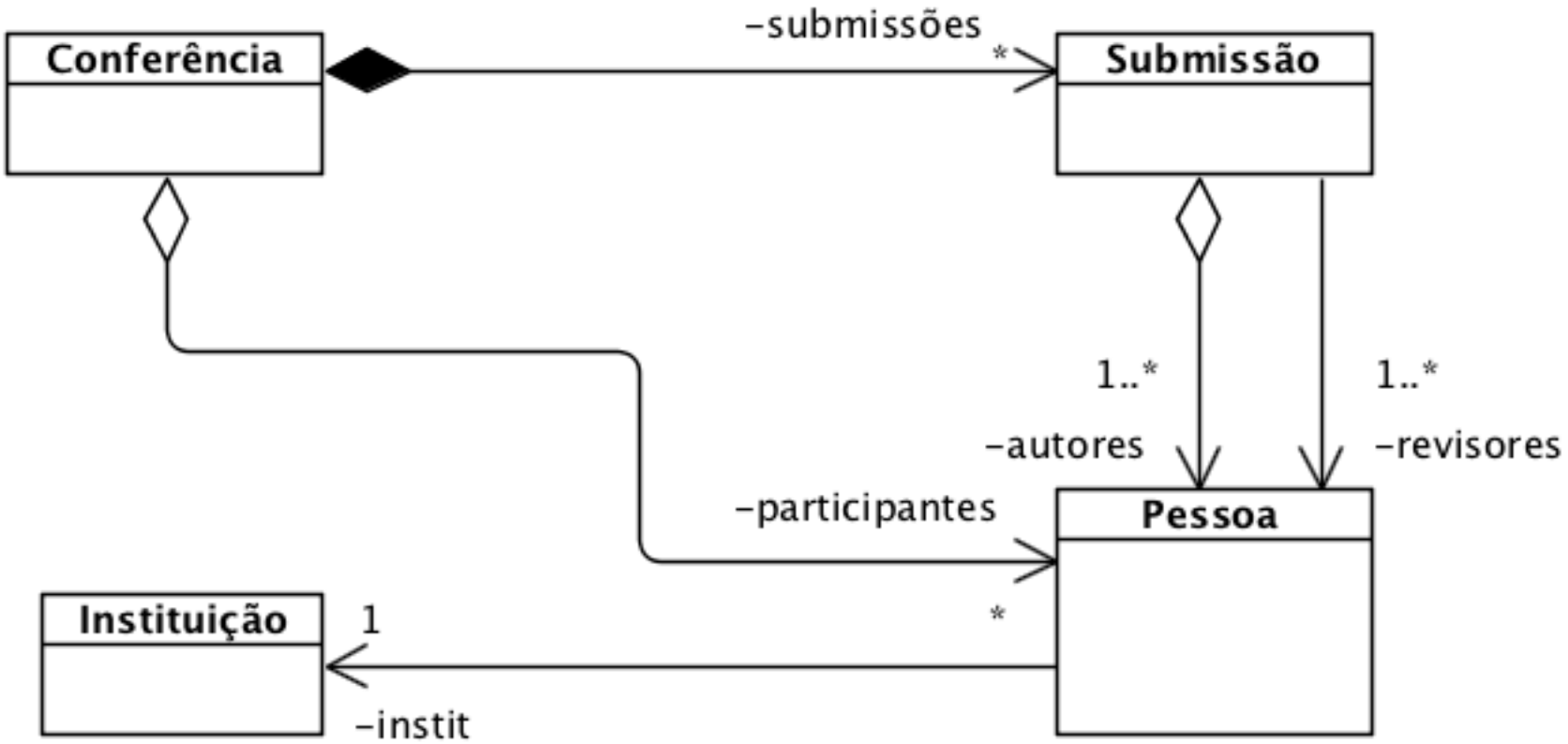
Desenvolvimento de Sistemas Software

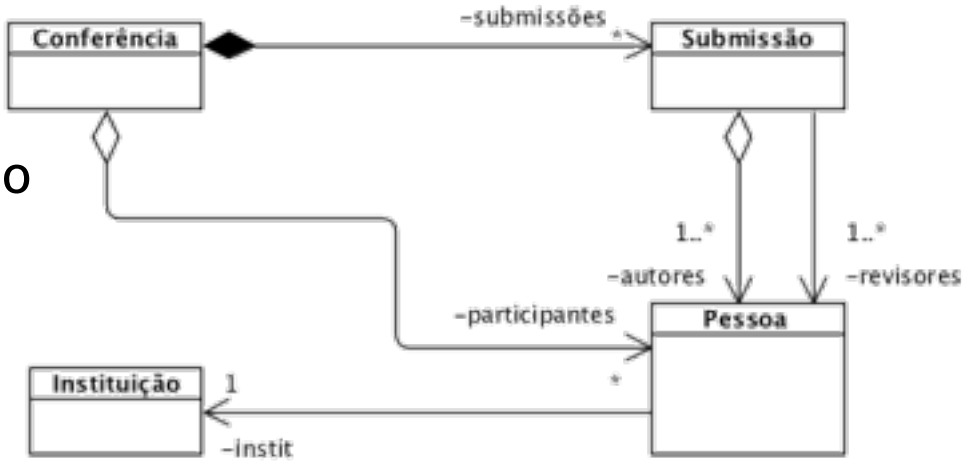
Aula Teórica 27: OCL

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*" ?
- Como trabalhar com *colecções* de objectos?

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

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

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

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

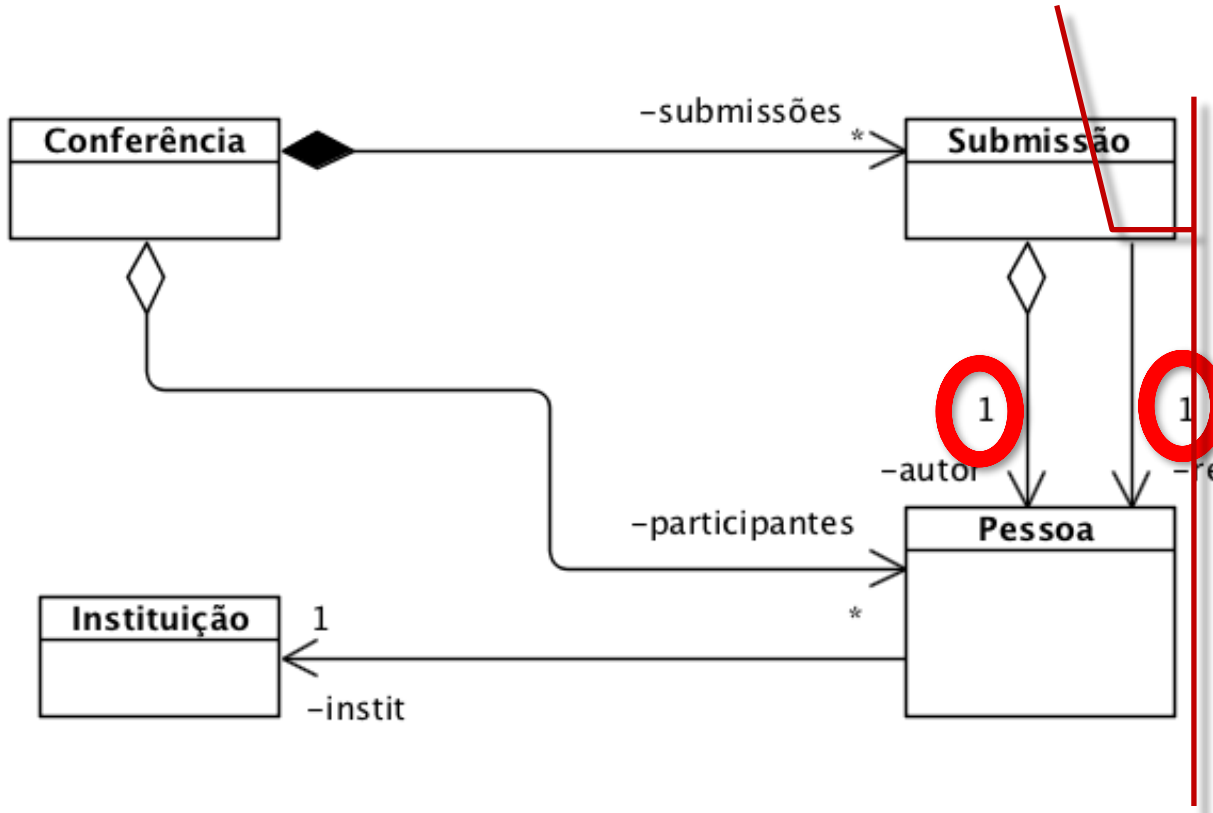
Tipos de Colecções

- 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, …

Invariantes

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

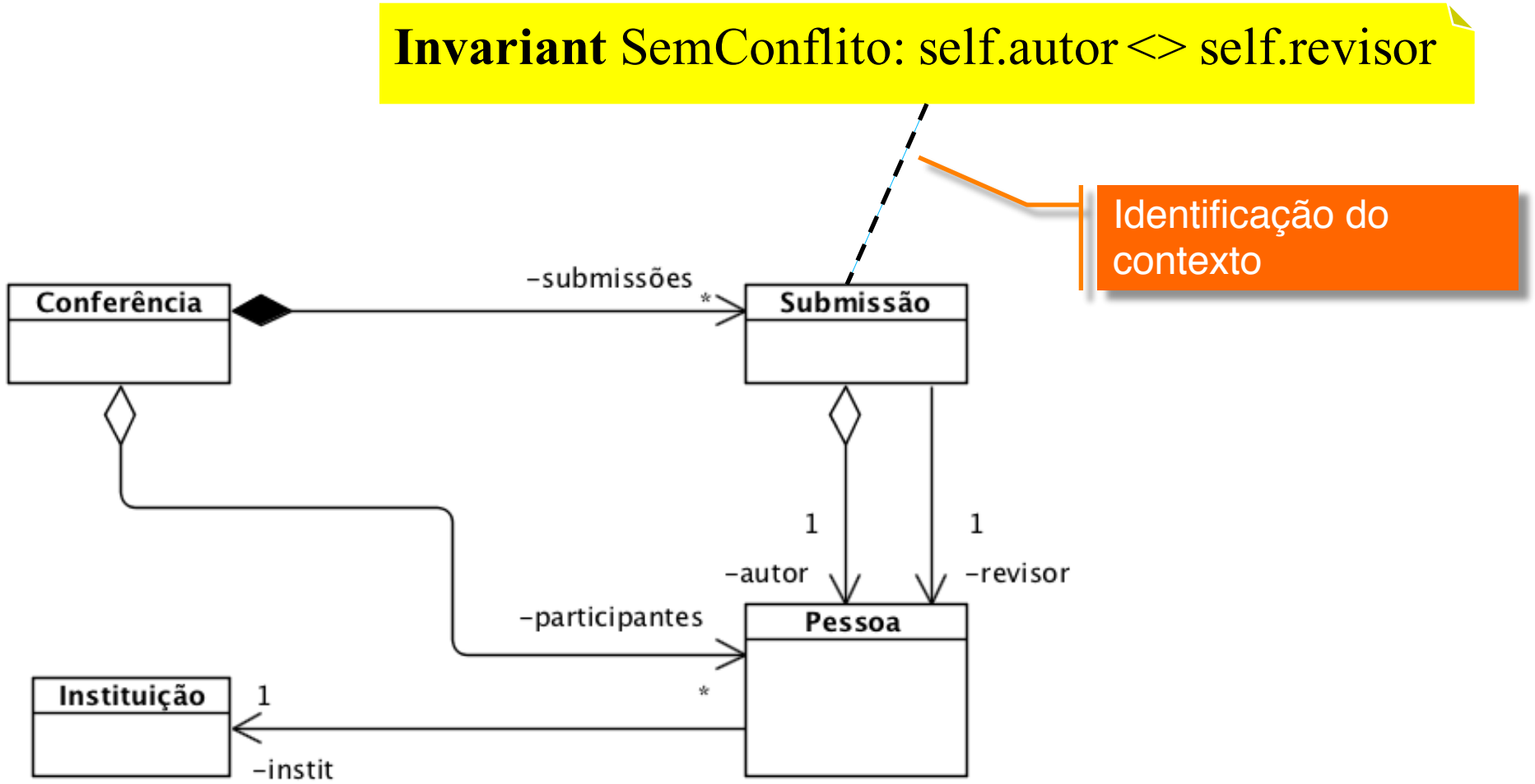
Context: Submissão
Invariant SemConflito: self.autor <> self.revisor



- 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 SemConflito)
- Os invariantes são expressões booleanas

