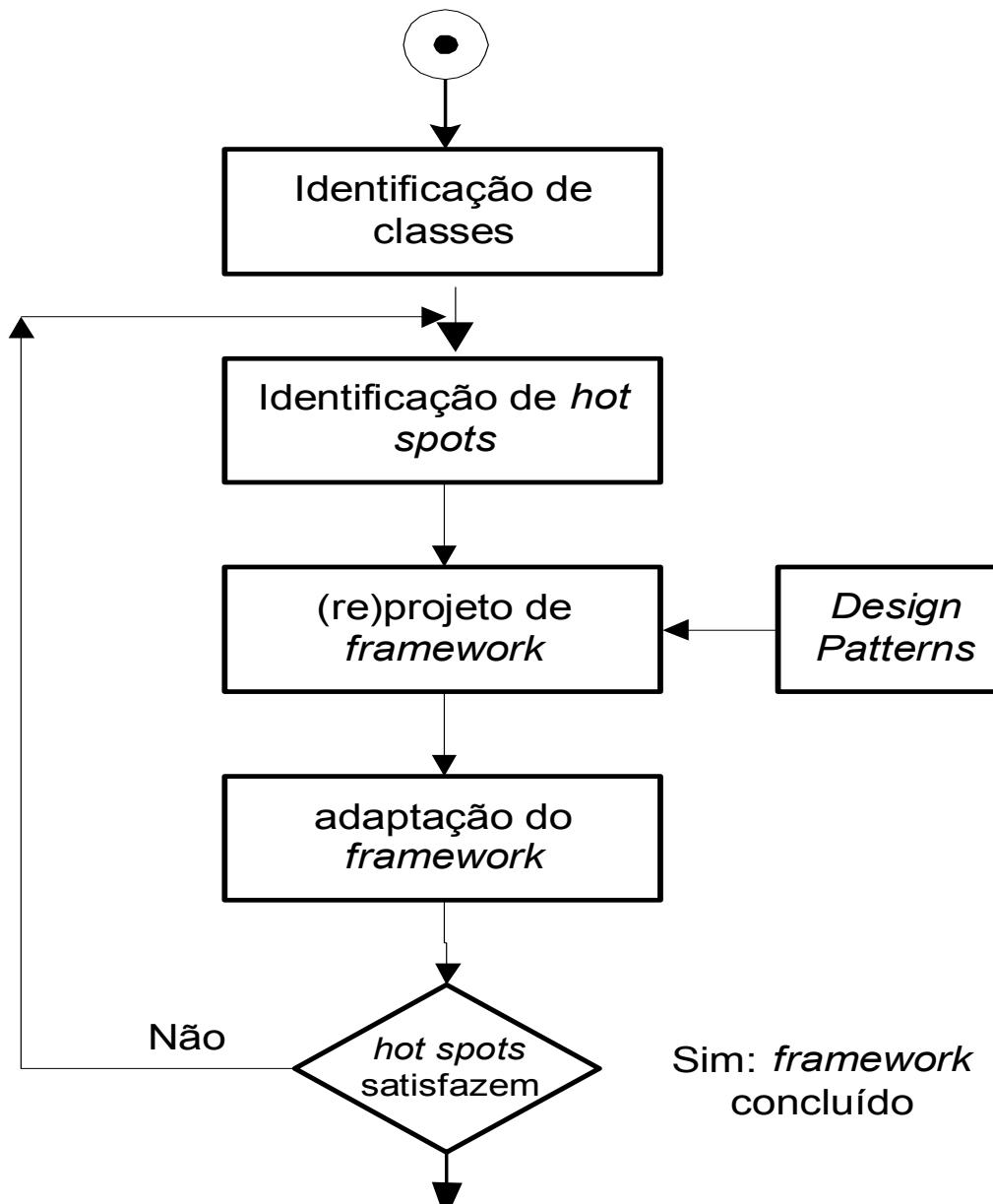
Projeto dirigido por “*Host Spot*” [PRE95]

01/02

- Um *framework* possui partes que propositadamente não são definidas. Isto o torna flexível, capaz de se moldar a diferentes aplicações. Os *Hot Spots* são as partes do *framework* mantidas flexíveis. A essência da metodologia é identificar os *hot spots* na estrutura de classes de um domínio e a partir disto construir o *framework*

Projeto dirigido por “Host Spot” [PRE95]

02/02



1. Identificar e caracterizar o domínio do problema

- analisar o domínio e identificar os *frameworks* necessários (para o desenvolvimento de aplicações), o que pode incluir *frameworks* desenvolvidos e a desenvolver;
- examinar soluções existentes;
- identificar as abstrações principais;
- identificar o limite de responsabilidade do *framework* em desenvolvimento;
- validar estas informações com especialistas do domínio.

2. Definir a Arquitetura e o Projeto

- refinar a estrutura de classes obtida no passo anterior, centrando atenção em como os usuários interagem com o *framework*:
 - que classes o usuário instancia?
 - que classes o usuário deriva (e que métodos sobrepõe)?
 - que métodos o usuário chama?
- aperfeiçoar o projeto com o uso de padrões;
- validar estas informações com especialistas do domínio.

3. Implementar o *framework*

- implementar as classes;
- testar o *Framework*;
- solicitar a terceiros o procedimento de teste;
- iterar para refinar o projeto.

4. Utilizando o *framework*

- fornecer documentação na forma de diagramas, receitas;
- manter e atualizar o *framework*, seguindo as seguintes regras:
 - corrigir erros imediatamente;
 - adicionar novas características ocasionalmente;
 - mudar interfaces tão infrequentemente quanto possível.

Método “*Catalysis*” [D’ S99]

- Utilização da notação UML;
- Processo de desenvolvimento bem definido;
- Desenvolvimento baseado em componentes e Frameworks;
- Reutilização de padrões e extensibilidade dos pacotes;
- Frameworks de Modelos (Model Frameworks).

Abordagem *Catalysis*

- Segundo Catalysis, não somente partes de código, podem ser reutilizáveis, projetos e especificações, também podem ser separadas em partes, as quais podem ser mantidas em bibliotecas e posteriormente combinados em diferentes configurações, como é o caso do *frameworks* de modelos.

Avaliação Geral (Métodos)

01/02

- aquisição de conhecimento do domínio;
- definição da estrutura de classes do *framework* (modelagem de *framework*);
- implementação do *framework*;
- avaliação do *framework*, através do desenvolvimento de aplicações;
- refinamento do *framework* a partir da aquisição de novos conhecimentos do domínio.

Avaliação Geral (Métodos)

02/02

O processo de desenvolvimento de *frameworks* proposto na abordagem Telligent enfatiza a idéia de construir pequenos *frameworks*, porém não é apoiado por uma notação consistente. Assim, o Catalysis é mais completo e eficiente, apoiado pela UML.

Exemplo de um estudo de caso no desenvolvimento de um Framework utilizando o Catalysis

Motivação

- Devido a semelhança entre WfMSs e os PSEEs, foi possível analisar a aplicação do padrão Process Manager em gerenciadores de *workflow*;
- Esta análise foi tomada como base para refinar o padrão e chegar a *Frameworks* que sejam reutilizáveis tanto em WfMSs como em PSEEs

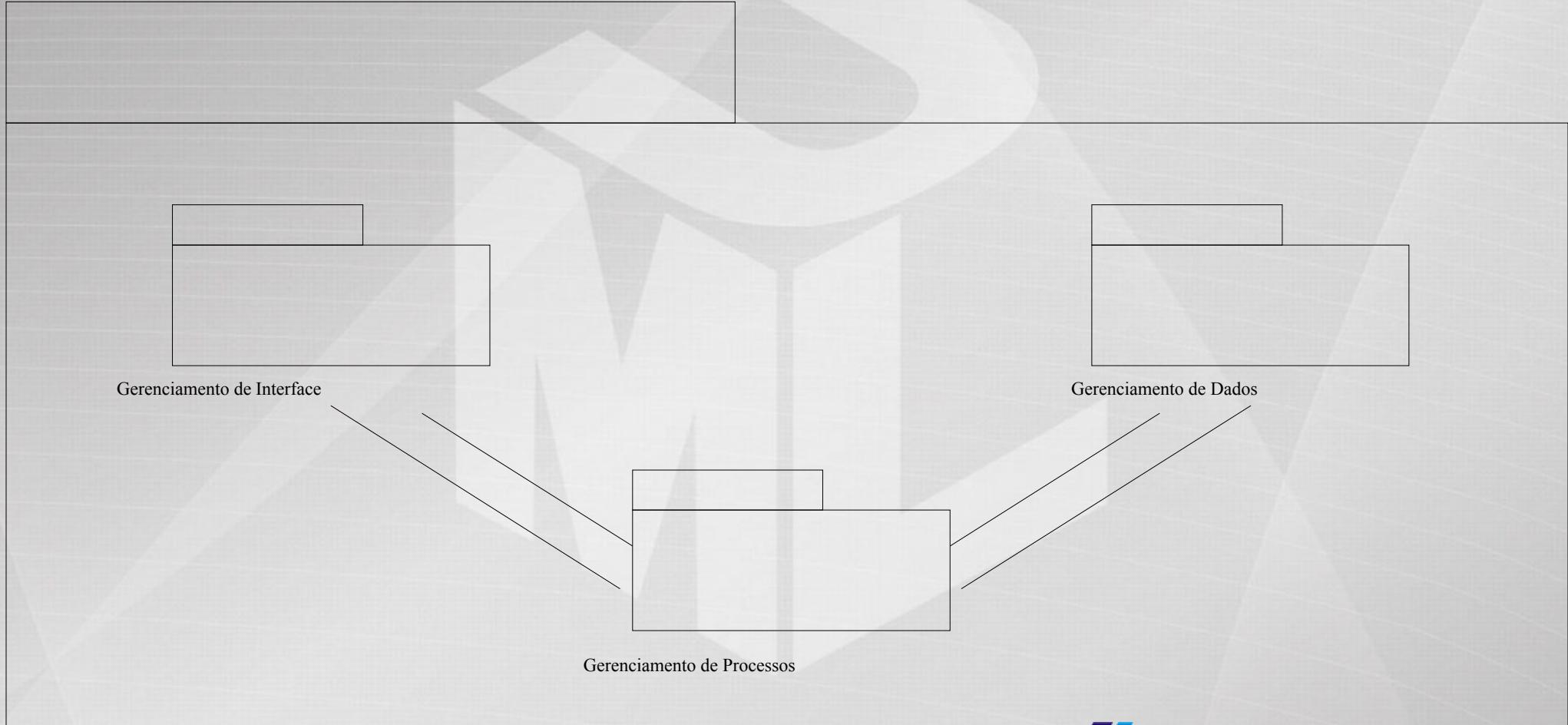
Objetivo do Trabalho

- Desenvolvimento de um *Framework* de agenda de tarefas para gerenciadores de processos baseado no padrão de gerenciadores de processos de software definido no projeto ExPSSE.

O Projeto ExPSEE (Gim99b)

- *ExPSEE - ambiente de engenharia de software orientado a processos que visa a investigação de tecnologias de processos de software e mecanismos para sua modelagem e automação;*

Construção do Padrão *Process Manager*

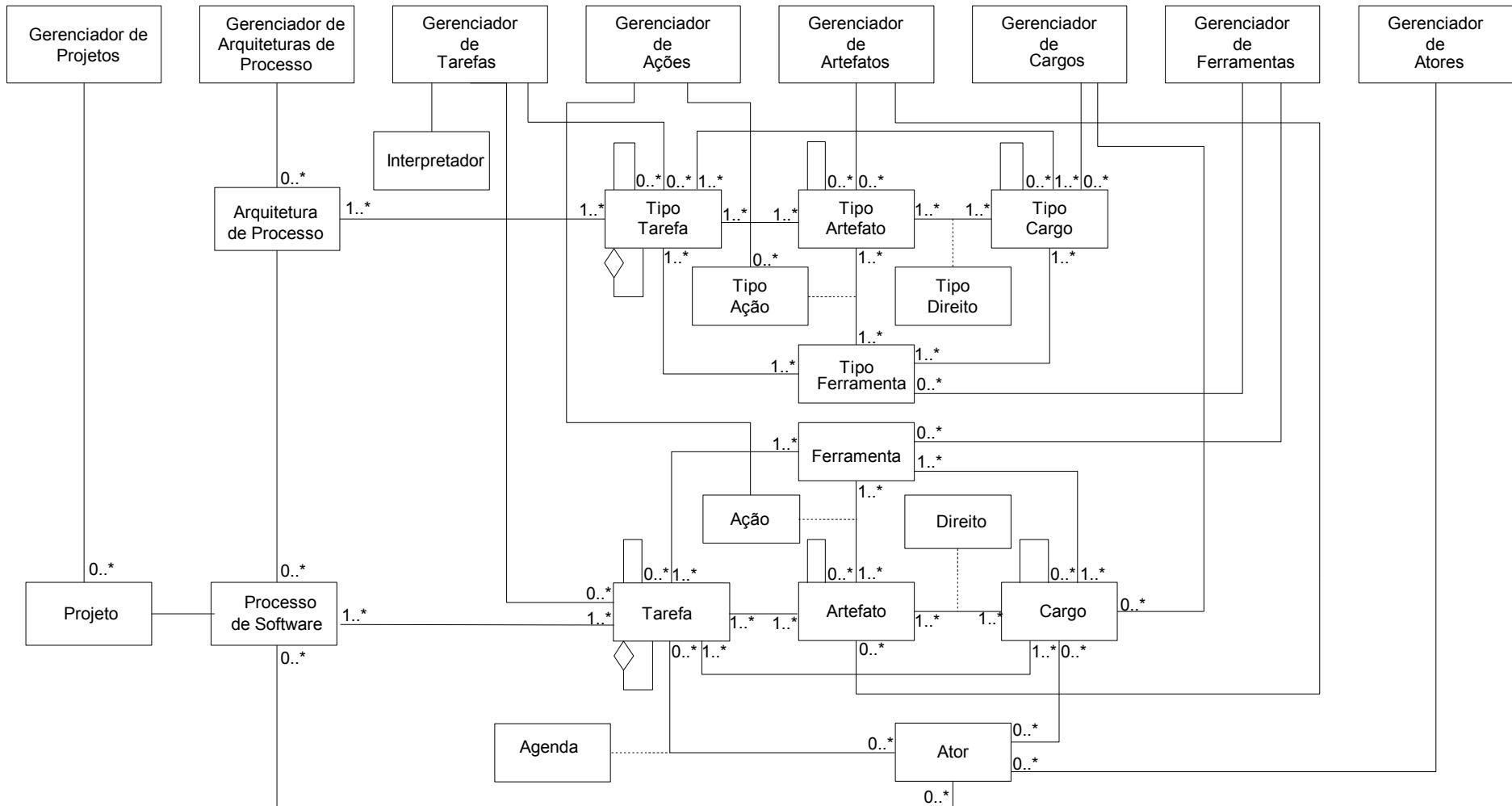


Ambiente de Engenharia de Software

UNIVEL

AUDARE
Engenharia de Software

Construção do Padrão *Process Manager*



Estrutura

- O gerenciador de processos é composto de módulos gerenciadores, responsáveis por registrar e manter a relação dos objetos existentes no ambiente. Os módulos são:
 - **gerenciador de arquitetura de processos** responsável pela definição de uma arquitetura de processo e os tipos relacionados a esta arquitetura;
 - **gerenciador de projetos** responsável pela definição e execução dos processos de software;
 - **gerenciador de ações** responsável pelo gerenciamento das funcionalidades que a ferramenta tem para que possa ser aplicada sobre um determinado artefato.
 - **gerenciador de tarefas** responsável pelo controle e gerenciamento das tarefas e ações a serem realizadas no processo de software;

Estrutura

- **gerenciador de artefatos** responsável pelo controle e gerenciamento dos artefatos utilizados e produzidos pelas tarefas através das ferramentas;
- **gerenciador de ferramentas** responsável pelo controle e gerenciamento das ferramentas utilizadas pelas tarefas no processo de software;
- **gerenciador de cargos** responsável pelo controle e gerenciamento dos cargos existentes no processo de software a serem ocupados pelos atores presentes no processo;
- **gerenciador de atores** responsável pelo controle e gerenciamento dos atores, e suas agendas, envolvidos no processo de software;
- **interpretador** responsável por executar os comandos da linguagem de programação de processos relacionados a cada tarefa.

