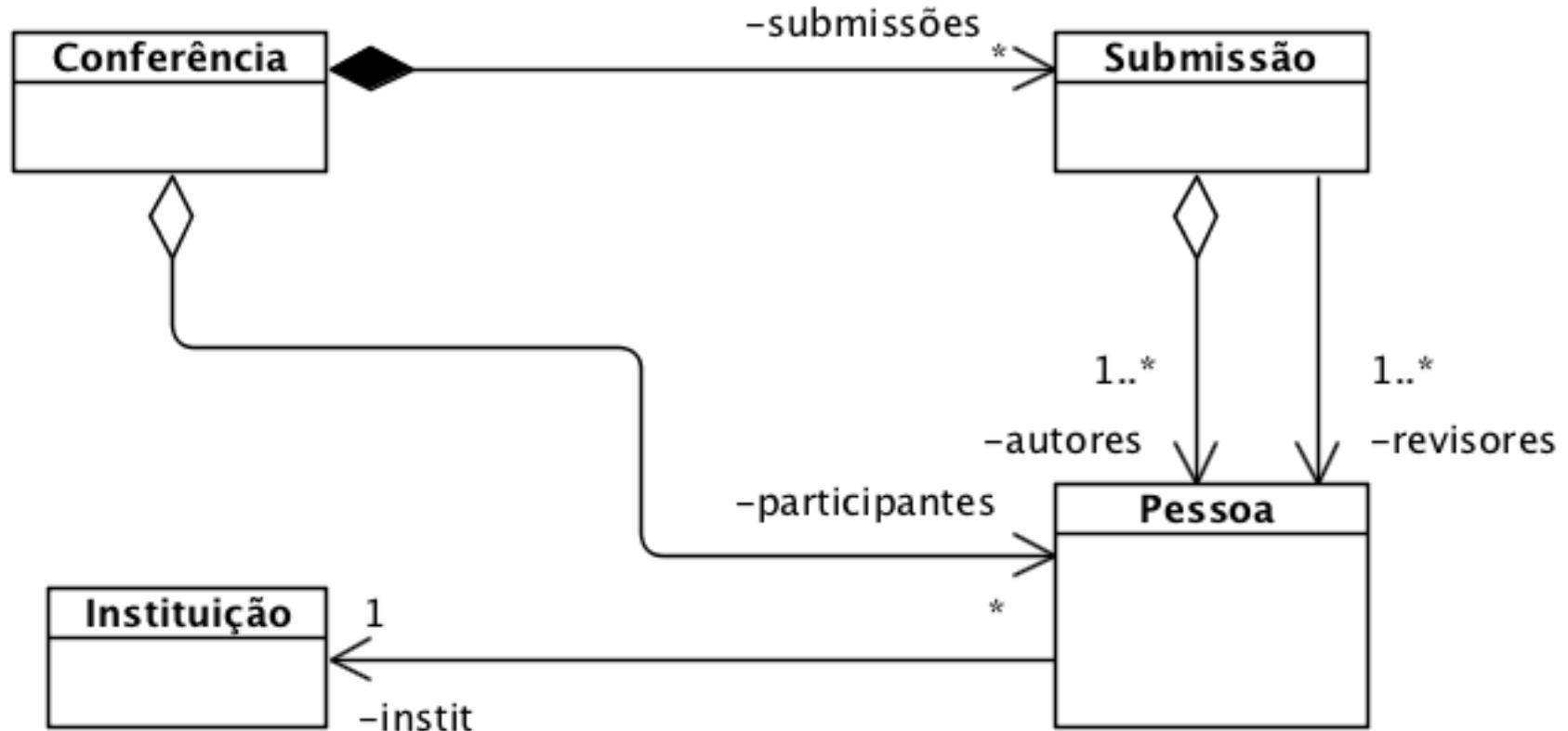




Desenvolvimento de Sistemas Software

Aula Teórica 26: OCL

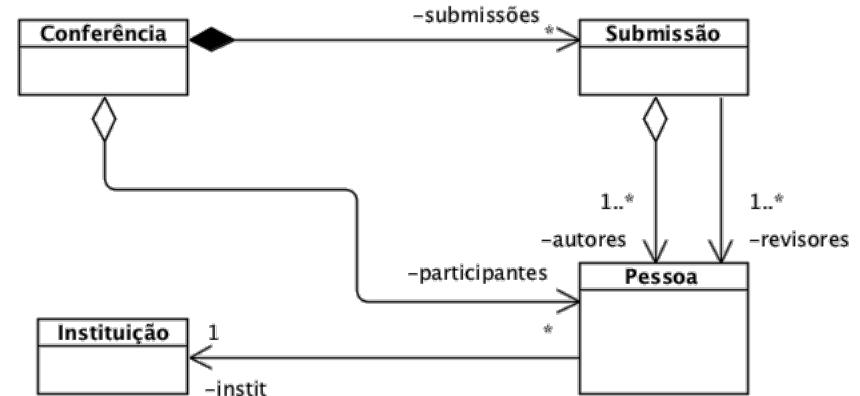
Diagramas UML nem sempre são suficientes





Diagramas UML nem sempre são suficientes

1. Os revisores de uma submissão não podem ser seus autores!
 2. Os revisores de uma submissão não podem ser da mesma instituição dos autores!
- Como expressar estas restrições de forma não ambígua e “mecanizável” ?



OCL: Object Constraint Language



Breve História da OCL

- Em 1995 a divisão de seguros da IBM desenvolve uma linguagem para modelação de negócio
 - IBEL (Integrated Business Engineering Language)
 - IBM propõe a IBEL ao OMG
 - OCL integrado na UML 1.1
 - A OCL foi utilizada para definir a UML 1.2



Empresas...

- Rational Software
- Microsoft
- Hewlett-Packard
- Oracle
- Sterling Software
- MCI Systemhouse
- Unisys
- ICON Computing
- IntelliCorp
- i-Logix
- IBM
- ObjecTime
- Platinum Technology
- Ptech
- Taskon
- Reich Technologies
- Softeam



Vantagens de Formalizar as Restrições

- Melhor documentação
 - As restrições adicionam informação àcerca dos elementos e suas relações aos modelos visuais da UML
 - Permitem documentar o modelo
- Maior precisão
 - As restrições escritas em OCL têm uma semântica formal
 - Ajudam a diminuir a ambiguidade dos modelos
- Melhor Comunicação
 - Se os modelos UML são utilizados para comunicar, as restrições OCL permitem comunicar sem ambiguidade

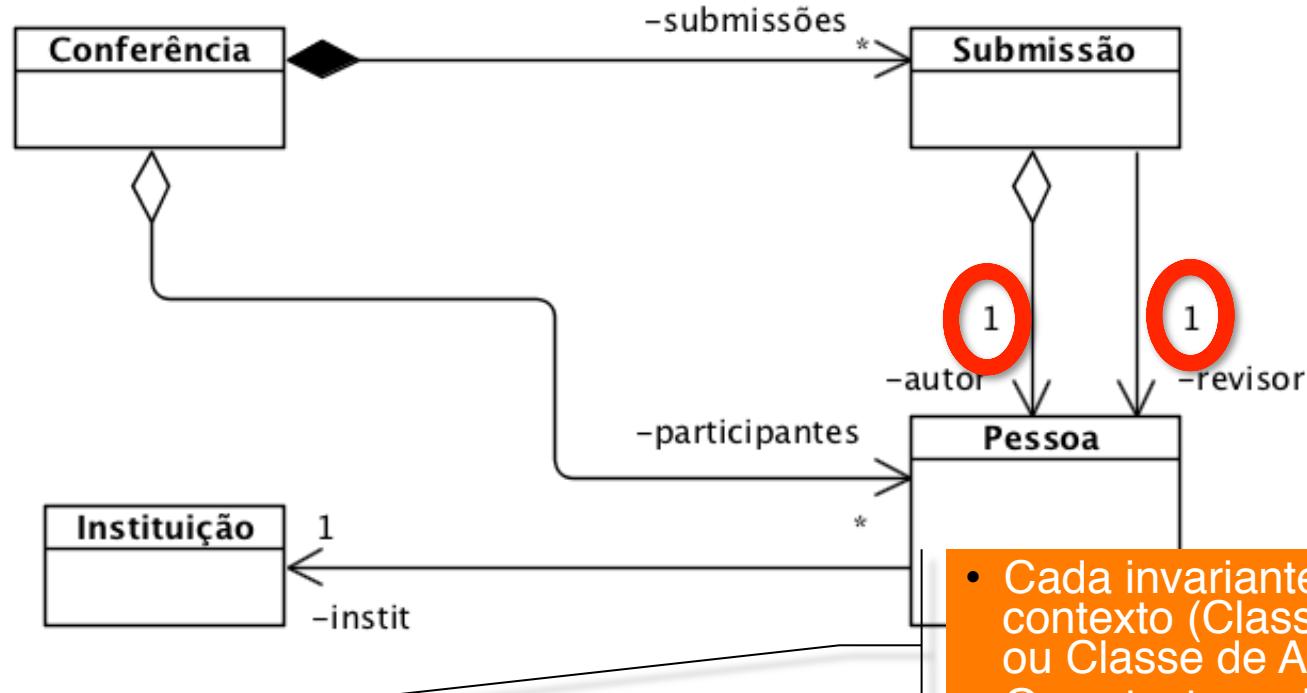


Onde utilizar OCL?

- Invariantes de classe e tipos
- Pré- e pós-condições dos métodos
- Restrições em operações e associações
- Requisitos e especificação de testes

Invariante

1. Os revisores de uma submissão não podem ser seus autores



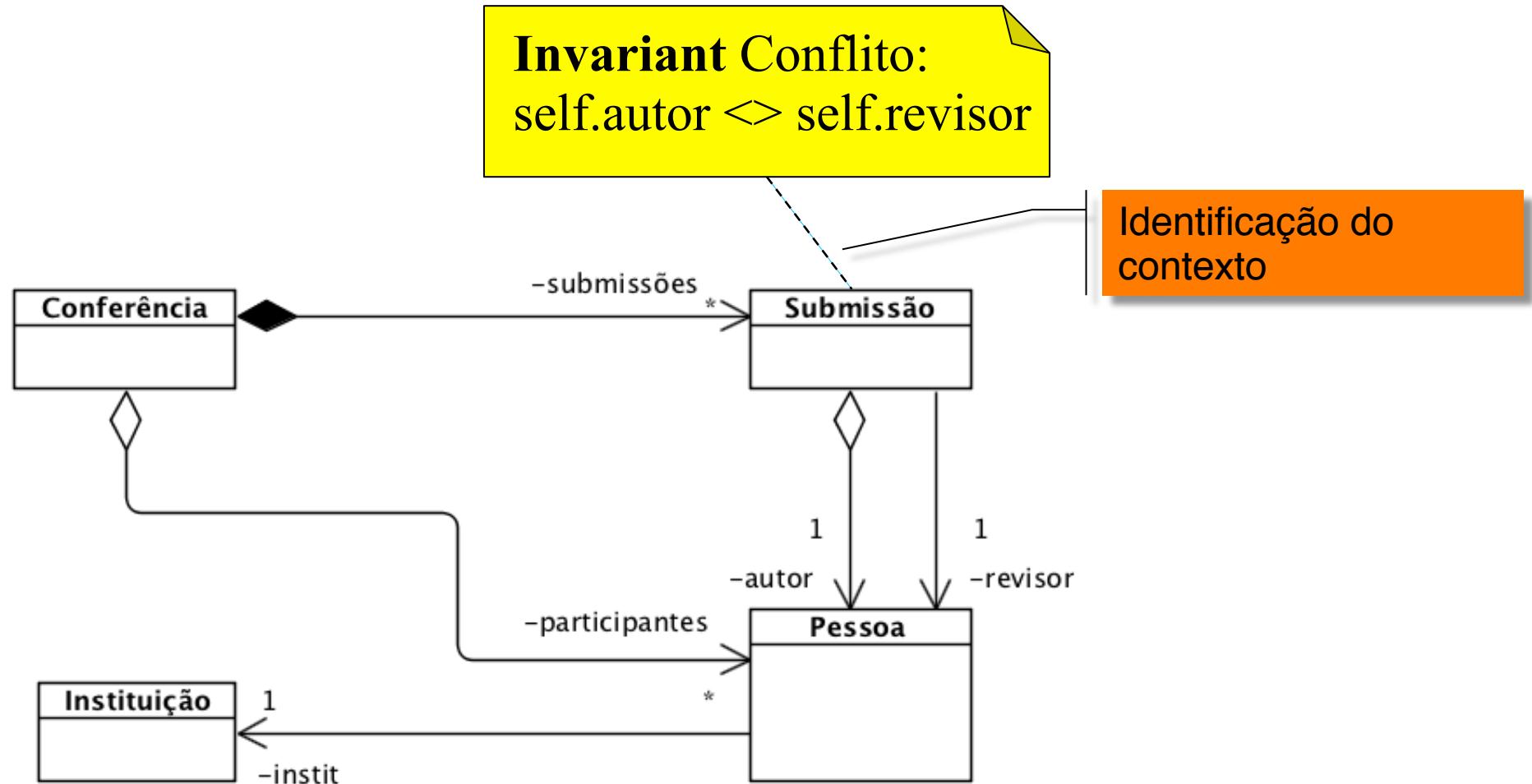
Context Submissão

Invariant Conflito: `self.autor <> self.revisor`

- Cada invariante tem um contexto (Classe, Interface ou Classe de Associação)
- O contexto pode ser referido utilizando *self* (opcional)
- Os invariantes podem ter um nome (neste caso é Conflito)
- Os invariantes são expressões booleanas