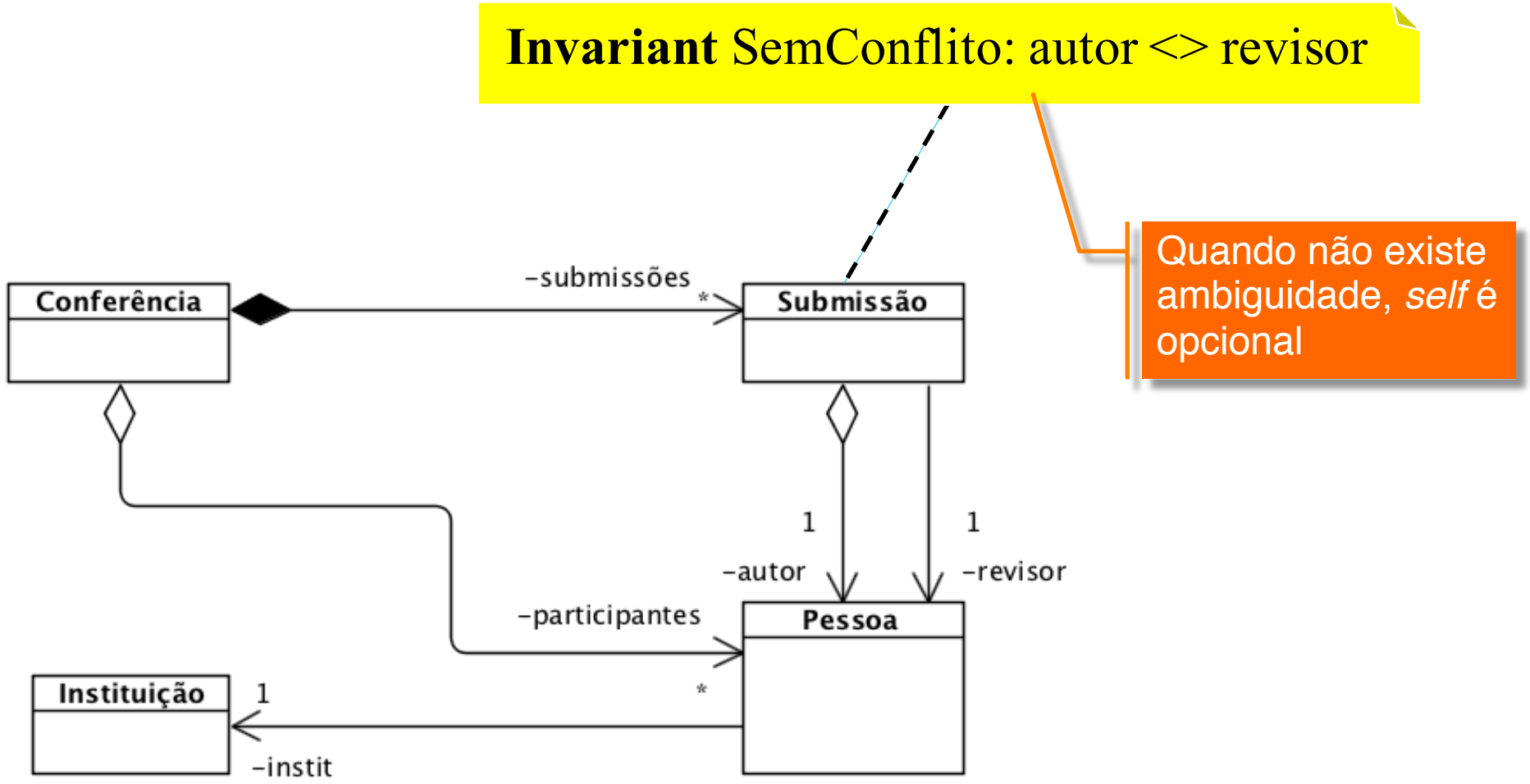
Invariantes

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

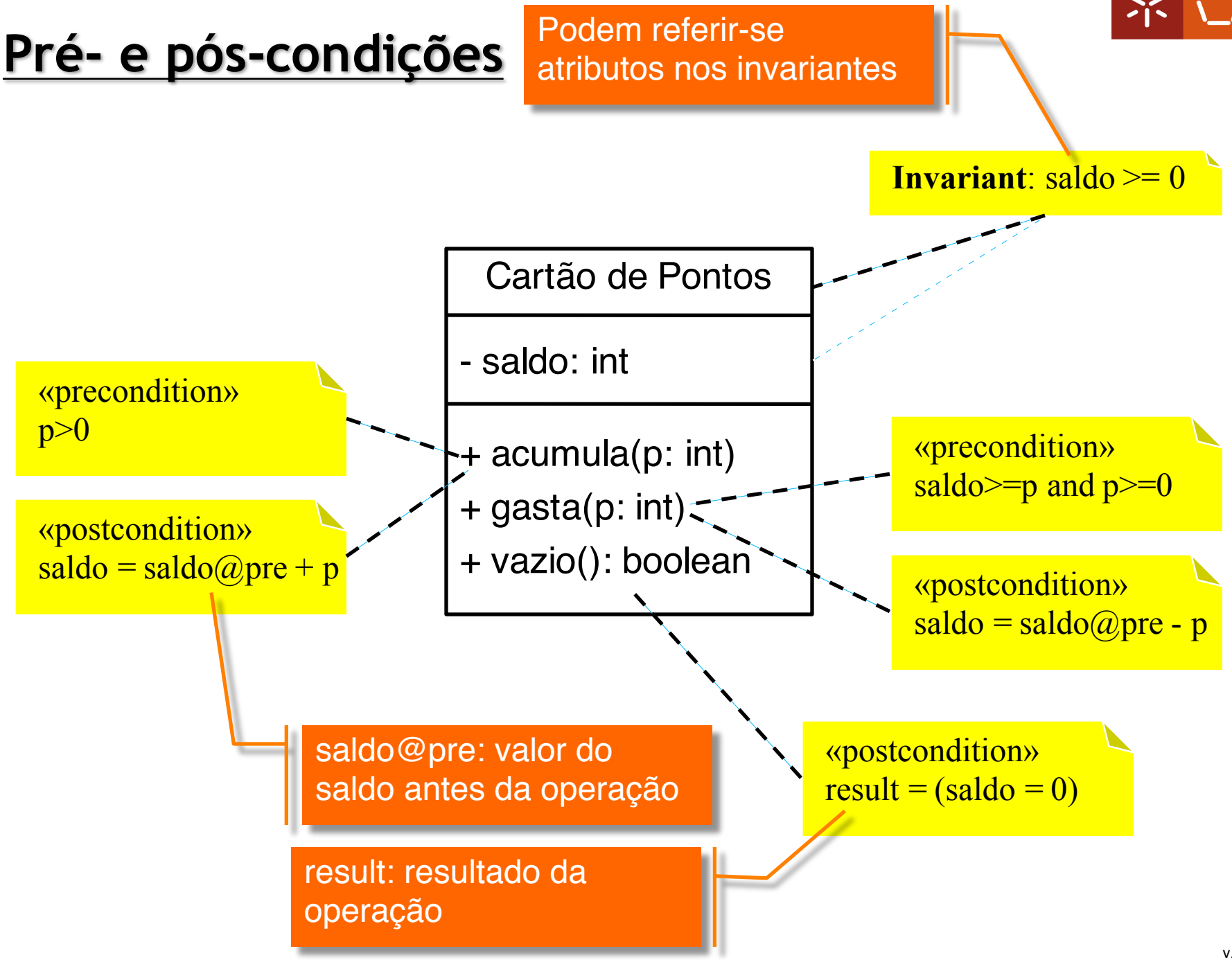


Invariantes

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

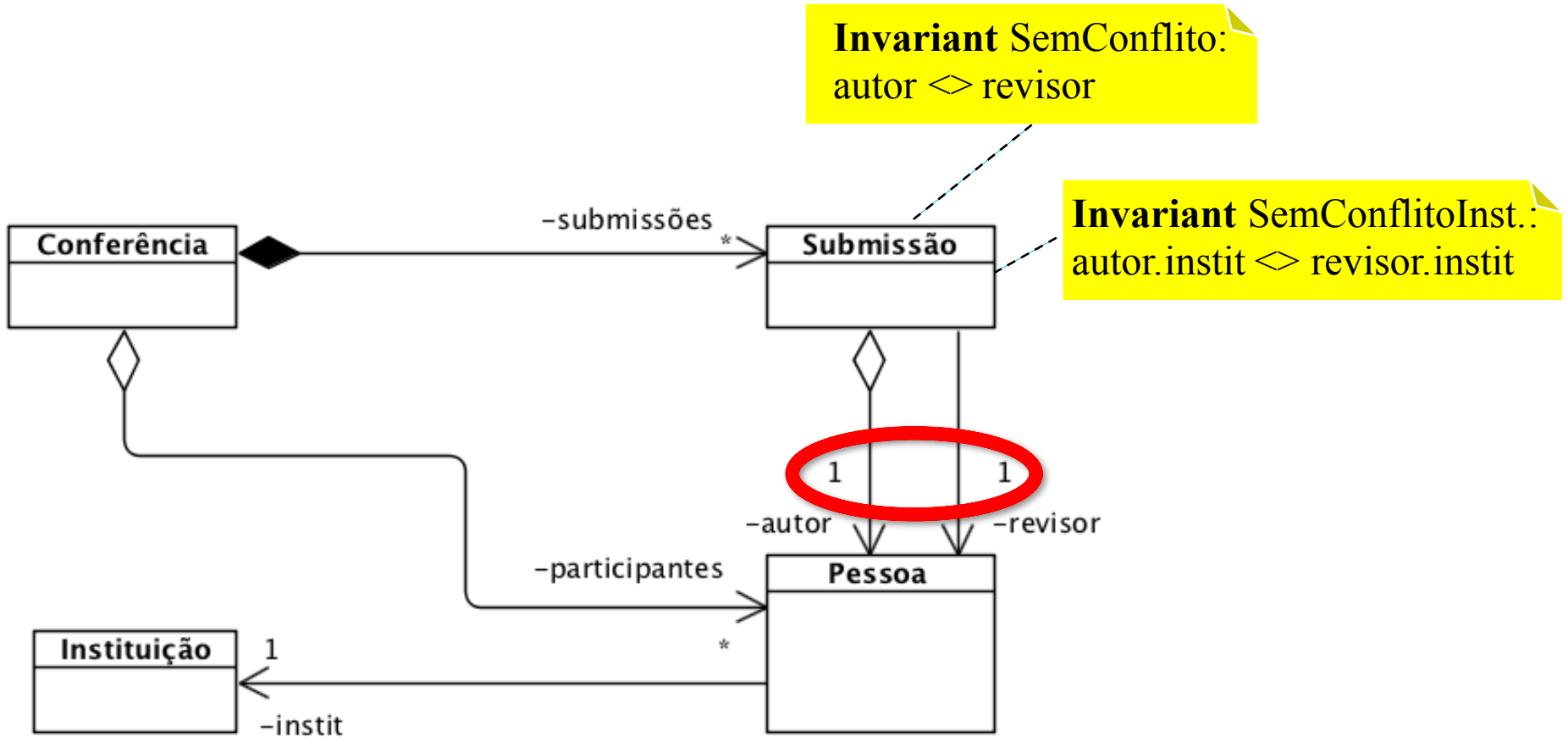


Pré- e pós-condições



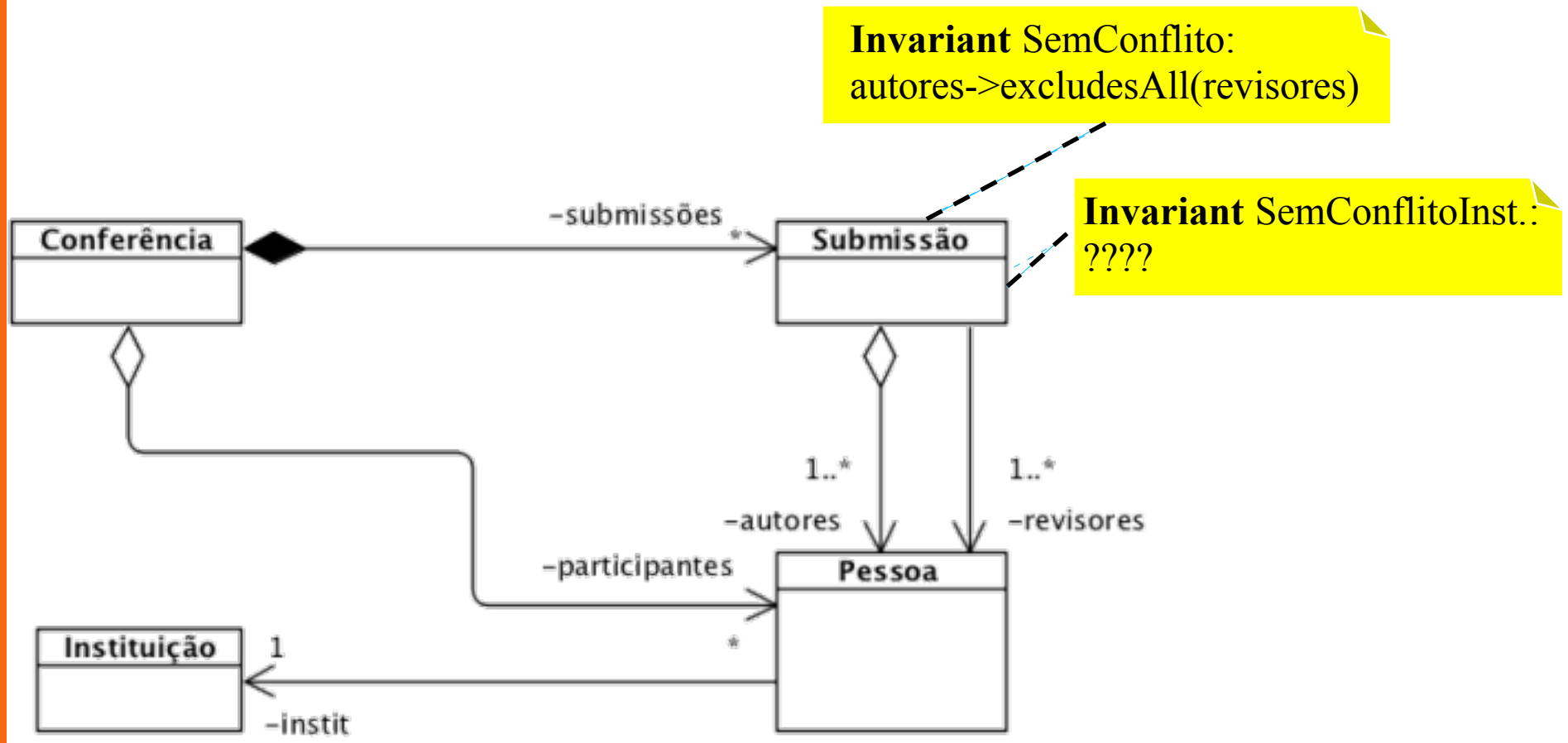
Navegação nas associações

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



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 *body* expression for every element in the *source* collection.
exists(iterators | body) : Boolean	True if *body* evaluates to true for at least one element in the *source* collection. Allows multiple iterator variables.
forAll(iterators | body): Boolean	True if *body* evaluates to true for each element in the source collection. Allows multiple iterator variables.
one(iterator| body): Boolean	True if there is exactly one element in the *source* collection for which *body* is true
isUnique(iterator| body): Boolean	Results in true if *body* evaluates to a different value for each element in the *source* collection.
any(iterator | body): T	Returns any element in the source collection for which *body* evaluates to true. The result is null if there is none.
collect(iterator| body): Collection(T2)	The Collection of elements resulting from applying *body* to every member of the *source* set. The result is flattened.

