UML – introdução

Paulo Trigo Silva

Before the UML

- 1960's - 70's
  - COBOL, FORTRAN, C
  - Structured analysis and design techniques
- 1980's - early 1990's
  - Smalltalk, Ada, C++, Visual Basic
  - Early generation OO methods
- Mid/late 1990's
  - Java
  - UML
  - Unified Process

Paulo Trigo Silva

Modelação Orientada a Objetos

- No *software* existem atualmente duas perspectivas essenciais
  - Algorítmica
  - Orientação por Objetos (O. O.)
- Perspectiva Algorítmica
  - elementos base são procedimentos ou funções
  - foco nas estruturas de controlo e na decomposição *top-down*
  - difícil contemplar alterações aos requisitos e crescimento do sistema
- Perspectiva da Orientação por Objetos (O. O.)
  - elementos base são classes e objetos
  - tem provado ser adequada em variados domínios de problema e diferentes níveis de dimensão e complexidade
  - atualmente grande parte das linguagens, sistemas operativos e ferramentas promovem uma "visão do mundo orientada a objetos"
  - ... UML existe para especificar, construir e documentar sistemas O.O.

Análise e Desenho O. O. – perspectiva histórica

- Métodos de Análise e Desenho Orientados a Objetos
  - surgiram entre meados da década 1970 e fins de 1980
- No período de 1989 a 1994
  - o número de métodos cresceu de cerca de 10 para mais de 50
  - isto fomentou as chamadas "guerras dos métodos" (*method wars*)
  - dificuldade em escolher um método perfeitamente ajustado ao problema
- Autores (metodologistas – *methodologists*) que se destacaram,
  - Grady Booch; Ivar Jacobson; James Rumbaugh
    - Booch Method;
    - OOSE – *Object Oriented Software Engineering*;
    - OMT – *Object Modeling Technique*
  - Yourdon; Coad
  - Shlaer; Mellor

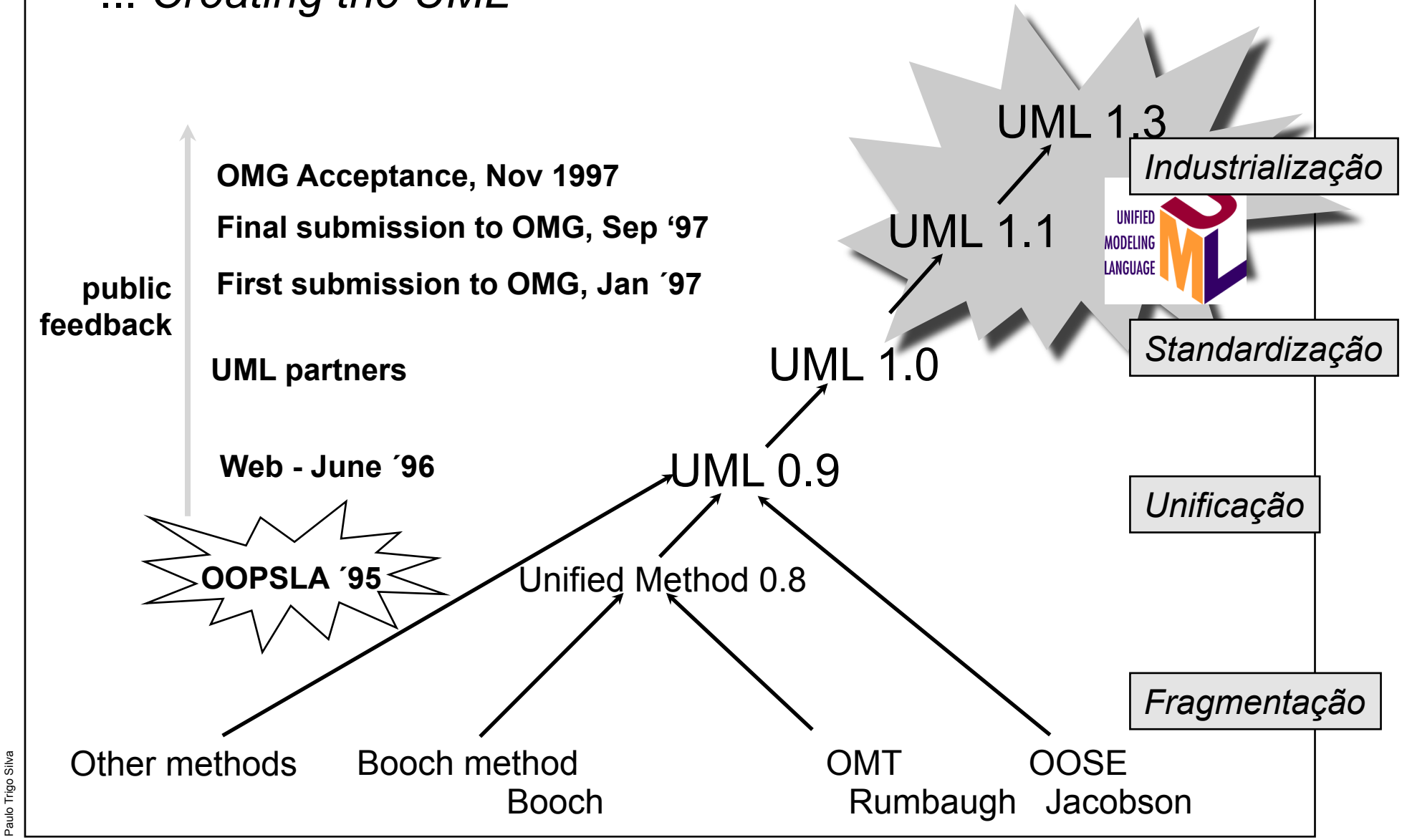
UML – perspetiva histórica

- Em meados de 1990, três autores (e empresas) "encontram-se"
  - Grady Booch – *Rational Software Corporation*
  - Ivar Jacobson – *Objectory*
  - James Rumbaugh – *General Electric*
  - ... e cada um começa a adoptar ideias dos métodos dos outros!
- Este "encontro" motivou o surgimento de uma linguagem "unificada"
  - cada autor já caminhava no sentido dos outros
    - evolução conjunta eliminou necessidade de "diferenças forçadas"
  - unificação dos métodos oferece "estabilidade" à área da O.O.
    - constitui suporte maduro a projetos e fabricantes de ferramentas
  - um método unificado captura experiência acumulada de cada autor e aborda questões por resolver em cada um dos métodos ...
- UML – *Unified Modeling Language*
  - oficialmente este esforço iniciou-se em Outubro de 1994

Unificação – objectivos e concretização

- ... do "encontro" resultaram 3 linhas orientadoras
  - visar problemas à escala dos sistemas complexos e de resposta crítica (*mission-critical*), p.e. banca, controlo aéreo, apoio ações militares, ...
  - modelar sistemas, desde a concepção até ao artefacto executável, utilizando técnicas de orientação a objetos
  - construir linguagem de modelação utilizável por pessoas e máquinas
- Marcos na concretização do projeto
  - outubro 1995 – versão 0.8 do *Unified Method*
    - Booch Method e OMT (nesta altura Jacobson junta-se à *Rational*)
  - junho 1996 – versão 0.9 da UML
    - unificação do OOSE (Jacobson) com o *Unified Method*
  - janeiro 1997 – versão 1.0 da UML proposta para *standard* da OMG
    - consórcio: Digital, HP, IBM, Microsoft, Oracle, Rational, e outros
  - 14 novembro 1997 – UML 1.1, adoptado como *standard* da OMG
    - no fim de 1998 a OMG *Revision Task Force*, lançou a UML 1.3

... *Creating the UML*



Paulo Trigo Silva

Ênfase da UML

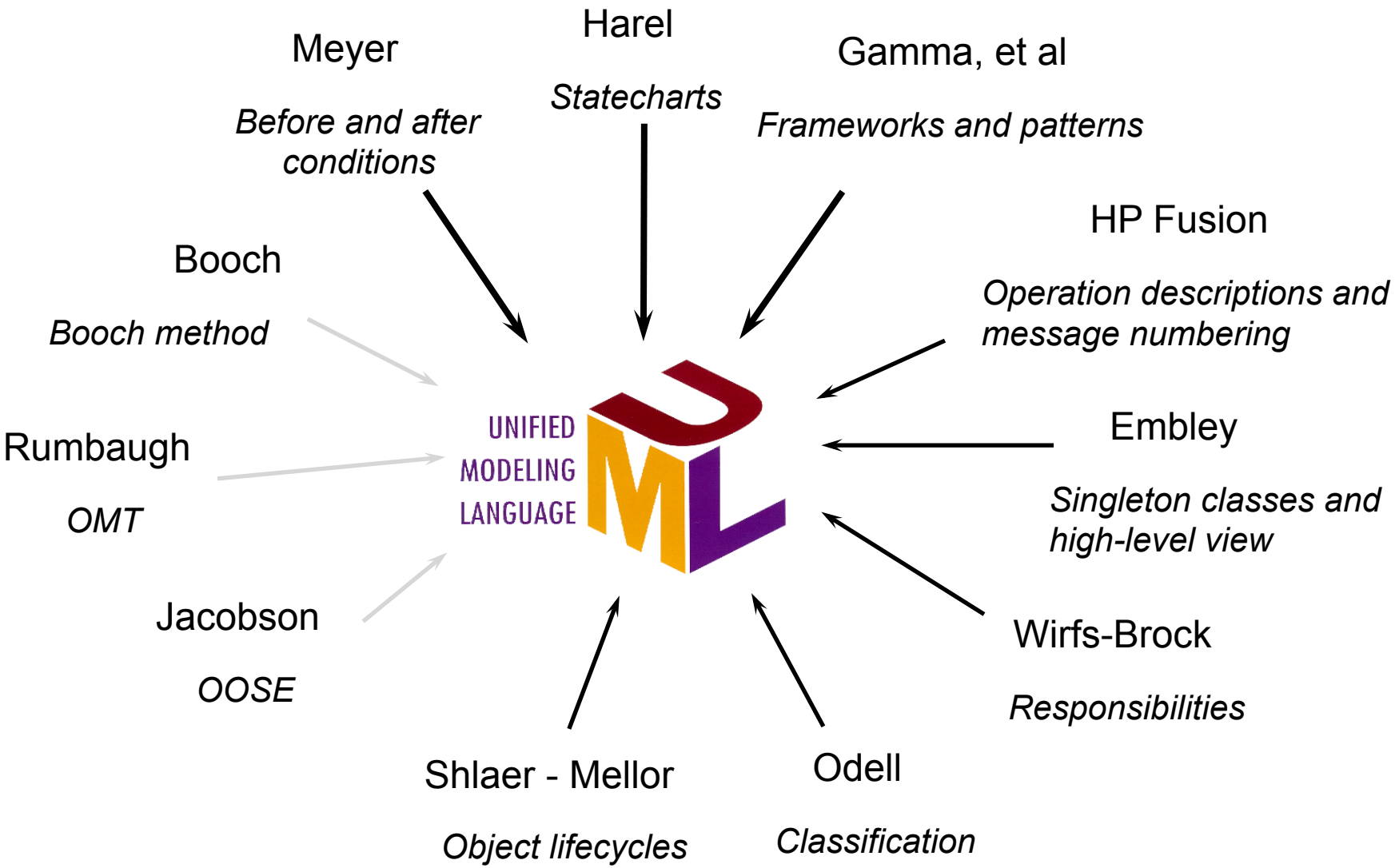
- Definição de uma linguagem de modelação *standard*
  - independente das linguagens de programação
  - independente das ferramentas CASE (Computer Aided Software Engineering)
  - independente dos processos de desenvolvimento
- O processo de desenvolvimento pode depender
  - do tipo de projecto
  - da ferramenta de suporte
  - da organização e *know-how* das equipas de projecto
- ... mas a mesma linguagem de modelação é sempre a mesma
  - UML
- Actualmente assiste-se à adopção generalizada da UML
  - como "a" linguagem de modelação de *software* na abordagem da O.O.
  - ... artigos, relatórios, ferramentas CASE, etc – industrialização da UML !

