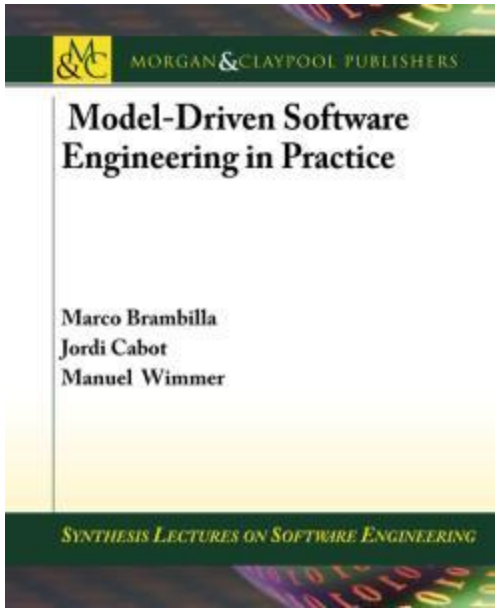
MORGAN & CLAYPOOL PUBLISHERS

Chapter #6

LINGUAGENS DE MODELAGEM EM RESUMO

Teaching material for the book
Model-Driven Software Engineering in Practice
by Marco Brambilla, Jordi Cabot, Manuel Wimmer.
Morgan & Claypool, USA, 2012.



Karen D. Rabelo **(NºUSP: 8437183)**
Stefany D. Rabelo **(NºUSP: 8770800)**

Conteúdo

- DSL vs. GPL
- Exemplo de GPL: UML
- Princípios e dimensões de DSL
- **OCL**

LINGUAGEM DE RESTRIÇÃO DE OBJETO
(OBJECT CONSTRAINT LANGUAGE – OCL)

Conteúdo

- Introdução
- Linguagem central OCL
- Biblioteca padrão OCL
- Ferramenta de apoio
- Exemplos

Motivação

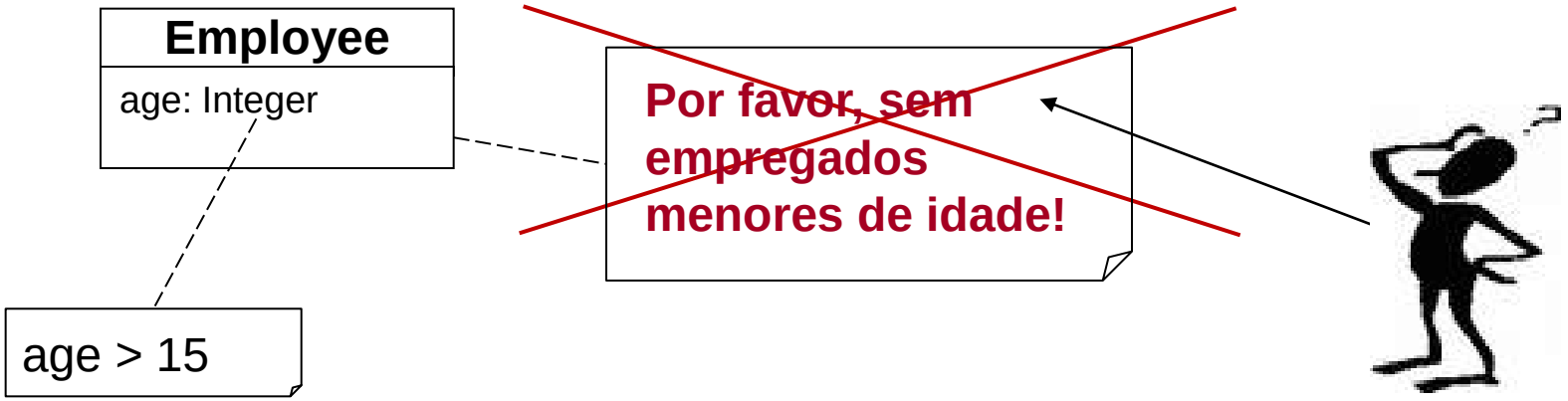
- Linguagens de modelagem gráfica geralmente não permitem a descrição detalhada de todas as facetas de um problema
  - *MOF, UML, ER, …*

- **Restrições** especiais são geralmente adicionadas aos diagramas em uma **linguagem natural**
  - Comumente **ambígua**
  - Sem validação **automática**
  - Sem geração **automática** do código

- A definição de restrições é também crucial na definição de novas linguagens de modelagem (DSLs).