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

Colecções - iteradores (2/2)

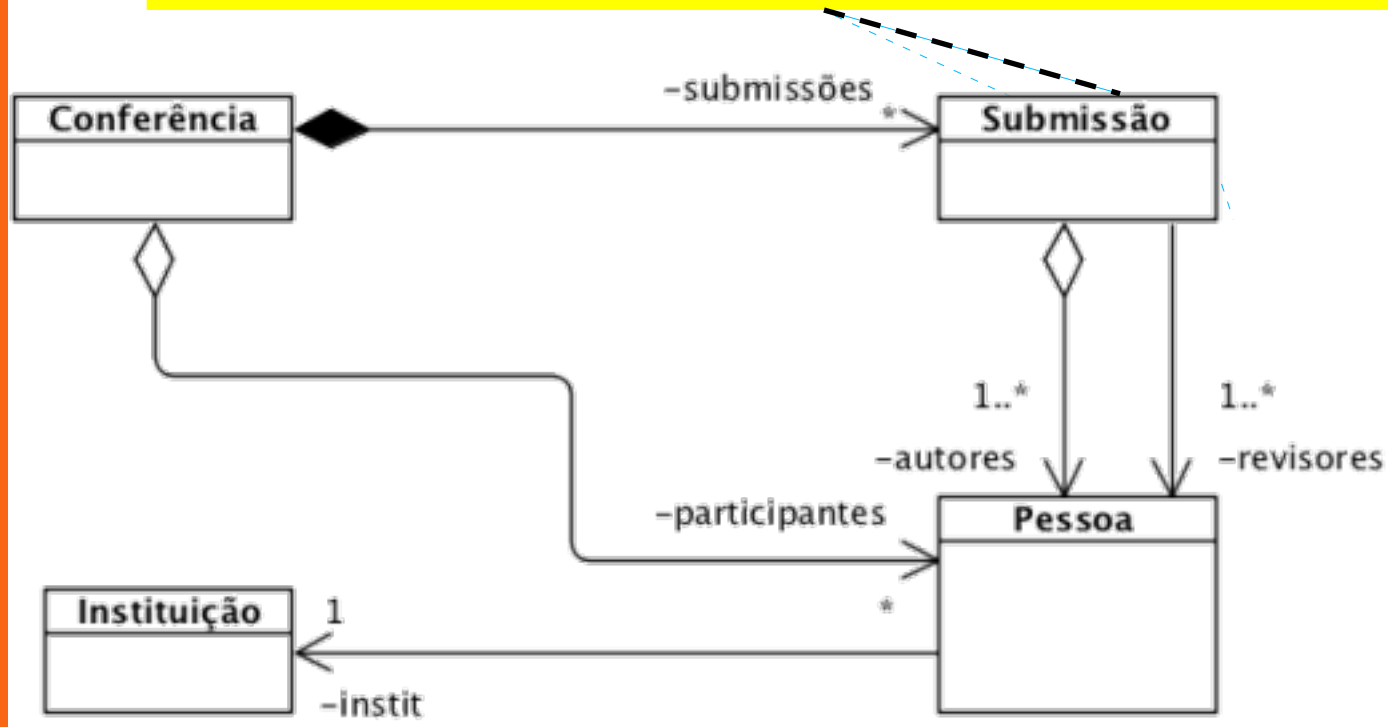
Iterator expression	Description
select(iterator | body): Collection(T)	The Collection of elements of the *source* collection for which *body* is true. The result collection is of the same type of the *source* collection.
reject(iterator | body): Collection(T)	The Collection of elements of the *source* collection for which *body* is false. The result collection is of the same type of the *source* collection.
collectNested(iterator | body): CollectionWithDuplicates(T2)	The Collection of elements (allowing duplicates) that results from applying *body* (of type T2) to every member of the *source* 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 *source* collection by ascending order of the value of the *body* expression. The type T2 of the *body* expression must support "<". Collection type conversions: Set -> OrderedSet, Bag -> Sequence.

Colecções - exemplos

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

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

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

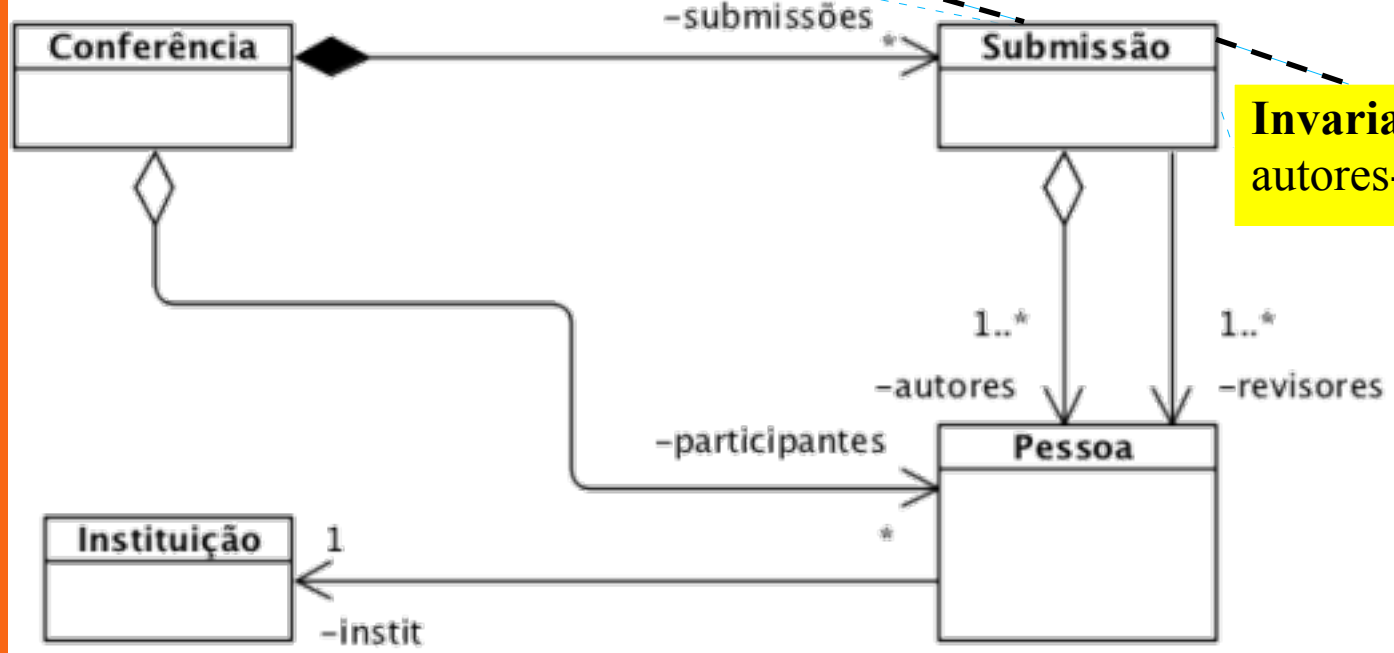


Colecções - exemplos

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

Invariant Conflito Inst.:
autores.instit->excludesAll(revisores.instit)

Como autores e revisores são colecções, o collect é aplicado automaticamente.



Invariant Conflito:
autores->excludesAll(revisores)

OCL

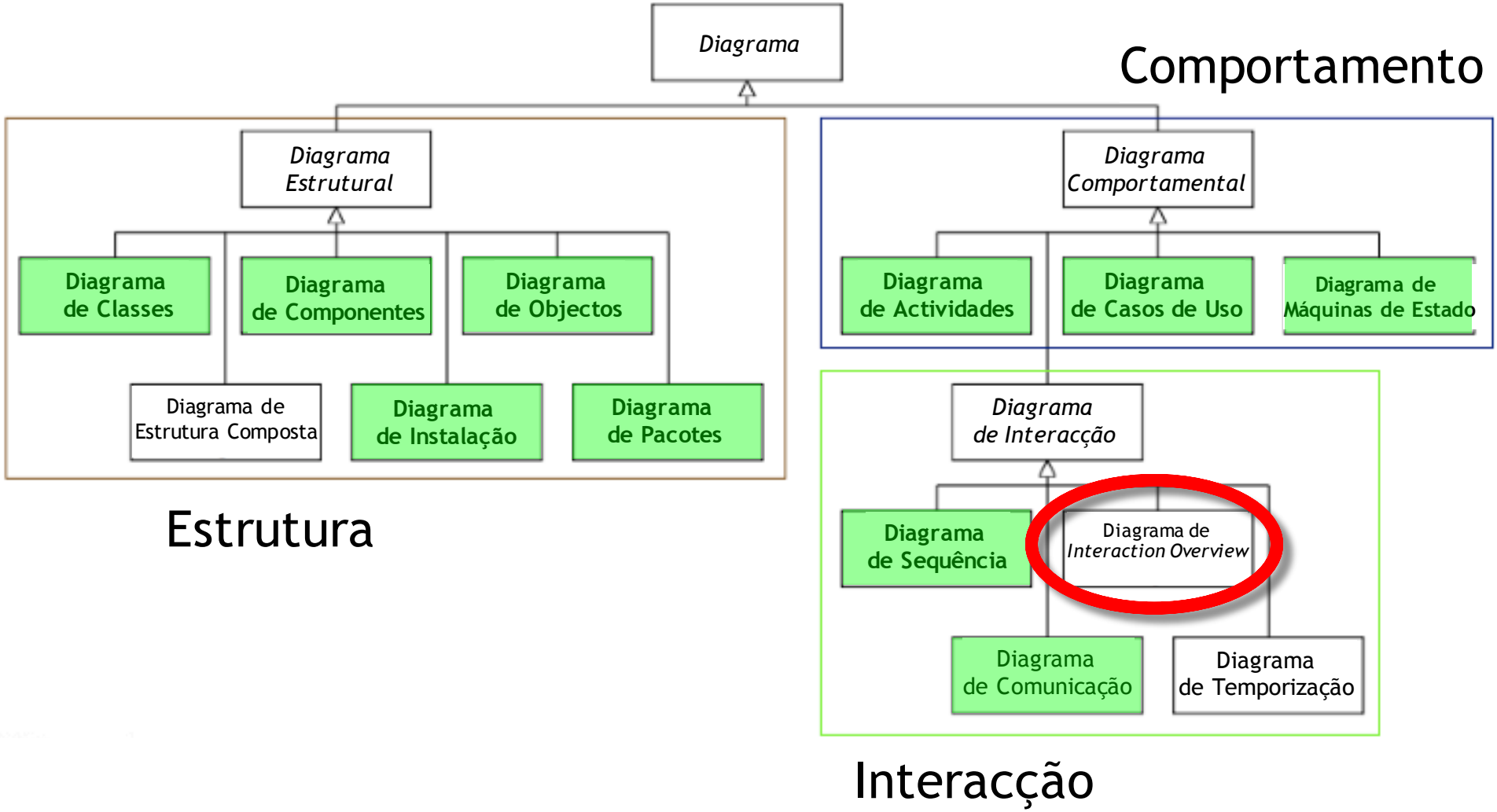
Sumário

- Object Constraint Language – OCL
  - *Sistema de tipos*
  - *Invariantes*
  - *Pré- e pós-condições*