UML – algumas referências

- Parcerias e contribuições à UML
  - Hewlett-Packard; IBM; Oracle; Microsoft; Platinum Technology; Rational Software; Intellicorp and James Martin and Company; Anderson Consulting; Ericson; Platinum Technology; Texas Instruments;
  - ObjectTime Limited; PTech; Reich Technologies; I-Logix; ICON; Softeam; Sterling Software; Taskon; MCI Systemhouse
- Endereços importantes
  - Rational Software
    - www.rational.com
  - Open Unified Process (em Eclipse Process Framework – EPF)
    - www.eclipse.org/epf/
  - OMG – Object Management Group
    - www.omg.org

Paulo Trigo Silva

... Contributions to the UML



... *Overwiew of the UML*

The UML is a language for

visualizing

specifying

constructing

documenting

the artifacts of a software-intensive system

Construir documentos / modelos – porquê ?

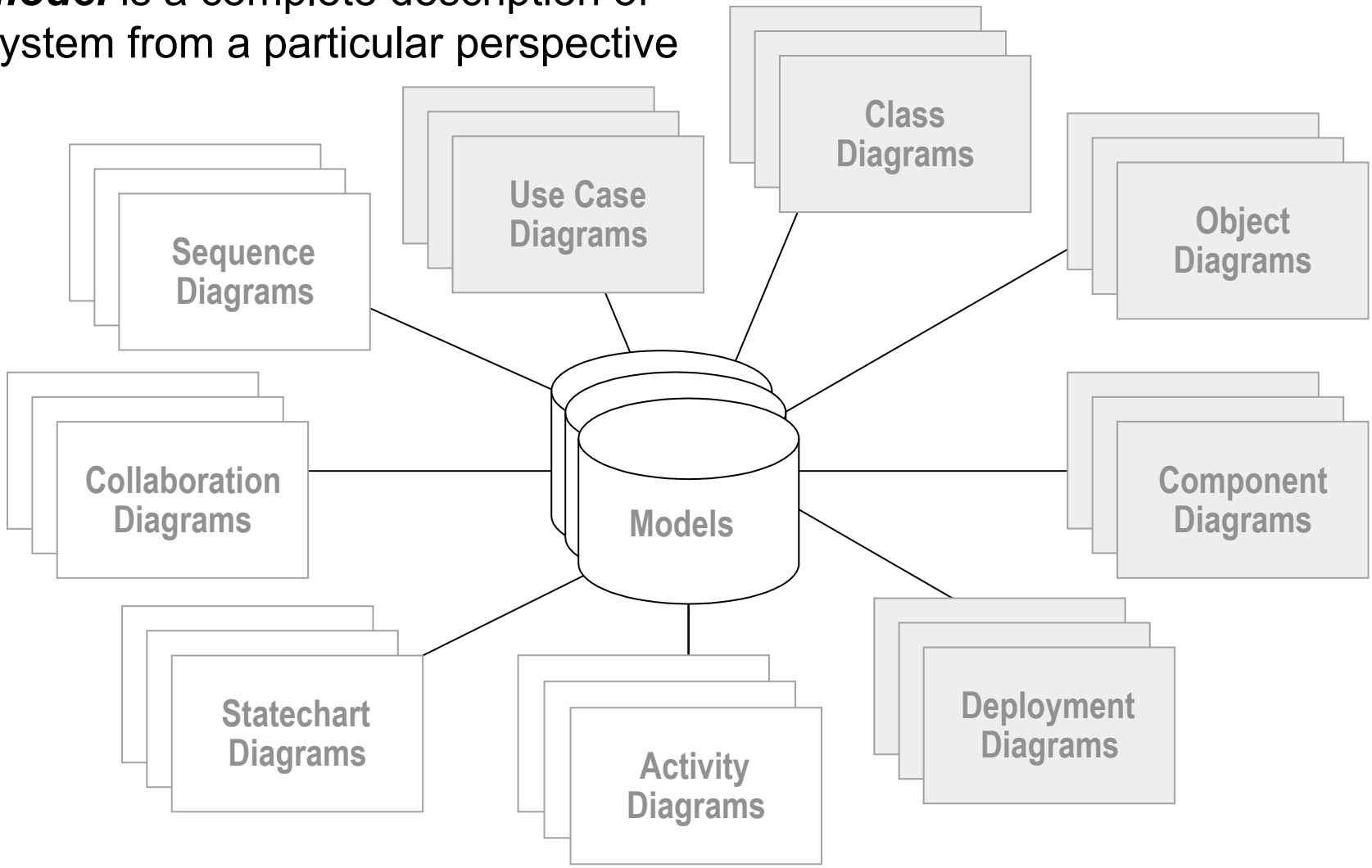
- ... em "The Unified Modeling Language – User Guide", p. 3
  - "*The primary product of a development team is not beautiful documents, world-class meetings, great slogans, or Pulitzer prize-winning lines of source code.*
  - *Rather, it is good software that satisfies the evolving needs of its users and the business. Everything else is secondary*."
- ... mas secundário não se pode confundir com irrelevante !
  - cumprir prazos, qualidade duradoura, desenvolvimento efectivo, ...
  - implica pessoas certas, foco correcto, ferramentas adequadas, ...
- Modelos – peças centrais a todas as actividades de desenvolvimento
  - fornecem base para partilha de ideias sobre estrutura e comportamento
  - permitem visualizar e controlar a arquitectura
  - realçam oportunidades de simplificação e reutilização
  - ... modelos servem para compreender o sistema, controlar e gerir o risco

A importância dos modelos

- "*A model is a simplification of reality.*"
  - um bom modelo omite elementos não relevantes para o nível de abstracção que pretende exibir
  - ... pode referir a estrutura – ênfase na organização do sistema
  - ... pode referir o comportamento – ênfase na dinâmica do sistema
- "*We build models so that we can better understand the system we are developing.*"
  - permite ver o sistema tal como ele é, ou como pretendemos que ele seja
  - constitui um guião, a seguir na construção do sistema
  - documenta as decisões que se tomam
- "*We build models of complex systems because we cannot comprehend such a system in its entirety.*"
  - permite uma abordagem do tipo "*divide-and-conquer*", [Dijkstra]
  - *amplify human intellect* – permite pensar a nível de abstracção mais alto

... *Models and Diagrams*

A ***model*** is a complete description of a system from a particular perspective



Paulo Trigo Silva

Linguagem UML – a perspectiva conceptual

- Para compreender uma linguagem é necessário
  - elaborar uma visão que permita raciocinar nessa linguagem
  - ... saber decidir em que contexto a linguagem é aplicável
  - ... conhecer os elementos que permitem comunicar na linguagem
- Em que situações usar a linguagem UML?
  - a grande motivação é a do desenvolvimento industrial de *software*
  - domínios em que tem sido utilizada,
    - serviços financeiros; banca; seguradoras; sistemas de informação empresariais; telecomunicações; transportes; defesa; comércio de retalho; meio científico; serviços distribuídos via *Internet*
- Aprender UML implica conhecer 3 noções essenciais
  - Blocos Base de construção (*basic building blocks*)
  - Regras de Integração dos Blocos Base (*rules to use the building blocks*)
  - Mecanismos Comuns às anteriores noções (*common mechanisms*)

UML – blocos base de construção

- O vocabulário da UML tem 3 espécies de Blocos Base
  - Conceitos (*Things*)
  - Relações (*Relationships*)
  - Diagramas (*Diagrams*)
- O que são esses Blocos Base ?
  - ... os Conceitos são os "cidadãos de primeira" num modelo
  - ... as Relações "ligam" os Conceitos
  - ... os Diagramas juntam "colecções de Conceitos" interessantes !

Paulo Trigo Silva

UML – como vamos prosseguir ?