Workflow

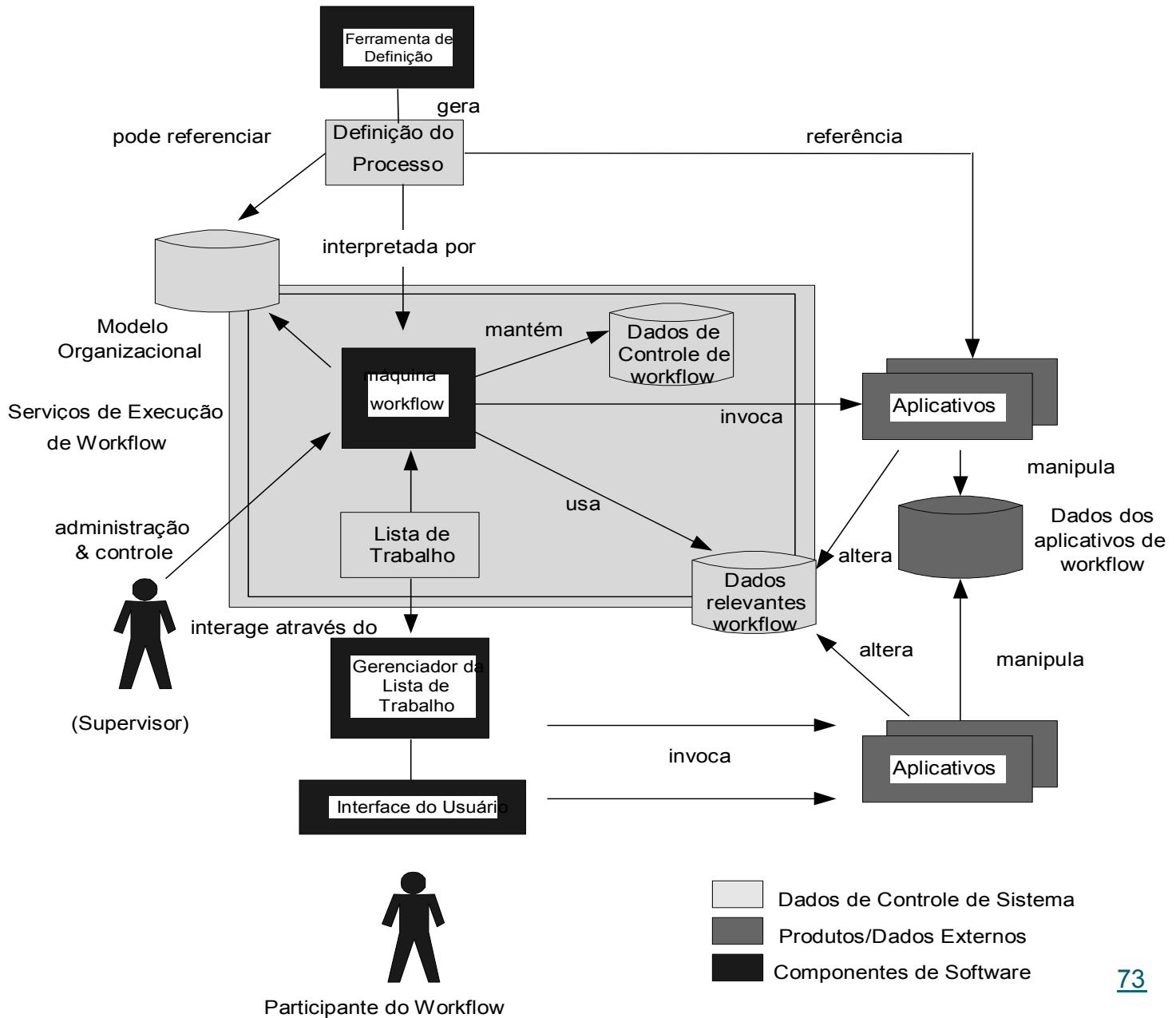
01/02

- Um *workflow* estabelece a ordem de execução das atividades e as condições em que cada atividade pode ser iniciada, assim como a sincronização das atividades e o fluxo de informações.
- WfMSs são sistemas que permitem a definição, a criação e o gerenciamento da execução de *workflows*.

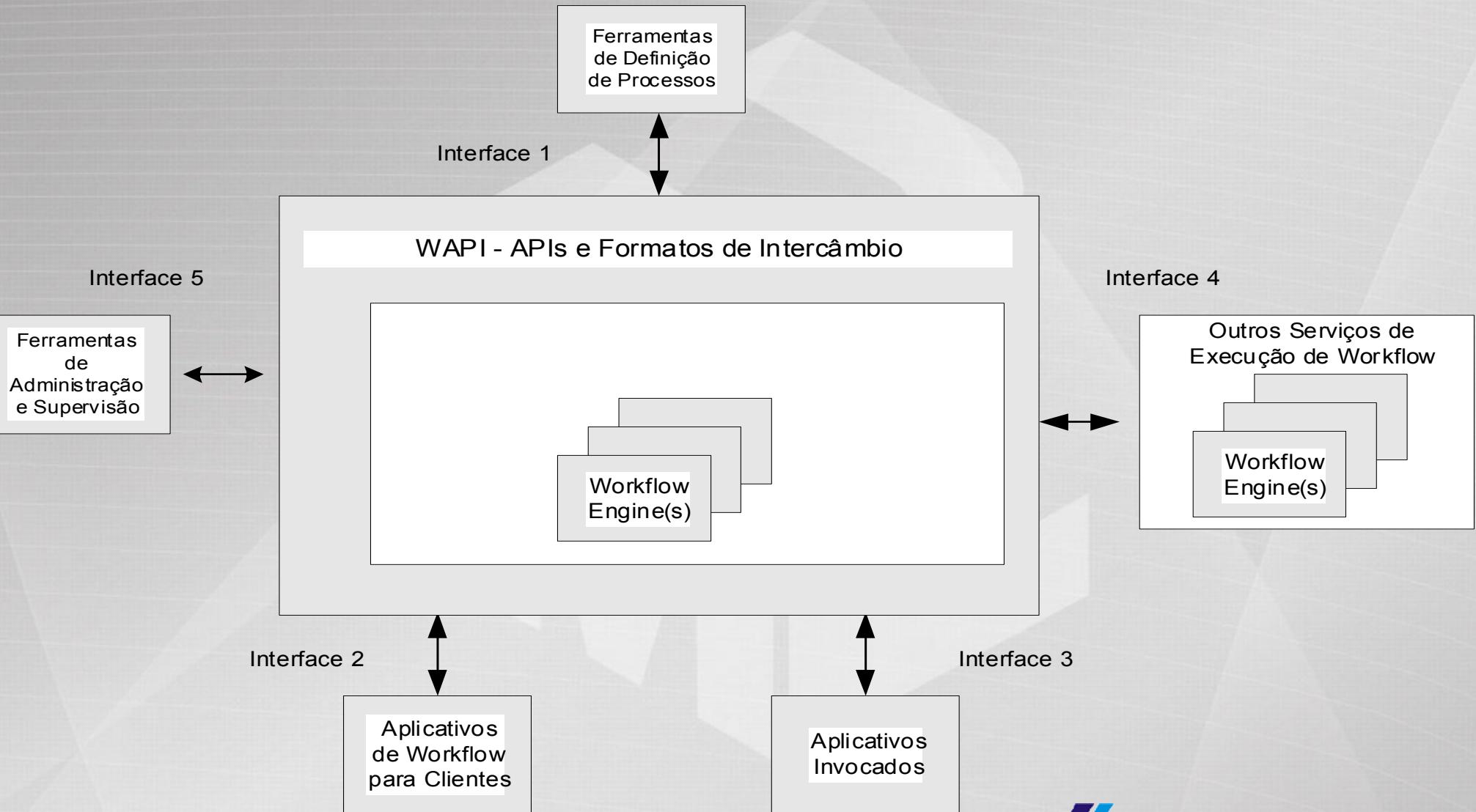
Workflow

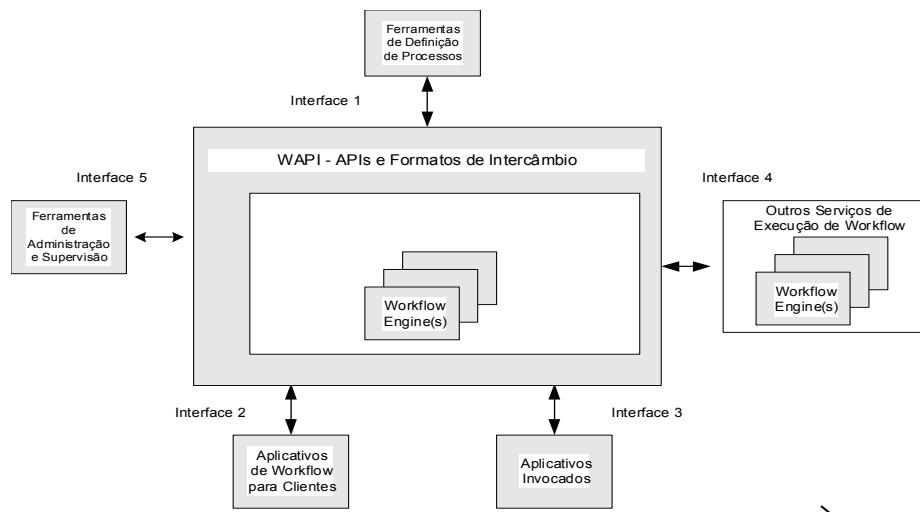


Arquitetura de um Sistema de Workflow

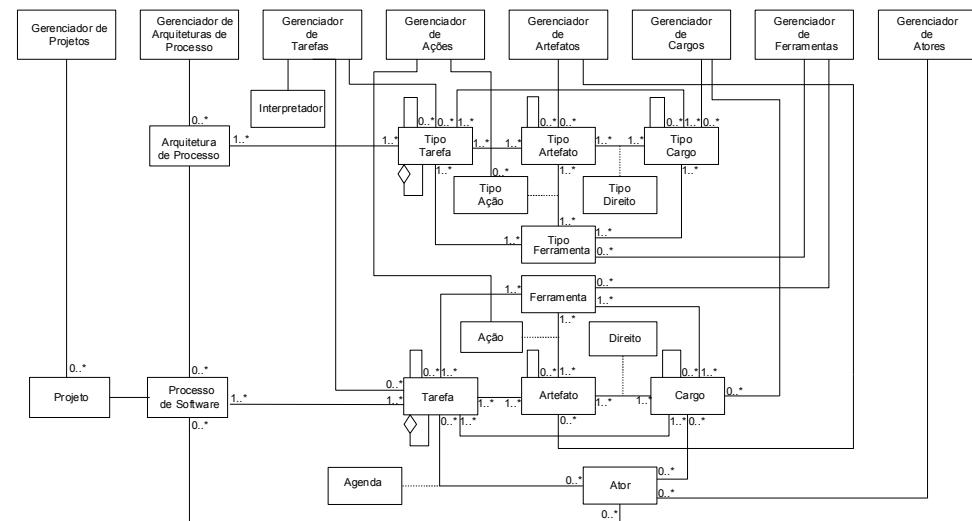
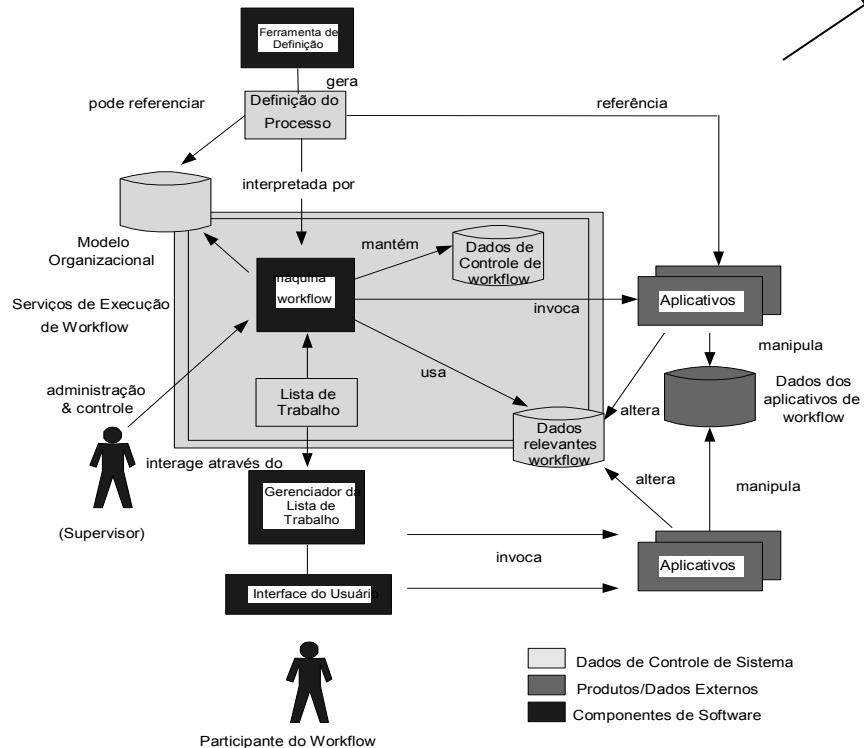


Modelo de Referência da WfMC





VERSUS



Comparação entre os modelos da WfMC e o padrão *Process Manager*

Modelos da WfMC		Gerenciador do Processo de Software
Interfaces	Elementos de arquitetura genérica	Elementos
WAPI	Máquina de <i>Workflow</i>	Gerenciador de Processo
Interface 1	Ferramentas de Definição	Gerenciador de Arquitetura de Processos
Interface 2	Gerenciador da Lista de Trabalho	Gerenciador de Tarefas
Interface 3	Dados de Controle do <i>Workflow</i> e Dados Relevantes do <i>Workflow</i>	Gerenciador de Artefatos
Interface 3	Aplicativos	Gerenciador de Ferramentas
Interface 4	Serviços de Execução de <i>Workflow</i>	Gerenciador de Ferramentas
Interface 5	Administração e Controle	Gerenciador de Atores
Interface 5	Modelo Organizacional	Gerenciador de Cargos

Definição do framework de agenda de tarefas

- Diagrama de Colaboração de Alto Nível;
- Diagrama de Camadas Verticais de Alto Nível;