Motivação

- Exemplo 1



Questão Adicional: Como obter todos os empregados menores de 30 anos?

Motivação

- **Linguagens de especificação formal** são a solução
  - Principalmente baseada em **teoria dos conjuntos** ou **lógica de predicados**
  - Requer um bom conhecimento de matemática
  - Principalmente utilizada na área acadêmica e dificilmente na indústria
  - Difícil aprendizagem e aplicação
  - Problemática quando utilizada em grandes sistemas

- ***Linguagem de Restrição de Objeto (OCL*)**: Combinação de linguagem de modelagem e linguagem de especificação formal
  - Formal, precisa, única
  - Uma sintaxe intuitiva é a chave para **grandes grupos de usuários**
  - Sem linguagem de programação (algoritmos, APIs, …)
  - Ferramentas de suporte: *parser, constraint checker, codegeneration,…*

Uso de OCL

- Restrições em modelos UML
  - Invariantes para classes, interfaces, estereótipos, …
  - Pré e pós condições para operações
  - Guardas para mensagens e transição de estados
  - Especificação de mensagens e sinais
  - Cálculo de atributos derivados e extremidades de associações
- Restrições em meta modelos
  - Invariantes para classes do meta modelo
  - Regras para a definição de uma boa formação do meta modelo
- Linguagem de consulta para modelos
  - Analogamente ao SQL para DBMS, XPath e XQuery para XML
  - Usado em linguagens de transformação