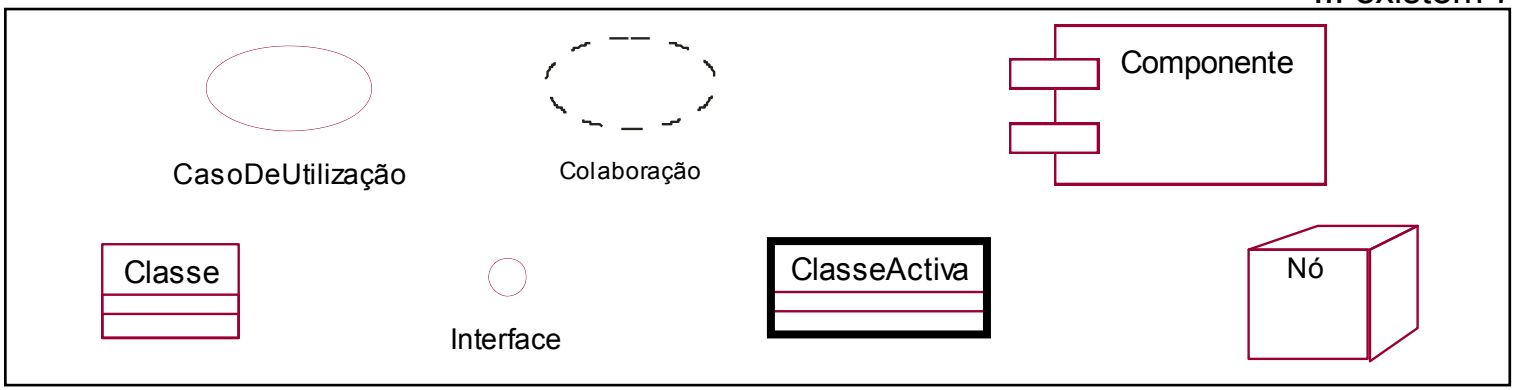
- ... já vimos as 3 espécies de Blocos Base do vocabulário da UML
  - Conceitos; Relações; Diagramas
- O vocabulário da UML tem 4 espécies de Conceitos (*Things*)
  - de Estrutura (*Structural*)
  - de Comportamento (*Behavioral*)
  - de Agrupamento (*Grouping*)
  - de Anotação (*Annotational*)
- O estudo da UML vai prosseguir do seguinte modo:
  - 1) Conceitos de Estrutura
  - 2) Relações
  - 3) Conceitos de Comportamento
  - 4) Conceitos de Agrupamento e Anotação
  - 5) Diagramas

Nota:
- A abordagem aos Conceitos está intercalada, no passo 2, pela abordagem às Relações
- Os Mecanismos Comuns aos diversos elementos serão abordados quando adequado

Resumo – UML ... e como vamos prosseguir ?

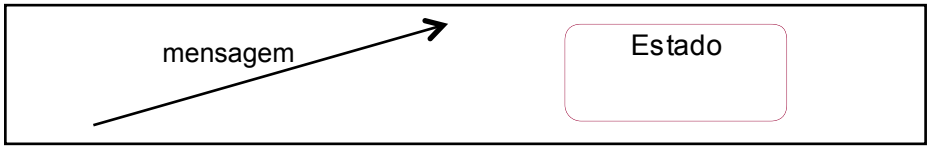
- Conceitos de Estrutura

... existem 7

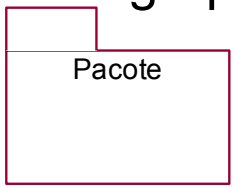


- Conceitos de Comportamento

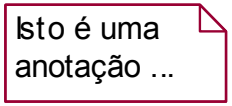
... existem 2



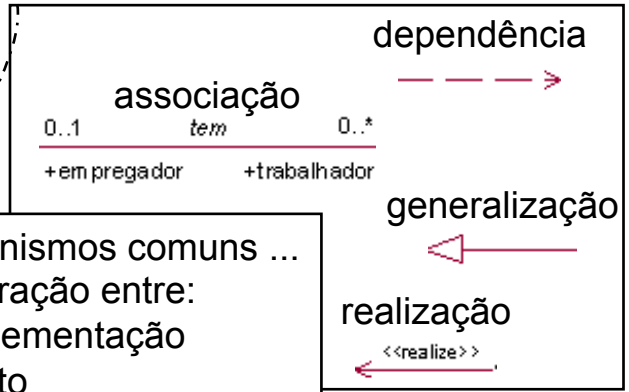
- Conceitos de Agrupamento



- Conceitos de Anotação



nesta passagem é preciso já conhecer as Relações ...



e também mecanismos comuns ... como o da separação entre:
- Interface / Implementação
- Classe / Objecto

... existem 4

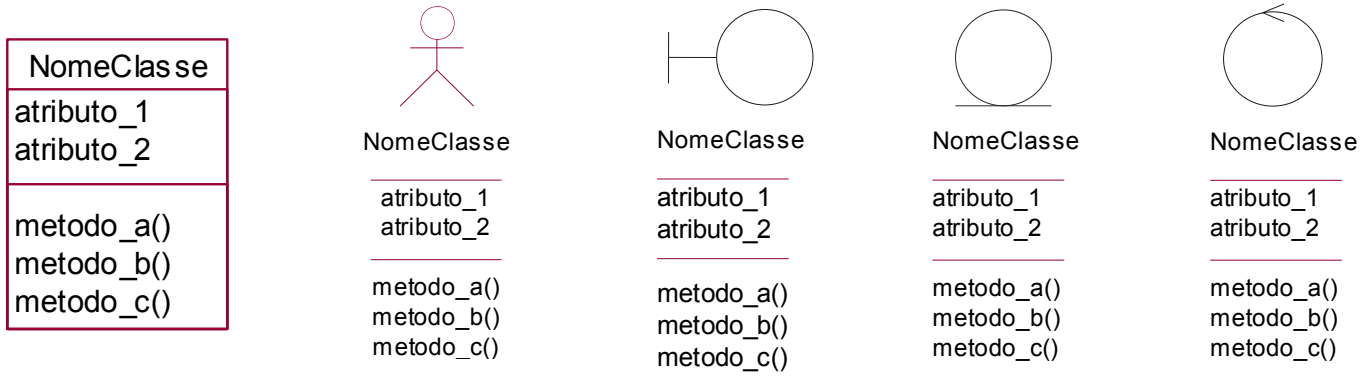
Conceitos de Estrutura

- São os elementos "mais estáticos" dos modelos UML
  - "nomes próprios" / "substantivos" do modelo
  - representam elementos lógicos (conceptuais) ou físicos
- Existem 7 espécies de "Conceitos de Estrutura"
  - os primeiros 4 são
    - Classe (*Class*)
    - Interface (*Interface*) – um "molde particular" / estereótipo de classe
    - Caso de Utilização (*Use Case*)
    - Colaboração (*Collaboration*) – um estereótipo de caso de utilização
  - os últimos 3 representam aspectos particulares do conceito de classe
    - Classe Activa (*Active Class*)
    - Componente (*Component*)
    - Nó (*Node*)

estes 2 representam conceitos <u>físicos</u>
... todos os anteriores eram <u>lógicos</u>

Conceitos de Estrutura – Classe

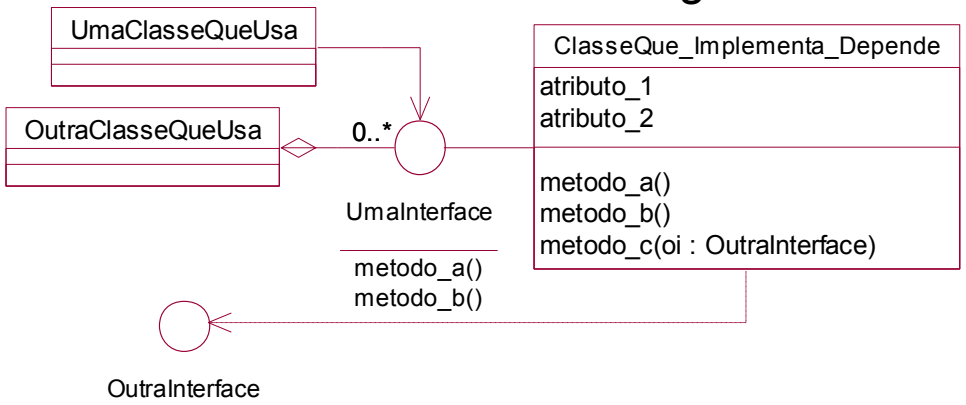
- Classe (*Class*)
  - descrição de um conjunto de objectos que,
    - partilham os mesmos atributos, operações, relações e semântica
  - implementa uma ou mais interfaces
  - normalmente apresentado como um rectângulo, que inclui,
    - nome, atributos e operações
  - a apresentação gráfica depende do estereótipo escolhido ...
    - o estereótipo estende o vocabulário da UML
    - permite ter blocos específicos, adequados a determinadas situações

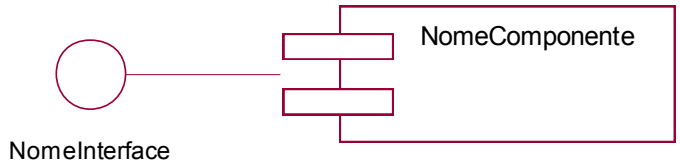


estereótipos:	nenhum	*actor*	*boundary*	*entity*	*control*

Conceitos de Estrutura – Interface

- Interface (*Interface*)
  - colecção de operações que especificam um serviço de,
    - uma classe ou componente
    - tipicamente apenas uma parte limitada do seu comportamento
  - define especificações de operações (assinaturas)
    - nunca a implementação dessas operações
  - normalmente apresentada por um circulo, que inclui, o seu nome
    - é um estereótipo de classe
  - raramente aparece sozinha
    - normalmente está ligada à classe ou componente que a concretiza

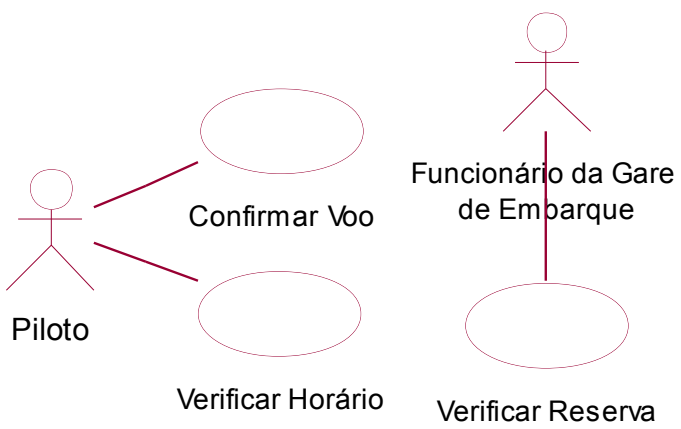


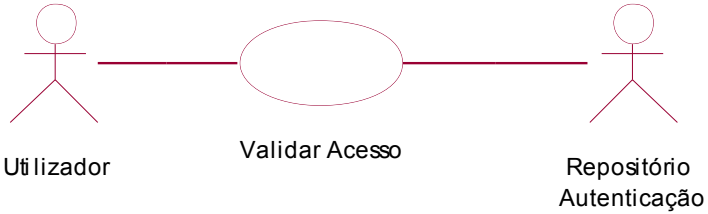


Cuidado – não "baralhar":
Interface, Classe, Componente ...