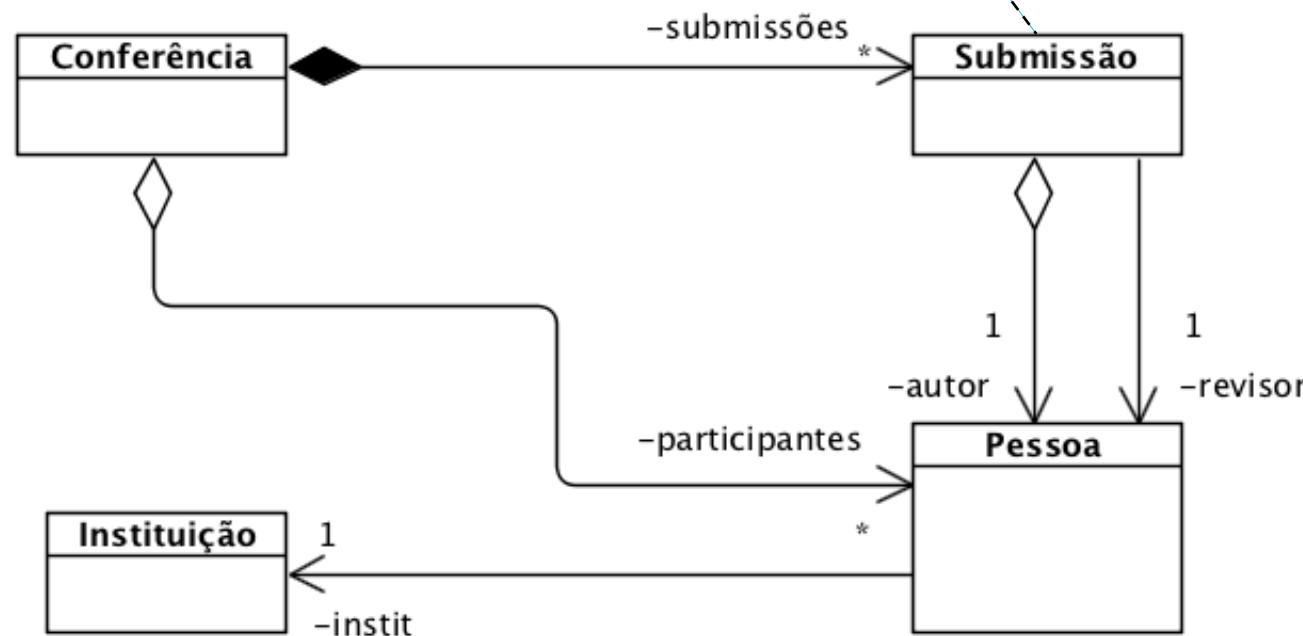
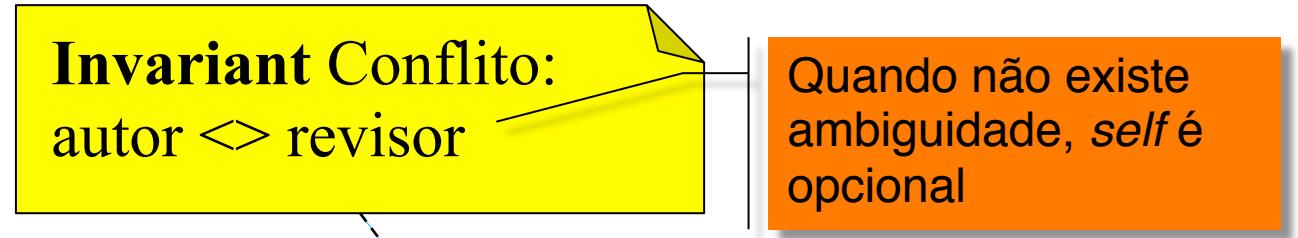
Invariante

- Os revisores de uma submissão não podem ser seus autores



Invariante

- Os revisores de uma submissão não podem ser seus autores



Pré- e pós-condições

Podem referir-se atributos nos invariantes

Invariant: saldo ≥ 0

Cartão de Pontos

- saldo: int

+ acumula(p: int)

+ gasta(p: int)

+ vazio(): boolean

«precondition»
 $p > 0$

«postcondition»
saldo = saldo@pre + p

«precondition»
saldo $\geq p$ and $p \geq 0$

«postcondition»
saldo = saldo@pre - p

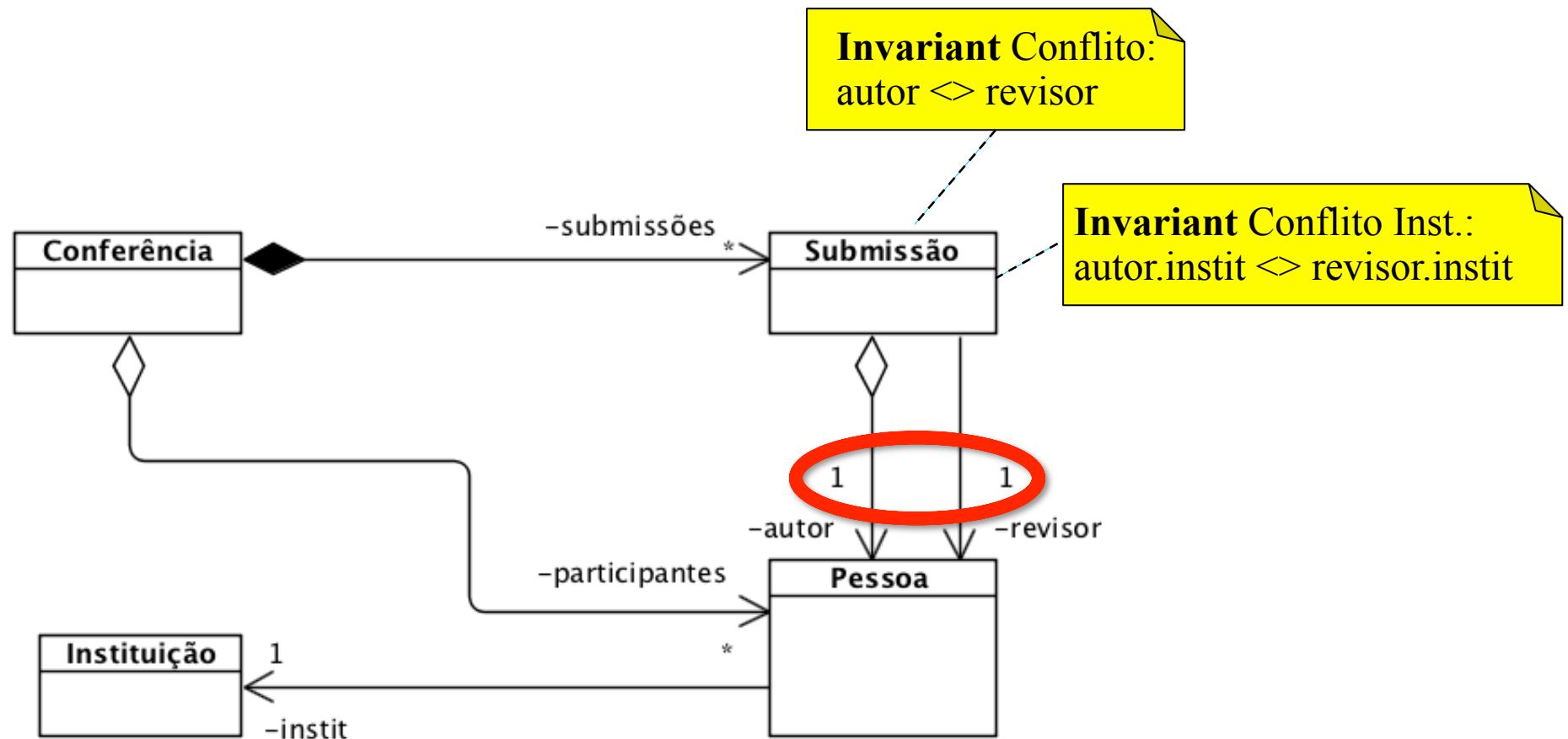
saldo@pre: valor do saldo antes da operação

«postcondition»
result = (saldo = 0)

result: resultado da operação

Navegação nas associações

1. Os revisores de uma submissão não podem ser seus autores
2. Os revisores de uma submissão não podem ser da mesma instituição dos autores



Sistema de tipos OCL

- Tipos primitivos

Type	Description	Values	Operations
Boolean		true, false	=, <>, and, or, xor, not, implies, if-then-else-endif
Integer	A whole number of any size	-1, 0, 1, ...	=, <>, >, <, >=, <=, *, +, - (unary), - (binary), / (real) abs(), max(b), min(b), mod(b), div(b)
Real	A real number of any size	1.5, ...	=, <>, >, <, >=, <=, *, +, - (unary), - (binary), / abs(), max(b), min(b), round(), floor()
String	A string of characters	'a', 'John'	=, <> size(), concat(s2), substring(from, to) (1<=from<=upper<=size), toReal(), toInteger()

Sistema de tipos OCL

- Colecções e Tuplos

Description	Syntax	Examples
Abstract collection of elements of type T	Collection(T)	
Unordered collection, no duplicates	Set(T)	Set{1 , 2}
Ordered collection, duplicates allowed	Sequence(T)	Sequence {1, 2, 1} Sequence {1..4} (same as {1,2,3,4})
Ordered collection, no duplicates	OrderedSet(T)	OrderedSet {2, 1}
Unordered collection, duplicates allowed	Bag(T)	Bag {1, 1, 2}
Tuple (with named parts)	 Tuple(field1: T1, ... fieldn : Tn)	Tuple {age: Integer = 5, name: String = 'Joe' } Tuple {name = 'Joe', age = 5}

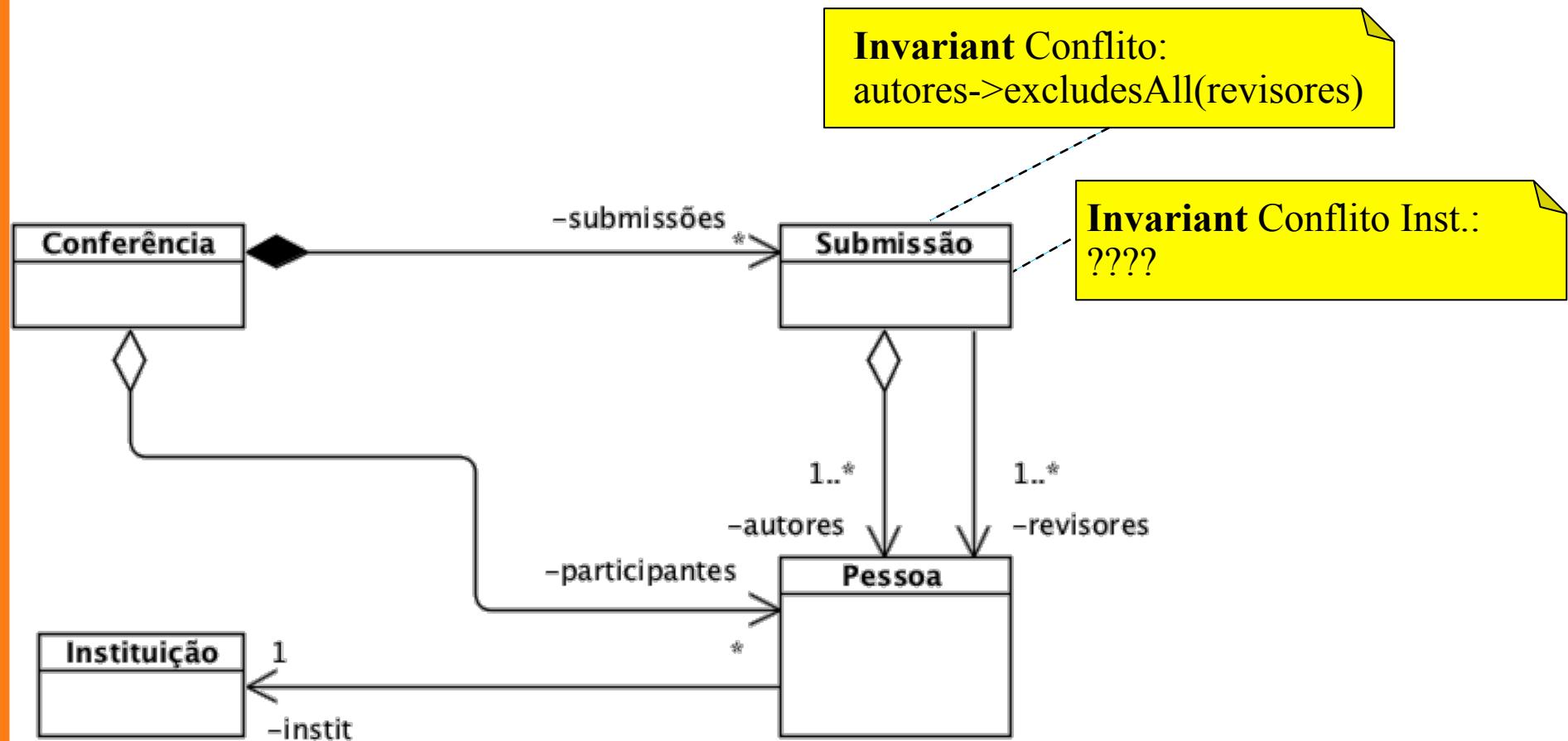
Colecções - Operações

Operation	Description
size(): Integer	The number of elements in this collection (<i>self</i>)
isEmpty(): Boolean	<i>size</i> = 0
notEmpty(): Boolean	<i>size</i> > 0
includes(object: T): Boolean	True if <i>object</i> is an element of <i>self</i>
excludes(object: T): Boolean	True if <i>object</i> is not an element of <i>self</i>
count(object: T): Integer	The number of occurrences of <i>object</i> in <i>self</i>
includesAll(c2: Collection(T)): Boolean	True if <i>self</i> contains all the elements of <i>c2</i>
excludesAll(c2: Collection(T)): Boolean	True if <i>self</i> contains none of the elements of <i>c2</i>
sum(): T	The addition of all elements in <i>self</i> (T must support "+")
product(c2: Collection(T2)) : Set(Tuple(first:T, second:T2))	The cartesian product operation of <i>self</i> and <i>c2</i> .