Desenvolvimento de Sistemas Software

Aula Teórica 30: Três diagramas

Diagramas da UML 2.x

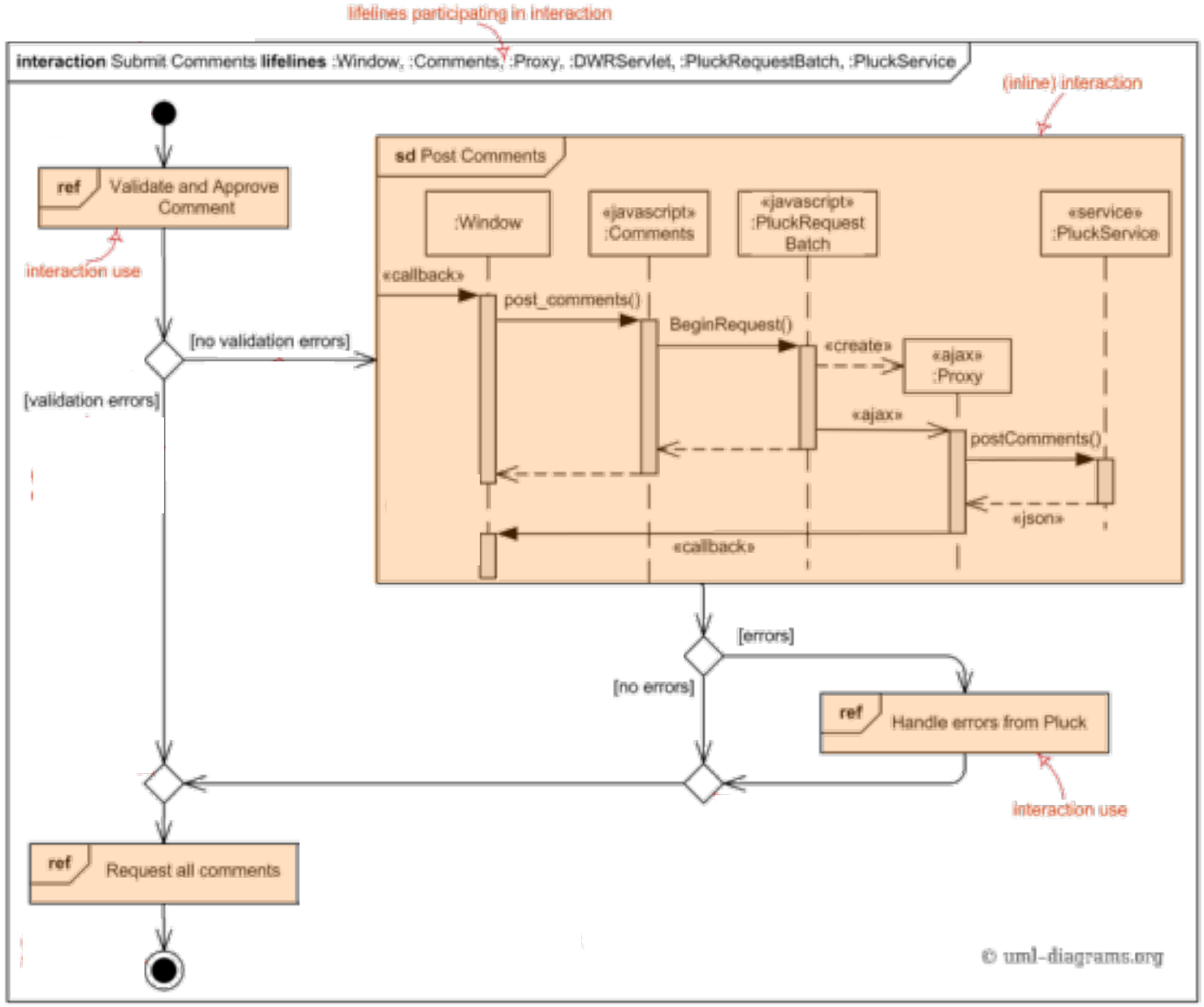


Interaction Overview Diagrams

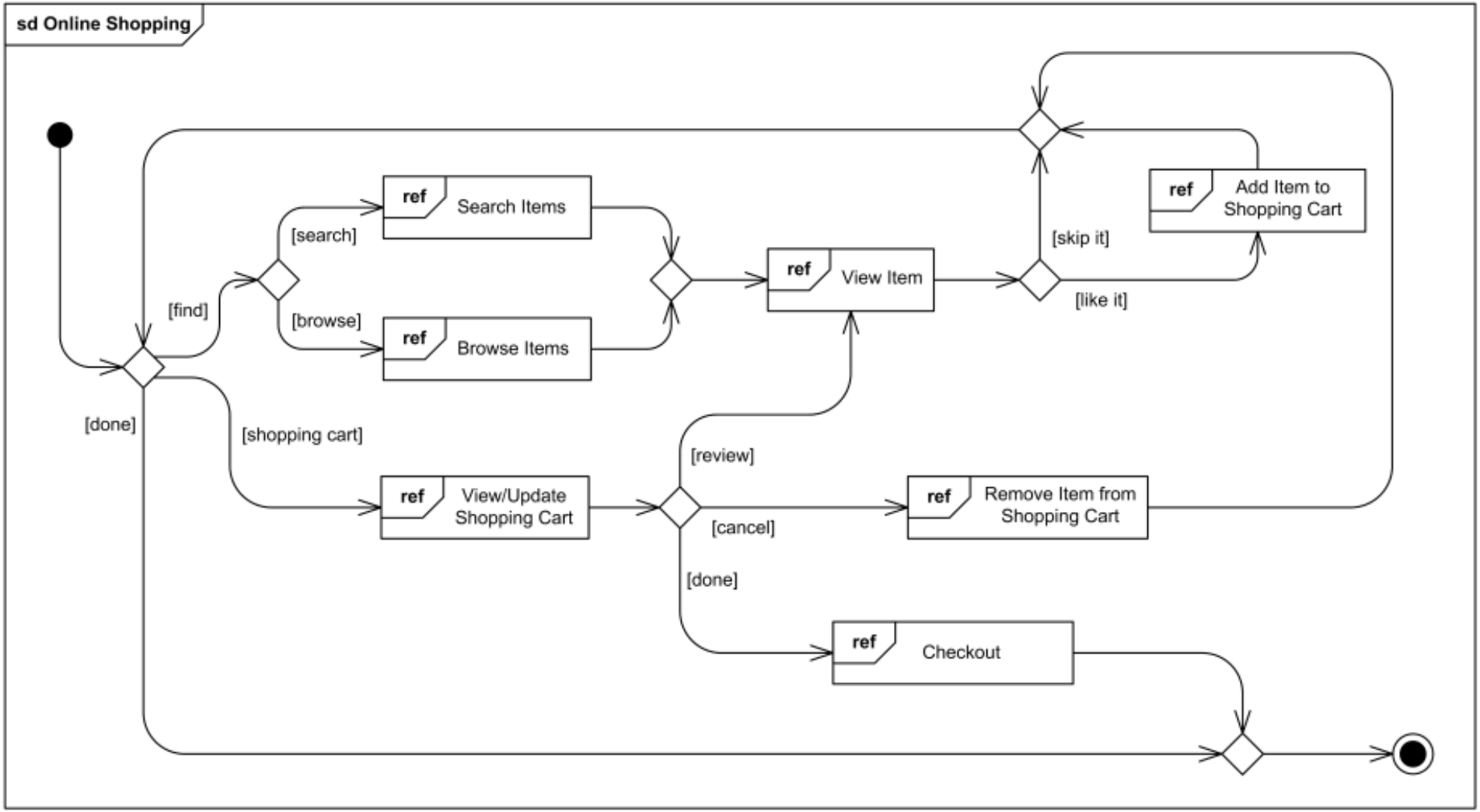
- Similares a Diagramas de Actividade
- Nodos de actividade subsituídos por interacções (Diagramas de Sequência)
- Permitem uma visão de alto nível sobre o fluxo de controlo entre interacções

- Poderiam ser utilizado como uma alternativa aos DSS para especificar Use Cases
  - Maior poder expressivo

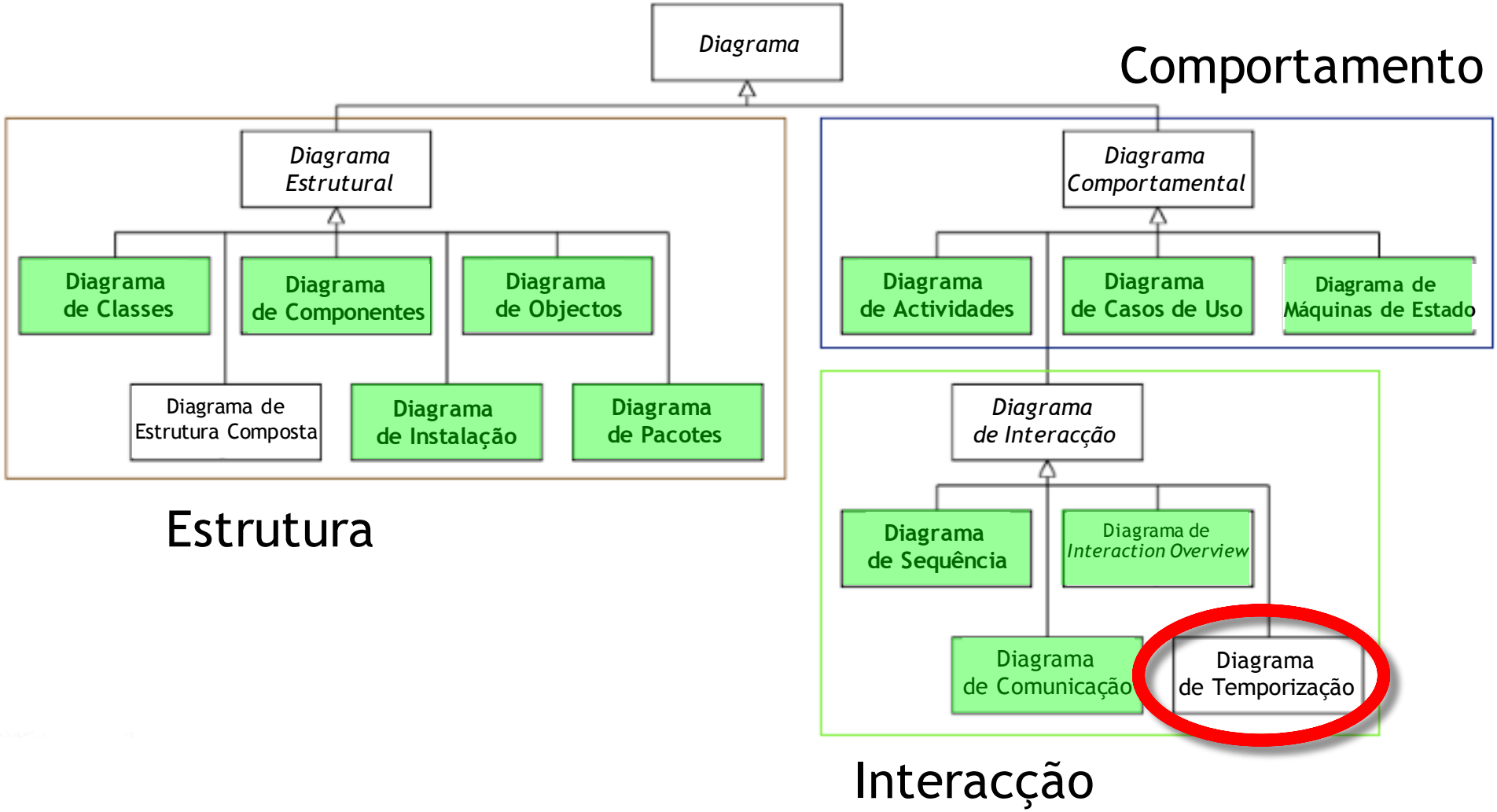
Interaction Overview Diagram - Exemplo



Outro Exemplo



Diagramas da UML 2.x

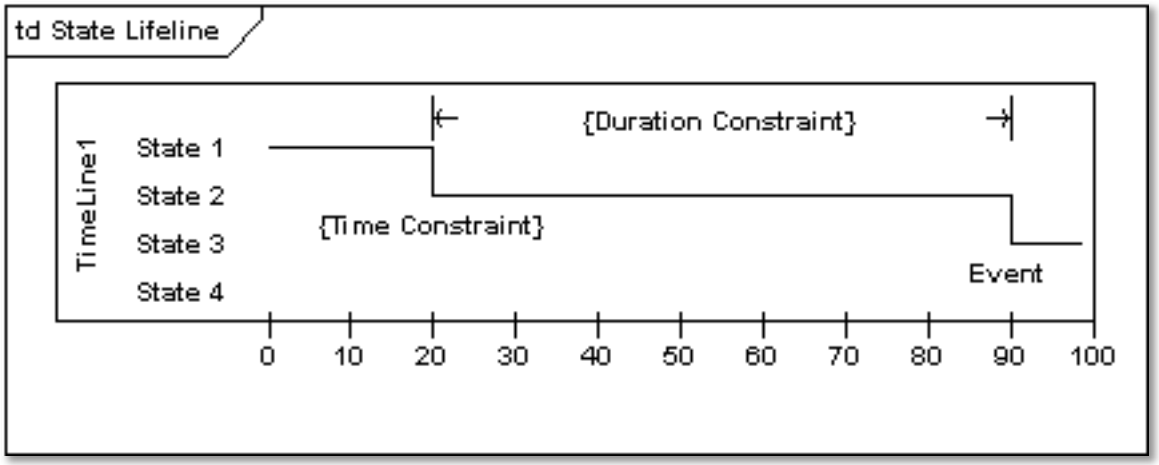


Timing diagram

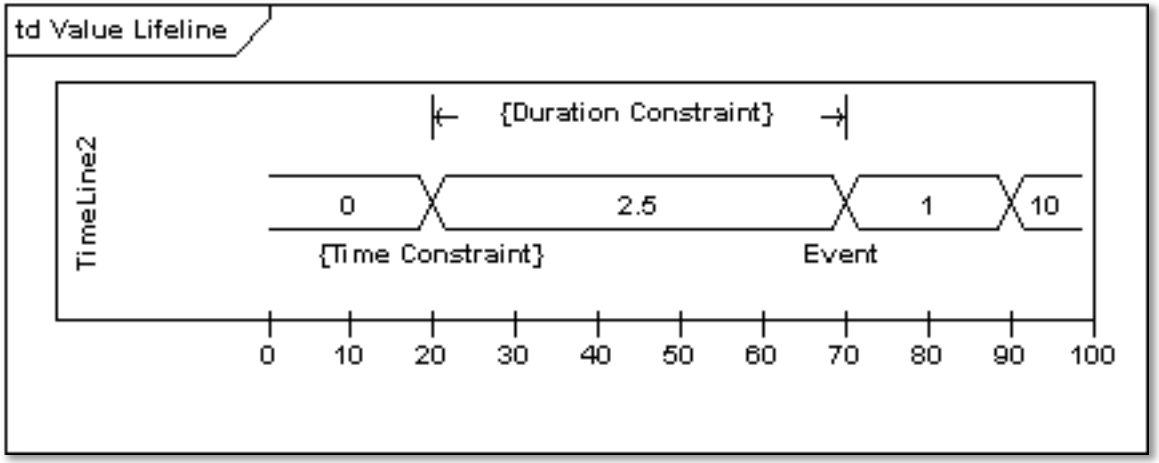
- Descrição de comportamento focada em restrições temporais
- Uma forma especial de diagrama de sequência
  - Eixos estão invertidos: tempo fluí da esquerda para a direita
  - Linhas de vida mostradas horizontalmente
  - Linhas de visa podem mostrar estado ou valor (do objecto que a linha de vida representa)