Diagrama de Colaboração de Alto Nível

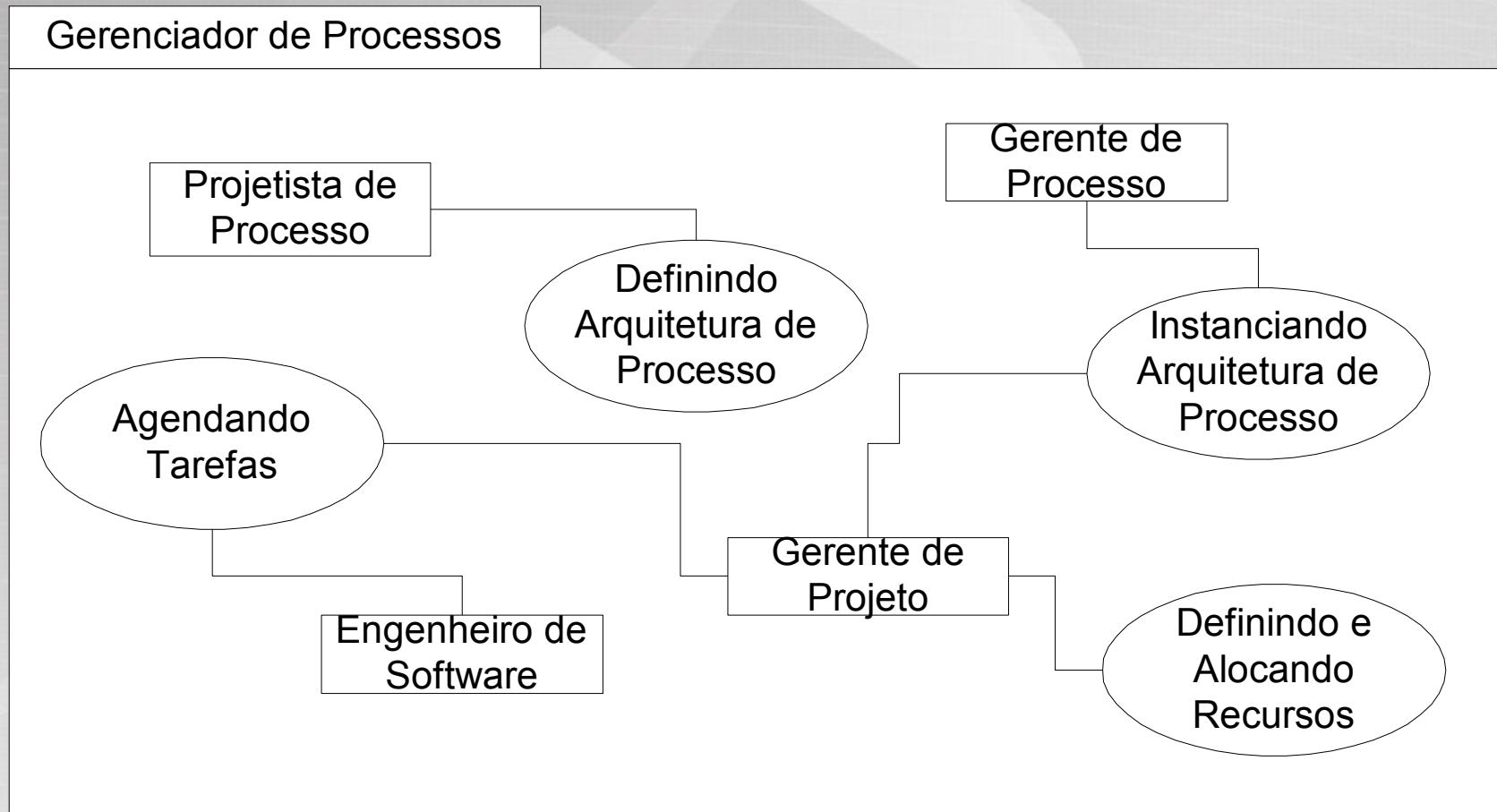
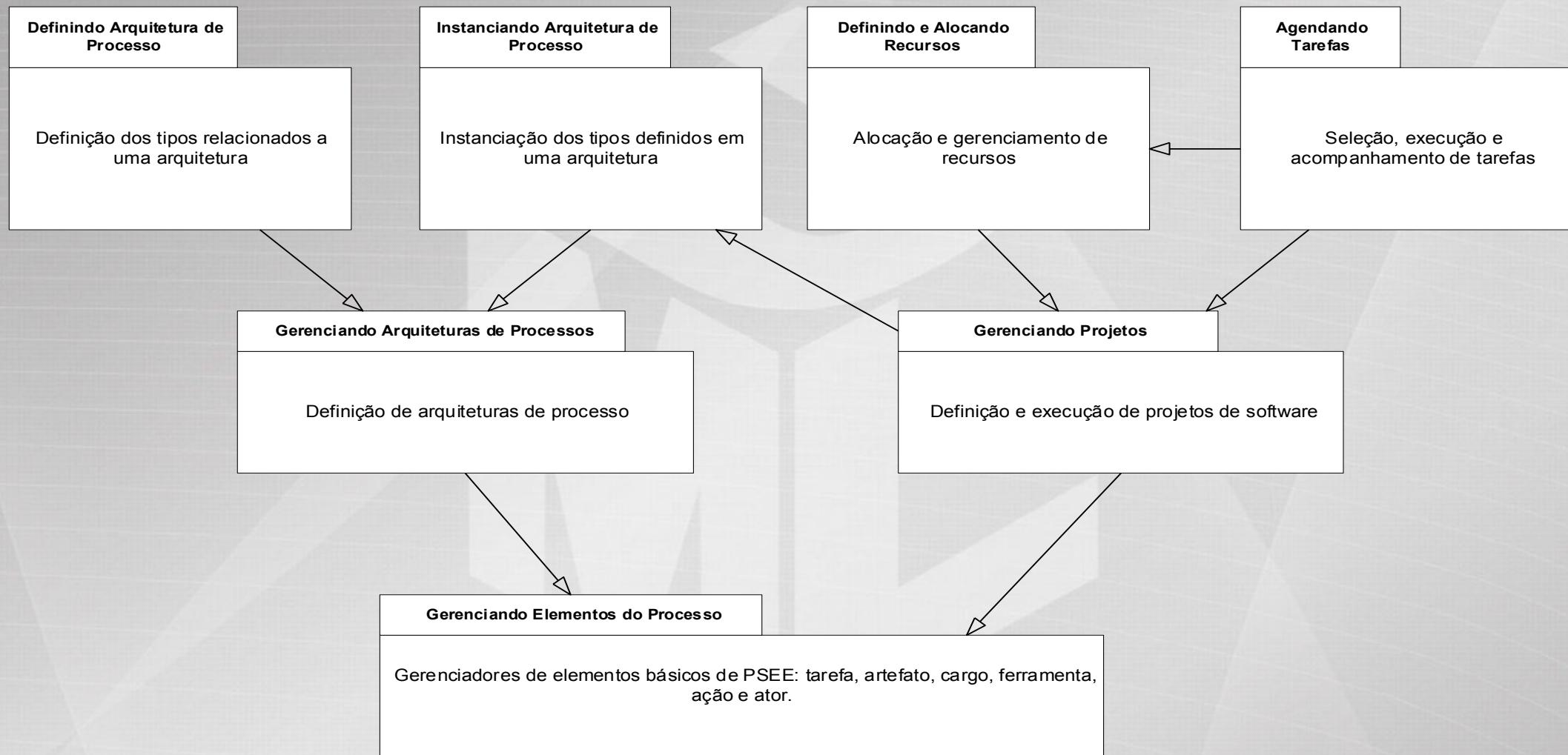
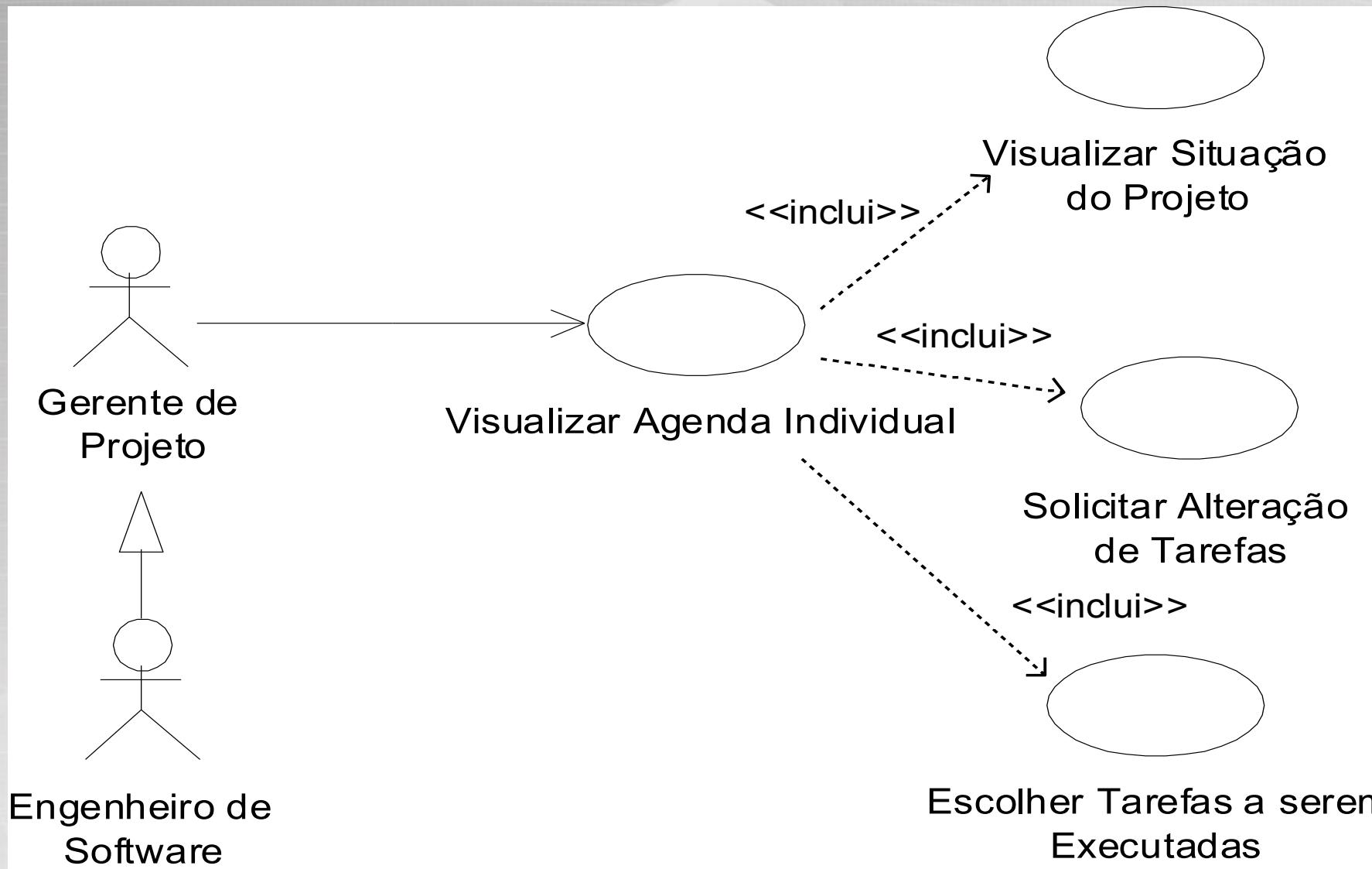


Diagrama de Camadas Verticais



Caso de Uso – Visualizar Agenda Individual de Tarefas



Interface Gráfica do *Framework* de Agenda de Tarefas do ExPSEE

 Visualizar tarefas agendadas

Entre com o agendamento :

Ator :	Pedro	Projeto :	Projeto 1
Cargo :	Cargo1		
Tarefa Pai :	Tarefa 1	Tarefa :	Tarefa 3
Data de Início :	17/10/1999	Data de Termino :	21/10/1999
Status :	em execucao	Andamento :	

Tarefas Agendadas :

Ator	Projeto	Cargo	Tarefa	ID_PAI_TA...	Data de Ini...
Pedro	Projeto 1	Cargo1	Tarefa 5	4	15/11/1999
Pedro	Projeto 1	Cargo1	Tarefa 4	0	13/11/1999
Pedro	Projeto 1	Cargo1	Tarefa 3	1	17/10/1999
Pedro	Projeto 1	Cargo1	Tarefa 2	1	15/10/1999
Pedro	Projeto 1	Cargo1	Tarefa 1	0	10/10/1999

Interface da Agenda de Tarefas Individual de um Determinado Ator

Agendar Tarefas

Entre com o agendamento :

Ator :

Cargo :

Projeto :

Tarefa :

Tarefa Pai :

Data de Inicio :

Data de Termíno :

Status :

Andamento : 0 25 50 75

Novo Cargo

Atores cadastrados :

- Pedro
- Ademir
- Simone
- Sergio
- Marcos
- Carlos
- Maria
- Adail
- Adriana

Digite os dados do novo ator :

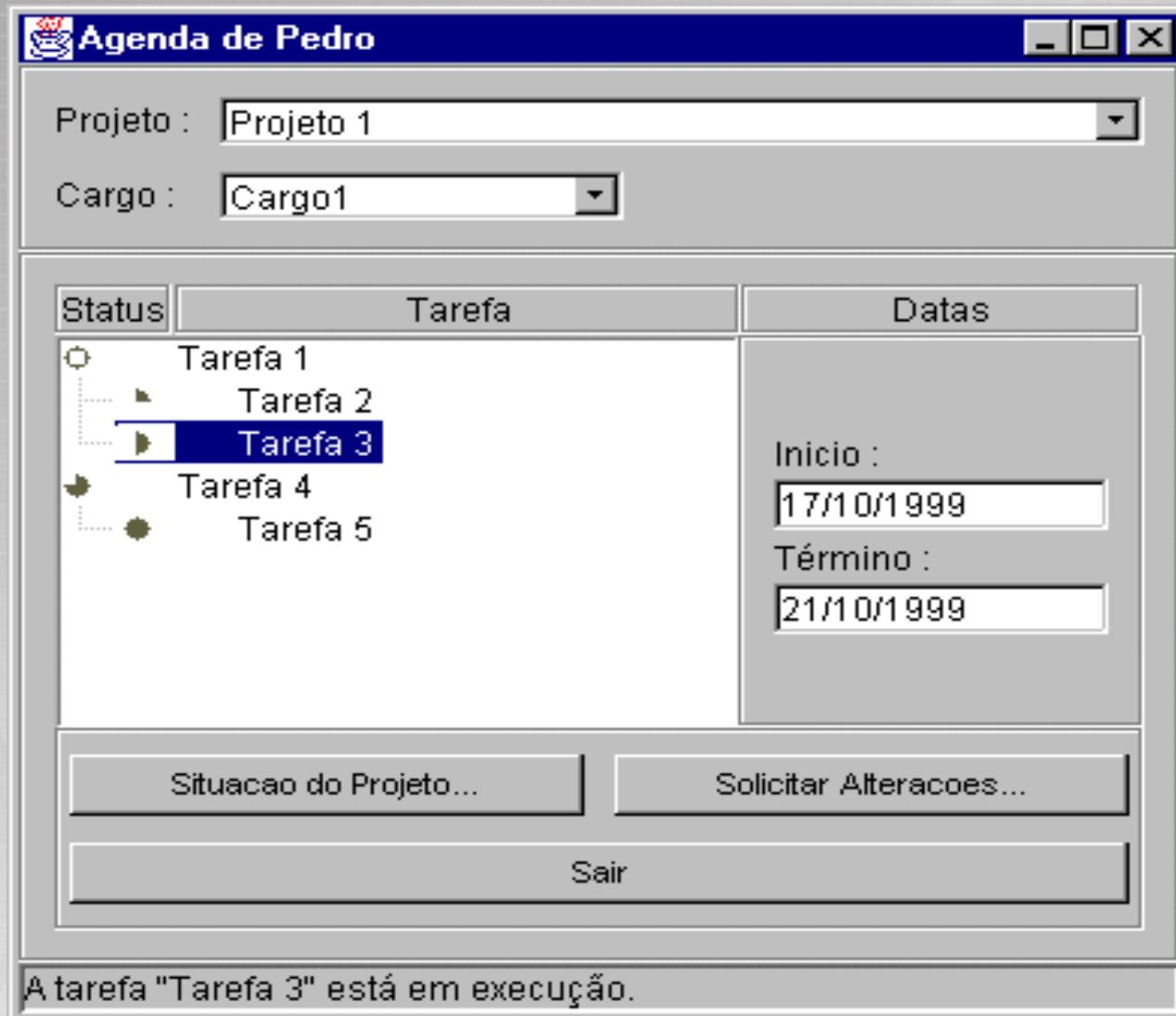
Nome :