Note: Operations on collections are applied with “->” and not “.”

Colecções - exemplos

1. Os revisores de uma submissão não podem ser seus autores
2. Os revisores de uma submissão não podem ser da mesma instituição dos autores



Colecções - iteradores (1/2)

Iterator expression	Description
iterate(iterator: T; accum: T2 = init body)	Returns the final value of an accumulator that, after initialization, is updated with the value of the <i>body</i> expression for every element in the <i>source</i> collection.
exists(iterators body) : Boolean	True if <i>body</i> evaluates to true for at least one element in the <i>source</i> collection. Allows multiple iterator variables.
forAll(iterators body): Boolean	True if <i>body</i> evaluates to true for each element in the <i>source</i> collection. Allows multiple iterator variables.
one(iterator body): Boolean	True if there is exactly one element in the <i>source</i> collection for which <i>body</i> is true
isUnique(iterator body): Boolean	Results in true if <i>body</i> evaluates to a different value for each element in the <i>source</i> collection.
any(iterator body): T	Returns any element in the <i>source</i> collection for which <i>body</i> evaluates to true. The result is null if there is none.
collect(iterator body): Collection(T2)	The Collection of elements resulting from applying <i>body</i> to every member of the <i>source</i> set. The result is flattened.

Note: The iterator variable declaration can be omitted when there is no ambiguity.

Colecções - iteradores (2/2)

Iterator expression	Description
select(iterator body): Collection(T)	The Collection of elements of the <i>source</i> collection for which <i>body</i> is true. The result collection is of the same type of the <i>source</i> collection.
reject(iterator body): Collection(T)	The Collection of elements of the <i>source</i> collection for which <i>body</i> is false. The result collection is of the same type of the <i>source</i> collection.
collectNested(iterator body): CollectionWithDuplicates(T2))	The Collection of elements (allowing duplicates) that results from applying <i>body</i> (of type T2) to every member of the <i>source</i> collection. The result is not flattened. Collection type conversions: Set -> Bag, OrderedSet -> Sequence.
sortedBy(iterator body): OrderedCollection(T)	Returns an ordered Collection of all the elements of the <i>source</i> collection by ascending order of the value of the <i>body</i> expression. The type T2 of the <i>body</i> expression must support “<”. Collection type conversions: Set -> OrderedSet, Bag -> Sequence.

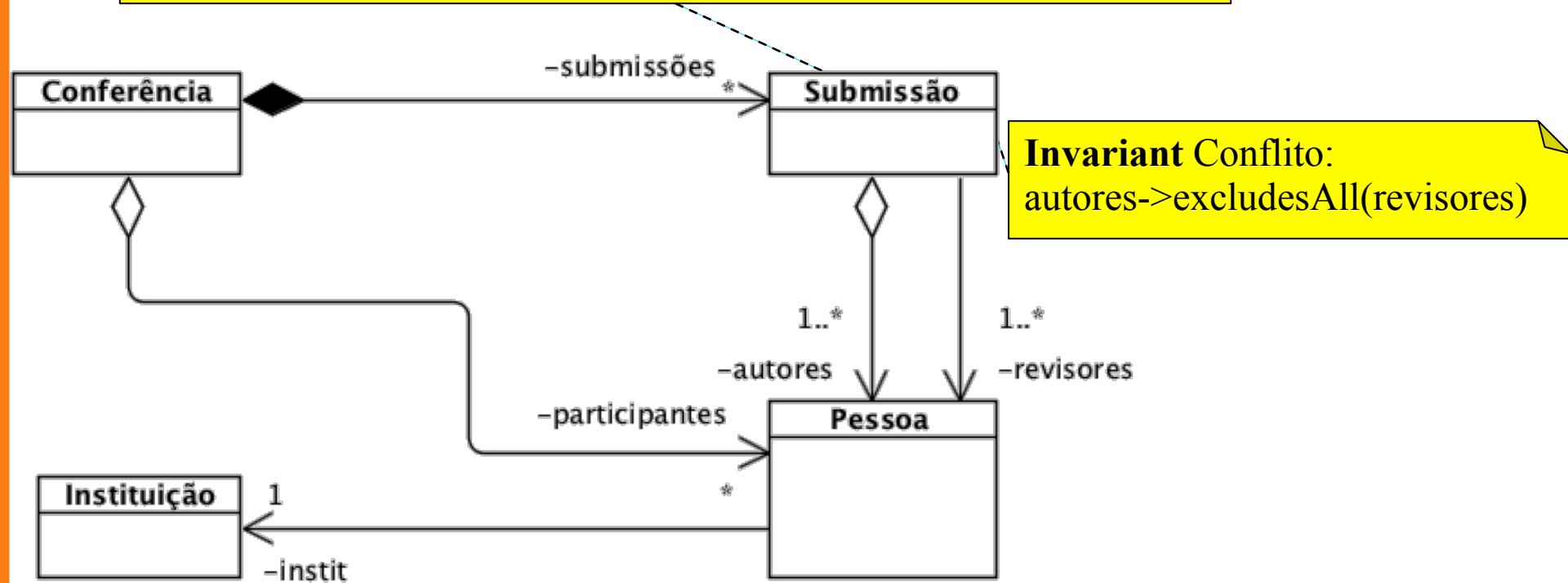
Colecções - exemplos

1. Os revisores de uma submissão não podem ser seus autores
2. Os revisores de uma submissão não podem ser da mesma instituição

Se o iterador não é ambíguo, pode ser omitido (para autores apresenta-se a notação completa)

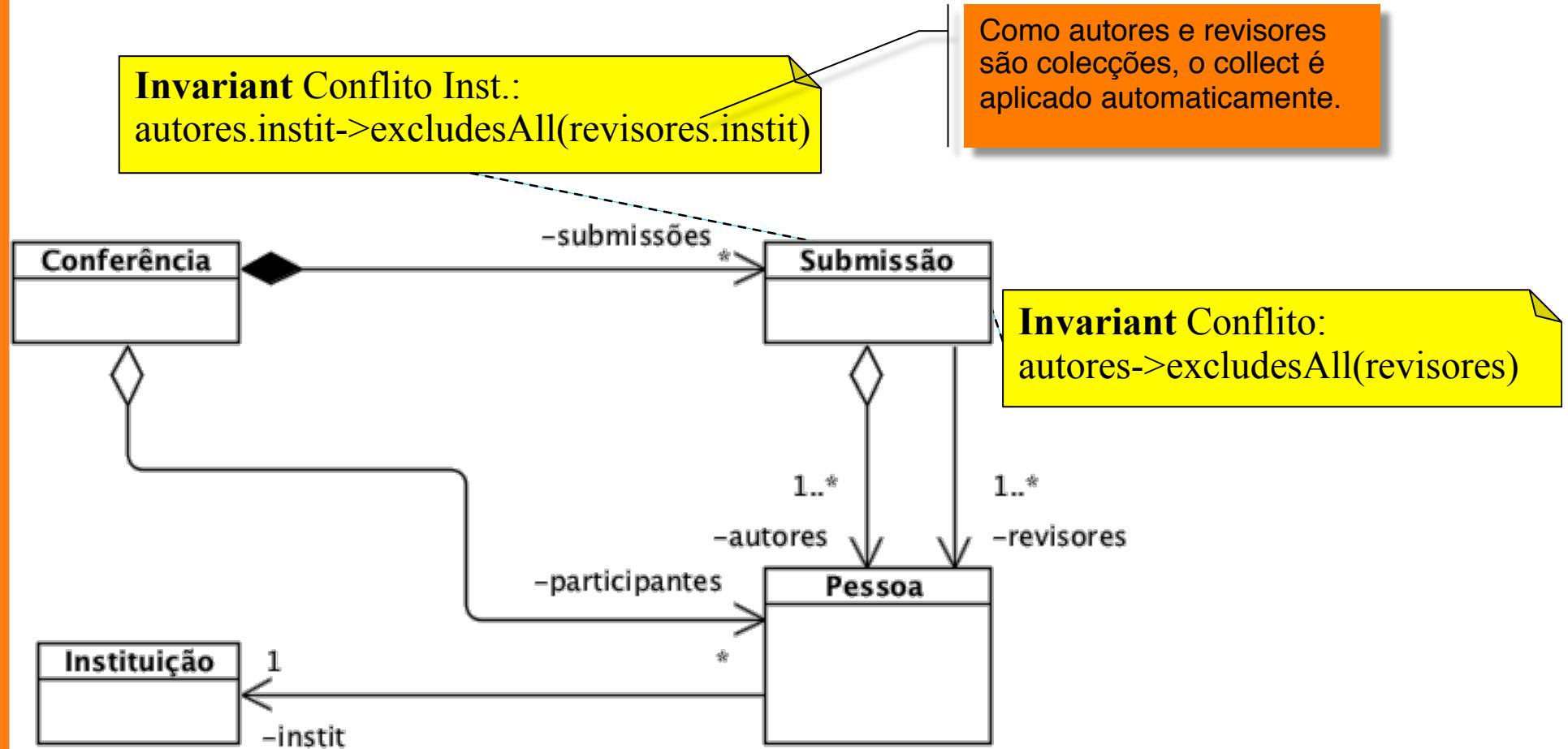
Invariant Conflito Inst.:

(autores->collect(a|a.instit))->excludesAll(revisores->collect(instit))



Colecções - exemplos

1. Os revisores de uma submissão não podem ser seus autores
2. Os revisores de uma submissão não podem ser da mesma instituição dos autores





Tipos de Colecções

- Colecções podem ser
 - Set
 - Sequence
 - Bag
- Cada um tem operações apropriadas ao tipo (herdam as das colecções)
 - Set: =, union, intersection, -(difference), ...
 - Bag: =, union, intersection, flatten, ...
 - Sequence: =, append, prepend, insertAt, subSequence, ...



OCL não é consensual

- Considerado demasiado orientado à implementação e, por vezes, demasiado verboso.
- Utilização de operações nas restrições levanta problemas
- As expressões em OCL são *locais*
- “*I would rather use plain English*” (Martin Fowler)
- No entanto... é a única norma que existe para UML!