Conceitos de Estrutura – Caso de Utilização

- Caso de Utilização (*Use Case*)
  - conjunto de sequências de acções que o sistema efectua
    - oferece um resultado observável, de valor para um actor específico
    - usado para estruturar os conceitos de comportamento do sistema
    - é concretizado pelo conceito de colaboração
  - normalmente apresentado por uma elipse, apenas com o seu nome
  - aparecem ligados ao(s) actor(es)
    - que dele tiram partido
    - ou de quem depende

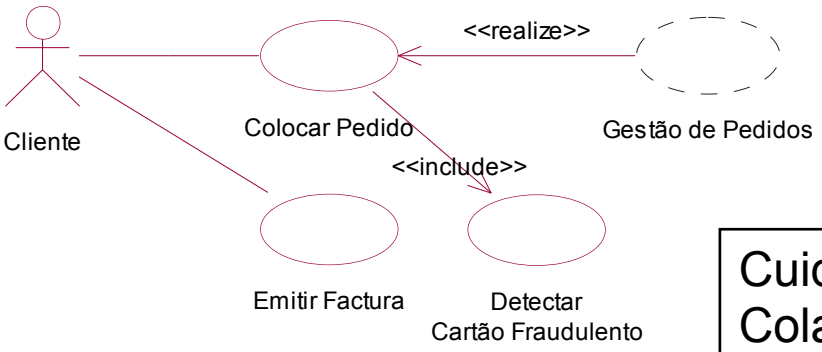


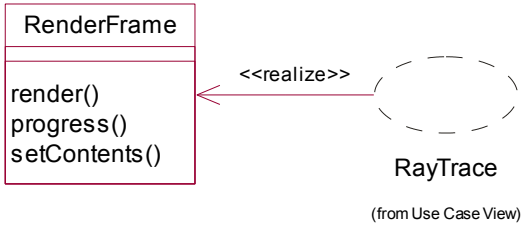


Cuidado – não "baralhar":
Caso de Utilização, Actor, Classe...

Conceitos de Estrutura – Colaboração

- Colaboração (*Collaboration*)
  - "sociedade" de elementos que "trabalham" em conjunto para,
    - fornecer comportamento superior ao da soma dos seus elementos
    - inclui estrutura e comportamento
    - uma mesma classe pode participar em diversas colaborações
  - as colaborações representam a concretização
    - dos "padrões" que constituem o sistema
  - normalmente apresentada por elipse tracejada, apenas com o seu nome
  - é usual aparecer como realização de
    - casos de utilização ou operações



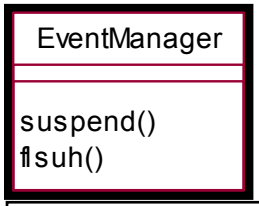


Cuidado – não "baralhar":
Colaboração, Caso de Utilização, Classe ...

Conceitos de Estrutura – Classe Activa

- Classe Activa (*Active Class*)
  - classe cujos objectos detém um ou mais processos ou *threads*
    - processo (*process*) – fluxo "pesado" (*heavyweight*) de execução que pode ocorrer em simultâneo com outros processos
    - *thread* – fluxo "leve" (*lightweight*) de execução que pode ocorrer em simultâneo com outras *threads*, no contexto do mesmo processo
  - classe que representa um fluxo de controlo independente
    - comportamento dos objetos pode ser concorrente com o de outros
  - as "outras classes" (as não activas) são passivas, ou seja,
    - não podem, de forma independente, iniciar actividade de controlo
  - normalmente apresentado por um rectângulo de traço mais forte e inclui,
    - nome, atributos e operações

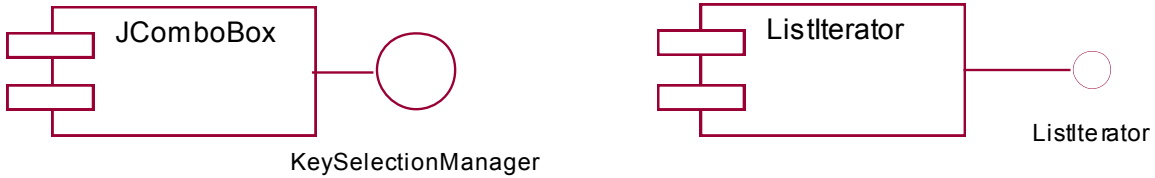
Nota – implementação UML no StarUML (v.5.0.2):
- Classe Activa tem grafismo igual à das outras Classes;
- Indica-se como *Stereotype thread* ou *process*;



Cuidado – não "baralhar":
Classe Activa, Classe ...

Conceitos de Estrutura – Componente

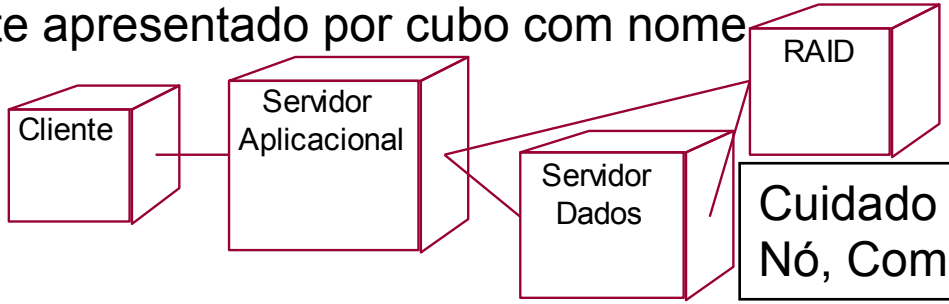
- Componente (*Component*)
  - elemento físico e que pode ser substituído
    - está em conformidade e realiza um conjunto de interfaces
  - representa o "empacotamento" físico de elementos lógicos, tais como,
    - classes, interfaces e colaborações
  - num sistema podem existir diferentes componentes
    - *Java Beans*, *Corba*, *COM*, ...
    - ficheiros de código (são artefactos do processo de desenvolvimento)
  - normalmente apresentado por rectângulo com alças, e inclui o seu nome



Cuidado – não "baralhar":
Componente, Interface, Classe, Colaboração ...

Conceitos de Estrutura – Nó

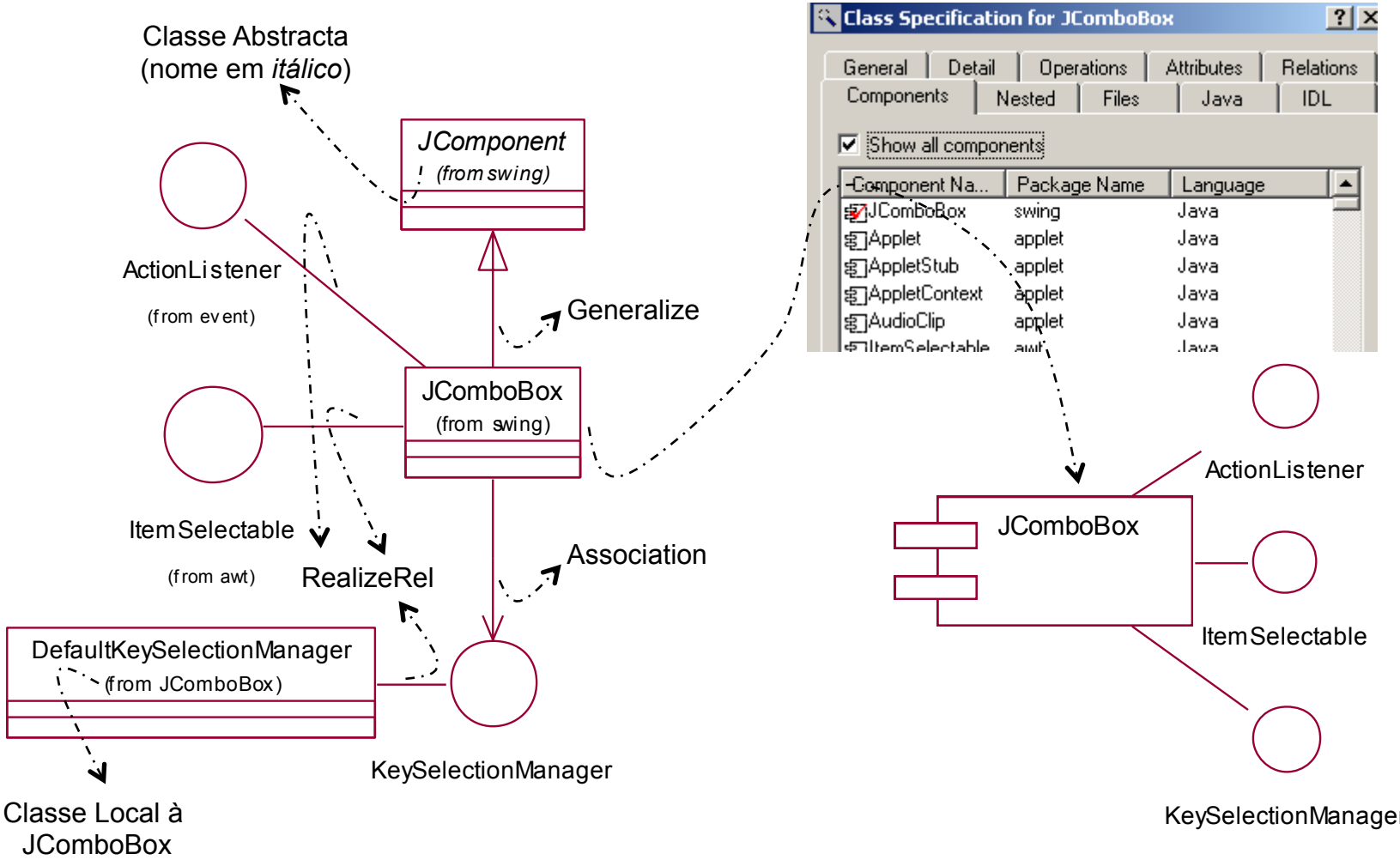
- Nó (*Node*)
  - elemento físico que existe em tempo de execução
    - recurso computacional com alguma memória
    - e possivelmente com capacidade de processamento
  - um conjunto de componentes pode,
    - residir num nó; migrar de um nó para outro
  - existem diferenças importantes entre nó e componente
    - componente participa na execução de um sistema
    - nó executa componentes
    - componente representa um "pacote" físico de conceitos lógicos
    - nó representa a implantação (*deployment*) física de componentes
  - normalmente apresentado por cubo com nome



Cuidado – não "baralhar": Nó, Componente ...

Classe, Interface, Componente ...