Timing diagram

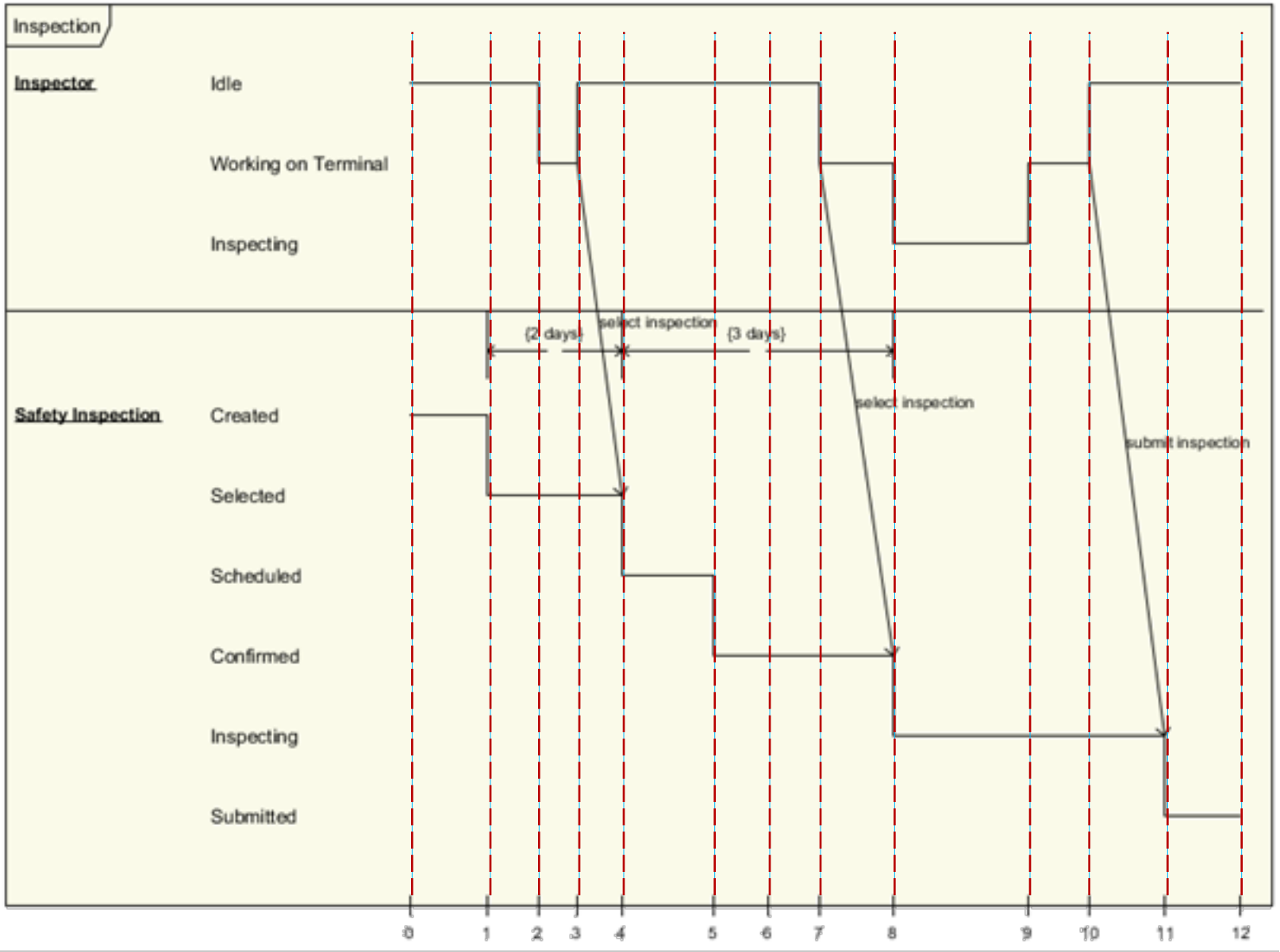
- Linha e vida de estado
  - mostra o estado de uma entidade ao longo do tempo



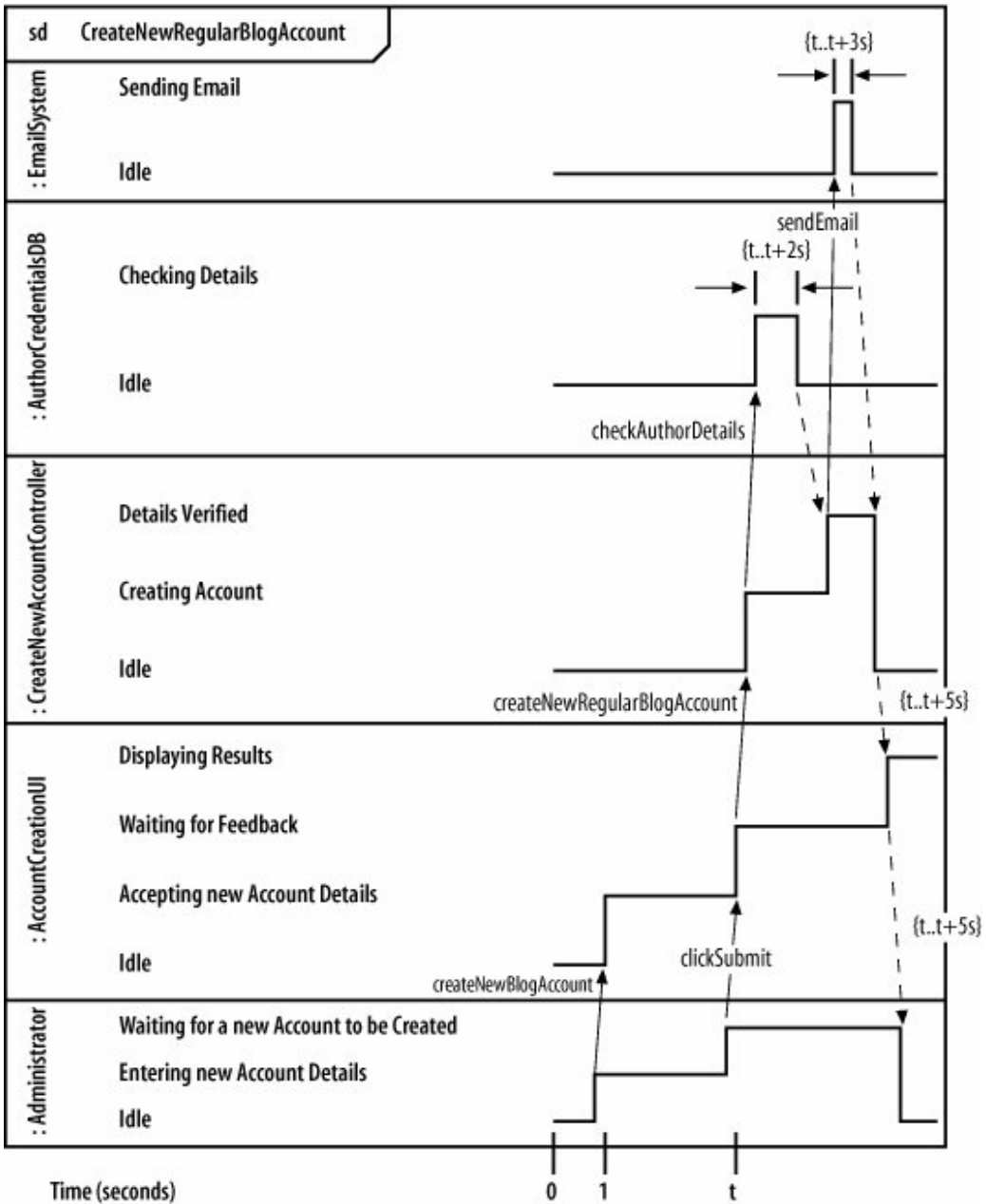
- Linha de vida de valor
  - mostra o valor de uma entidade ao longo do tempo