OCL

Sumário

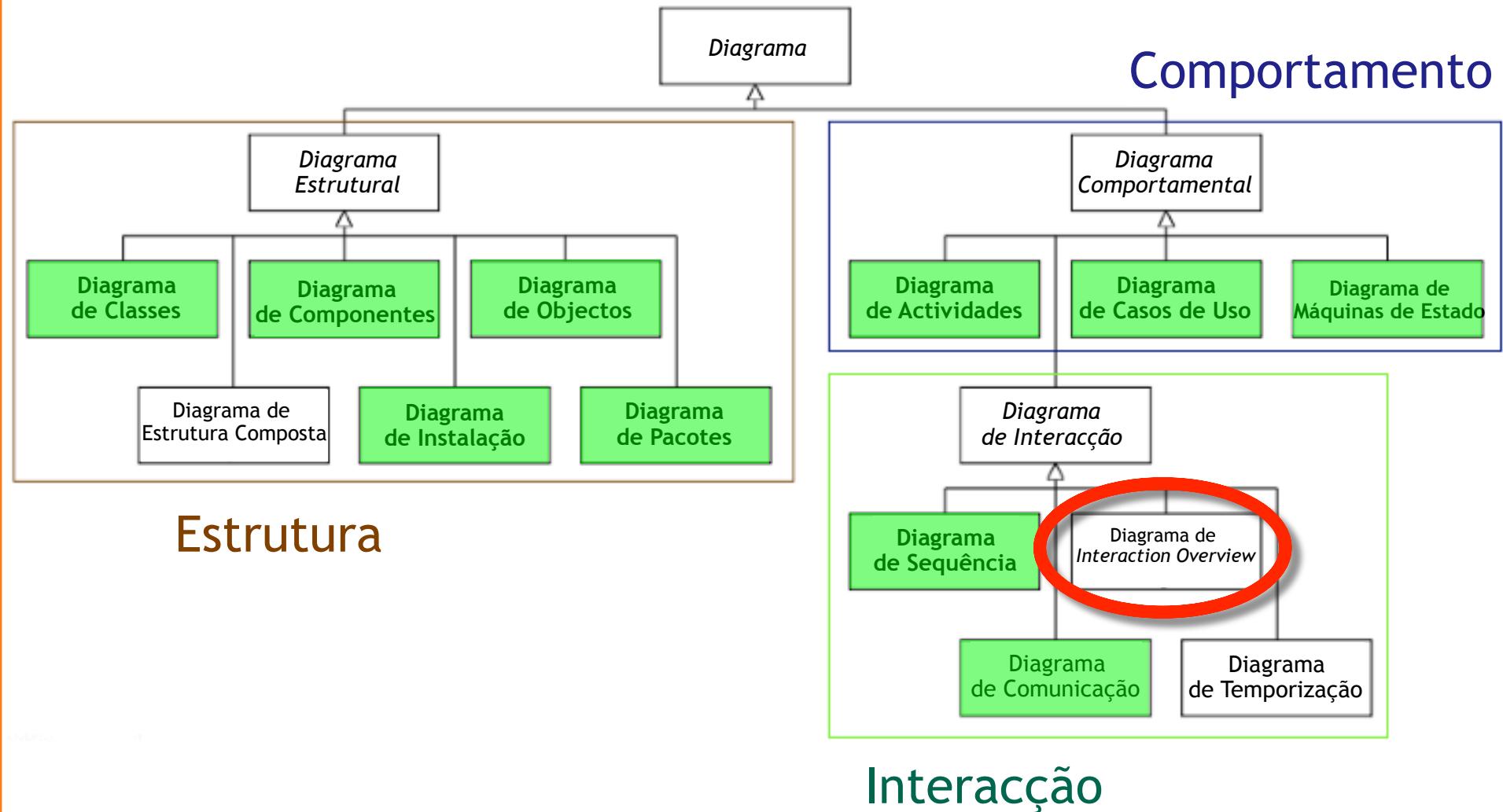
- Necessidade da OCL
- Breve história da OCL
- Invariantes, pré- e pós-condições
- Sistemas de tipos
 - Colecções
- Limitações e Críticas



Desenvolvimento de Sistemas Software

Aula Teórica 26: Três diagramas

Diagramas da UML 2.x

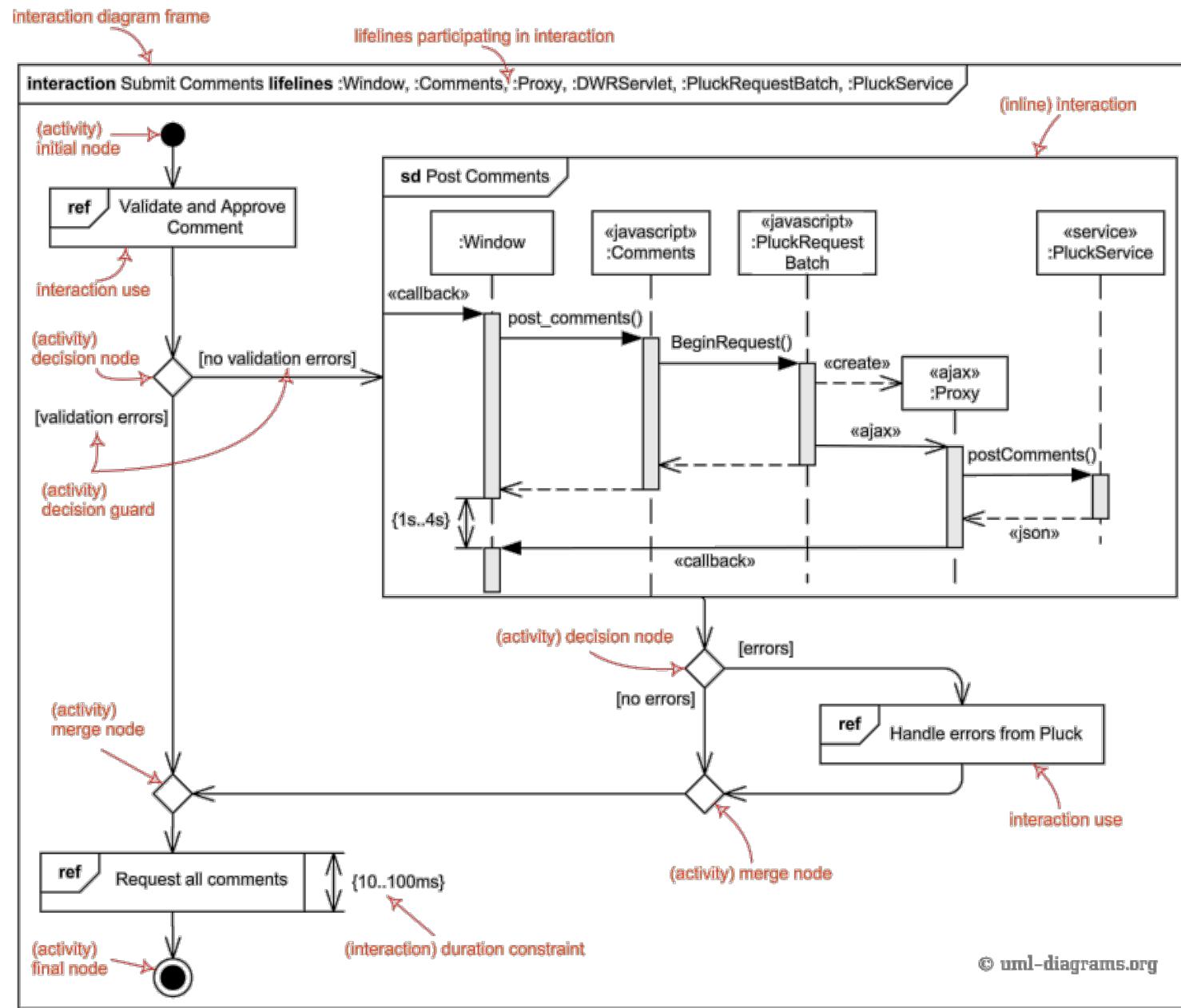




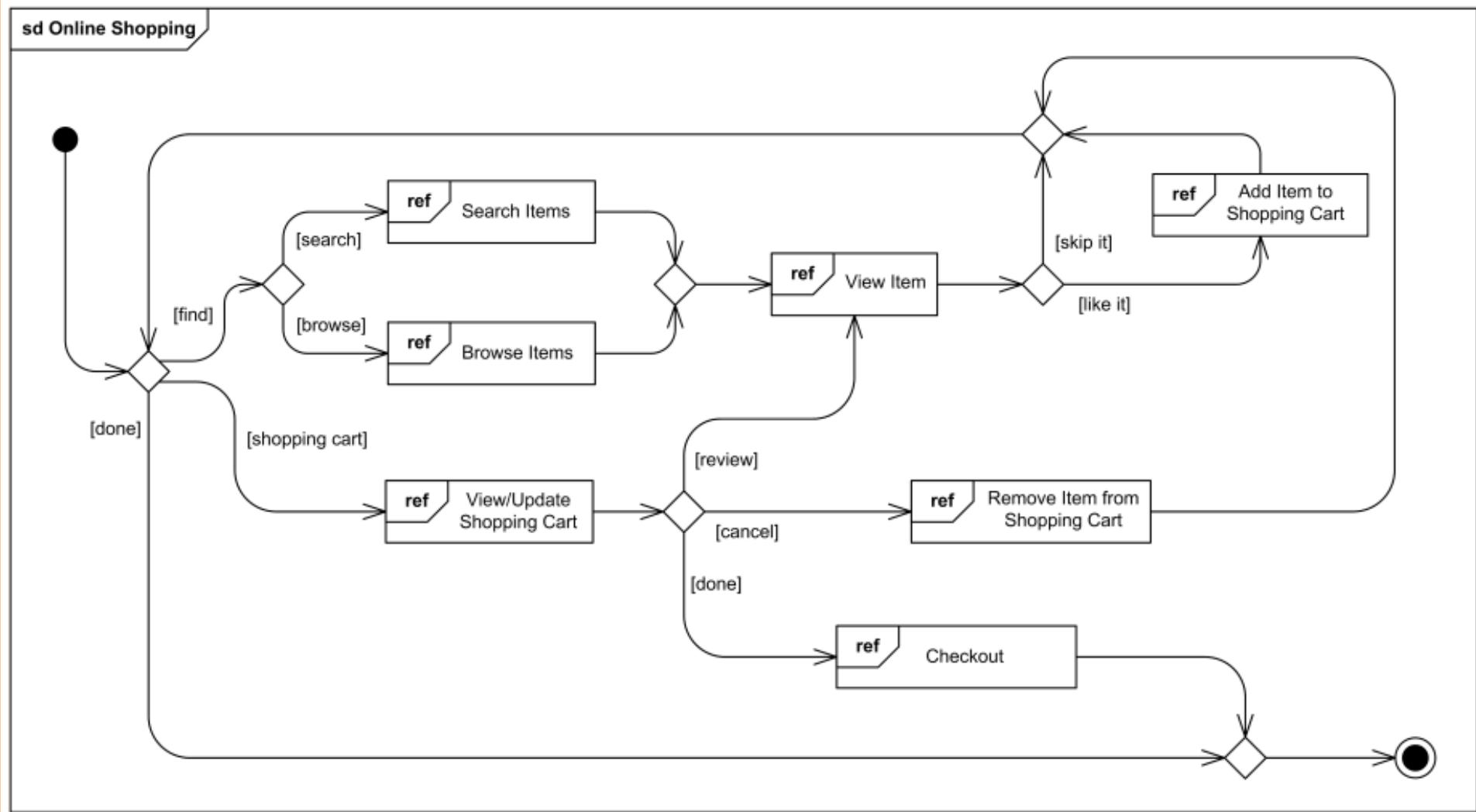
Interaction Overview Diagrams

- Similares a Diagramas de Actividade
- Nodos de actividade substituídos por interacções (Diagramas de Sequência)
- Permitem uma visão de alto nível sobre o fluxo de controlo entre interacções
- Poderiam ser utilizado como uma alternativa (com maior poder expressivo) aos DSS para especificar Use Cases