- ... do pacote *Swing* do Java, no *Rational Rose*

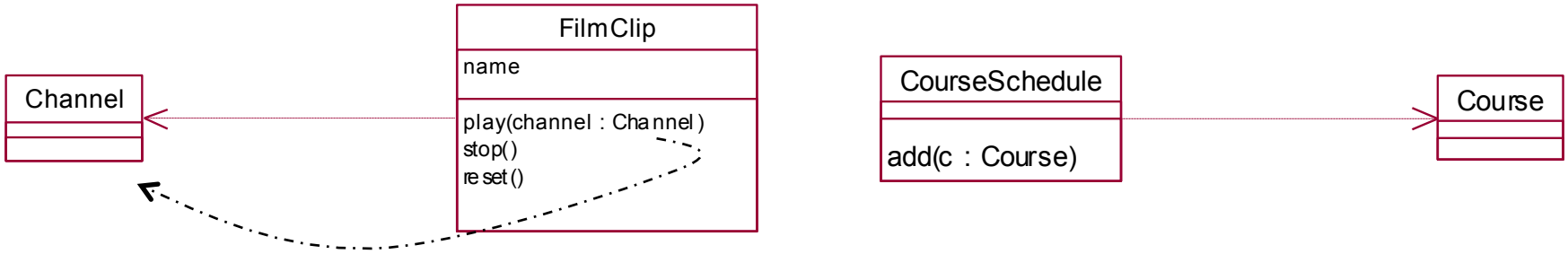


Relações

- Existem 4 espécies de "Relações"
  - Dependência (*Dependency*)
  - Associação (*Association*)
    - Qualificação (*Qualification*) – um adorno (*adornment*) importante
    - Agregação (*Aggregation*) – "um outro tipo de Associação"
  - Generalização (*Generalization*)
  - Realização (*Realization*)
- Estas Relações são,
  - as formas base de relacionar Conceitos
- Estas Relações utilizam-se,
  - para construir Diagramas "bem–formados", ou seja,
  - ... construídos de acordo com as regras (sintáticas) da linguagem

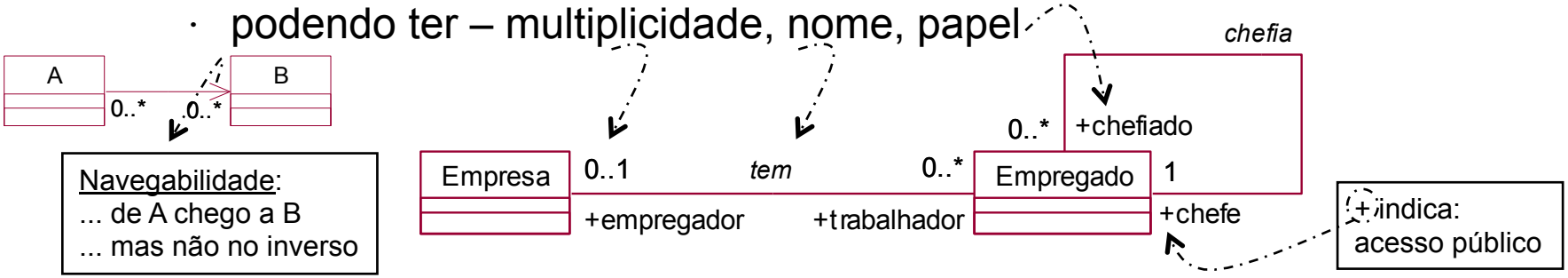
Relações – Dependência

- Dependência (*Dependency*)
  - relação semântica de utilização (*using*) entre dois conceitos
    - a alteração de um Conceito (o independente) pode afectar
    - a semântica do outro Conceito (o dependente)
    - mas o inverso não é necessariamente verdade
  - muitas vezes, no contexto de uma classe é usada para mostrar que,
    - a classe usa outra como argumento na assinatura de uma operação
    - a operação pode ser afectada por uma alteração na classe usada
    - ... se passar a exibir outro comportamento ou interface
  - normalmente apresentada por uma linha tracejada, orientada
    - para o sentido do conceito de que se depende



Relações – Associação

- Associação (*Association*)
  - relação estrutural que descreve
    - um "conjunto de ligações" (*set of links*)
    - onde cada ligação (*link*) "é um tuplo" de vários objectos
  - tendo uma associação, é possível navegar entre os objectos,
    - que são instâncias das classes participantes nessa associação
  - pode-se estabelecer uma associação,
    - de uma classe para ela mesma
    - entre duas classes (associação binária)
    - entre mais que duas classes (associação n-ária)
  - normalmente apresentada por uma linha ligando as classes participantes
    - podendo ter – multiplicidade, nome, papel



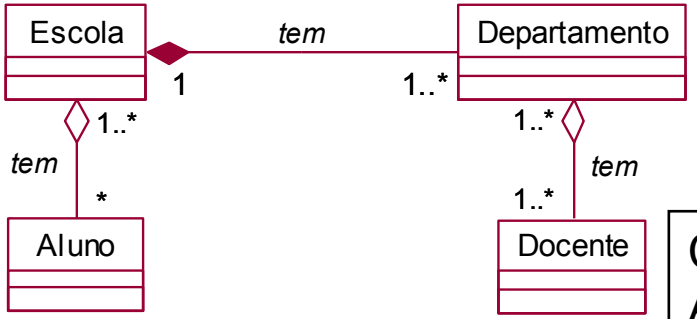
Relações – Associação / Qualificação (um adorno)

- Qualificação (*Qualification*) como adorno de uma Associação
  - um problema muito comum das associações é o de,
    - ... partindo de um objecto num lado de uma associação,
    - identificar o objecto (ou conjunto de objectos) que,
    - do outro lado da associação a ele estão ligados ...
  - exemplo – "*um gato pode ser sempre vadio, ou ter ao longo da sua vida, diversos donos, assim como um dono pode ter diversos gatos ... mas cada gato, em determinado data, ou tem um dono ou nenhum !*"
    - "data" não é atributo de nenhuma classe participante na associação
    - ou seja, "data" é um atributo da associação
    - um atributo da associação representa-se como uma qualificação
    - ... qualificação pode "reduzir" a multiplicidade da associação
    - ... objecto qualificado + valor do qualificador → objecto do outro lado

Cuidado – não "baralhar":
Qualificação, Multiplicidade, Associação ...

Relações – Associação / Agregação (outro tipo)

- Agregação (*Aggregation*)
  - uma associação entre duas classes, representa uma relação estrutural
    - entre pares – entre elementos que estão ao mesmo nível conceptual
  - numa relação "todo/parte" (*whole/part*)
    - uma classe representa "algo maior" – o todo
    - que é composta por "constituintes mais pequenos" – as partes
  - uma Agregação é uma relação "todo/parte", também designada por *has*
  - normalmente apresentada por um losango perto do "todo"
    - contorno – partes têm uma existência própria (*has by reference*)
    - cheio – partes só existem no contexto do todo (*has by value*)

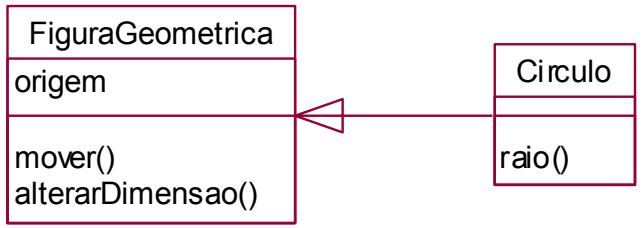


Cuidado – não "baralhar":
Agregação por Valor / Referência, Associação ...

Paulo Trigo Silva

Relações – Generalização

- Generalização (*Generalization*)
  - relação estrutural entre
    - um conceito geral – super-classe ou classe-pai
    - e uma "espécie" mais específica desse conceito – classe filho
  - os objectos da classe filho
    - podem ocorrer em qualquer local onde ocorra a classe pai
    - o filho pode substituir o pai – o inverso não é verdade ...
  - o filho herda as propriedades (atributos e operações) dos pais
    - e o filho pode adicionar propriedades que só são suas
  - uma operação de um filho com a mesma assinatura que a do pai
    - sobrepõe (*overrides*) a operação do pai – polimorfismo
  - normalmente apresentada por uma linha orientada,
    - com uma seta apontando o pai

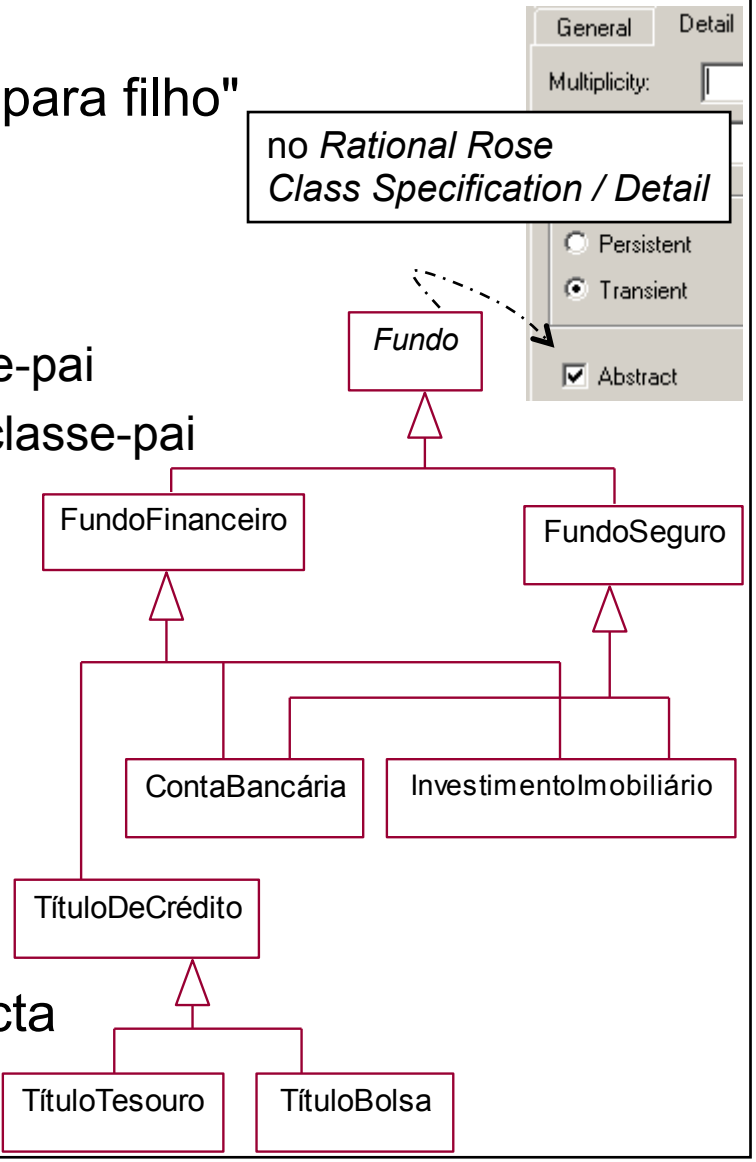


Polimorfismo
... do grego *polýs* – *muito* + *morphé* – forma
... propriedade do que se apresenta em várias formas

Paulo Trigo Silva

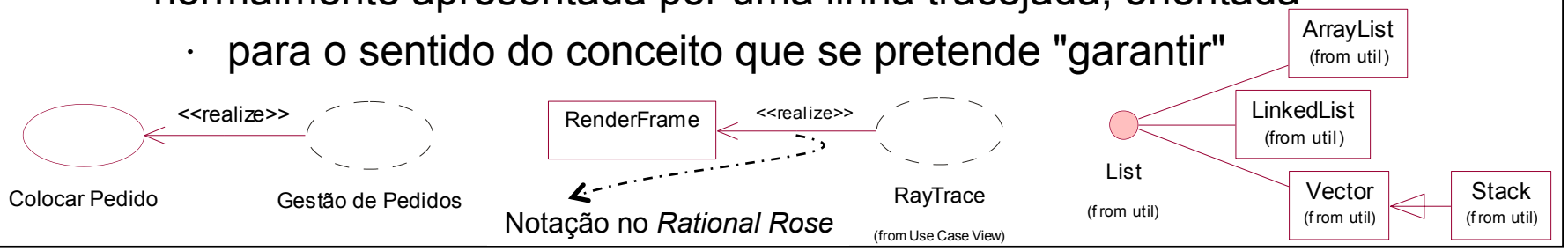
Relações – Generalização (cont.)