Login :

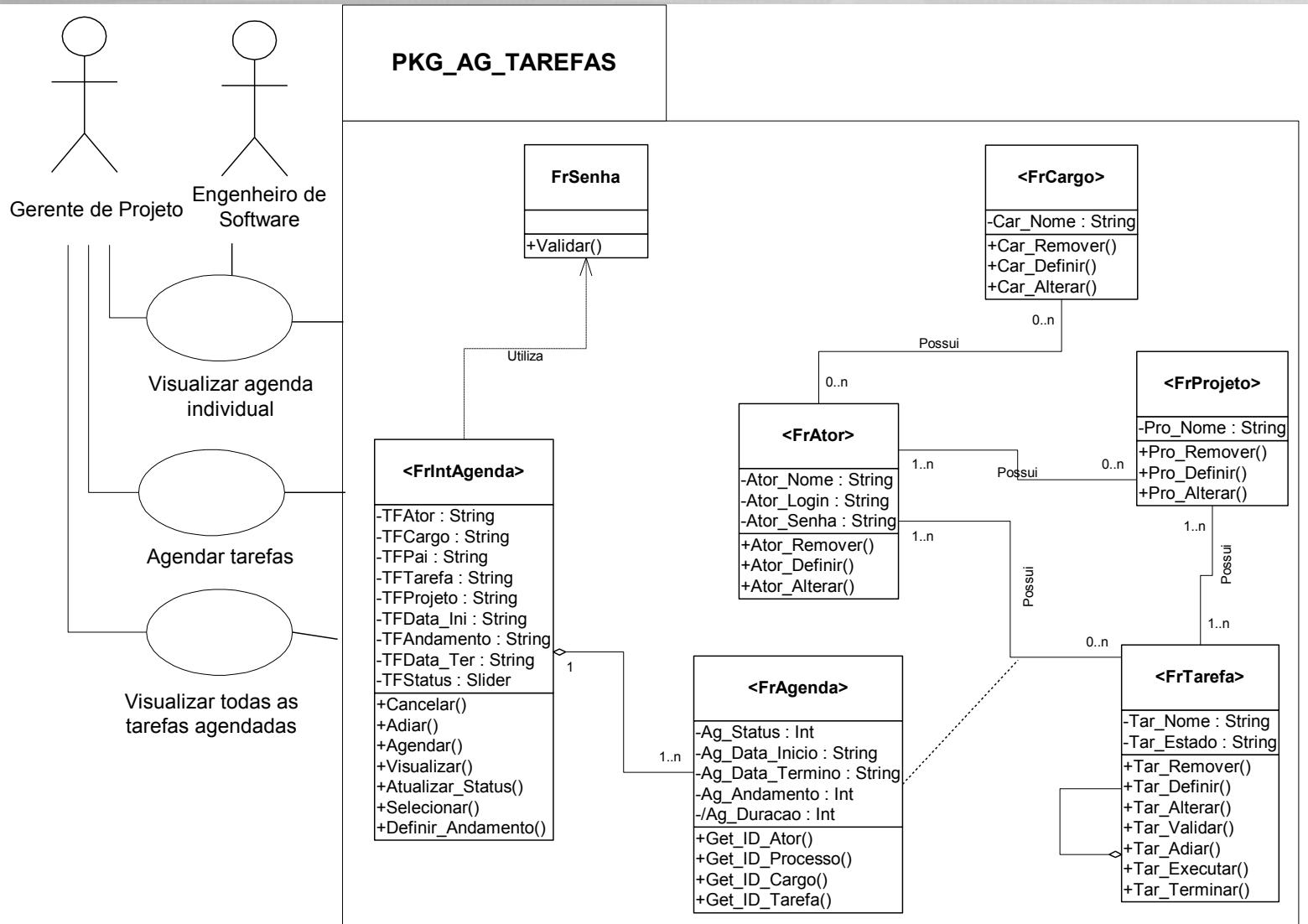
Senha :

Confirmacao :

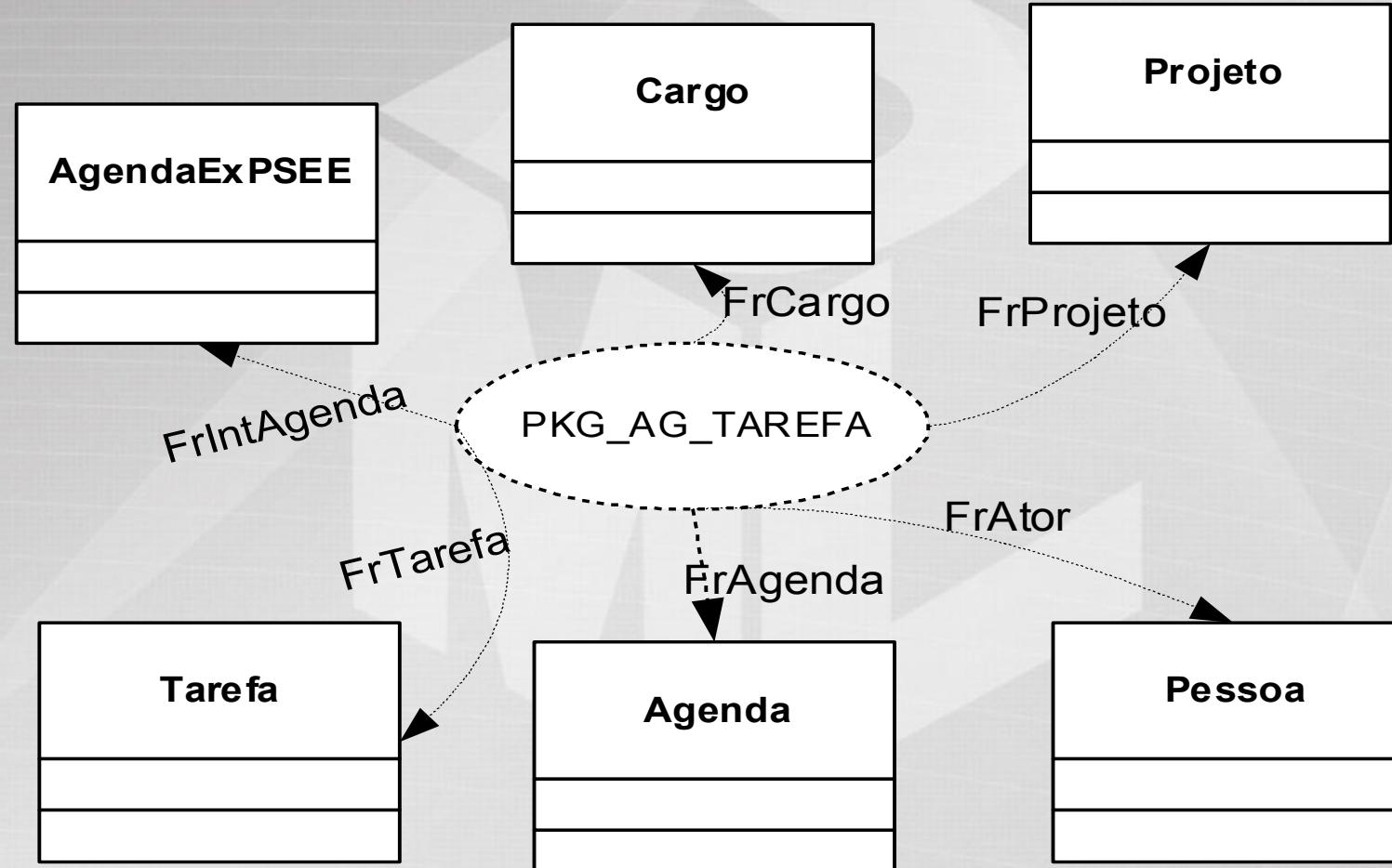
Interface da Agenda de Tarefas da Visualização das Tarefas no Ambiente ExPSEE.



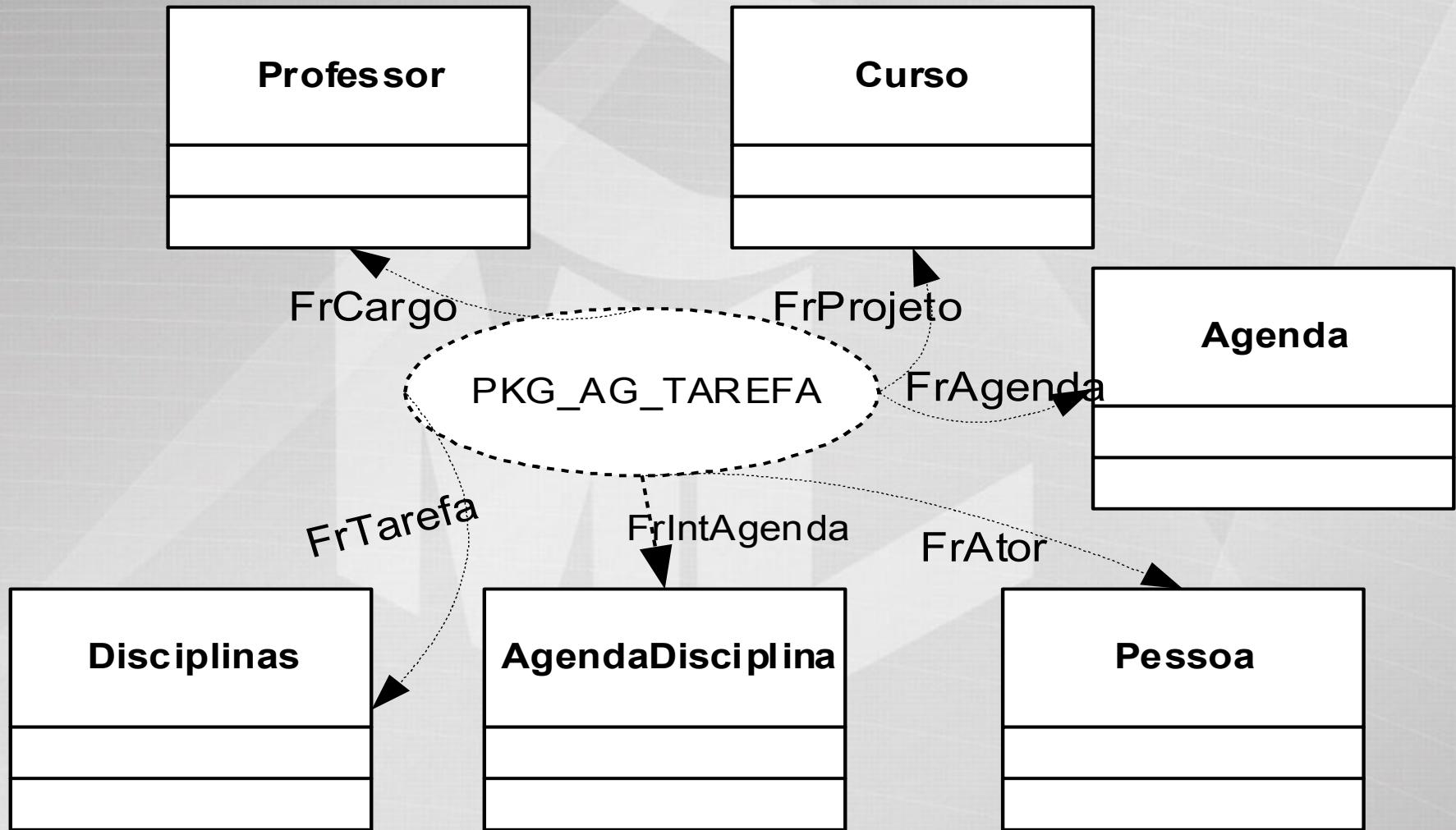
Framework de Modelo



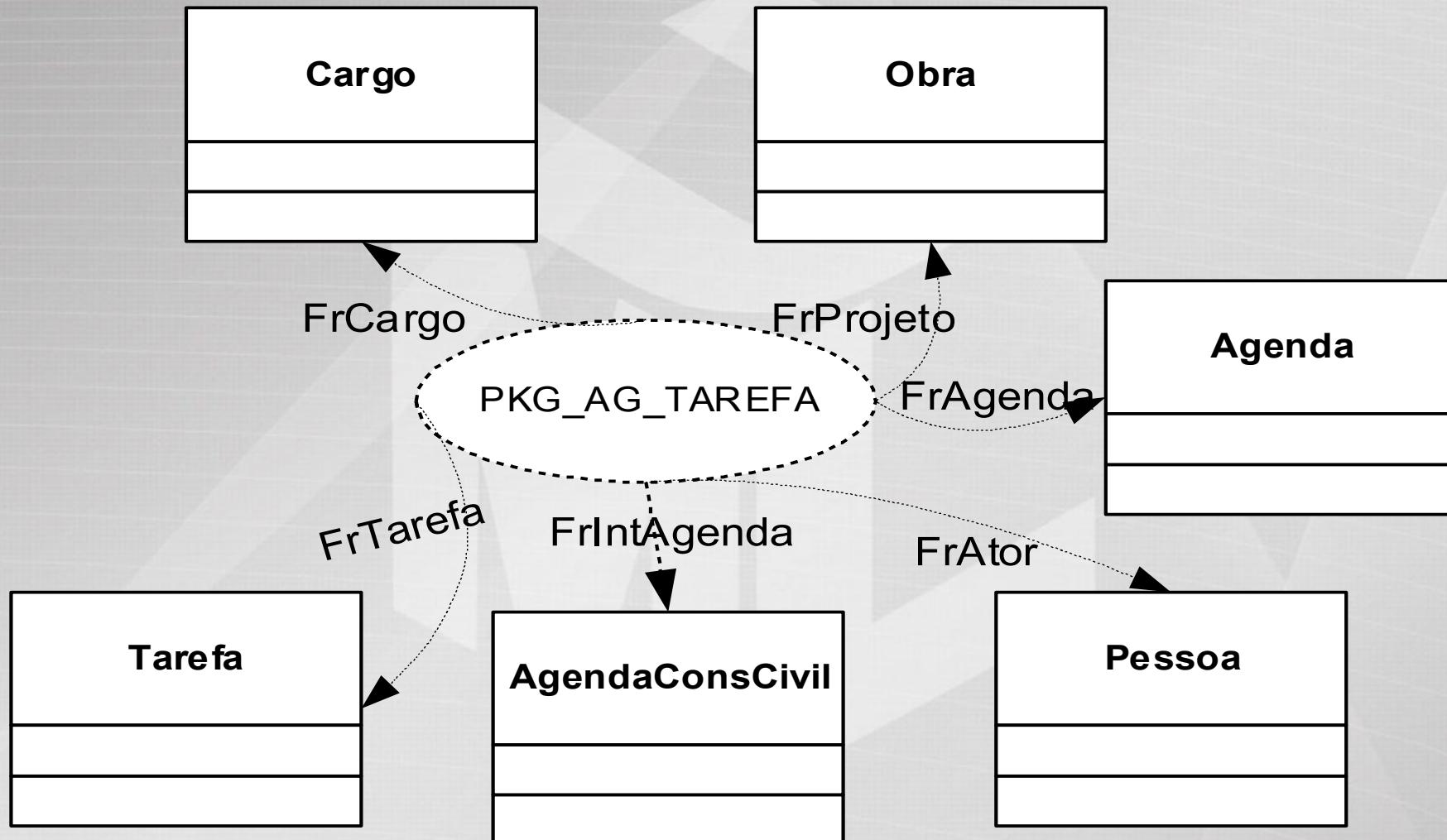
Aplicação do Framework de Agenda de Tarefas no ExPSEE