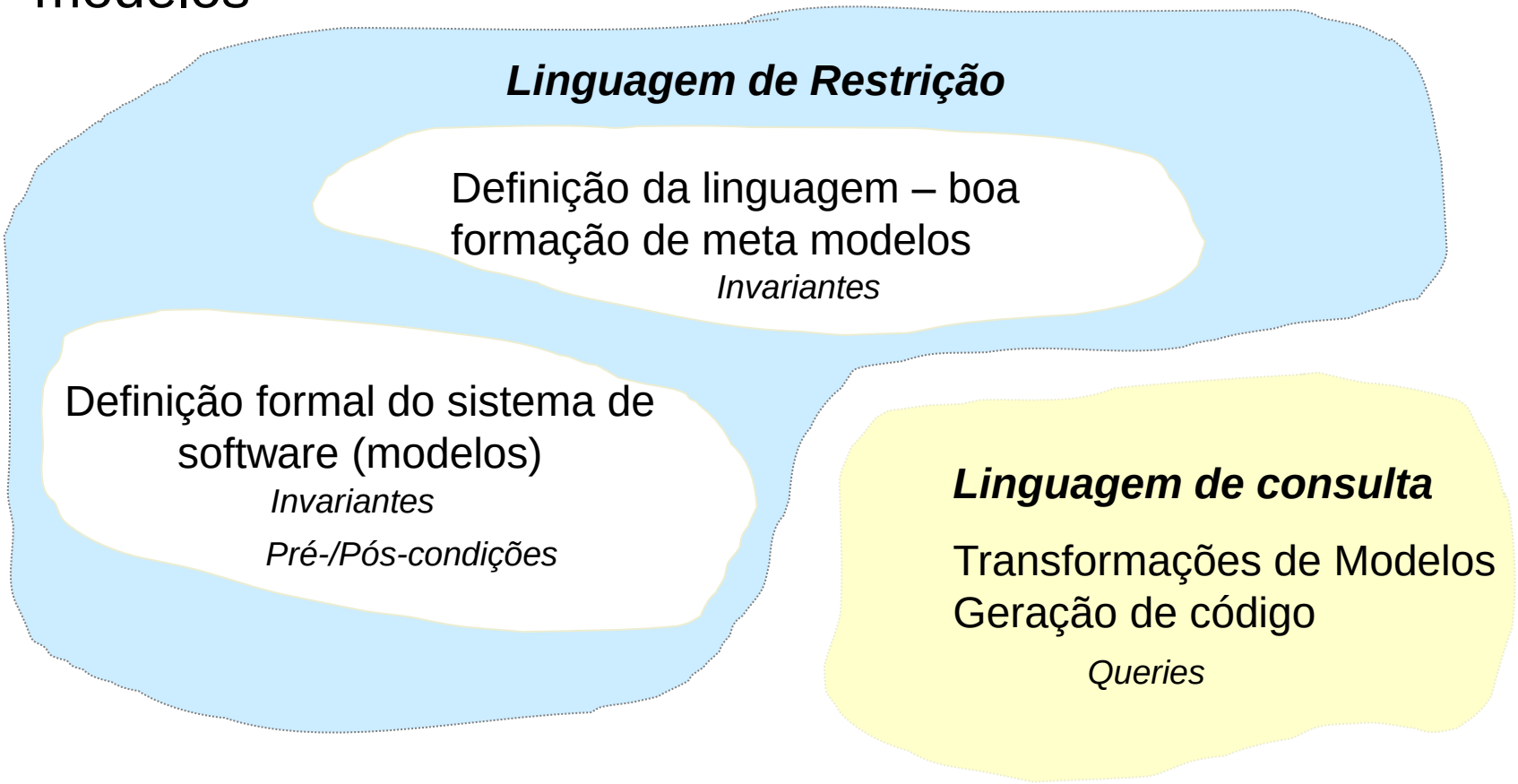
Uso de OCL

- Campo de aplicação de OCL
  - Invariantes — **context** *C* **inv:** *I*
  - Pré/Pós condições — **context** *C::op() : T* **pre:** *P* **post:** *Q*
  - Operações de consulta — **context** *C::op() : T* **body:** *e*
  - Valores iniciais — **context** *C::p : T* **init:** *e*