- Generalização implica Herança de "pai para filho"
  - de estrutura e comportamento
- Herança Simples e Múltipla
  - Simples – conceito com um única classe-pai
  - Múltipla – conceito com mais que uma classe-pai
- Classe Abstracta
  - não tem instâncias
  - é base para definição de
    - operações e
    - estado
    - ... a herdar por subclasses
- Se pelo menos uma operação é abstracta
  - a Classe é também Abstracta



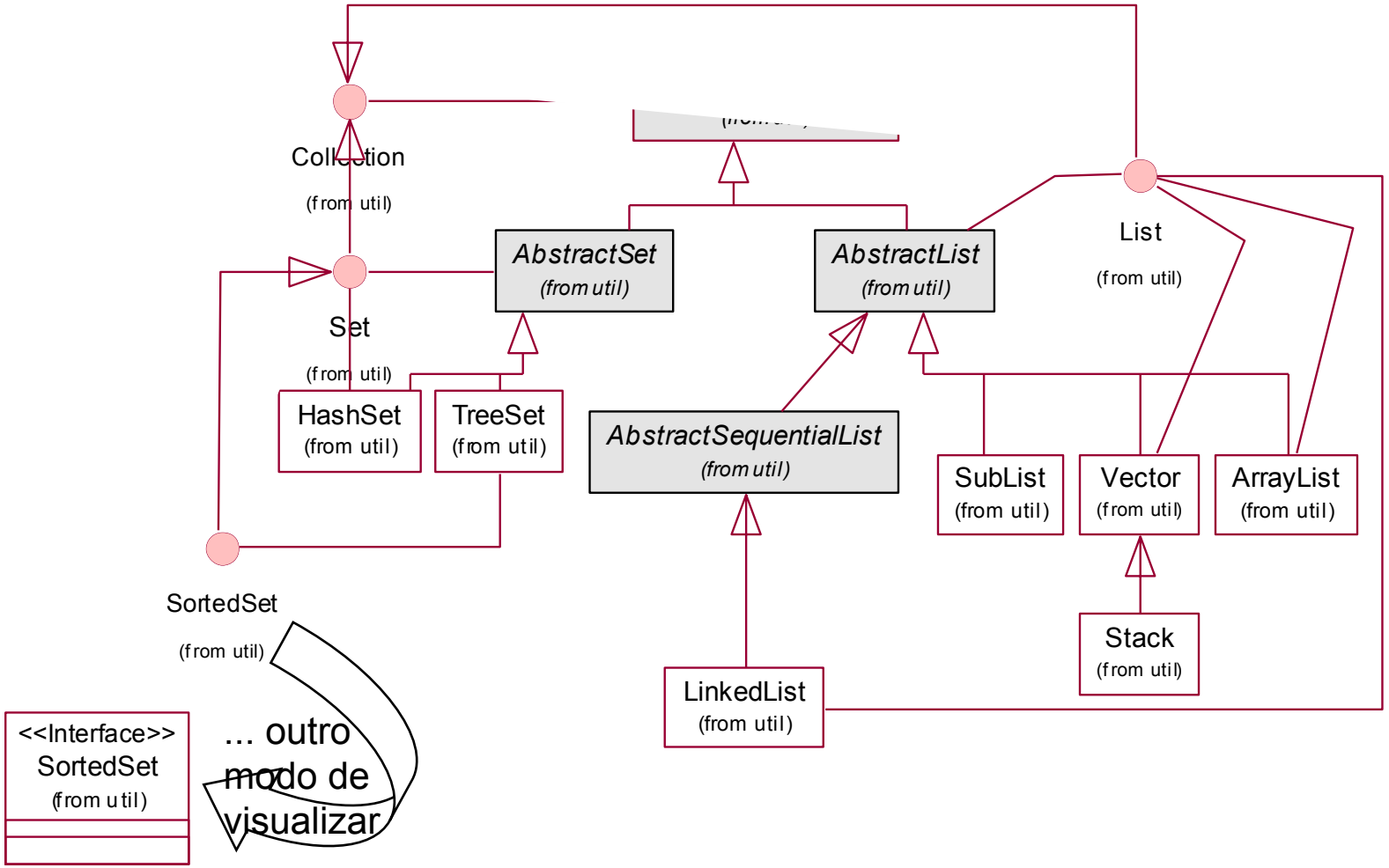
Relações – Realização

- Realização (*Realization*)
  - ligação semântica do género "estabelecimento de contrato"
  - contrato entre dois Conceitos
    - um "define" serviços
    - outro "garante" levar a cabo os serviços especificados
  - semanticamente a Realização está "a meio caminho" entre
    - a Dependência e a Generalização
    - ... depende da definição dos serviços; é uma concretização particular
  - é usual usar-se em dois contextos
    - interface – operações / serviços de classes ou componentes
    - colaboração – relação entre um *use case* e a sua concretização
  - normalmente apresentada por uma linha tracejada, orientada
    - para o sentido do conceito que se pretende "garantir"



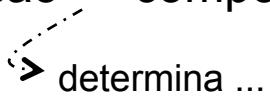
Classes, Interfaces, Generalização, Realização, ...

- ... do pacote *util* do Java



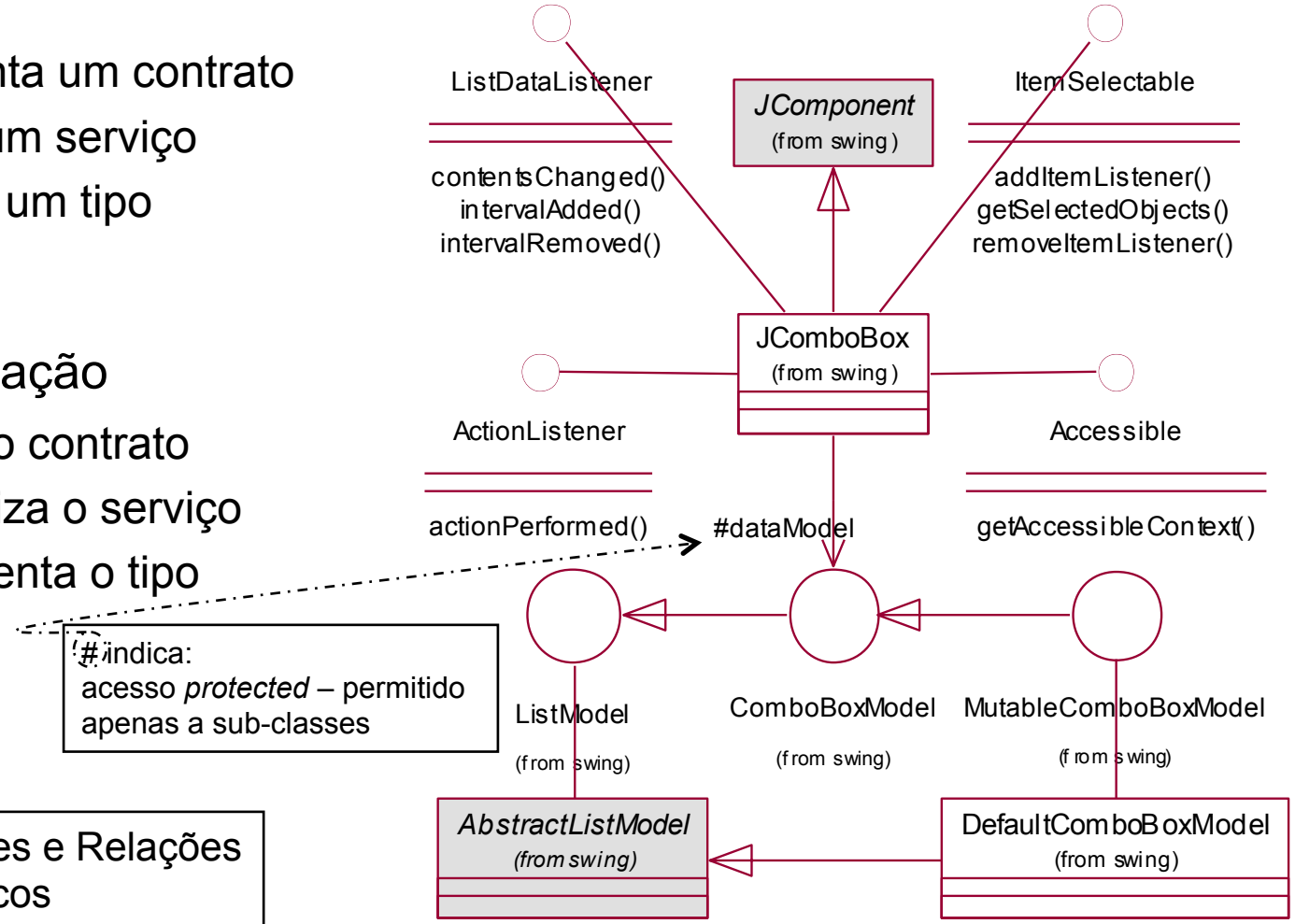
Mecanismos Comuns – "divisão do mundo"