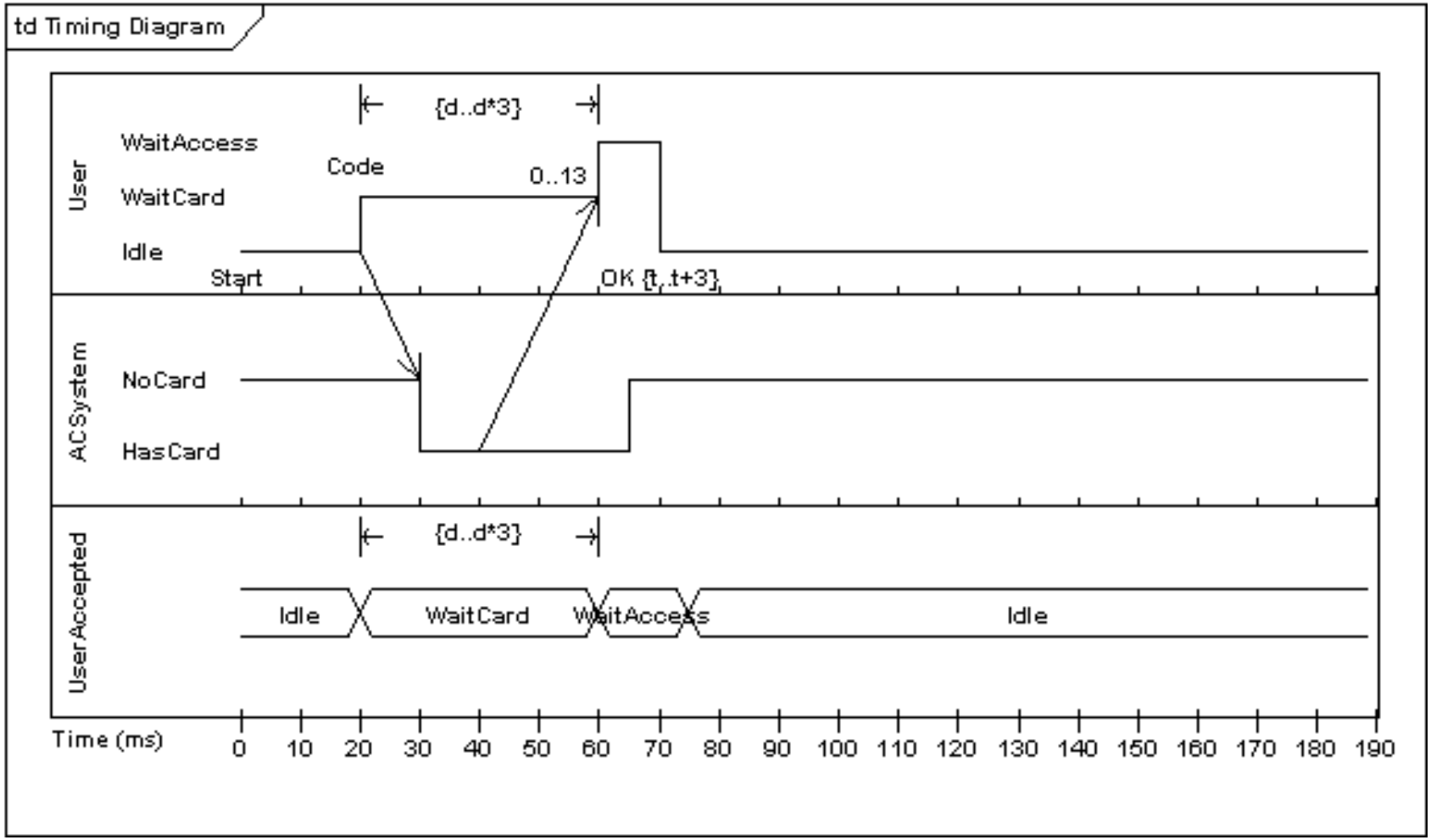
Timing Diagram - Outro Exemplo



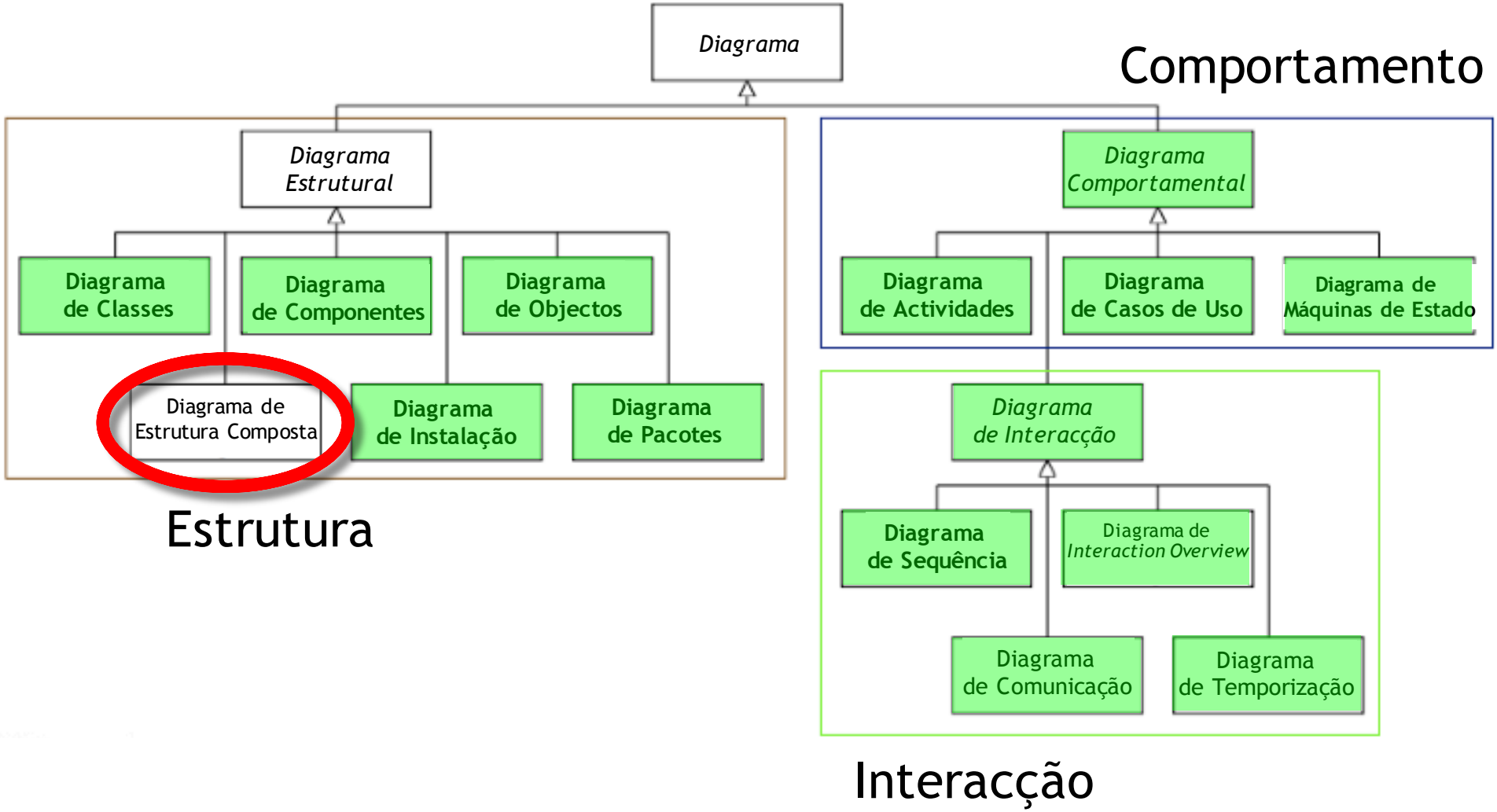
Timing diagram - Exemplo



Timing Diagram - Diferentes linhas de vida



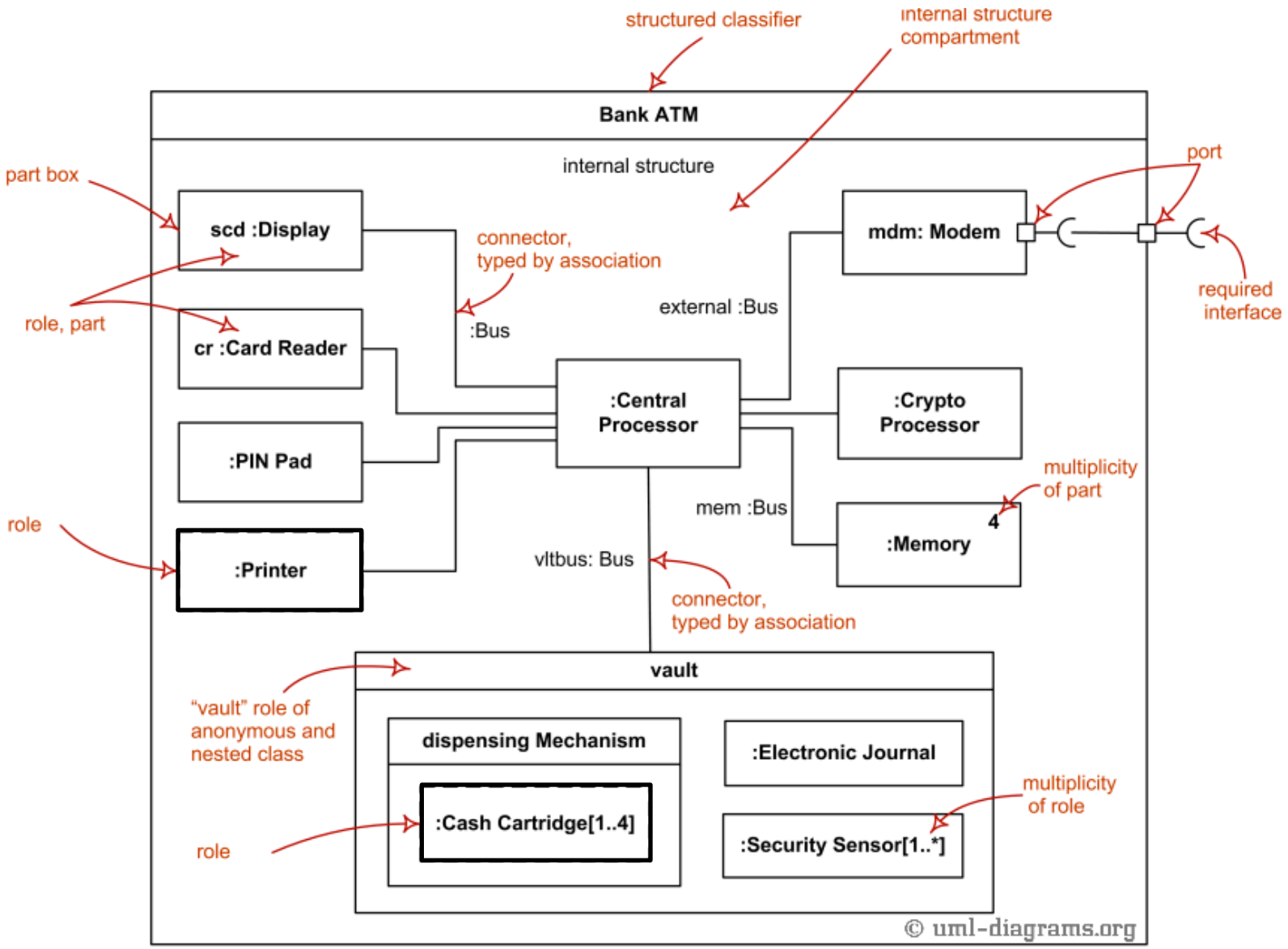
Diagramas da UML 2.x



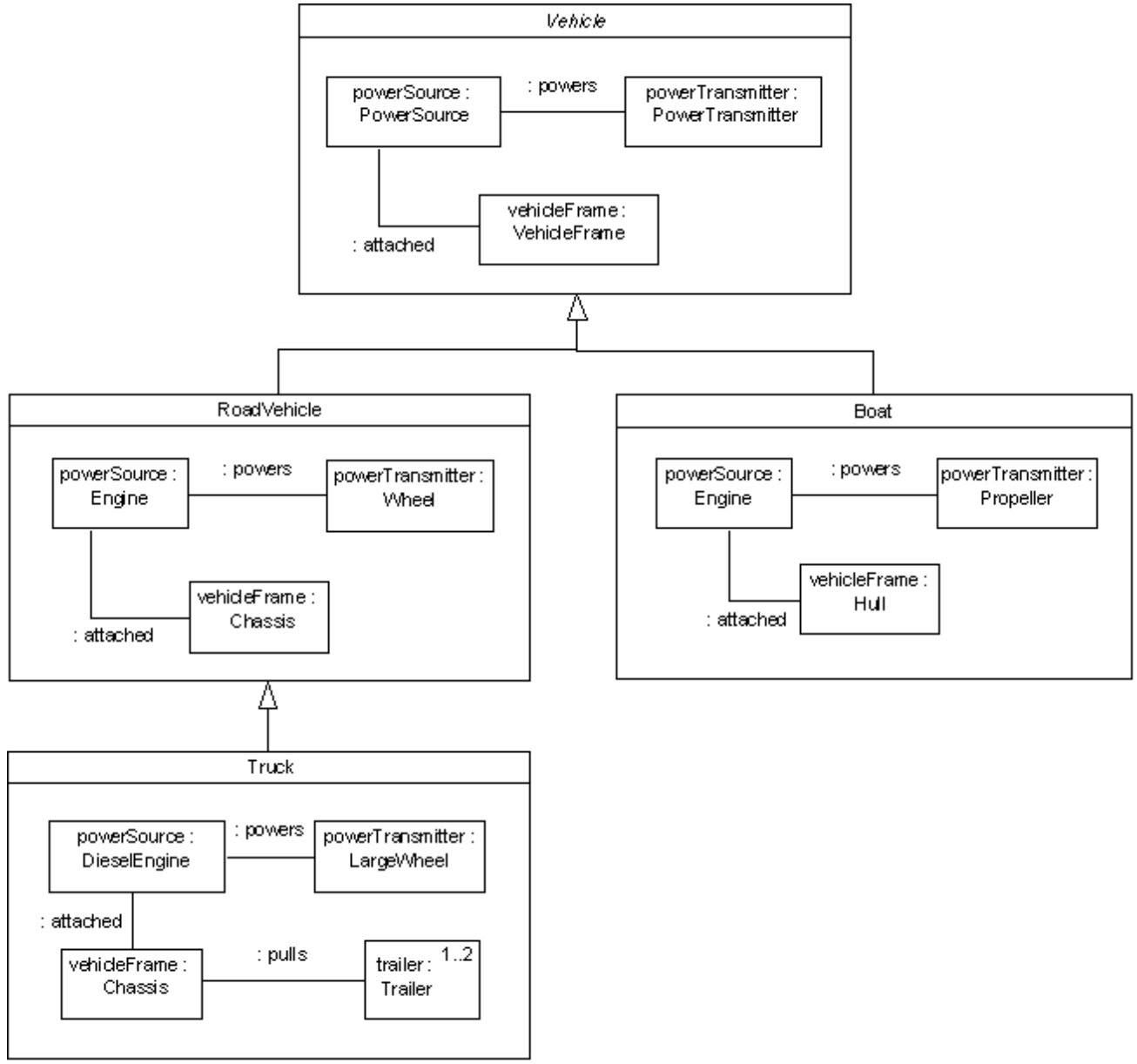
Diagramas de Estrutura Composta

- *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 experior)
  - Conecções a ligar *parts* e portas
- Existe a partir da UML 2.0
  - Pouco documentados