Interaction Overview Diagram - Exemplo

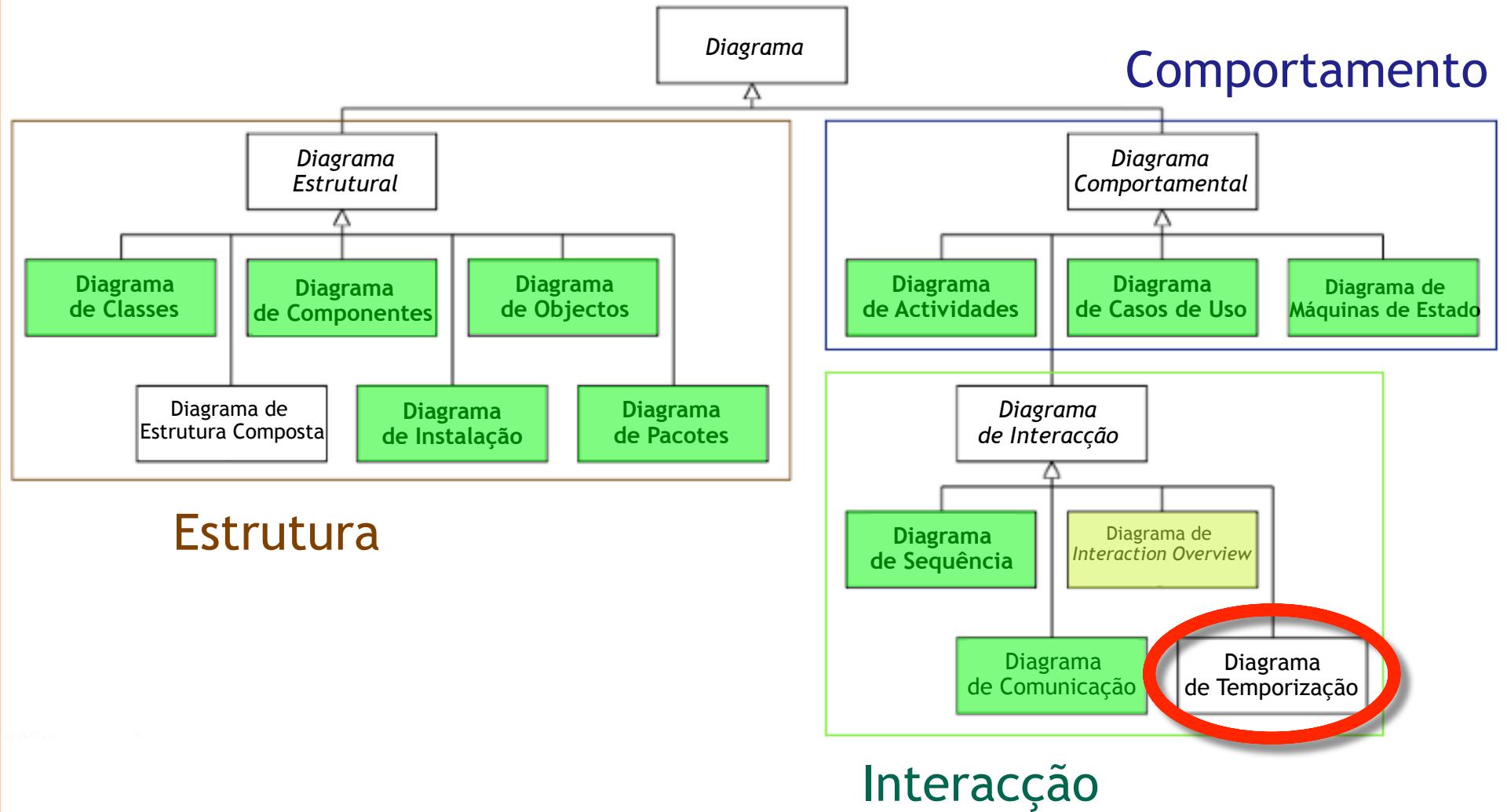


Outro Exemplo





Diagramas da UML 2.x





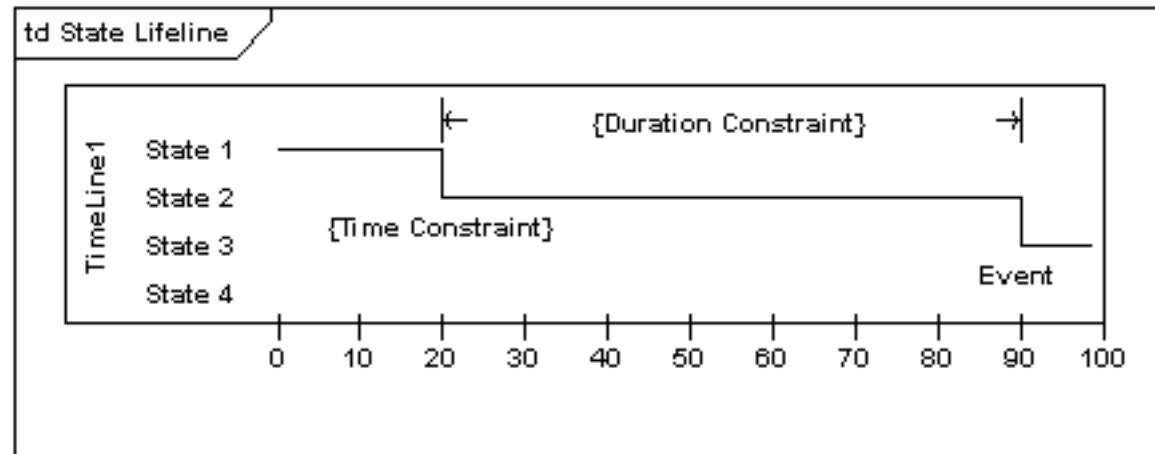
Timing diagram

- Descrição de comportamento focada em restrições temporais
- Uma forma especial de diagrama de sequência
 - Eixos estão invertidos: tempo fluí da esquerda para a direita
 - Linhas de vida mostradas horizontalmente
 - Linhas de vida podem mostrar estado ou valor (do objecto que a linha de vida representa)