- Na modelação Orientada por Objectos é usual "dividir o mundo" em
  - Interface / Implementação
  - Classe / Objecto
- Interface e Classe
  - Interface tem semelhanças com Classe Virtual
    - nenhuma pode ter instâncias directas
  - Interface é diferente de Classe Virtual (CV)
    - Interface não pode ter atributos (estado); CV pode
    - Interface delimita fronteiras do modelo; CV é Abstracção de topo
  - a mesma Interface pode ser realizada por diferentes
    - Classes (Abstracção lógica); Componentes (Abstracção física)
- Implementação e Objecto
  - implementação → comportamento + estado (de cada Objecto)



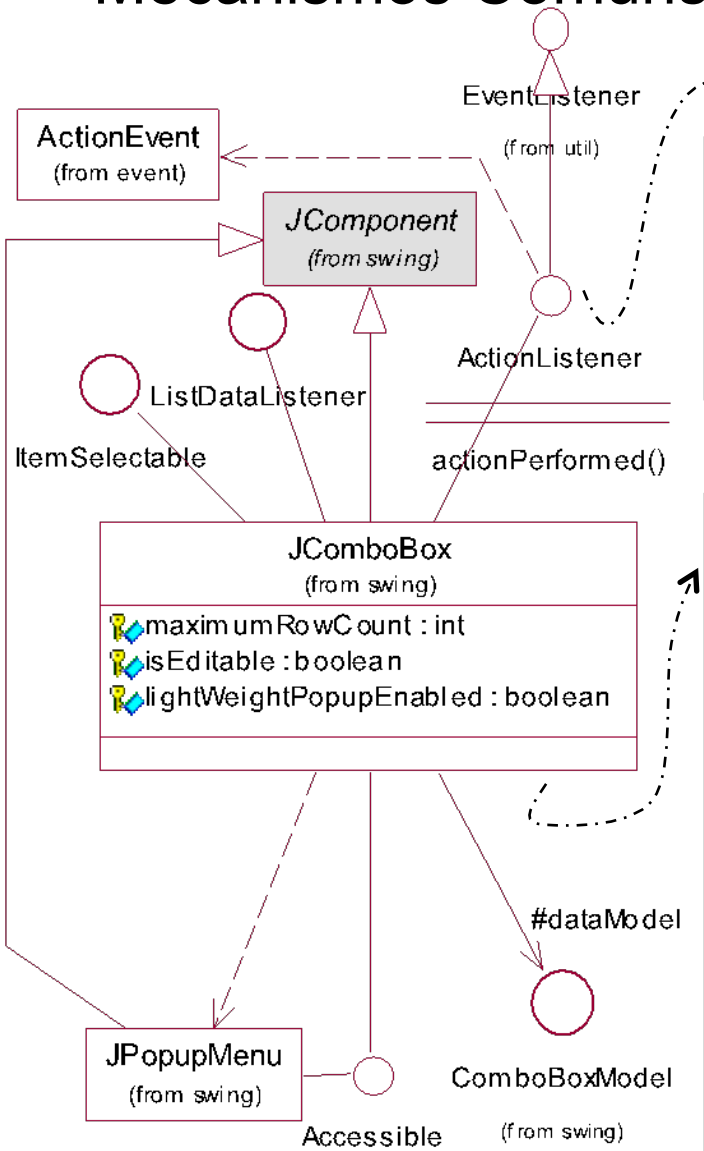
Mecanismos Comuns – Interface / Implementação

- Interface
  - apresenta um contrato
  - define um serviço
  - declara um tipo
  - ...
- Implementação
  - realiza o contrato
  - concretiza o serviço
  - implementa o tipo
  - ...



Classes, Interfaces e Relações
- Aspectos Estáticos
- Diagrama de Classes

Mecanismos Comuns – Interface / Implementação (cont.)



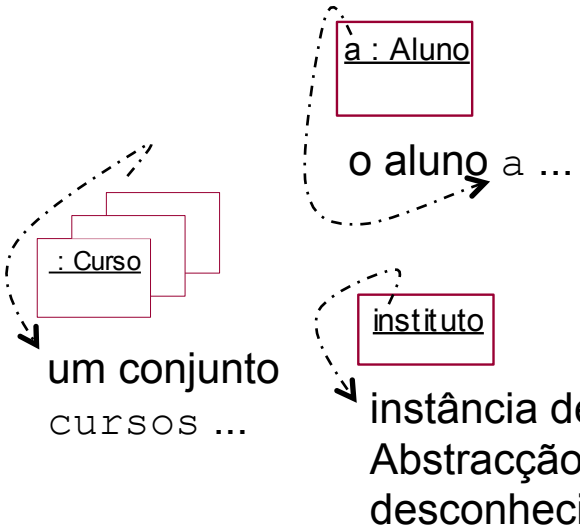

```
public interface ActionListener
extends EventListener
{
   public void actionPerformed(ActionEvent e);
}
```

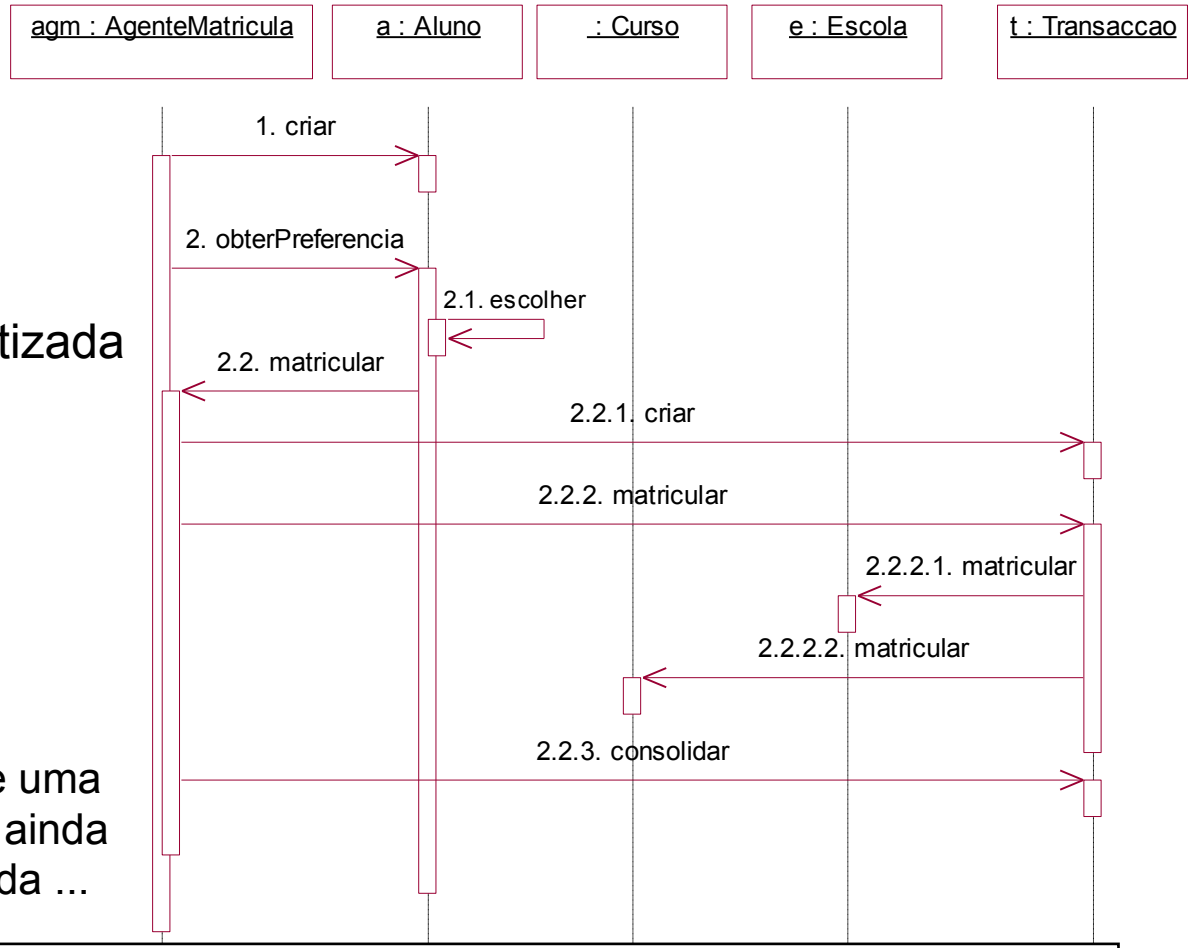
```
public class JComboBox
extends JComponent
implements ItemSelectable, ListDataListener,
           ActionListener, Accessible
{
  protected ComboBoxModel dataModel;
  protected int maximumRowCount = 8;
  protected boolean isEditable  = false;
  protected boolean lightWeightPopupEnabled =
     JPopupMenu.
        getDefaultLightWeightPopupEnabled();
  ...
}
```