  - Atributos derivados — **context** *C::p : T* **derive:** *e*
  - Definição de atributo/operação — **context** *C* **def:** *p : T = e*

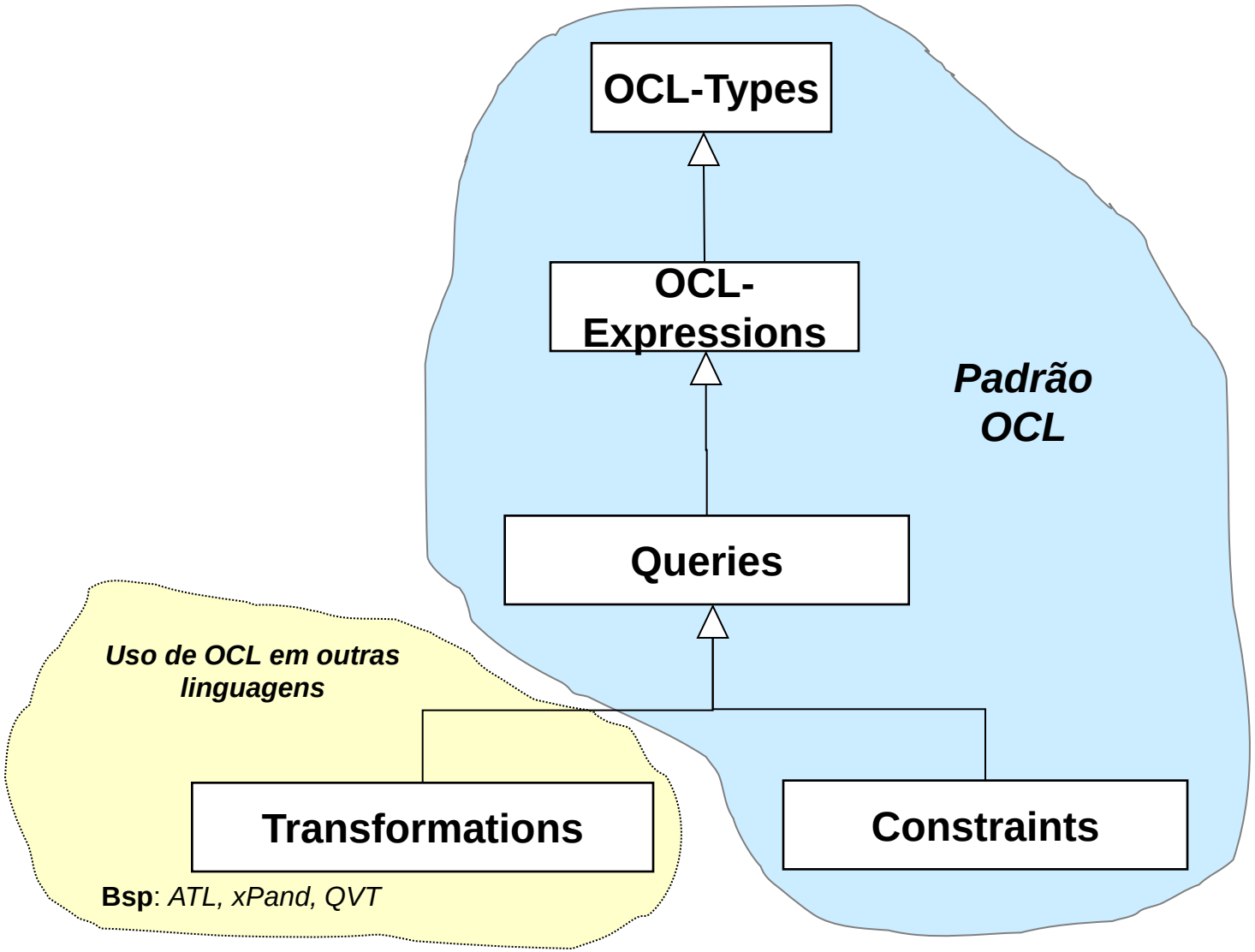
- Atenção: Efeitos colaterais não são permitidos!
  - Operação `C::getAtt : String body: att` permitido em OCL
  - Operação `C::setAtt(arg) : T body: att = arg` **não** permitido em OCL