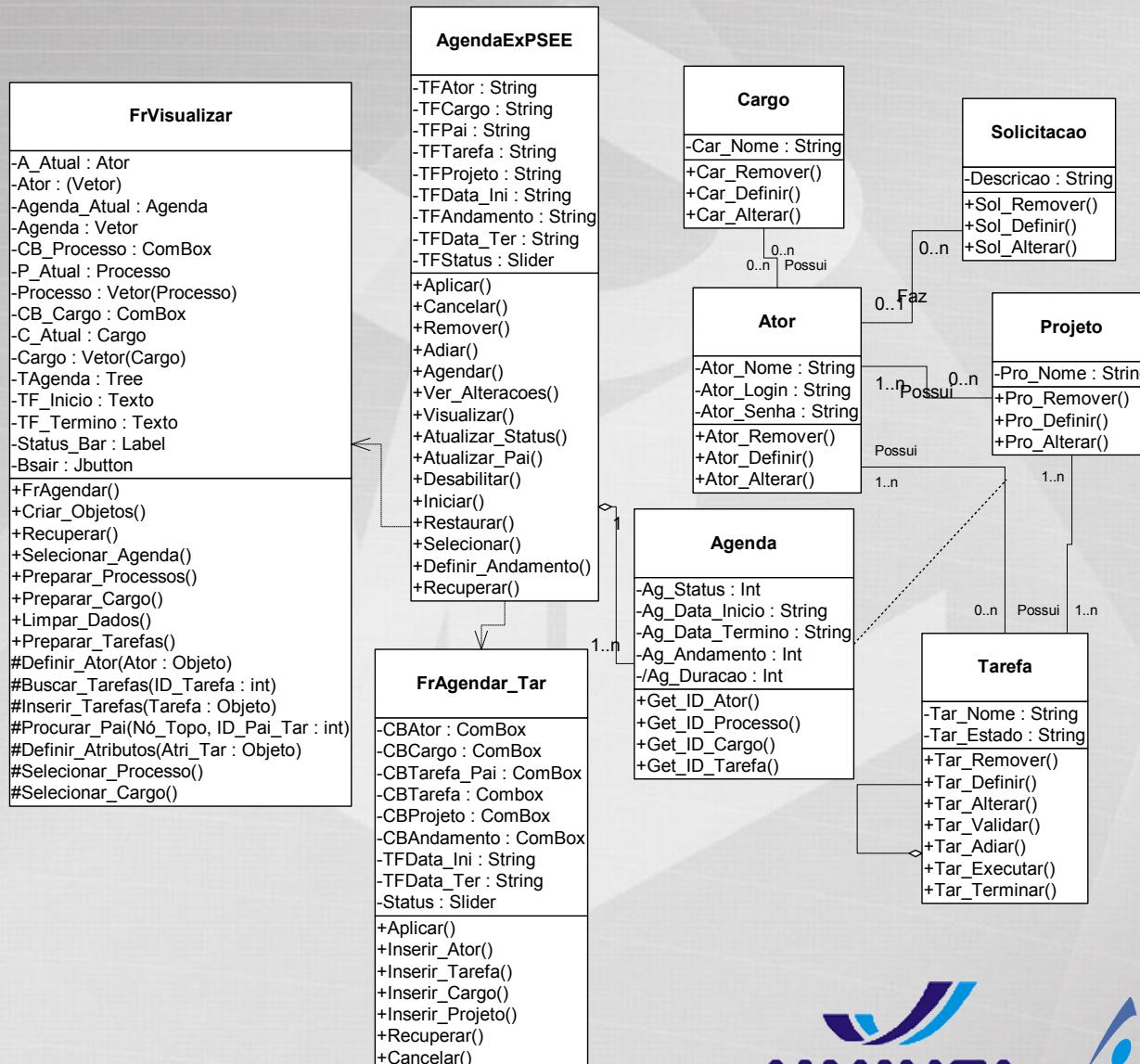
Aplicação do Framework de agenda do conteúdo aplicado a disciplinas



Aplicação do *framework* de agenda de obras de construção civil.



Desdobramento da aplicação do *framework* de agenda de tarefas do ExPSEE.



EXERCÍCIOS

- 1. Elaborar o Modelo de Framework basedo no estudo de caso;**
- 2. Analisar o Estudos de Caso e desenhar 3 (três) aplicações de frameworks;**
- 3. Documentar o Framework principal utilizando a documentação de Projeto do padrão do Gamma;**
- 4. Analisar os Modelos Arquiteturais do MDA: CIM, PIM e PSM em relação ao Estudo de Caso.**

MATERIAL COMPLEMENTAR

Modelo MDA (*Model Driven Architecture*) no Desenvolvimento Baseado em Componentes

Origem da MDA

- A OMG (*Object Management Group*), uma associação internacional sem fins lucrativos tem o seu foco voltado na integração de padrões, como a UML, CORBA, etc.
- Em 2001 a OMG adotou a *Model Driven Architecture* ou MDA, que foi definida como uma abordagem para o uso de modelos no desenvolvimento de software.

MDA – O que é o Modelo MDA ?

As três metas primordiais da MDA são portabilidade, interoperabilidade e reusabilidade.

A MDA fornece uma abordagem e habilita ferramentas a serem fornecidas para:

- especificar um sistema independente da plataforma que o suporta (Ex.: OCL, UML, SQL, etc.)
- especificar plataformas baseado em modelos (Ex.: Bold da Borland para Delphi, NakedObject para Java, ambos fazem o papel intermediário entre a modelagem e a implementação utilizando a abordagem MDA).
- escolher uma determinada plataforma (Arquitetura) para o sistema;
- transformar a especificação do sistema (ex.: Classes de Análise) em uma especificação de Projeto para uma determinada plataforma (ex.: J2EE, .NET, etc).

Modelos da MDA

A MDA trabalha com dois modelos:

- PIM (*Platform Independent Model*), o modelo PIM trata do domínio do problema com uma visão abstrata, sem uma plataforma específica de implementação. (UML, OCL)
- PSM (*Platform Specific Model*), após o término do PIM o mapeamento para uma determinada plataforma é feito pela elaboração do PSM.

Esses dois modelos definem dois estágios diferentes da elaboração do projeto de construção de um software.

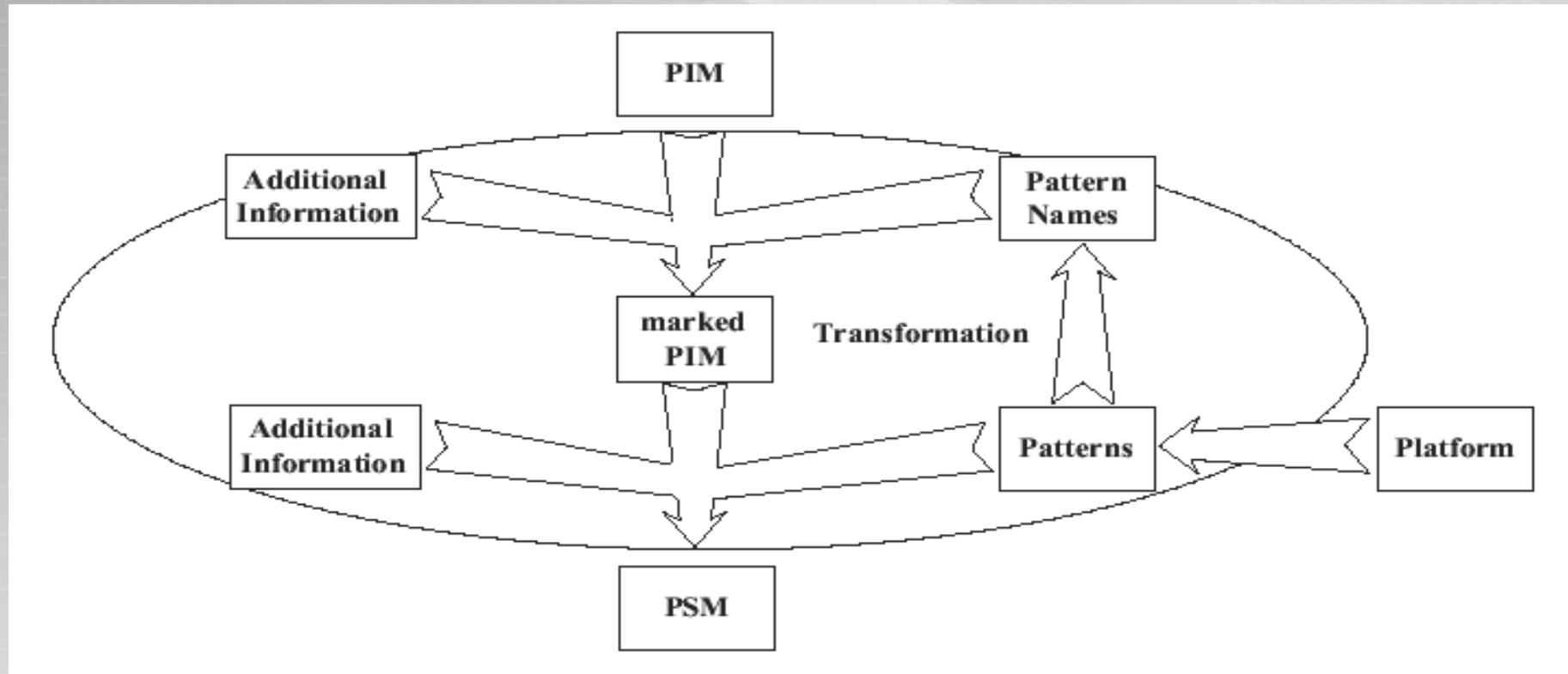
PIM para PSM

Transformação de Modelo

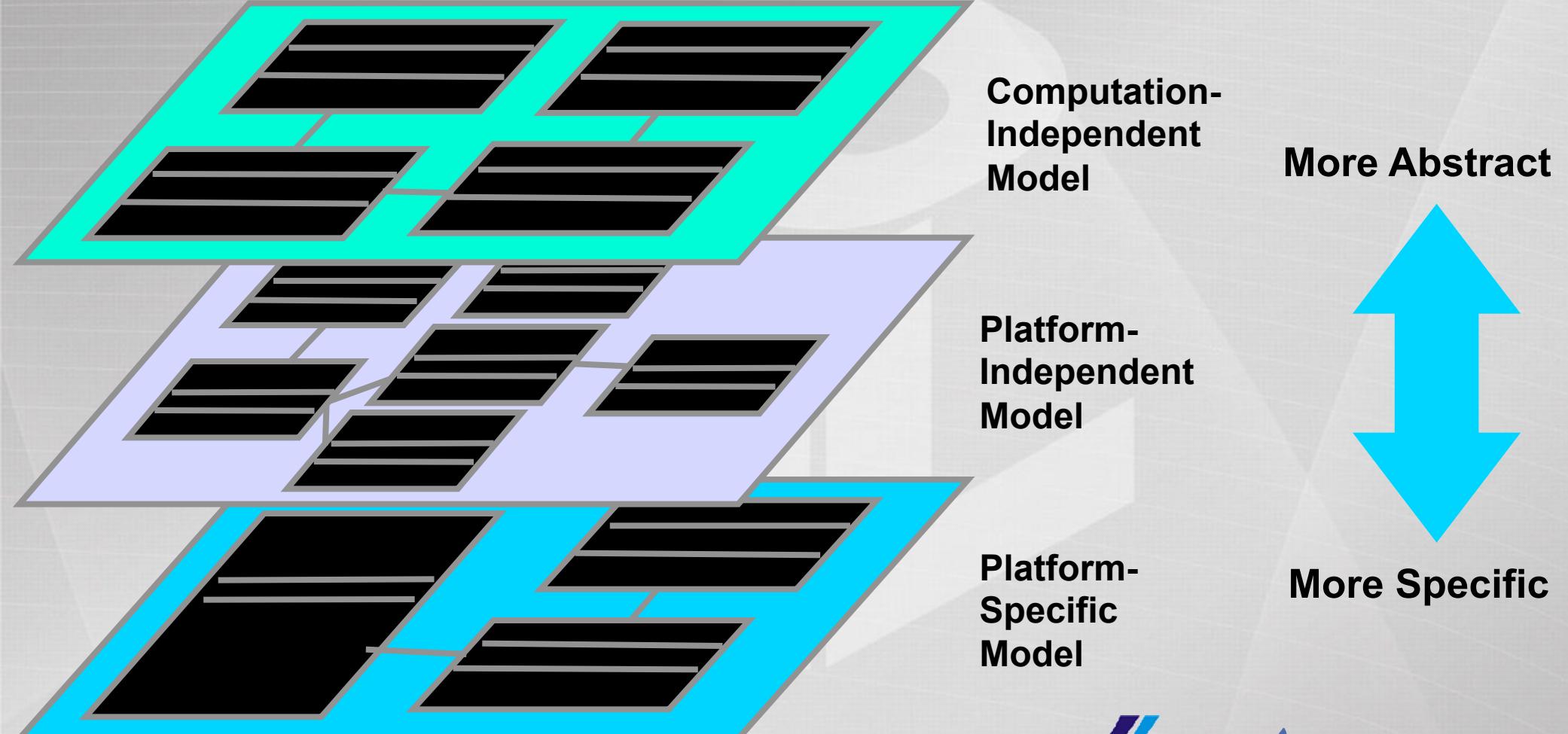
Transformação de modelo é o processo de converter um modelo para outro do mesmo sistema. Que pode ser de forma automática, através de ferramentas, ou manual, através de decisões de projeto.

O *platform independent model* e outras informações são combinadas pela transformação para produzir um modelo específico de plataforma. Exemplo de informações: marcações, informações adicionais, templates, patterns, etc.