Timing diagram

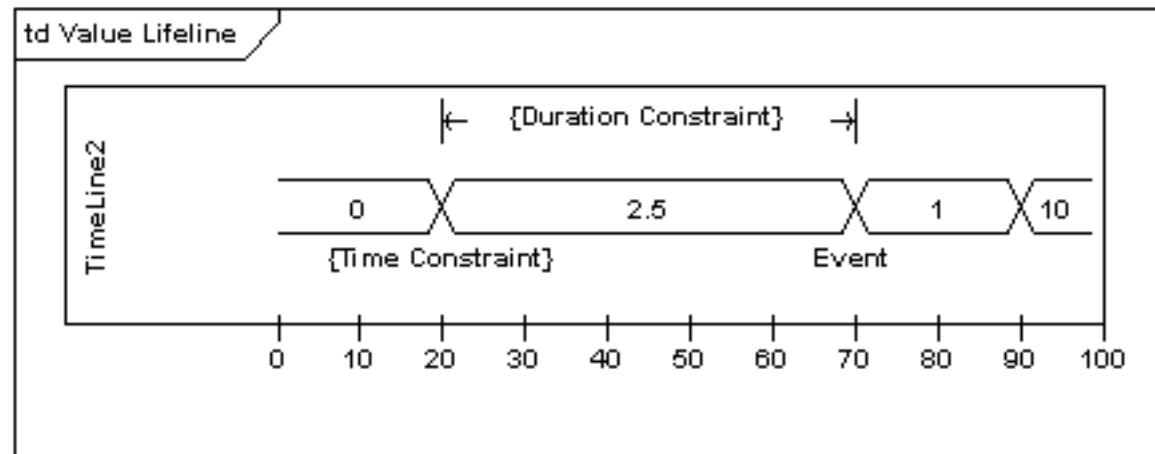
- Linha e vida de estado

- mostra o estado de uma entidade ao longo do tempo



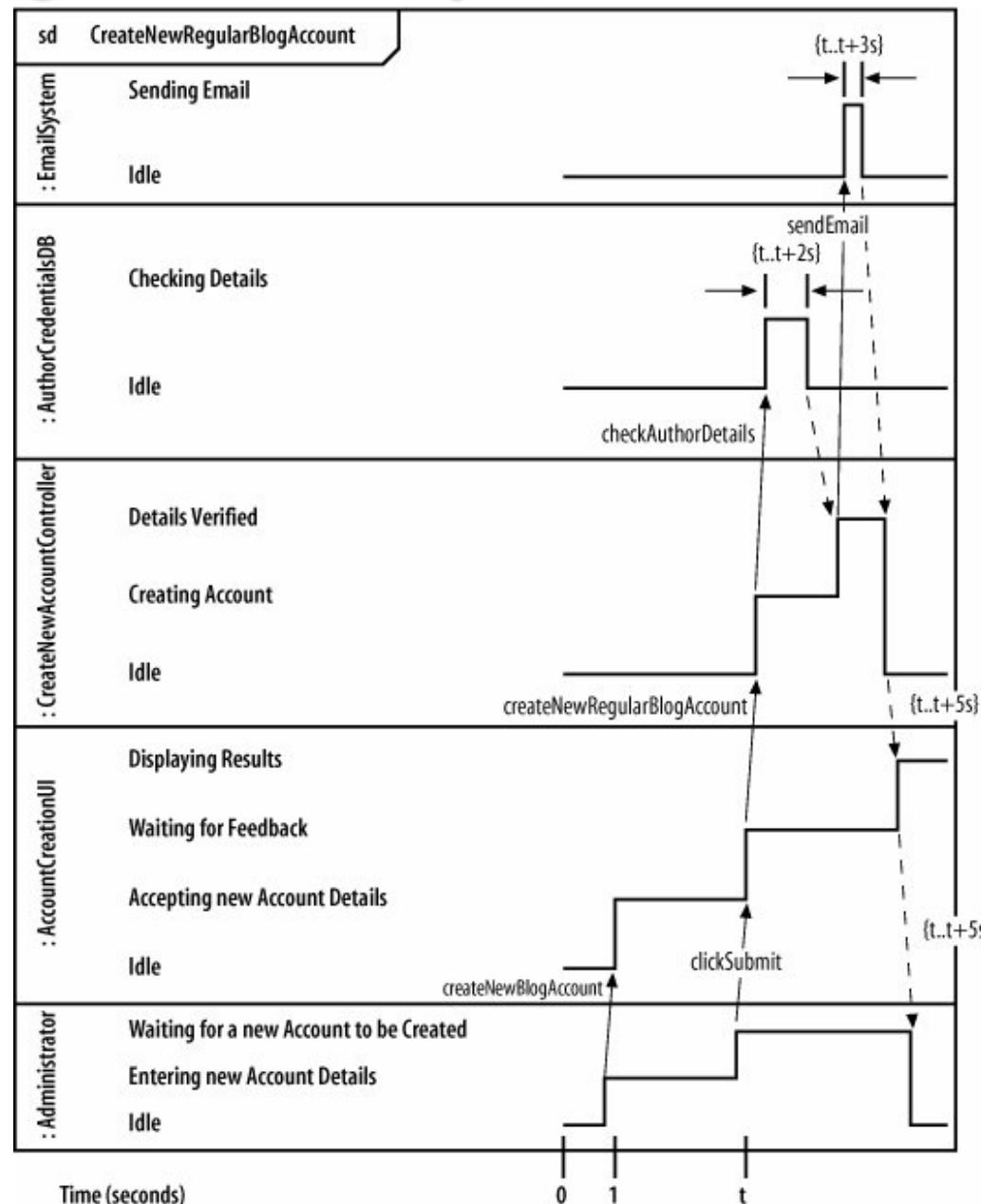
- Linha de vida de valor

- mostra o valor de uma entidade ao longo do tempo

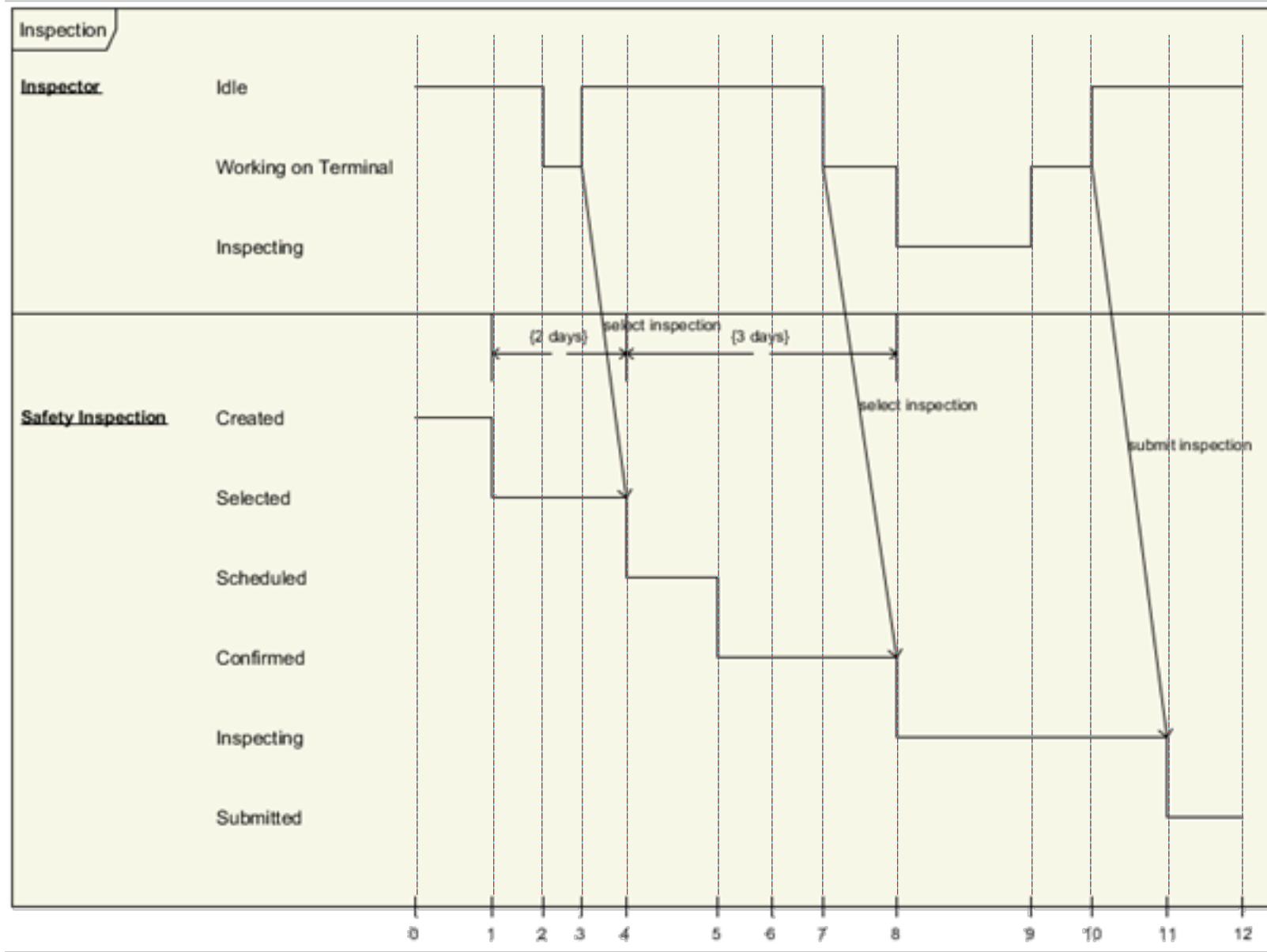




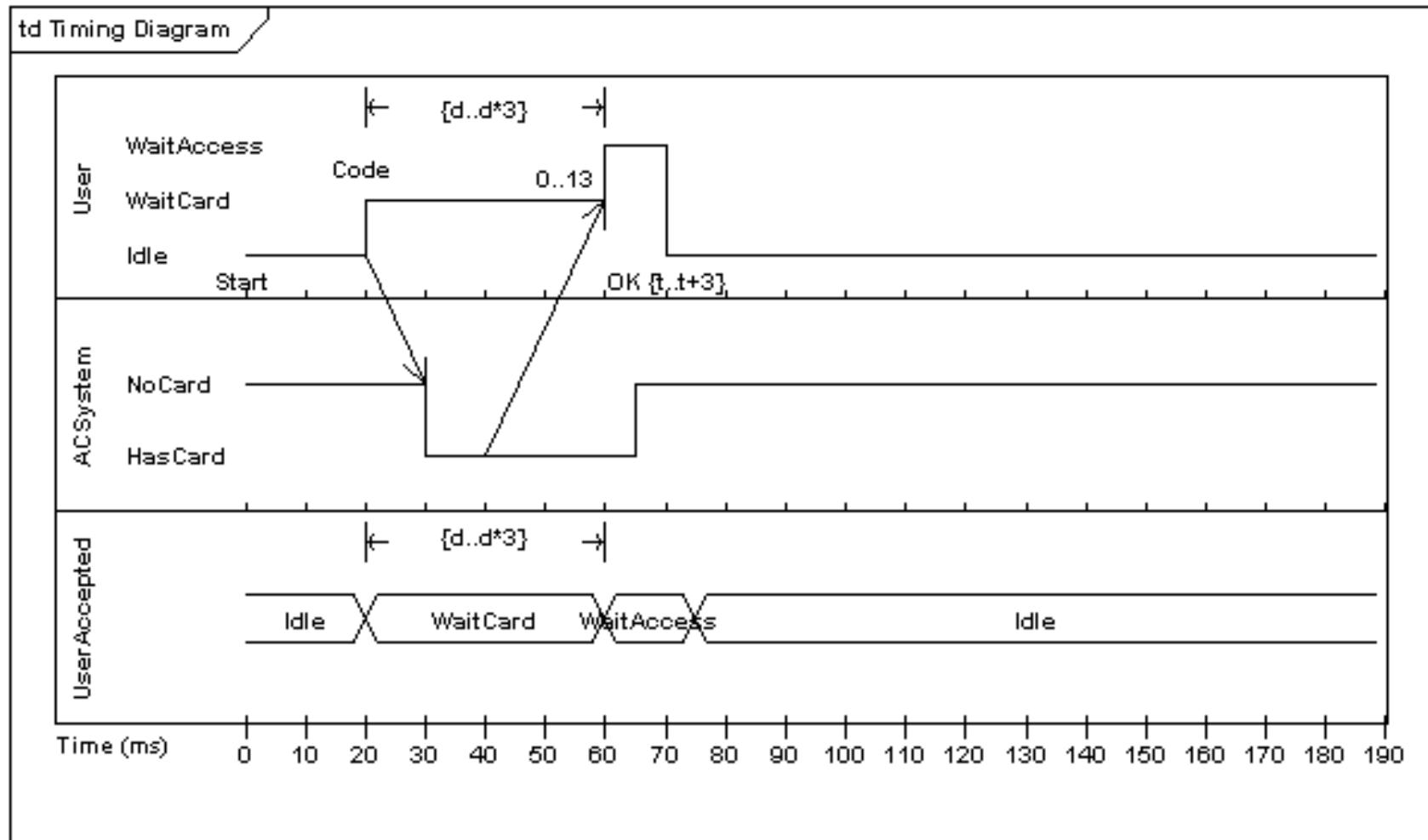
Timing diagram - Exemplo



Timing Diagram - Outro Exemplo

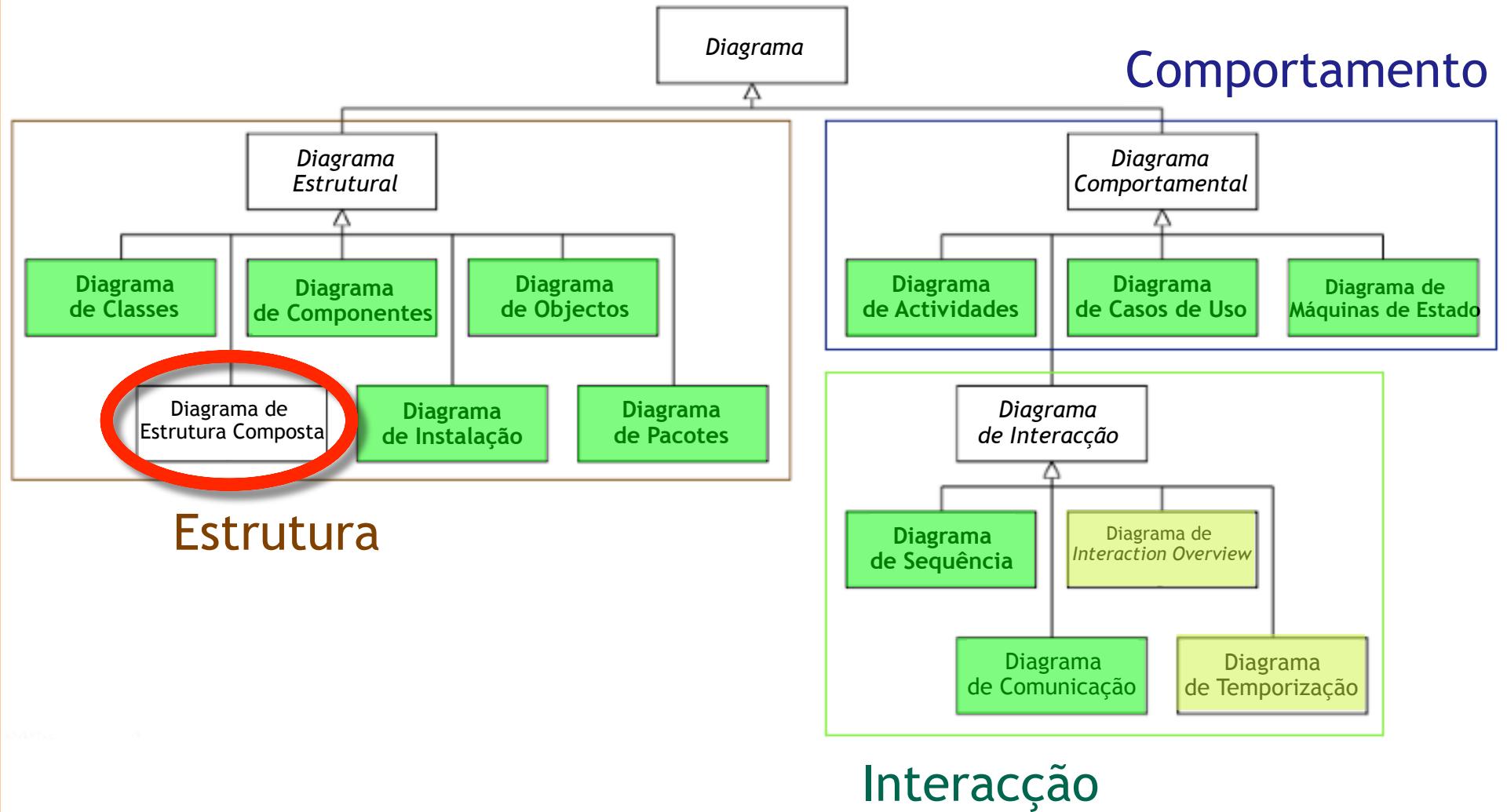


Timing Diagram - Diferentes linhas de vida





Diagramas da UML 2.x



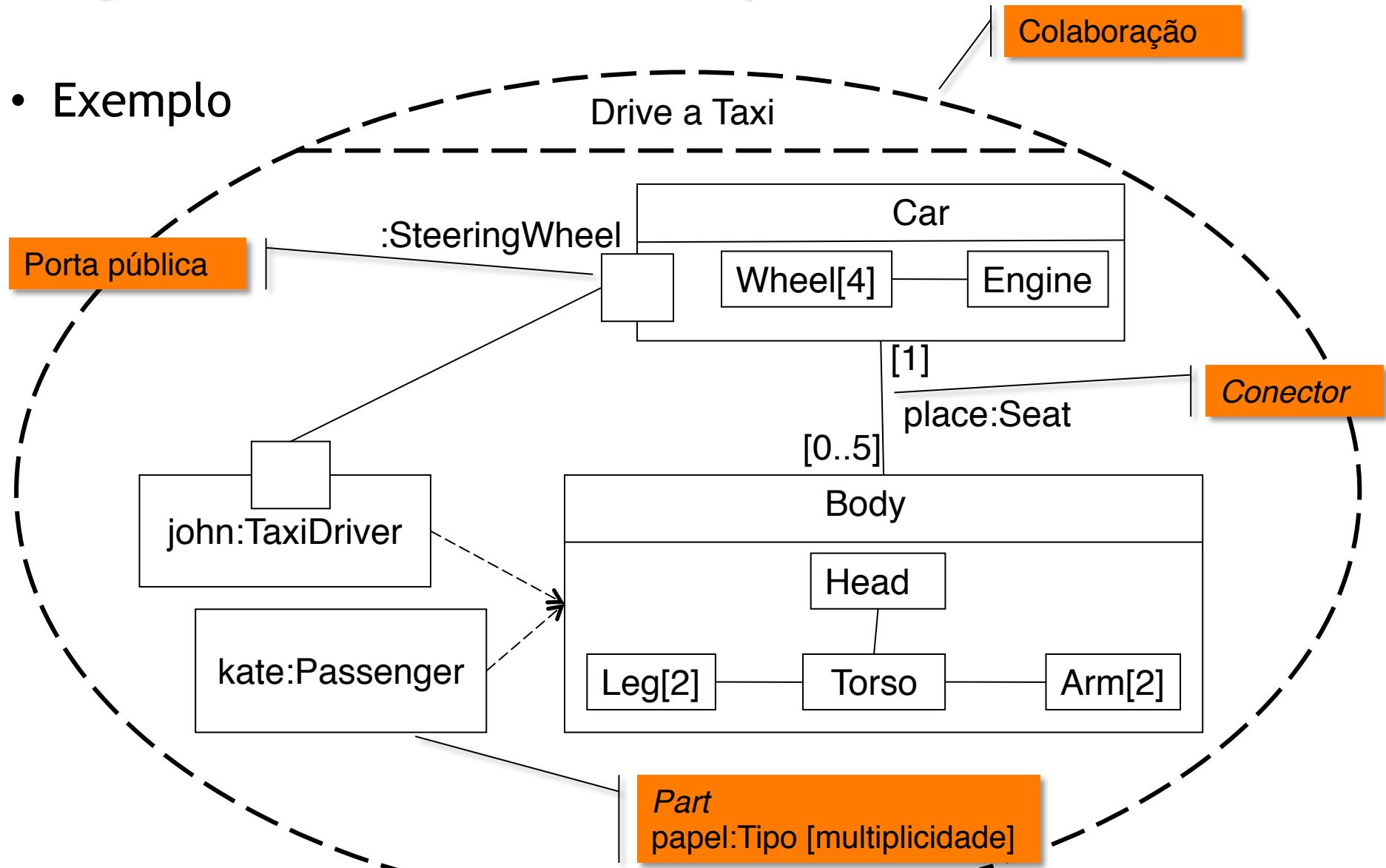


Diagramas de Estrutura Composta

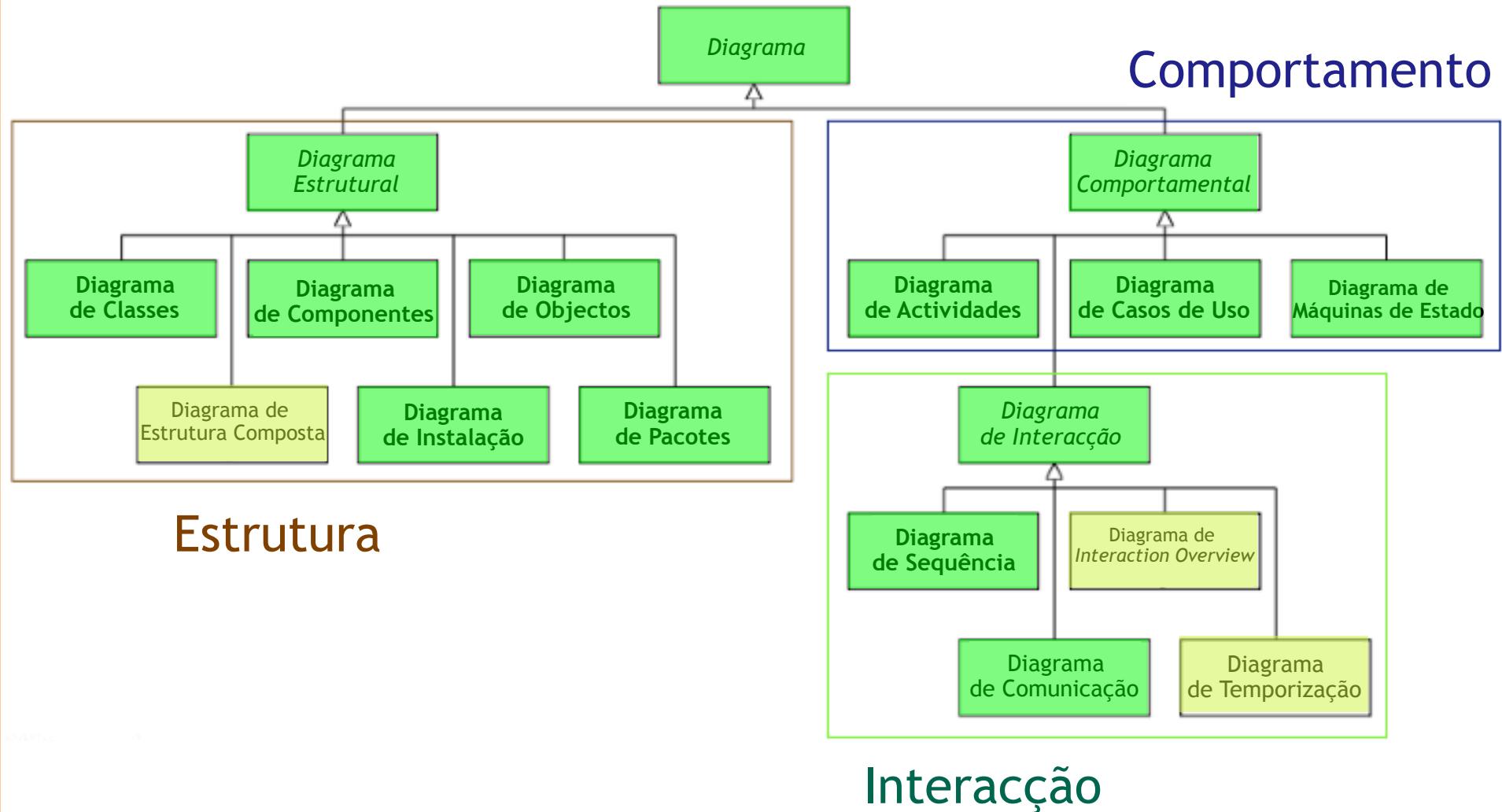
- *Composite Structure Diagrams*
- Mostram a estrutura interna de uma classe e as colaborações que essa estrutura permite
- Inclui
 - *Parts* internas
 - Portas pelas quais estas interagem (entre elas e com o exterior)