Uso de OCL

- **Campo de aplicação** de OCL em engenharia dirigida a modelos



Uso de OCL

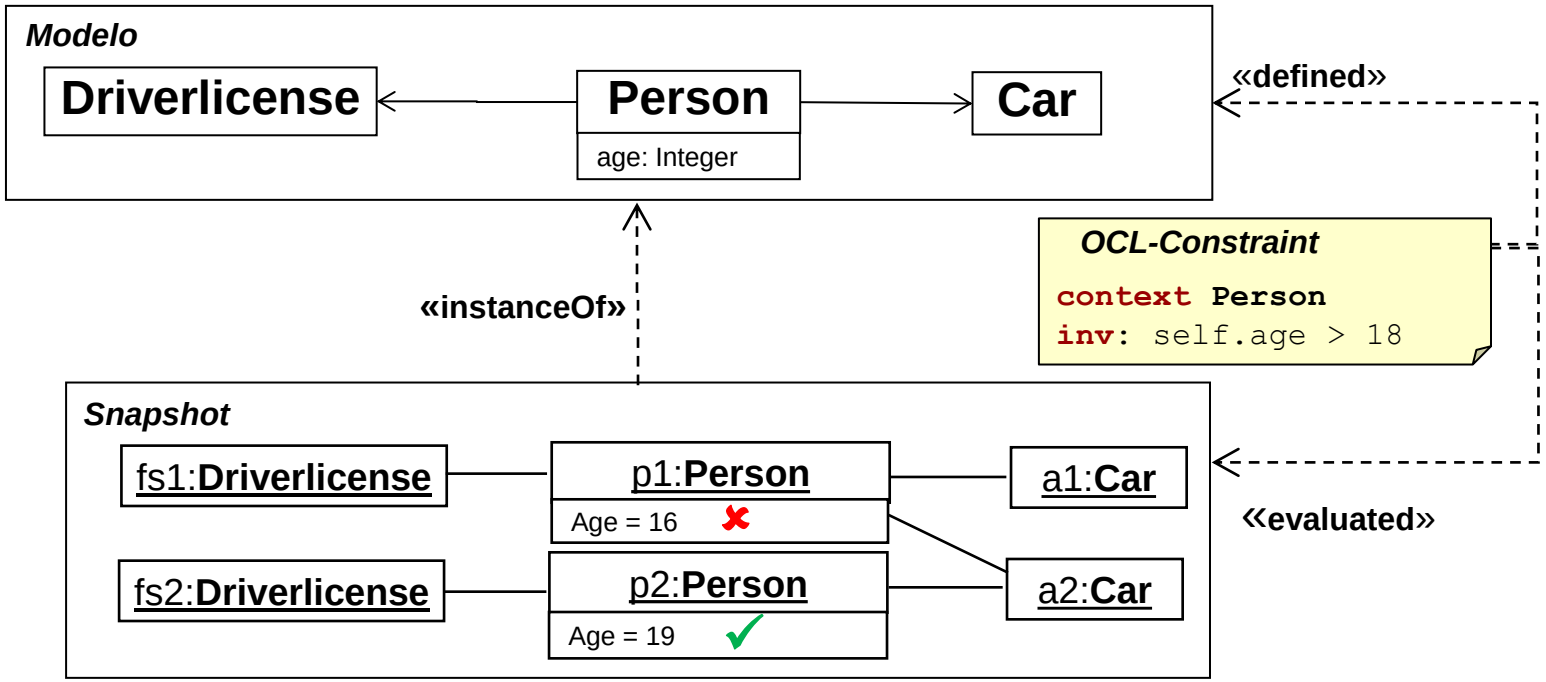

OCL-Types
OCL-Expressions
Padrão OCL
Queries
Uso de OCL em outras linguagens
Transformations
Constraints
Bsp: ATL, xPand, QVT

Uso de OCL

Como OCL funciona?

- **Restrições** são definidas em nível de modelagem
  - Base: Classes e suas propriedades
- Informações sobre o **gráfico do objeto** são **consultadas**
  - Representa o status do sistema, também chamado de ***snapshot***
- **Analogamente** as linguagens de consulta XML
  - XPath/XQuery consultam documentos XML
  - Scripts são baseados em informações do esquema XML