Paulo Trigo Silva

Mecanismos Comuns – Classe / Objecto

- Classe
  - uma Abstracção
- Objecto
  - Abstracção concretizada





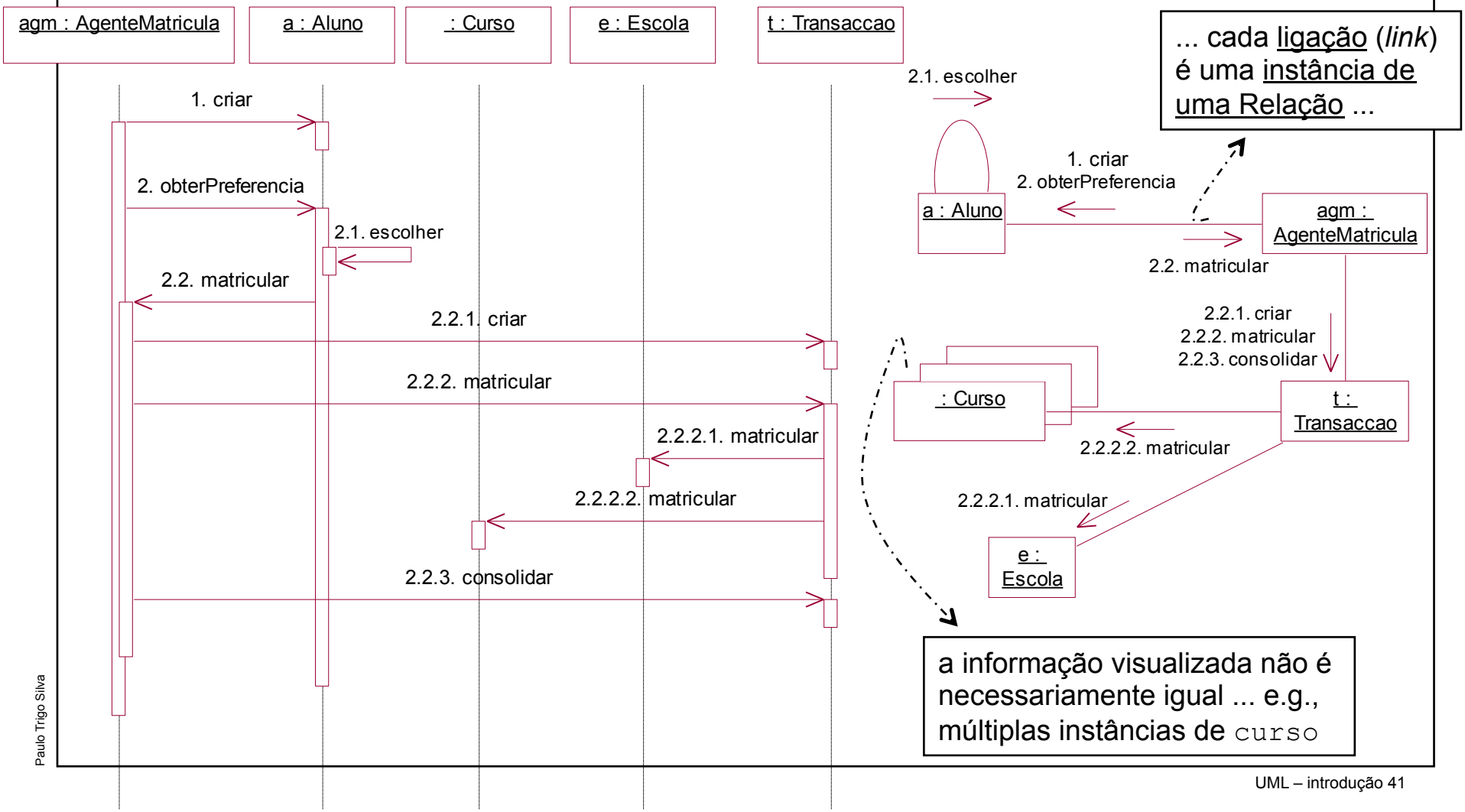
Objectos, Instâncias de Relações (Ligações – *Links*) e Mensagens
- Aspectos Dinâmicos
- Diagrama de Interacção (Sequência e Colaboração – semanticamente equivalentes)

Mecanismos Comuns – Classe / Objecto (cont.)

Diagrama de Sequência – ênfase na ordenação temporal das mensagens

semanticamente equivalentes
conversão automática

Diagrama de Colaboração – ênfase na organização / ligação dos objectos



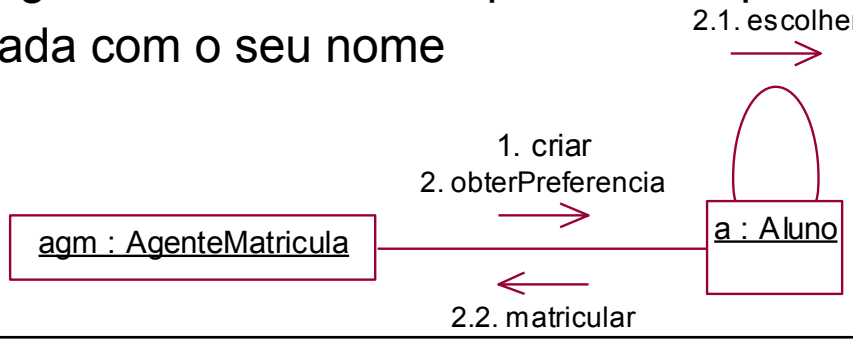
Conceitos de Comportamento

- São a parte "dinâmica" dos modelos UML
  - "acções" / "verbos" do modelo
  - representam comportamento ao longo do tempo e espaço
- Existem 2 espécies de "conceitos de comportamento"
  - Interacção (*Interaction*)
    - ênfase na ordem temporal das mensagens
    - ênfase na descrição do modo como as mensagens fluem, no contexto de determinada organização estrutural entre Objectos
    - ... Diagramas de Interacção – Sequência e Colaboração
  - Máquina de Estados (*State Machine*)
    - ênfase na descrição do ciclo de vida de um Objecto – que pode ser uma instância de classe, um caso de utilização ou todo o sistema
    - ... Diagramas de Transição de Estados

Paulo Trigo Silva

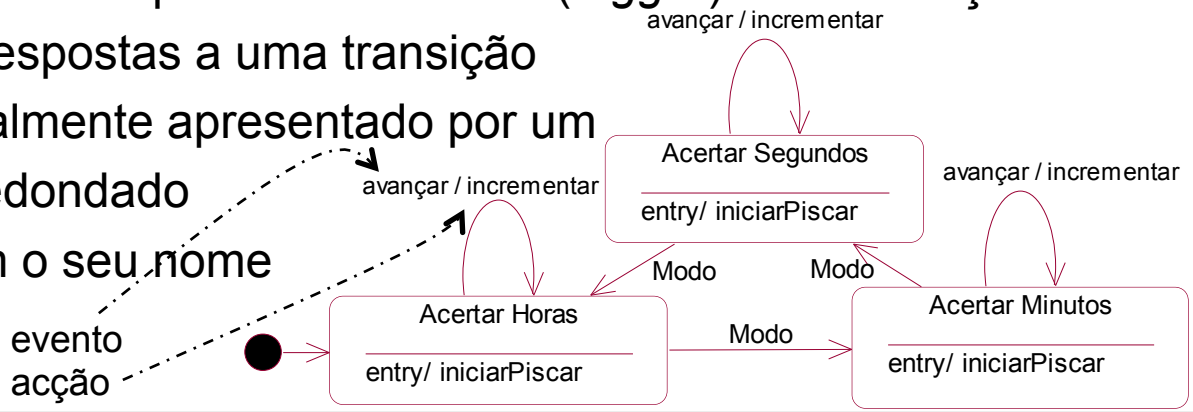
Conceitos de Comportamento – Interacção

- Interacção (*Interaction*)
  - num contexto particular e para atingir determinado objectivo, inclui,
    - o conjunto de mensagens trocadas entre objectos nesse contexto
  - especifica o comportamento
    - de uma "sociedade" de objectos, ou
    - de uma operação individual
  - uma interacção envolve outros elementos,
    - mensagens
    - sequência de acções – fluxo desencadeado pela mensagem
    - associações (*links*) – ligações entre objectos
  - uma mensagem é normalmente apresentada por uma linha direccionada
    - etiquetada com o seu nome



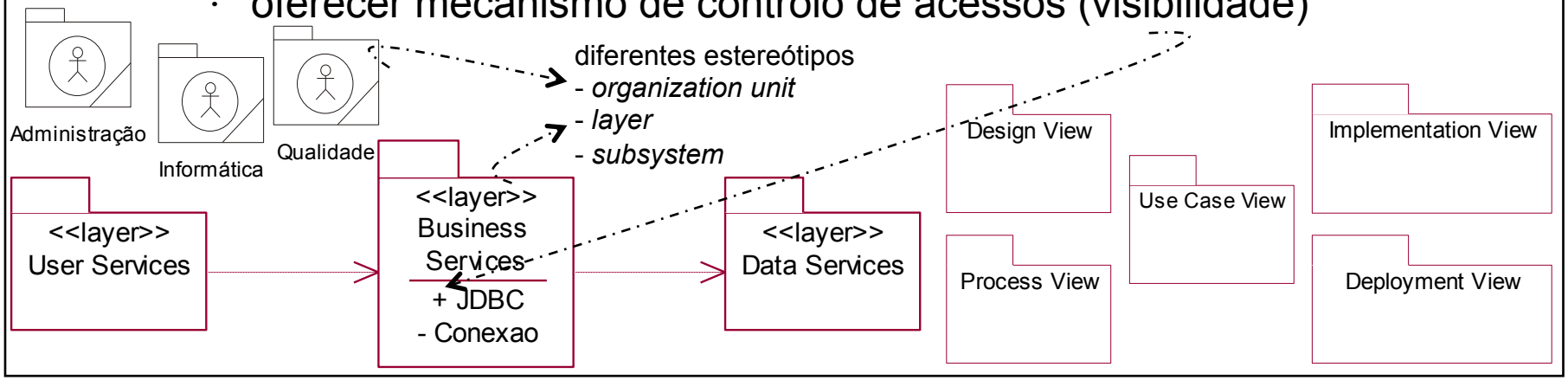
Conceitos de Comportamento – Máquina de Estados

- Máquina de Estados (*State Machine*)
  - especifica a sequência de estados que um Objecto ou uma Interacção
    - percorre em resposta a eventos,
    - durante a sua vida
  - o comportamento de uma Classe ou Interacção de Classes
    - pode ser especificado por uma Máquina de Estados
  - uma Máquina de Estados envolve outros elementos,
    - estado
    - transições – fluxos entre estados
    - eventos – elementos que desencadeiam (*trigger*) uma transição
    - actividades – respostas a uma transição
  - um estado é normalmente apresentado por um
    - rectângulo arredondado
    - etiquetada com o seu nome



Conceitos de Agrupamento

- Elemento organizacional, designado por pacote – *package*
  - permite agrupar diferentes elementos que, façam sentido na perspectiva,
    - semântica ou estrutural
  - um *package* pode conter
    - classes, interfaces, casos de utilização, ... e mesmo outros pacotes
  - um elemento está definido num único *package*
  - algumas das vantagens na sua utilização são,
    - facilitar gestão e procura de artefactos
    - evitar conflitos de nomes – X::A é diferente de X::Y::A e de Z::A
    - oferecer mecanismo de controlo de acessos (visibilidade)



Conceitos de Anotação

- Anotações ou Notas são os "adornos autónomos" mais importantes
  - autónomos – podem existir não associados a qualquer outro elemento
    - outro exemplo de "adorno" – Qualificação é adorno de Associação
  - são comentários (textos e/ou gráficos) usados para,
    - descrever, realçar, ilustrar ... algum elemento do modelo
  - o seu conteúdo não altera o significado do modelo em que encontra
  - é normal utilizar Anotações para descrever informalmente,
    - algum detalhe dos requisitos; observações; revisões; explicações; ...
    - restrições adicionais a impor ao modelo
  - algumas considerações na utilização de Anotações,
    - localização – graficamente perto dos elementos que descrevem
    - evolução – as mais antigas devem ser eliminadas (modelo evolui)
    - extensão – referir outros documentos para comentários extensos