Figura do Modelo de Transformação



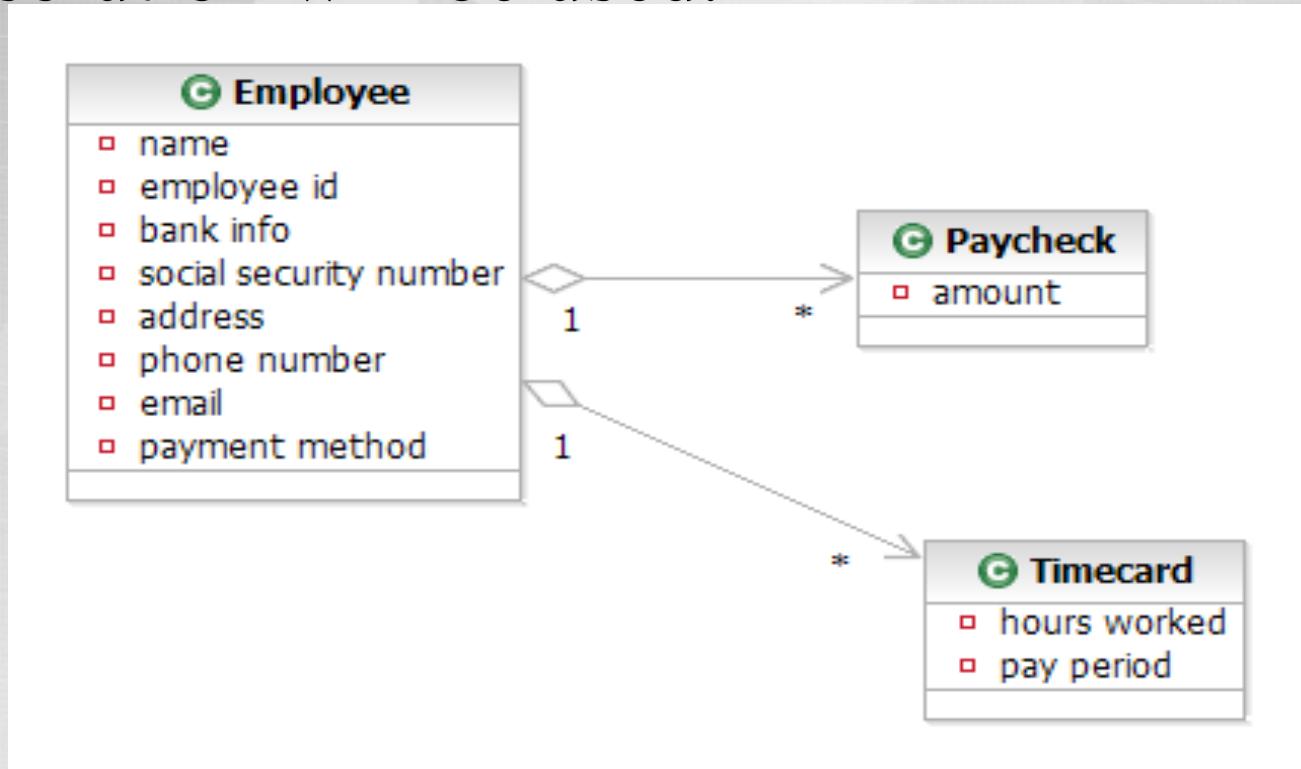
Classifications of Models in MDA



Computation-Independent Model (CIM)

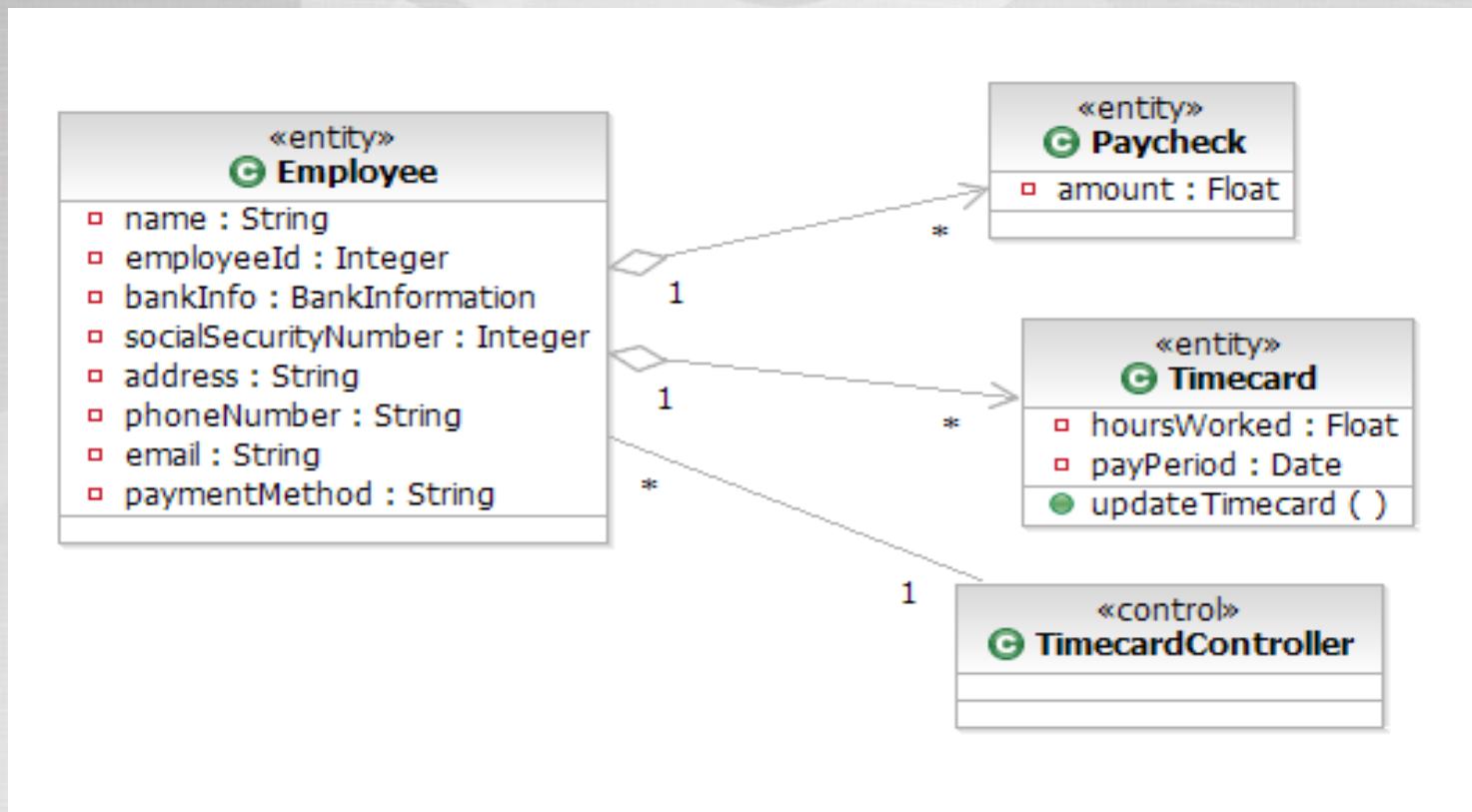
Example

- ◆ Uses the vocabulary of the domain.
- ◆ No information in the model indicates that a computer-based solution will be used.



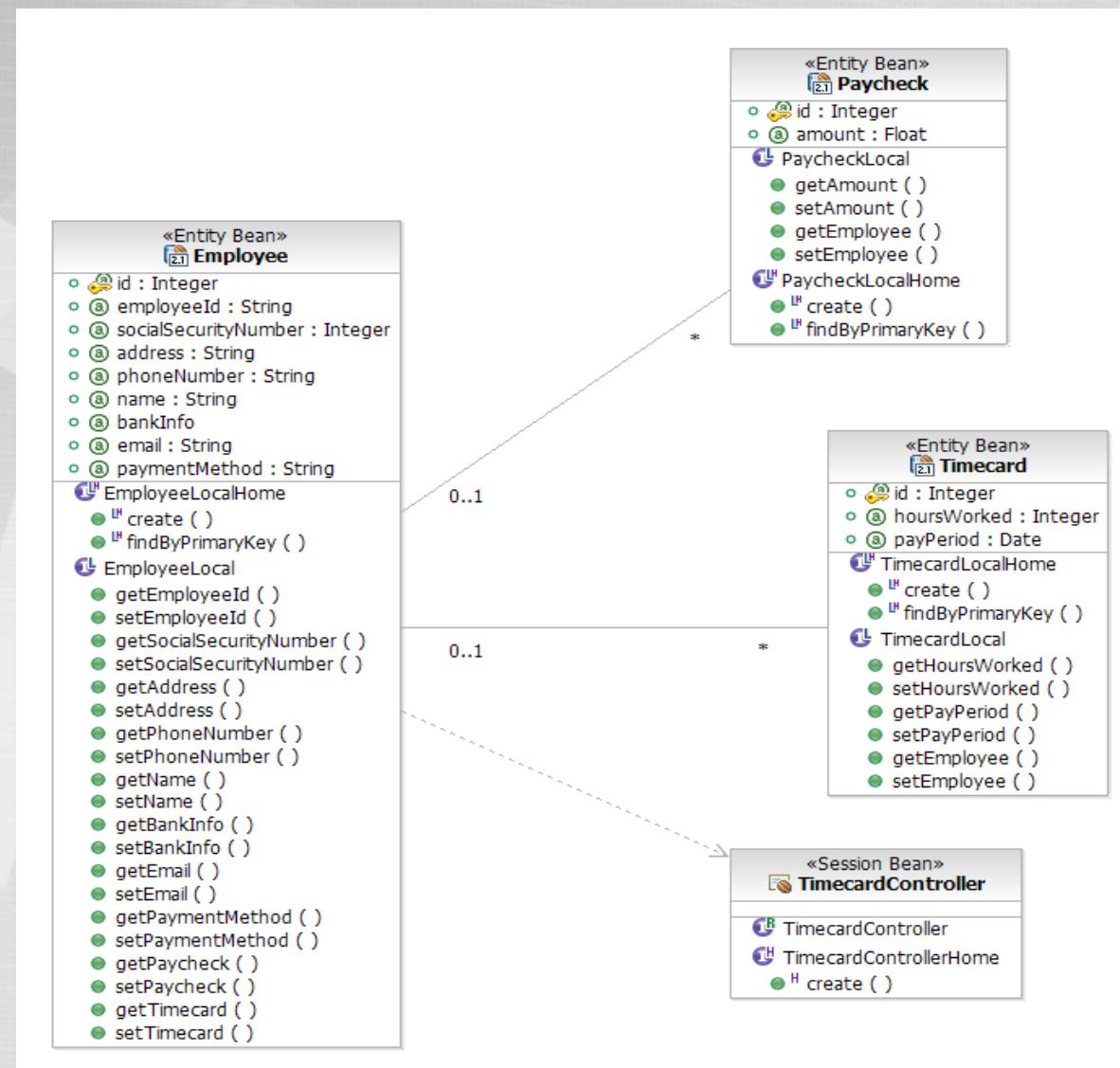
Platform-Independent Model Example

- ◆ Less abstract than CIM
- ◆ Closer to implementation but not tied to a platform.

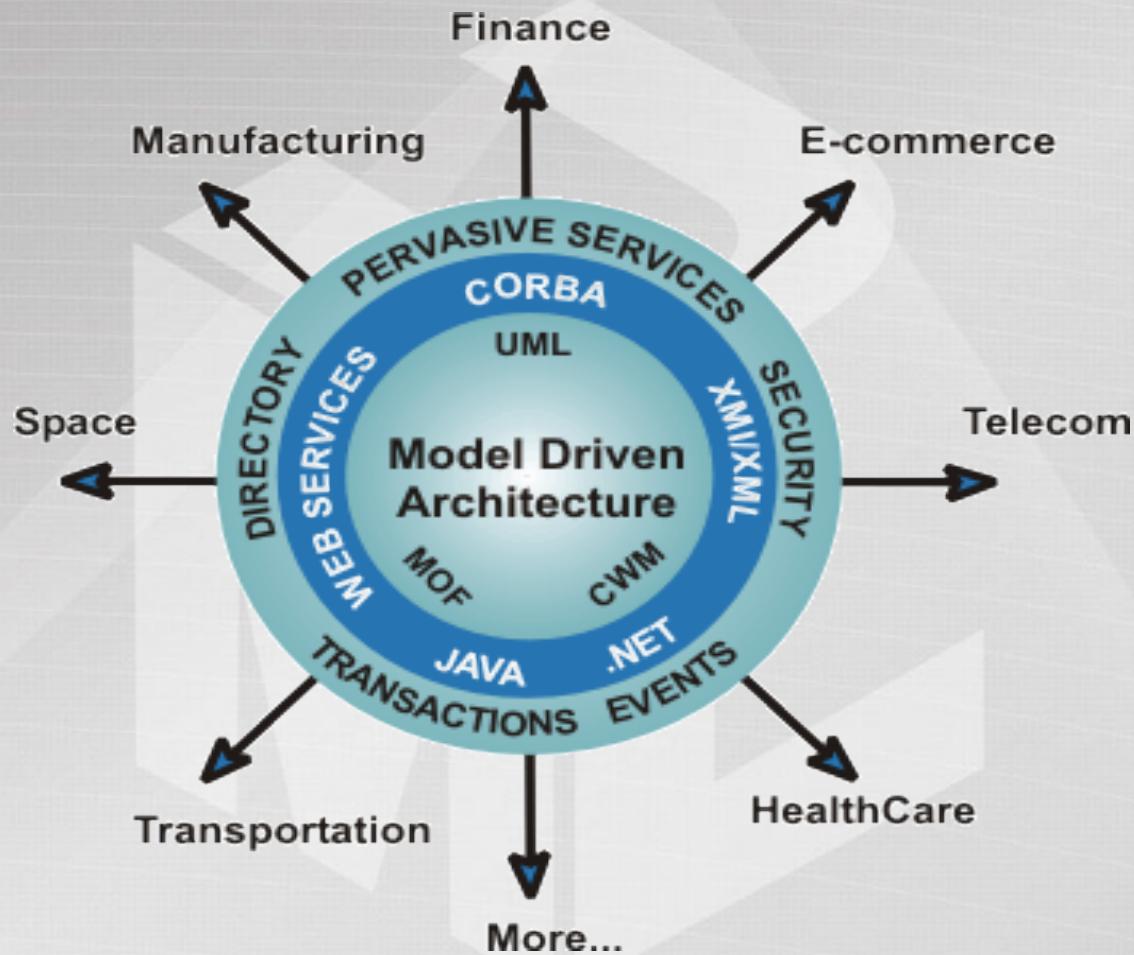


Platform-Specific Model Example

- ◆ Less abstract than PIM
- ◆ Closer to implementation
- ◆ J2EE elements captured in model



Desenho Padrão da MDA



Tecnologias Suportadas

CWM - Common Warehouse Metamodel

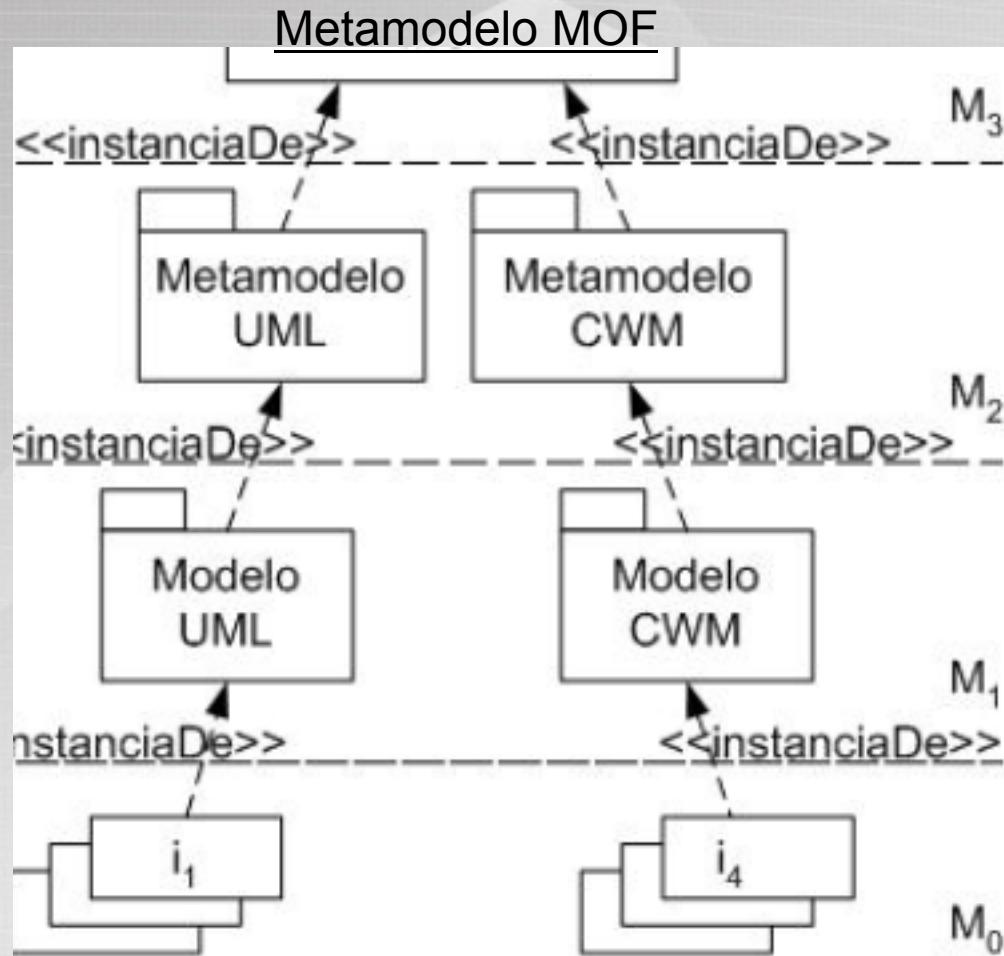
O CWM estabelece padrões para a representação dos Metadados de um repositório, com o objetivo de reduzir os tipos de conflitos entre os esquemas de Banco de Dados.