Diagramas de Estrutura Composta

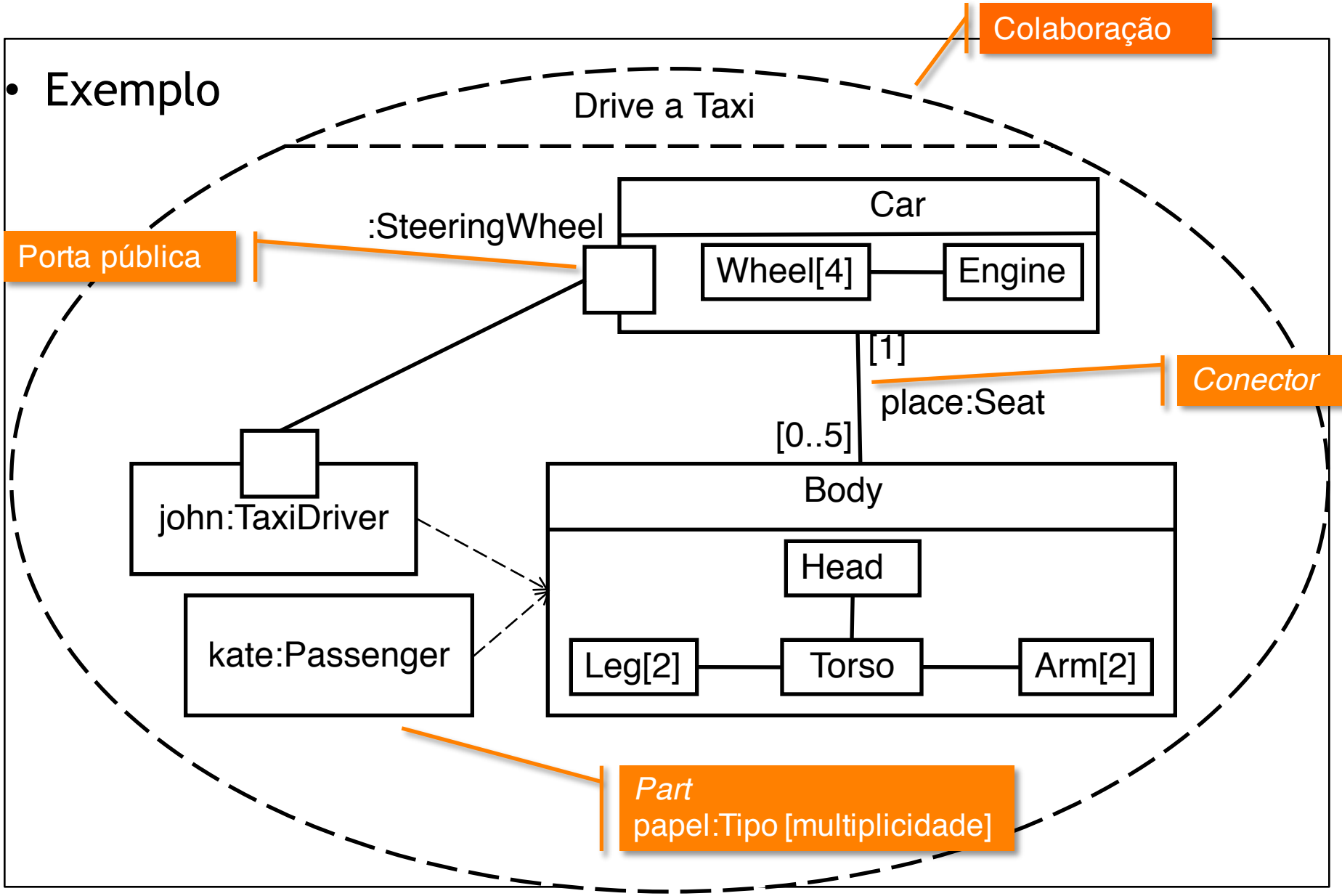


Diagramas de Estrutura Composta

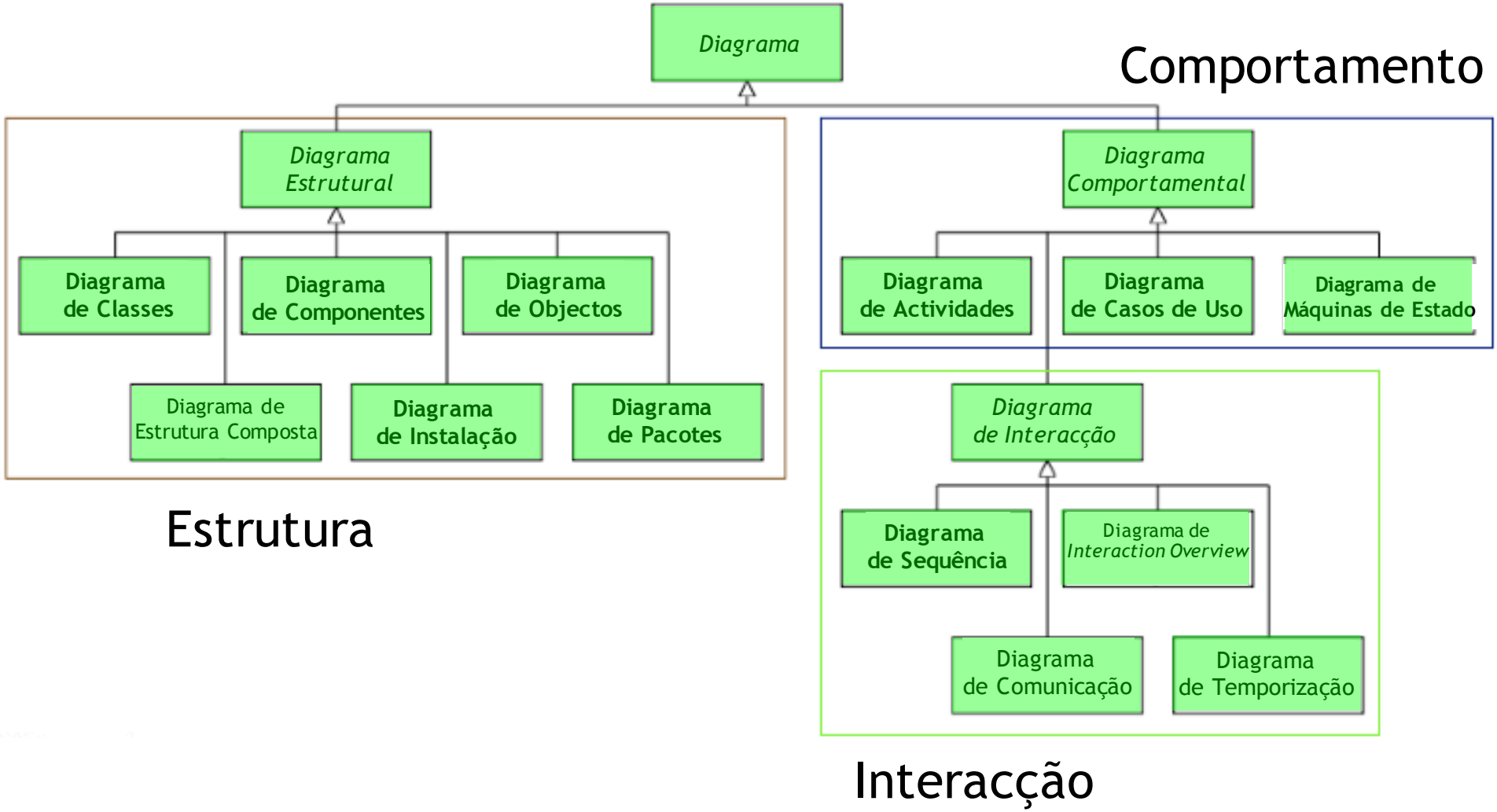


Diagramas de Estrutura Composta

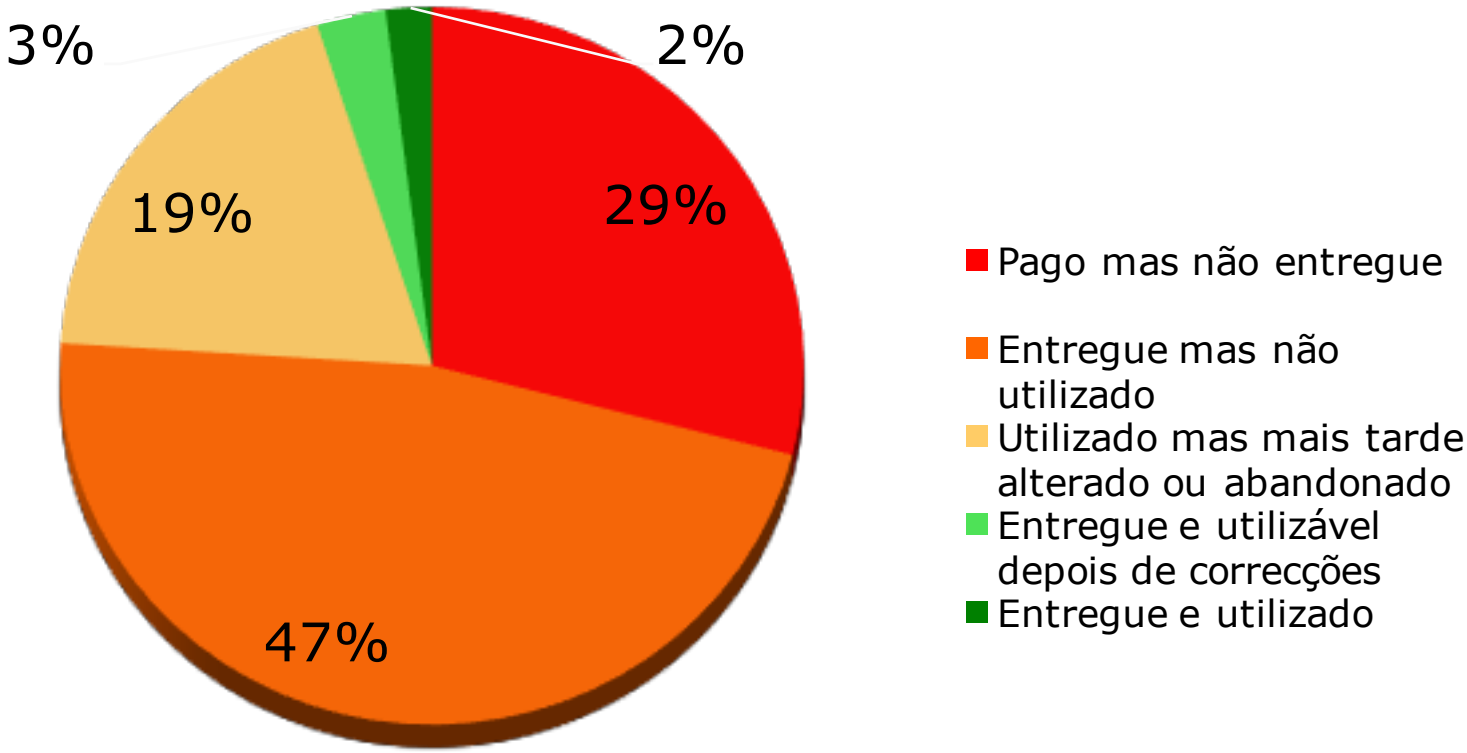
- Exemplo



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).

Fonte: GAO, 1992

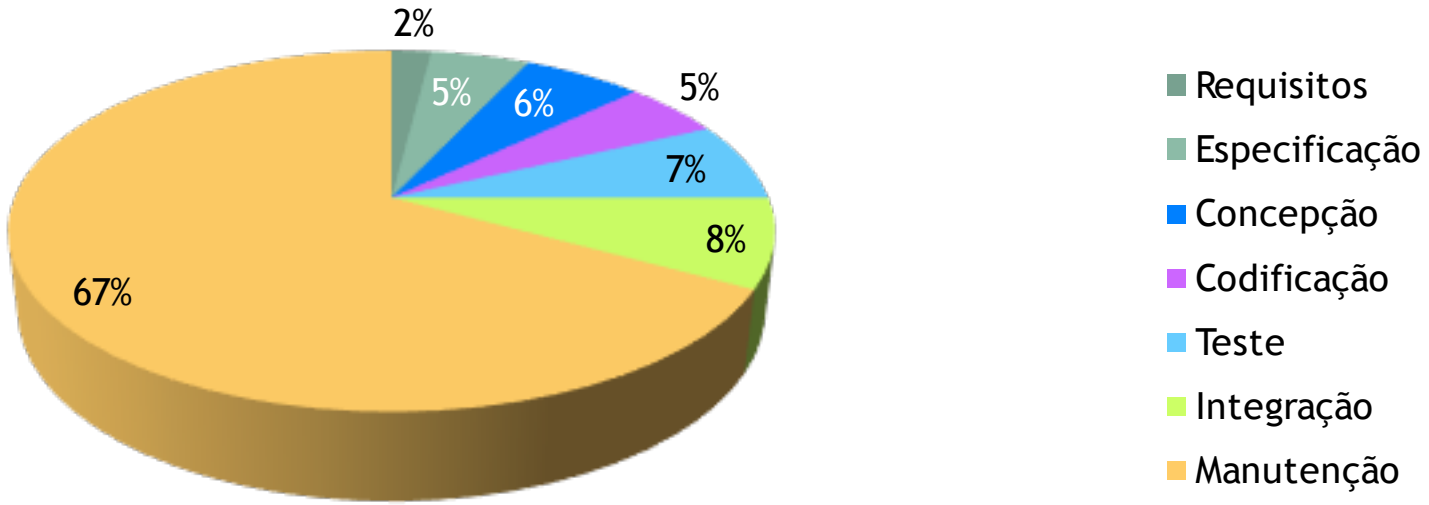
Como vai o desenvolvimento de Software?

- 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!)