 - Conexões a ligar *parts* e portas
- Existe a partir da UML 2.0
 - Pouco documentados

Diagramas de Estrutura Composta

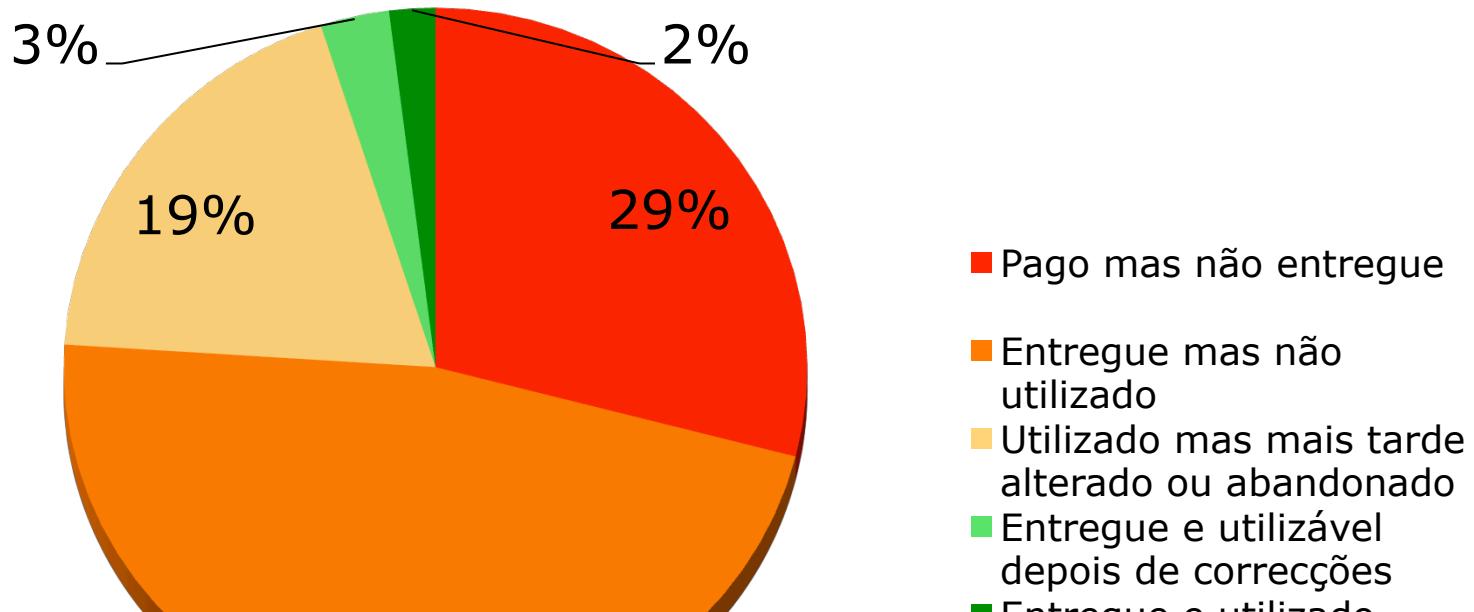
- Exemplo



Diagramas da UML 2.x



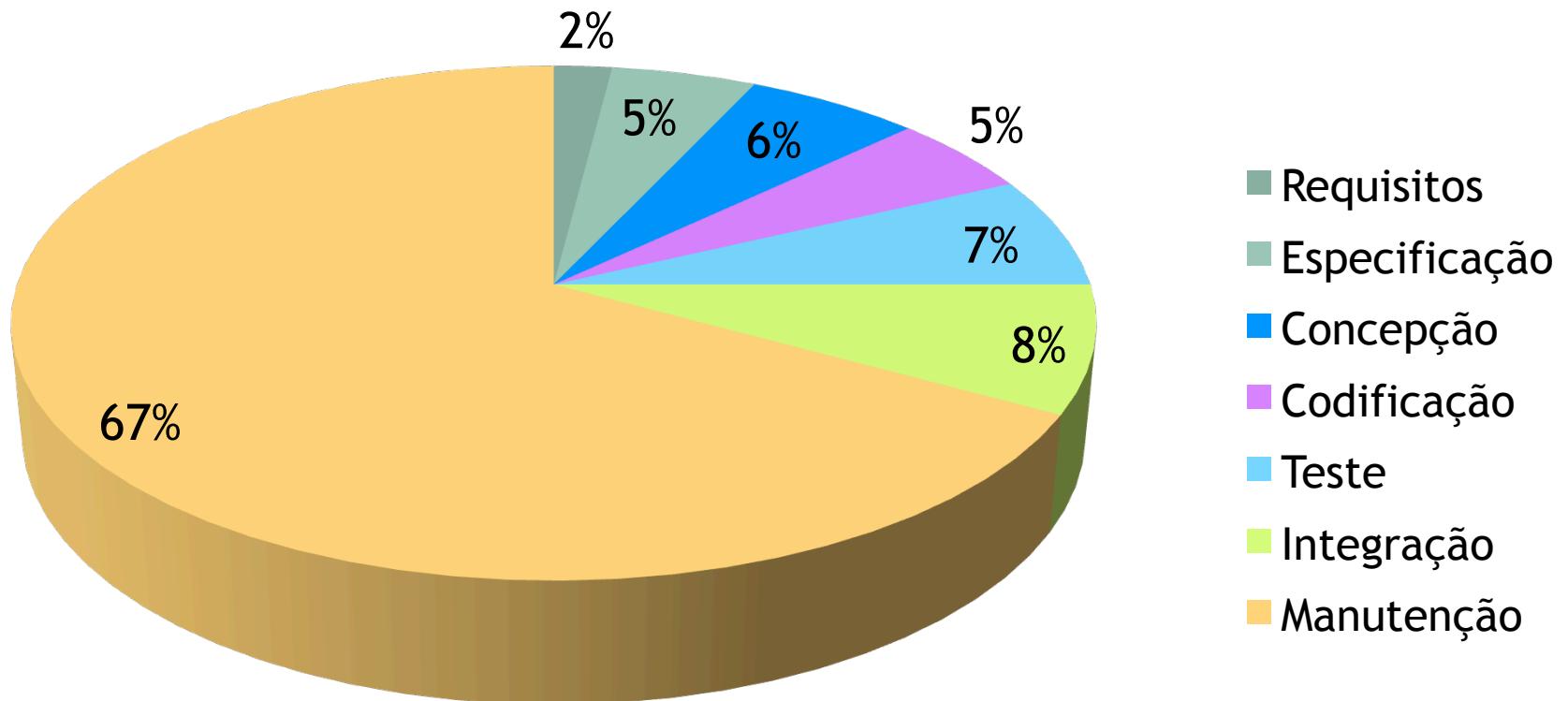
Estado da arte...



- Mais de 75% do software pago não chegou a ser utilizado!
- Apenas 5% do software pago foi utilizado continuadamente (deste, 3% necessitou de correcções).

Estado da arte...

Repartição de custos dos projectos





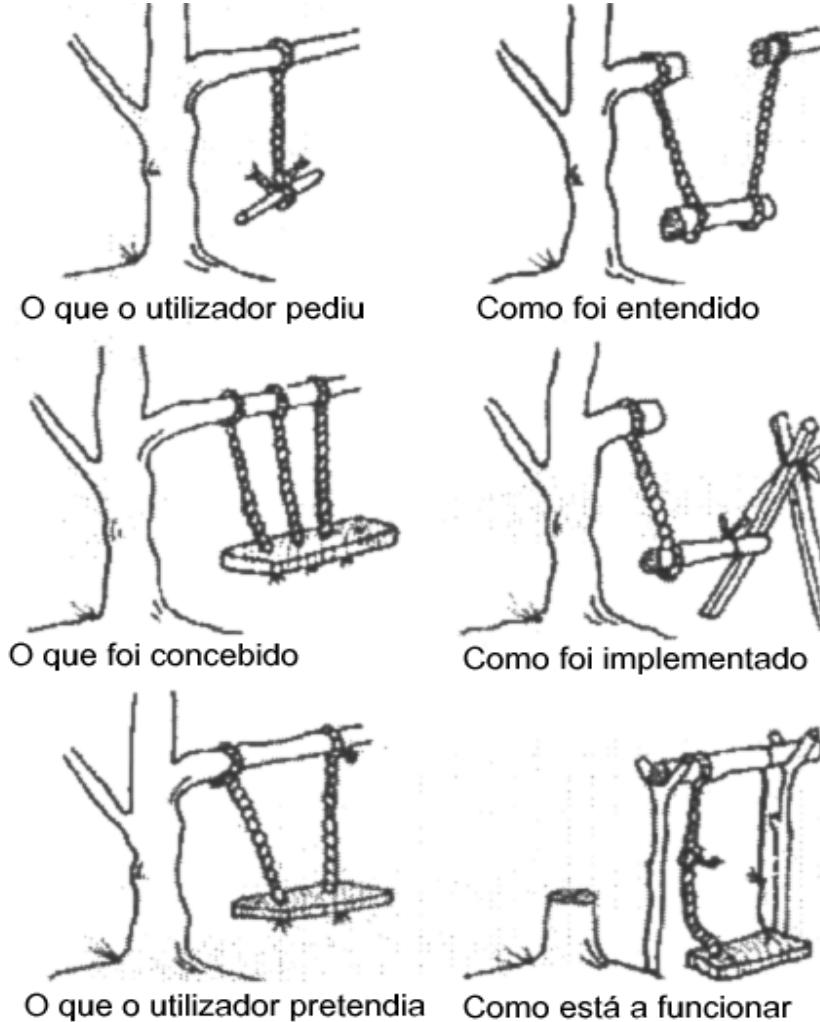
Como vai o desenvolvimento de Software?

- 56% de todos os *bugs* pode ser atribuídos a erros cometidos durante a fase de análise (i.e., não se esteve a construir o sistema certo!)

alguns dados sobre grandes projectos (>50,000 loc):

- produtividade média está abaixo das 10 linhas de código por dia;
- em média, encontram-se 60 erros por cada 10,000 linhas de código.