MOF - Meta-Object Facility

Conjunto de classes UML para a representação de metadados. A especificação MOF define um meta-metamodelo padrão para construção de metamodelos. O propósito principal do MOF é fornecer um mecanismo básico para definição e manipulação de metamodelos em vários domínios, com o foco inicial em Análise e Projeto de Metamodelos de Objetos.

Metadados, ou Metainformação, são dados sobre outros dados. Um item de um metadado pode dizer do que se trata aquele dado, geralmente uma informação inteligível por um computador. Os metadados facilitam o entendimento dos relacionamentos e a utilidade das informações dos dados.

NÍVEIS DE ABSTRAÇÃO DA MDA



Fundamentos de Linha de Produto de Software

Abordagens de Linhas de Produtos de Software

- Family-Oriented Abstraction, Specification and Translation (FAST) (Weiss; Chi Tau, 1999);
- Feature-Oriented Domain Analysis (FODA) documentado pelo SEI (Kang, 1990) e a abordagem proposta por Bosch (2000).
- *Product Line Practice (PLP)* do SEI/CMU (Clements; Northrop, 2001);
- *Product Line Software Engineering (PuLSE)* do Centro de Fraunhofer (Bayer et. al., 1999);
- método KobrA (Atkinson; et. al., 2001).

O Que é uma Abordagem de Linha de Produto?

A abordagem de linha de produto de software tem como objetivo principal promover a reutilização de arquiteturas e permitir o compartilhamento de seus principais artefatos. A abordagem de linha de produto é aplicável aos domínios em que existe uma demanda por produtos que possuem um conjunto de características comuns e pontos de variabilidades bem definidos

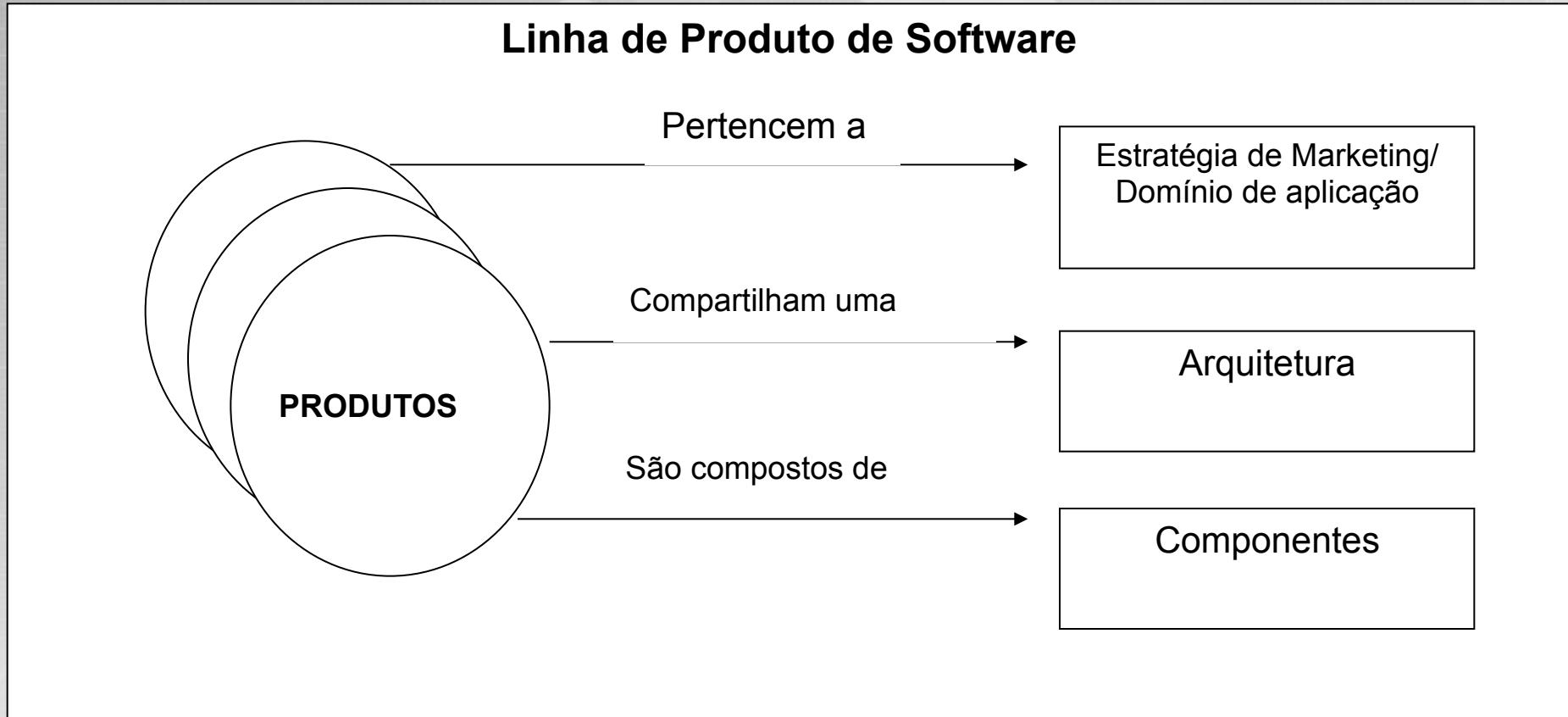
VARIABILIDADE

Um dos objetivos da engenharia de software é preparar o software para mudanças, especialmente quando é projetada uma arquitetura para uma família de produtos (Coplien, 1998). Uma arquitetura de software incorpora previsão de mudanças e variações.

VARIABILIDADE

Feature é uma forma de representar pontos de variação, sendo que este conceito tem origem na engenharia de domínio. Uma *feature* é uma característica de um produto que é considerada importante na descrição e distinção de membros de uma família de produtos. Pontos de variabilidade podem ser introduzidos em vários níveis de abstração: na descrição da arquitetura, na documentação do projeto, no código fonte, no código compilado, em tempo de ligação e na execução do código.

Itens Principais da Linha de Produto de Software



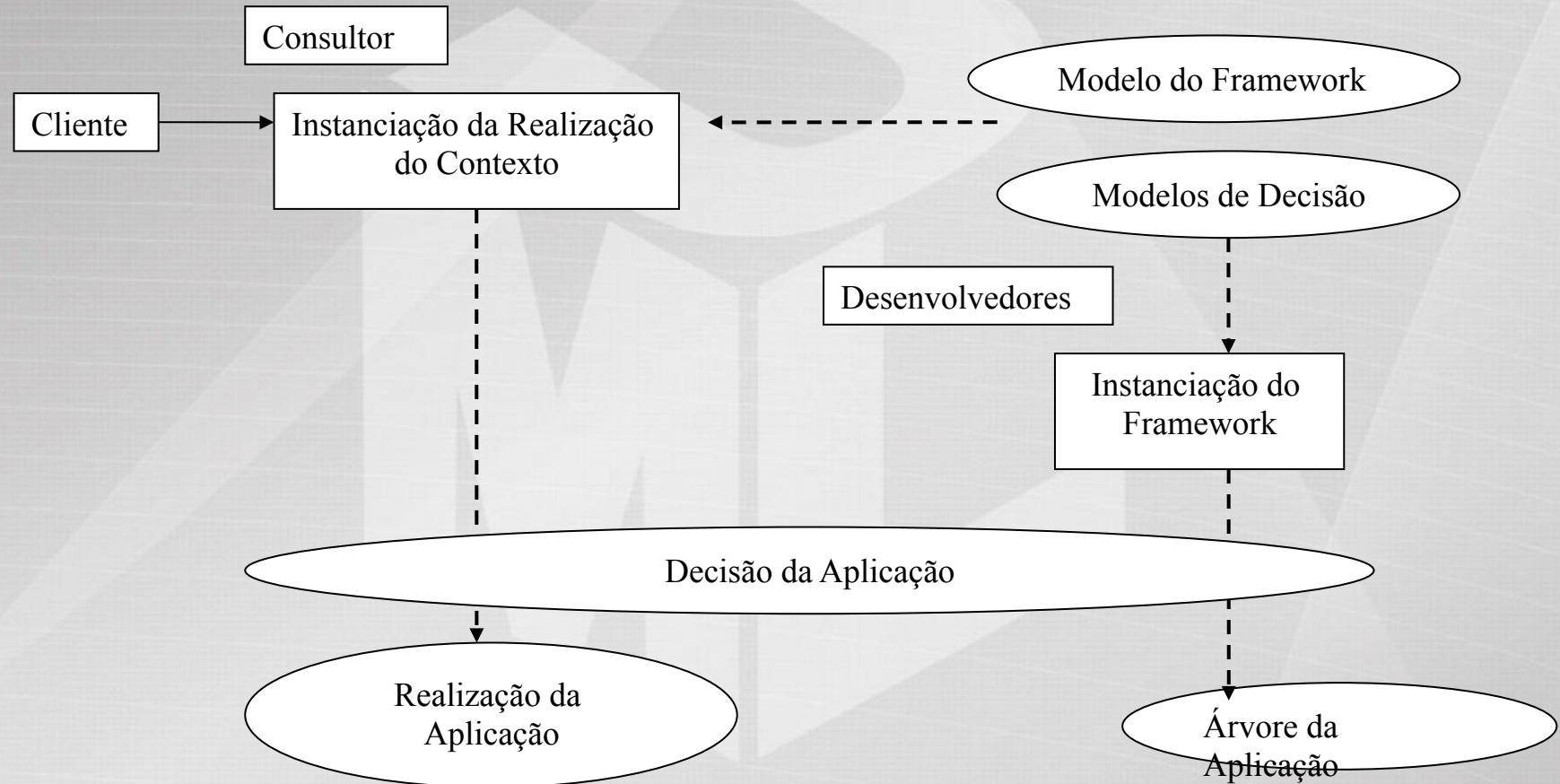
Atividades Essenciais de uma Linha de Produto



Exemplo de Linha de Produto de Software (1/2)

- Component-Based Product Line Engineering with UML
 - KobrA (em Alemão “Komponentenbasierte Anwendungsentwicklung”, em inglês “component-based application development”);

Visão Geral do Processo da Engenharia da Aplicação (2/2)



Introdução a SOA

Afinal, o que é SOA?

- Diferentes interpretações, dependendo do interlocutor...



Diretor Negócios

“SOA é uma tecnologia que cria um ambiente de negócio ágil e provê vantagem competitiva ou maior valor.”



Gerente TI

“SOA é conjunto de processo, estrutura e diretrizes de governança que permite alinhar TI às necessidades do negócio.”



Arquiteto SW

“SOA é uma arquitetura de software baseada em padrões abertos que permite integrar aplicações novas e existentes.”

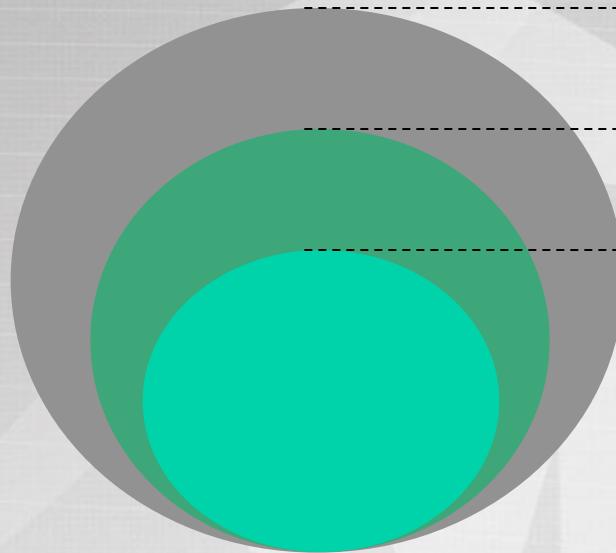


Desenvolvedor

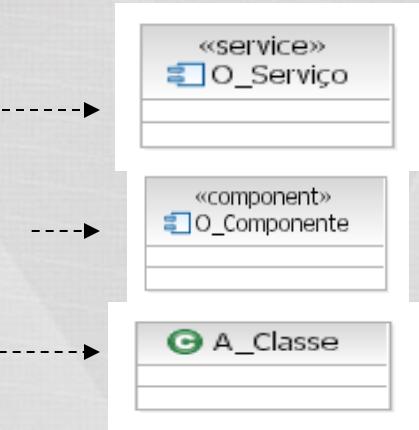
“SOA é um *framework* baseado em *webservices* que permite invocar objetos remotamente utilizando protocolo SOAP, baseado em XML.”

SOA e UML

Abstração



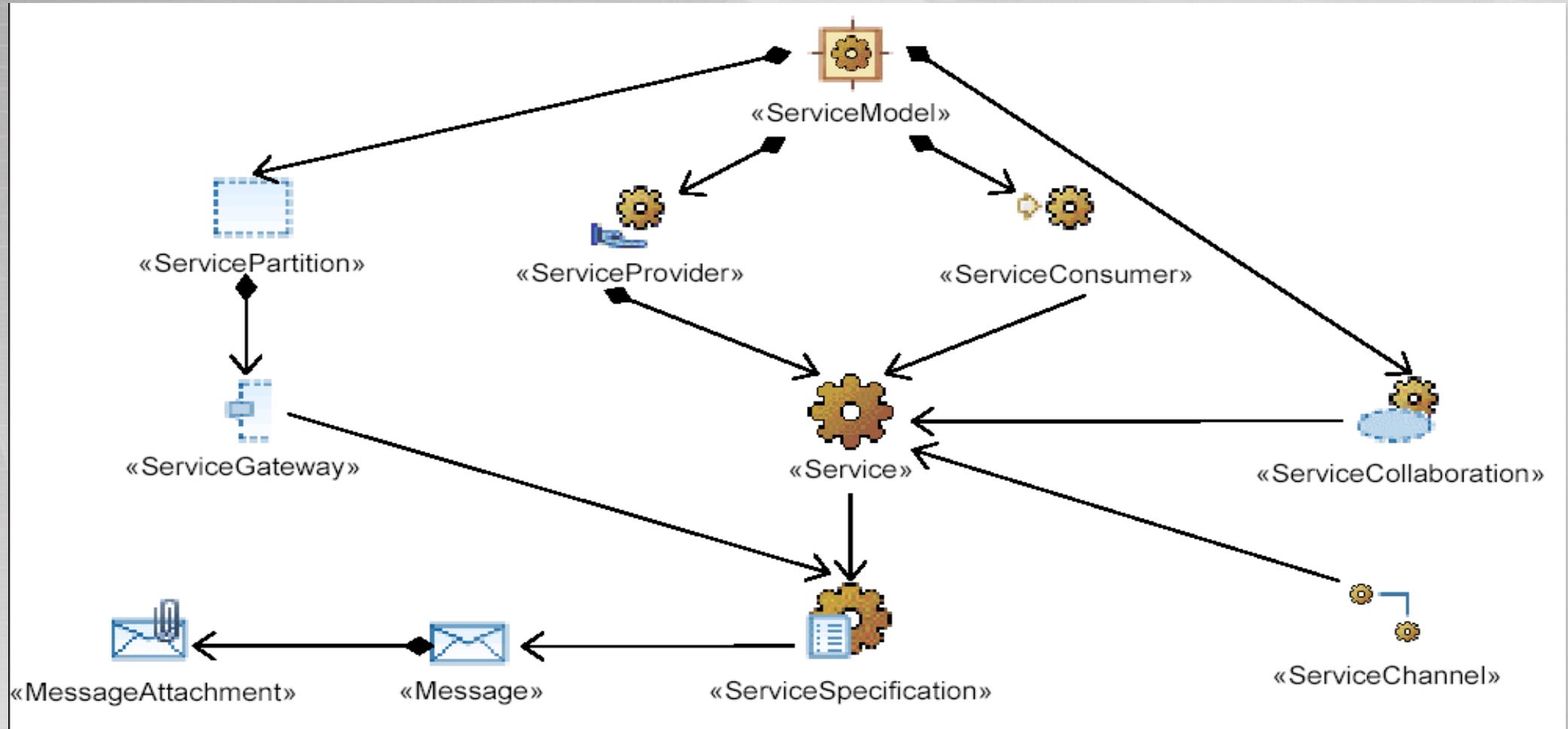
- Serviço
- Componente
- Classe



A utilização de OO e uma arquitetura de componentes é um facilitador para a adoção de SOA, não um pré-requisito

A UML permite projetar a arquitetura orientada a serviços.