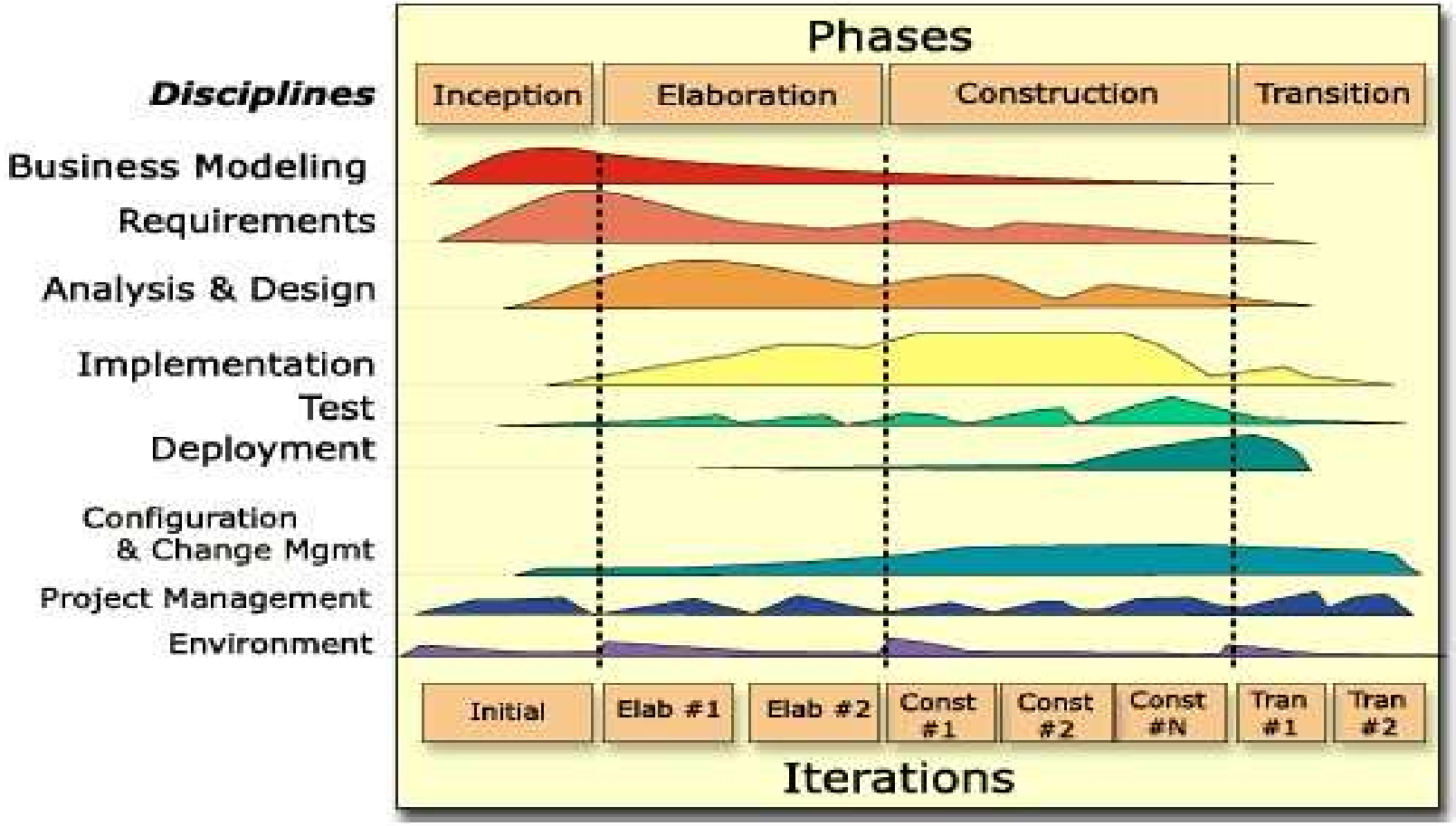
alguns dados sobre grandes projectos (>50,000 loc):

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

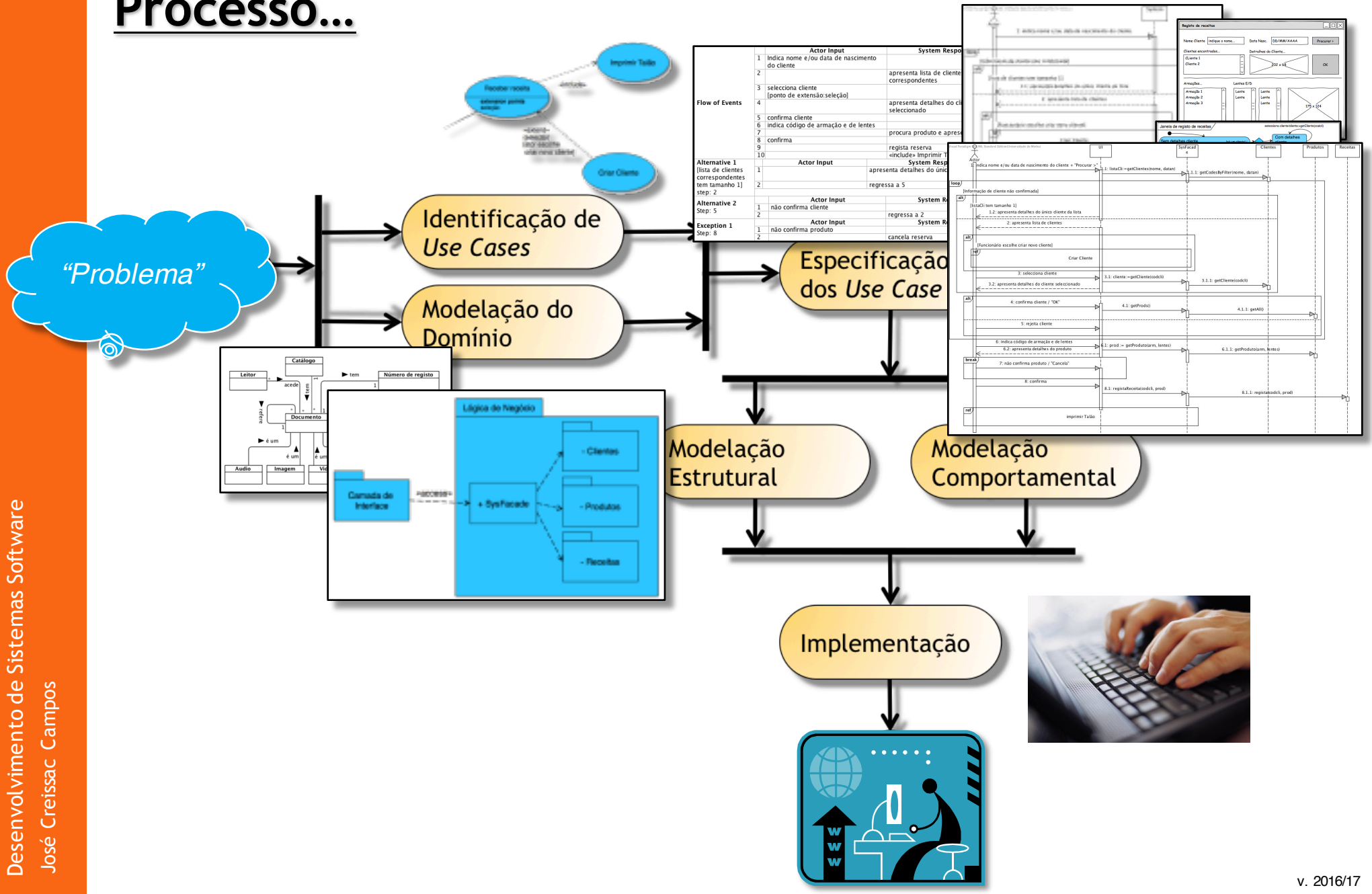
Repartição de custos dos projectos



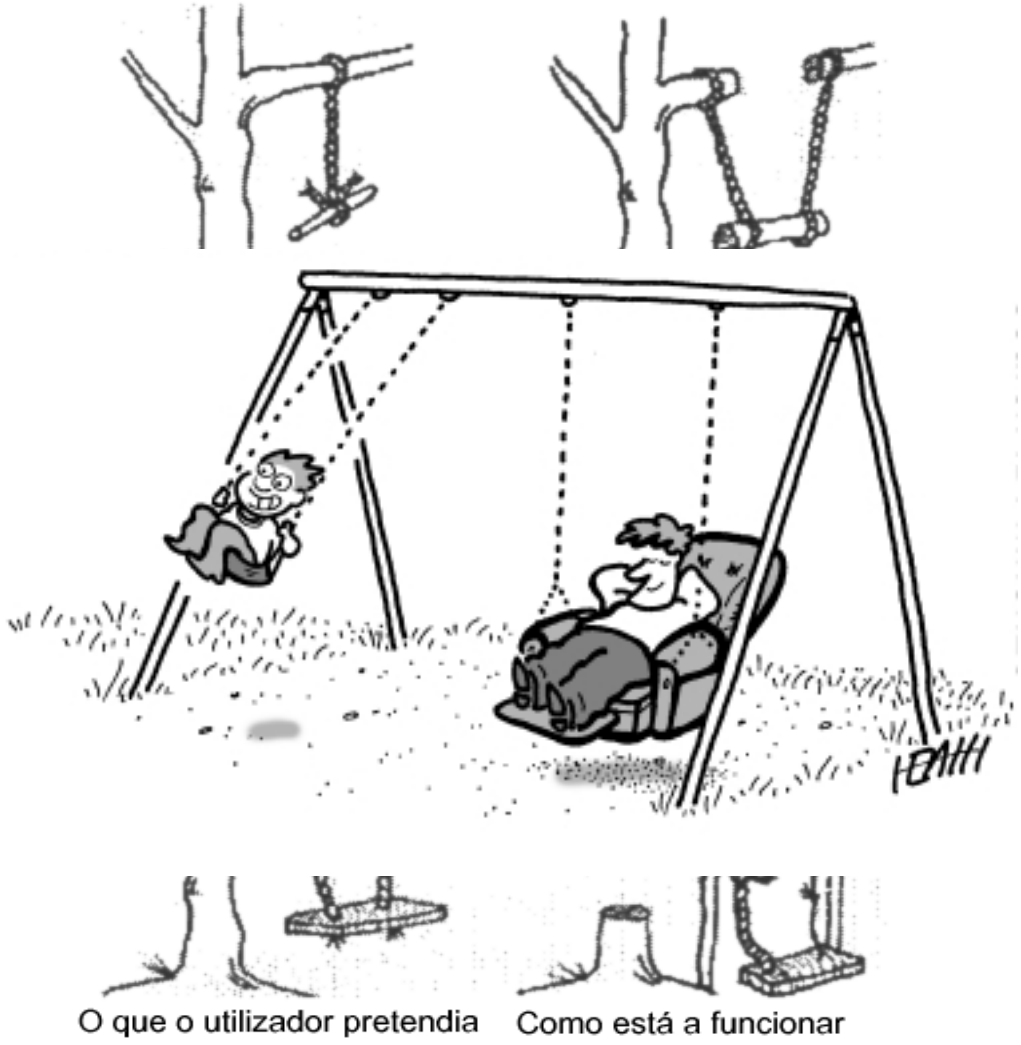
(Rational) Unified Process



Processo...

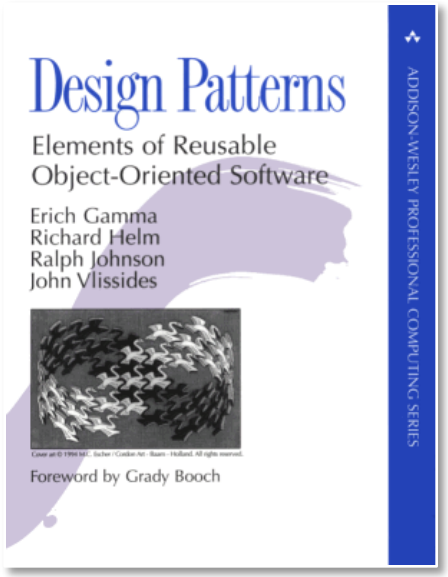

"Problema"
Identificação de Use Cases
Modelação do Domínio
Especificação dos Use Case
Modelação Estrutural
Modelação Comportamental
Implementação
Flow of Events
Alternative 1
Alternative 2
Exception 1
Catálogo
Leitor
Documento
Número de registo
Audio
Imagem
Lógica de Negócio
Camada de Interface
+ SysFacade
- Clientes
- Produtos
- Receitas
UI
SysFacad
Clientes
Produtos
Receitas

Desenvolver um bom sistema não é tarefa trivial

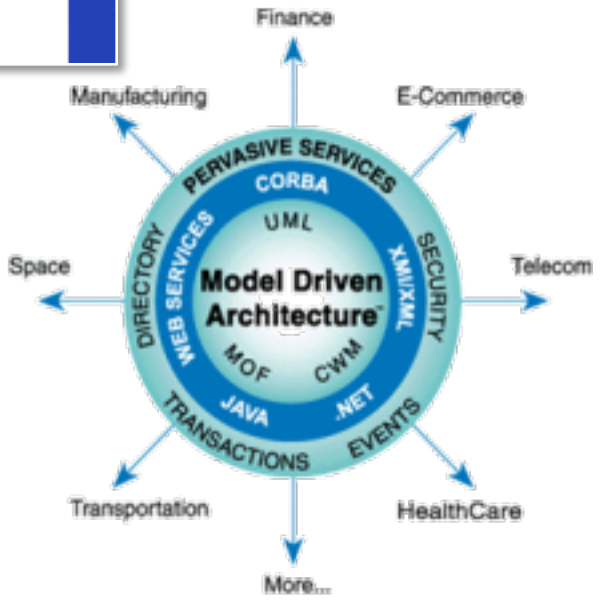


Mais...

- Software design patterns



- Model Driven Engineering
  - Model Driven Arquitecture (OMG)



- Engenharia de Aplicações
  - Desenvolvimento de Aplicações Multi-camada (Web)

Avaliação

- Exame (≥ 9.0) - uma prova escrita sobre a matéria leccionada
  - Exame de consulta
  - Trabalho Prático (≥ 10.0)
  - 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.

- Classificação Final (**≥ 10.0**)

  .6 Exame + .4 Trabalho

Modelação Estrutural/Modelação Comportamental

Sumário

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