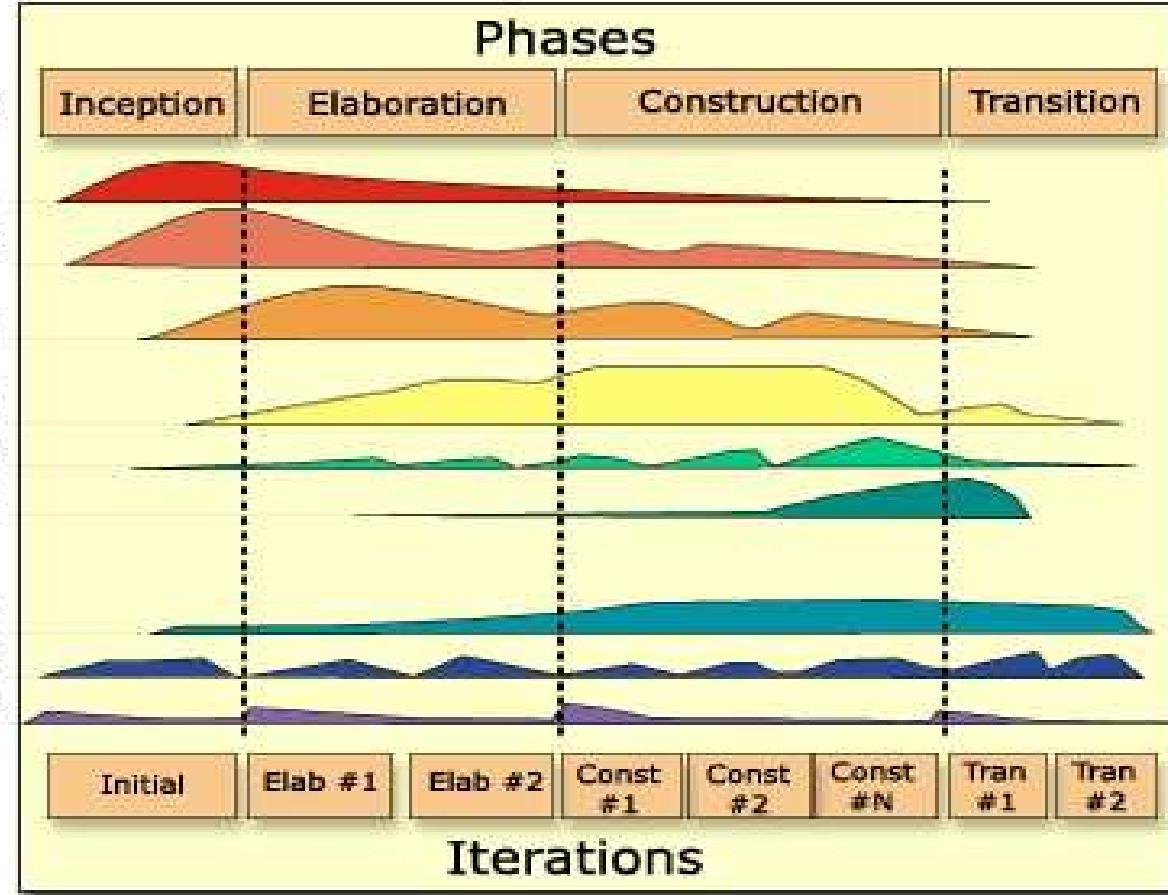
Desenvolver um bom sistema não é tarefa trivial

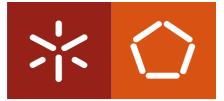


(Rational) Unified Process

Disciplines

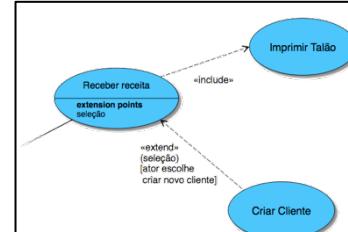
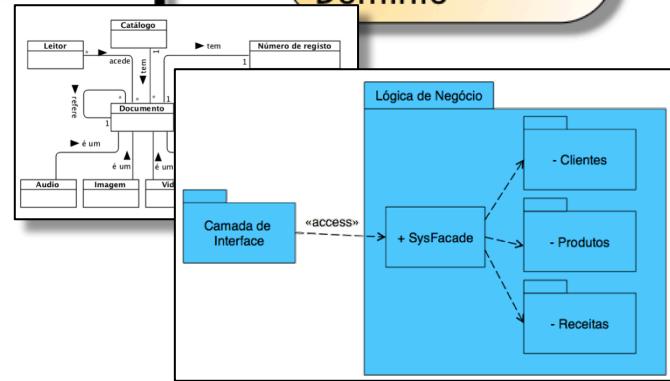
- Business Modeling
- Requirements
- Analysis & Design
- Implementation
- Test
- Deployment
- Configuration & Change Mgmt
- Project Management
- Environment





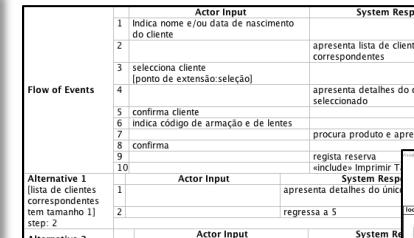
Processo...

"Problema"

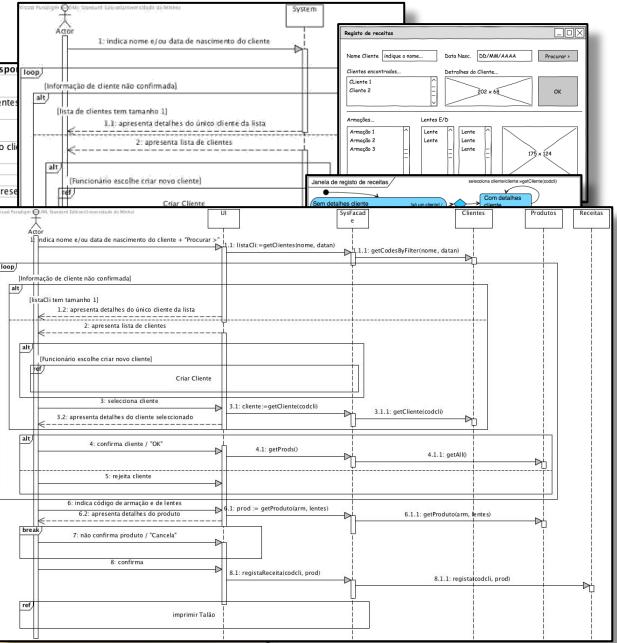


Identificação de
Use Cases

Modelação do
Domínio



Especificação
dos Use Case



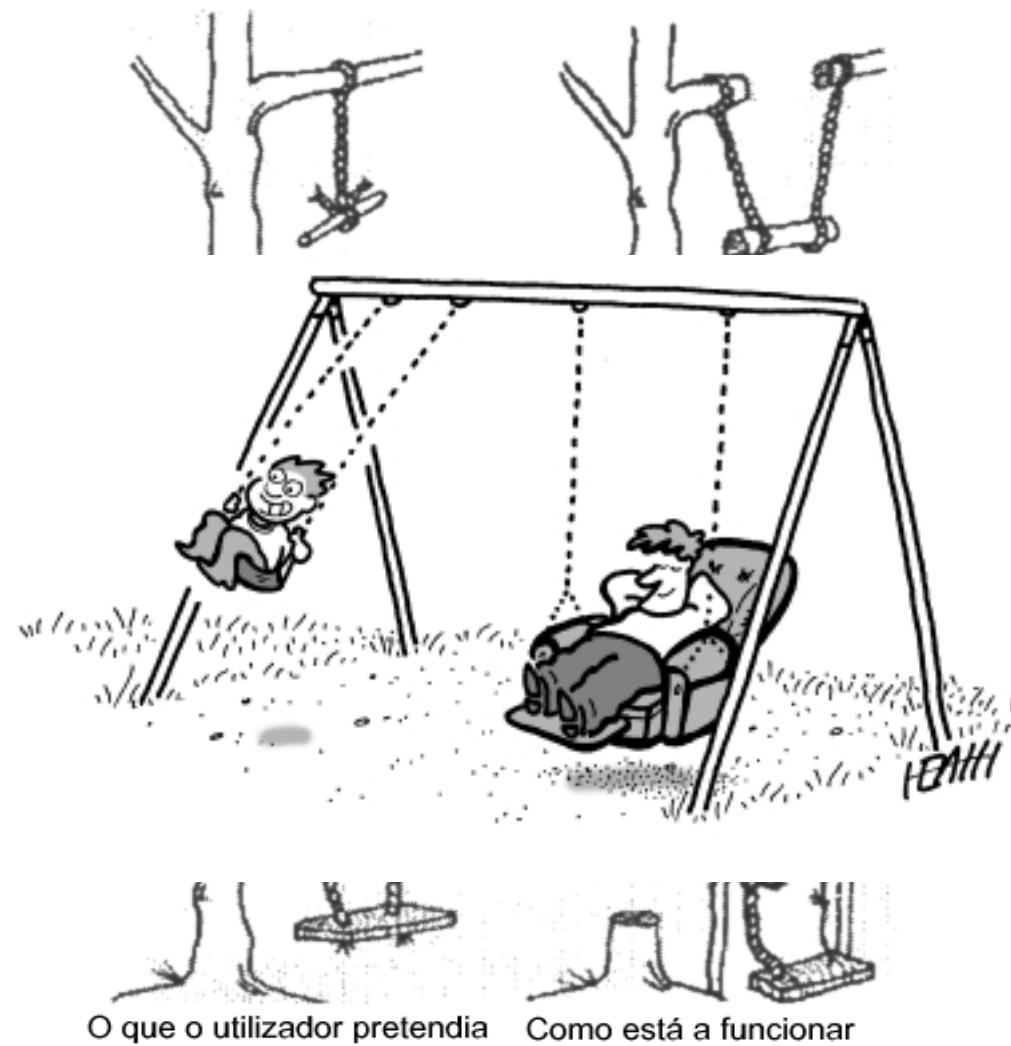
Modelação
Estrutural

Modelação
Comportamental

Implementação



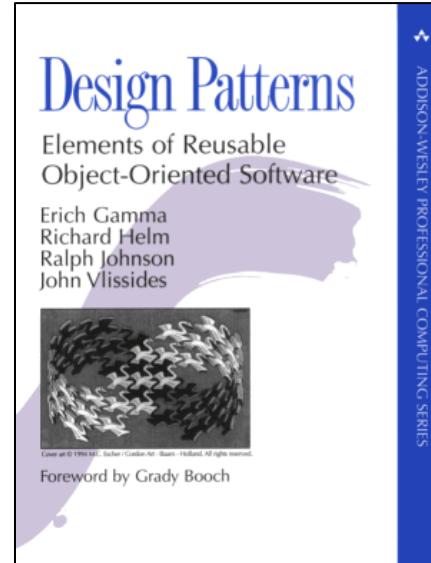
Desenvolver um bom sistema não é tarefa trivial



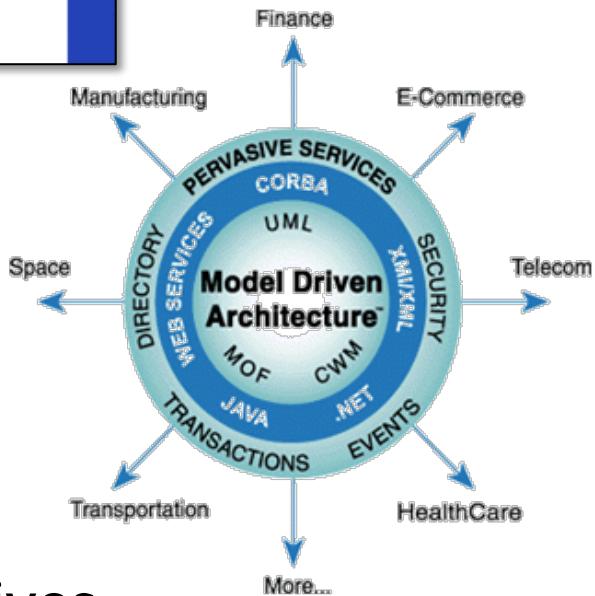


Mais...

- Software design patterns



- Model Driven Engineering
 - Model Driven Architecture (OMG)
- Desenvolvimento de Sistemas Interactivos
 - Model Based User Interface Development





Avaliação

- Exame (≥ 9.0) - uma prova escrita sobre a matéria teórica
 - Exame de consulta
 - Objectivos
 - Reconhecer os diferentes tipos de diagramas da UML ;
 - Compreender modelos (de requisitos/estruturais/comportamentais) descritos em UML;
 - Conceber sistemas de software utilizando UML;
 - Implementar sistemas de software a partir de modelos UML.
- Trabalho Prático (≥ 10.0)
- Classificação Final (≥ 10.0)
.6 Exame + .4 Trabalho



Três Diagramas

Sumário

- Modelação Estrutural
 - Diagramas de Estrutura Composta
- Modelação Comportamental
 - *Timing Diagrams*
 - *Interaction Overview Diagrams*

Fim!