Elementos do “UML2 Profile for Software Services”



ESTILOS ARQUITETURAIS

- Pipes e Filtros (Ex. Sistema Operacional Unix / Compilador);
- Camadas (Ex.: Modelo OSI);
- Objetos (Ex.: Abordagem Orientada a Objetos);
- Invocação Implícita (Ex.: JavaBeans – Tratamento de Eventos);
- Quadro Negro (Ex.: IA, BD compartilhado, Repositório Ativo);
- Repositório Passivo (depende da notificação das transações existentes);
- Arquiteturas de Aplicações Distribuídas (Processos Comunicantes e Cliente Servidor);

Tecnologias envolvidas

Diretrizes da Divisão em Camadas

A divisão em camadas fornece um **particionamento lógico de subsistemas em diversos conjuntos**, com determinadas regras sobre como os relacionamentos podem ser formados entre camadas. **Essa divisão permite restringir dependências entre subsistemas**, fazendo com que o sistema seja acoplado mais livremente e, dessa forma, mantido com mais facilidade.

Número de Camadas

Para um sistema pequeno, três camadas são suficientes. Para um sistema complexo, cinco a sete camadas costumam ser suficientes. Para qualquer grau de complexidade, o uso de mais de dez camadas deve ser visto com suspeita, que deverá aumentar com o número de camadas.

Algumas regras práticas são apresentadas abaixo:

Número de Classes	Número de Camadas
0 - 10	A divisão em camadas não é necessária
10 - 50	2 camadas
25 - 150	3 camadas
100 - 1000	4 camadas



Material Complementar Padrões Arquiteturais

Padrões de Software

BUSCHMANN subdivide os padrões em três categorias que representam diferentes níveis de abstração:

- Padrões Arquiteturais (*Architectural Pattern*): Descrevem um esquema de organização estrutural e fundamental para um sistema de software. Eles fornecem um conjunto de subsistemas especificando seus relacionamentos e definindo um conjunto de regras e diretrizes para esses relacionamentos.

Padrões de Software

- **Padrões de Projeto (*Design Pattern*):** Possuem uma descrição mais refinada dos subsistemas, componentes e seus relacionamentos. Neste nível, os mecanismos de cooperação entre componentes são descritos e definidos para encontrar soluções de questões de projeto, em um dado domínio ou contexto.
- **Padrões em Nível de Idioma (*Idioms*):** Descrevem como os componentes são internamente estruturados e interagem, em nível de implementação, incluindo construções típicas de linguagem de programação.

Padrões de Software

- Os padrões podem ser representados a partir de um esquema básico de três partes, denominado de *Contexto-Problema-Solução*, que denota o relacionamento entre um dado contexto, um certo problema proveniente deste contexto e uma solução apropriada para o problema.

Padrões de Software

- Segundo Buschmann [BUS 96], grupos de padrões podem ser organizados de três maneiras:
 - **catálogos de padrões:** apresentam vários padrões, cada um de uma forma relativamente independente. São coleções de padrões de soluções relativamente independentes para problemas comuns de projeto. Podemos citar como exemplo o catálogo de padrões apresentado em [GAM 95];

Padrões de Software

- **sistemas de padrões:** apresentam um relacionamento entre os padrões e cobrem aspectos específicos de um domínio. São vistos como linguagens de padrões incompletas;
- **linguagens de padrões:** cobrem todos os aspectos importantes de um determinado domínio de aplicação e são completas, apresentando grupos de padrões relacionados que cobrem domínios e disciplinas específicos.

Padrões de Software – Segundo Buschmann

01/03

- 1. nome:** contém o nome do padrão, deve ser intuitivo;
- 2. exemplo:** um exemplo do mundo real demonstrando a existência do problema e a necessidade de um padrão;
- 3. contexto:** as situações em que o padrão pode ser utilizado. Descreve o contexto dessas situações;
- 4. problema:** descreve a questão que o padrão resolve;
- 5. solução:** contém o princípio fundamental da solução;
- 6. estrutura:** descreve uma especificação detalhada dos aspectos estruturais da solução encontrada pelo padrão;

Padrões de Software – Segundo Buschmann

02/03

- 7. dinâmica:** descreve o aspecto dinâmico da solução encontrada pelo padrão;
- 8. implementação:** algumas diretrizes para implementação do padrão;
- 9. variantes:** descrição de variantes ou especializações do padrão;
- 10.usos conhecidos:** contém uma breve descrição de sistemas existentes que podem se beneficiar do uso do padrão;
- 11.conseqüências:** descreve os benefícios do padrão e suas habilidades potenciais;

Padrões de Software – Segundo Bruschmann

03/03

12. também conhecido como: outros nomes para padrões, se quaisquer são conhecidos;

13. resolução do exemplo: discussão de qualquer aspecto importante para resolução do exemplo que ainda não estão cobertos nas seções de solução, estrutura, dinâmica e implementação;

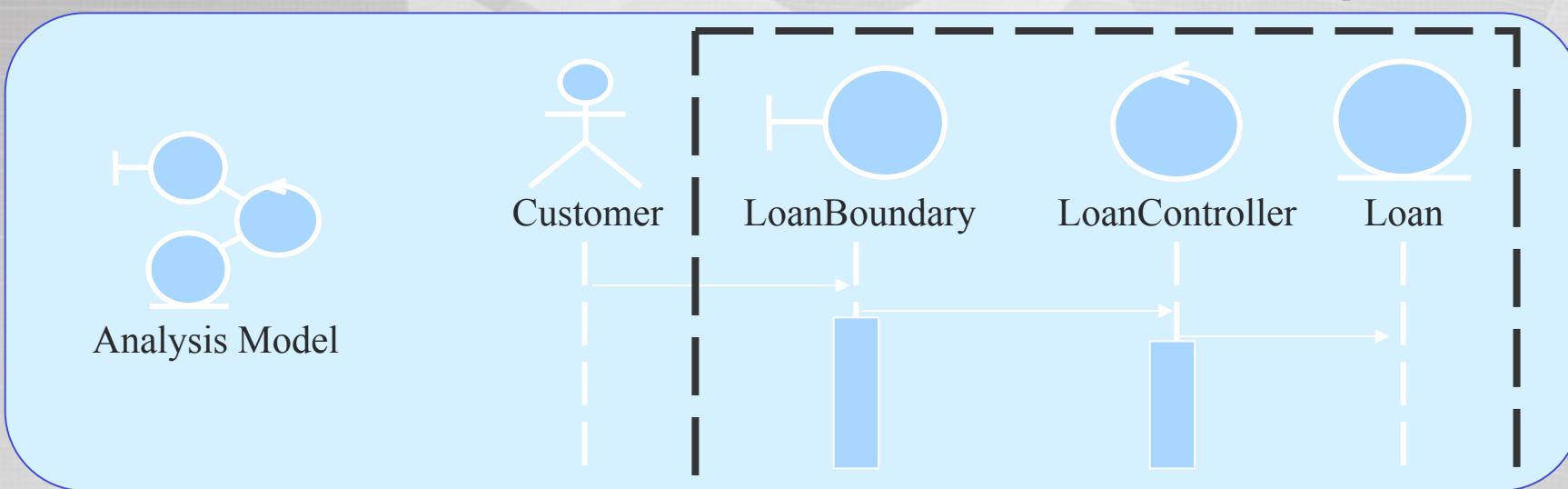
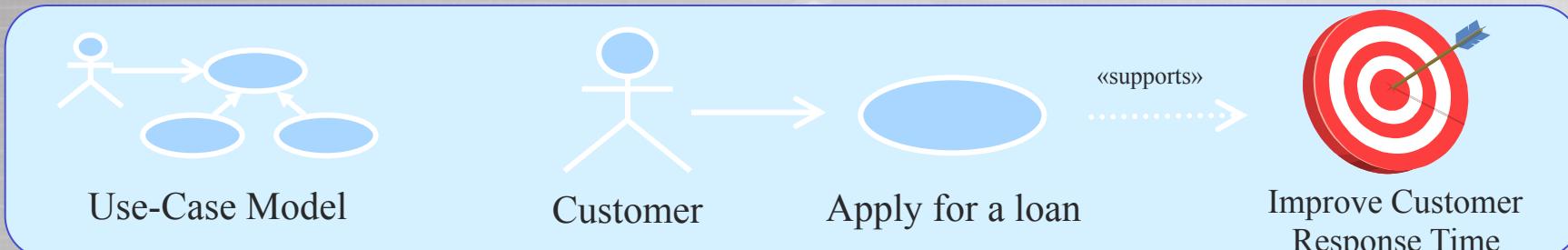
14. veja também: referências para padrões que resolvem problemas similares, e para padrões que ajudam a refinar o padrão que está sendo descrito.

Material Complementar SOA

Fundamentos da SOA

- Na maioria dos casos, uma SOA será implementada usando vários padrões, sendo o mais comum o HTTP, o WSDL (Linguagem de Definição de Serviços Web), o SOAP (Protocolo Simples de Acesso a Objeto) e a XML (Linguagem de Marcação Extensiva). Esses últimos três padrões funcionam em conjunto para distribuir mensagens entre serviços de um modo muito parecido com um posto de correio.
- WSDL → Entrada em um catálogo de endereço
- HTTP → Carteiro (transporte)
- SOAP → Envelope (encapsulamento)
- XML → Carta (mensagem)

Abordagem Top-Down, Orientada por Casos de Uso

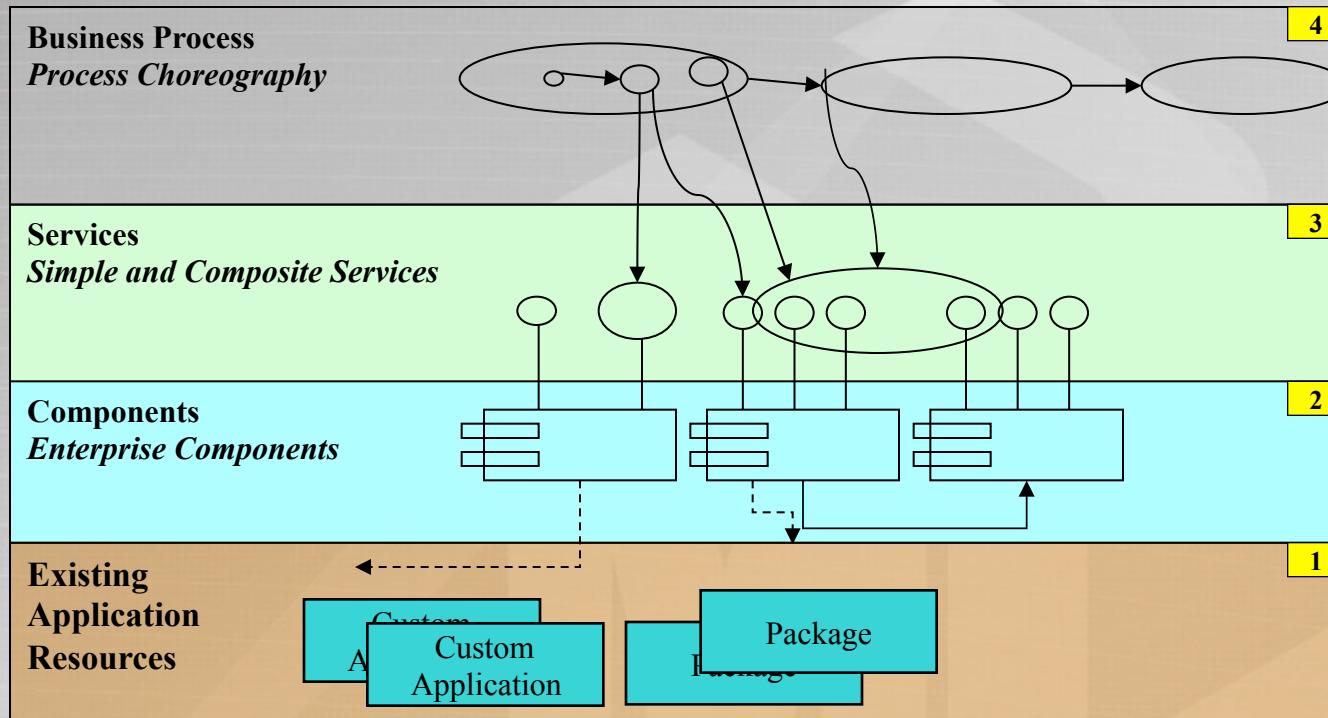


Características de SOA

- Atividades de negócio são realizadas através de uma série de serviços que possuem maneiras bem definidas de “pedir” e “responder” informações .
- Não interessa como o serviço foi implementado, contanto que ele responda aos comandos da forma correta com a qualidade necessária.
- Isto significa que o serviço precisa ser adequadamente seguro e confiável, além de rápido o suficiente.
- Isto faz de SOA uma tecnologia ideal para ser utilizada em um ambiente de TI que possua hardware e software de múltiplos fabricantes.
- As idéias tem suas origens na metade dos anos 80.

*Webservices
é uma
maneira que
temos de
implementar
SOA, mas
WS é
diferente de
SOA*

SOA muda as regras do jogo



SOA cria um modelo digital do negócio <= A GRANDE SACADA!

- Ideal = 1 processo de negócio é desempenhado por 1 serviço de TI reutilizável
- Valor de negócio mensurável - meça o valor do serviço e terá seu valor para o negócio
- Substitui as tradicionais discussões de ROI e as discussões de provas de conceito de TI.

Afinal, o que diferencia SOA das tentativas passadas?

Padrões

- Padrões adotados garantem interfaces bem definidas.
- Antes, padrões proprietário limitavam os padrões

Comprometimento Organizacional

- A área de negócio percebe o valor de SOA (63% dos projetos hoje são conduzidos pelas LOB)*
- Antes, os canais de comunicação e o “vocabulário” não se encaixavam

Foco

- Os serviços SOA focam em atividades no nível do negócio e suas interações
- Antes, o foco era técnico e focado nas subatividades e como seriam construídas

Conexões

- Os serviços SOA são ligados dinamicamente e de maneira flexível
- Antes, as interações eram codificadas dentro da aplicação

Nível de Reusabilidade

- Serviços SOA podem ser extensivamente reutilizados para alavancar ativos de SW existentes
- Antes, o reuso estava restrito a uma determinada aplicação ou tecnologia