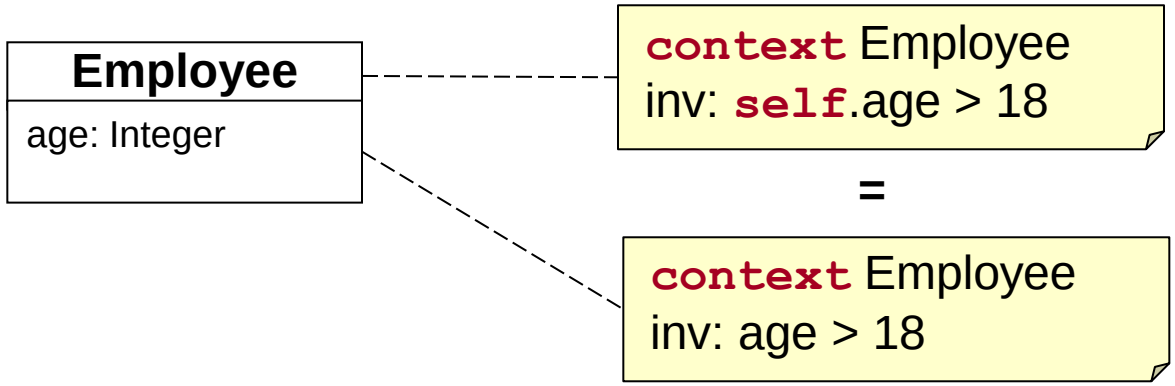
- Exemplos



Projeto de OCL

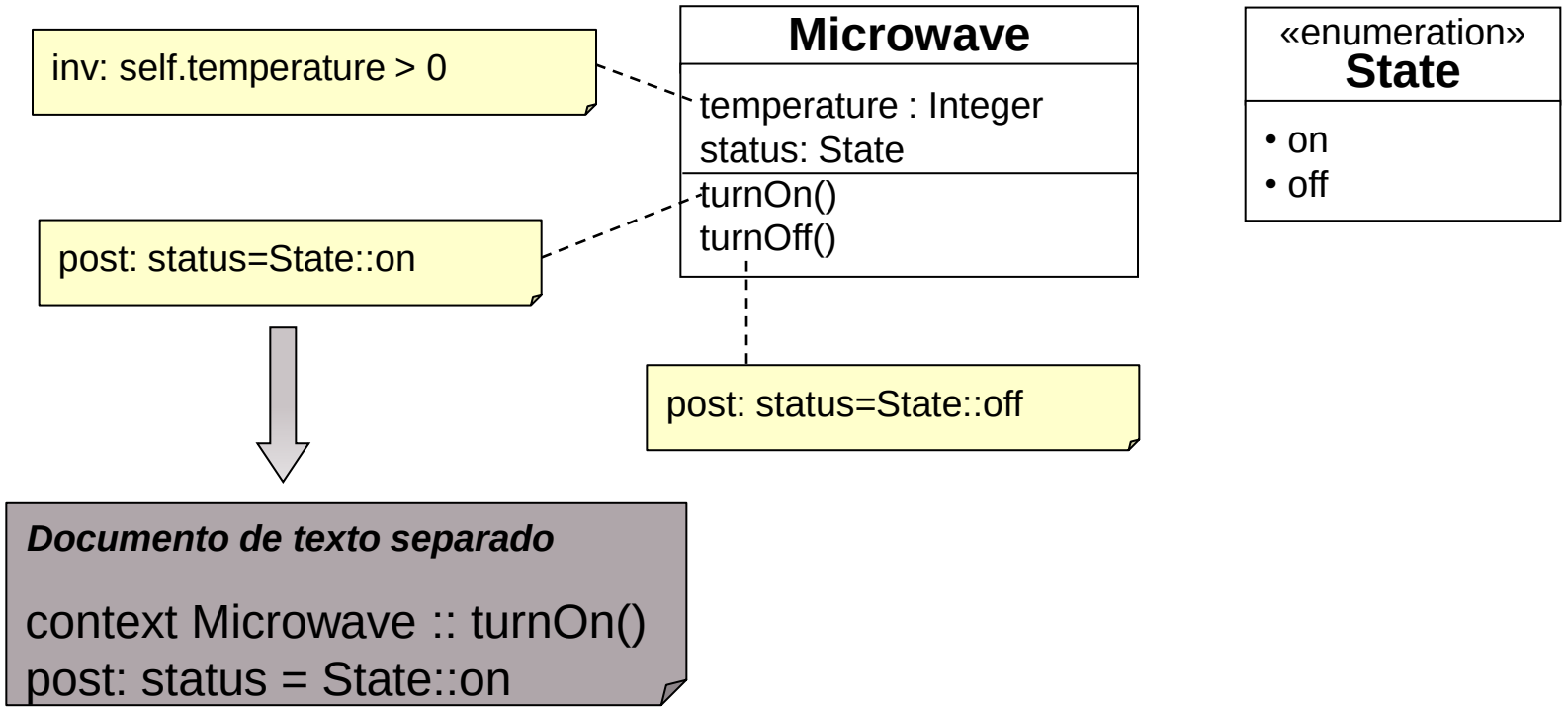
- Um contexto deve ser associado a cada declaração OCL
  - **Endereço inicial –** para qual elemento do modelo a declaração OCL se aplica
  - Especifica quais elementos do modelo serão afetados com o uso das expressões
- O contexto é especificado pela palavra reservada **`context`** seguida do nome do elemento do modelo (principalmente nomes de classe)
- A palavra reservada **`self`** especifica a instância atual, a qual será avaliada pela invariante (contexto da instância).
  - **`self`** pode ser omitido se o contexto da instância é único

- Exemplo:



Projeto de OCL

- OCL pode ser especificado de **duas** diferentes formas
  - Como um comentário **diretamente** no diagrama de classes (contexto descrito por conexão)
  - Documento separado



Tipos

- **OCL** é uma linguagem tipada
  - Cada **objeto, atributo** e **resultado** de uma operação ou navegação é atribuído a uma **range de valores** (tipo)

- **Tipos pré-definidos**
  - **Básicos**
    - Tipos simples: *Integer, Real, Boolean, String*
    - Tipos específicos de OCL: *AnyType, TupleType, InvalidType, …*
  - **Valores definidos, tipos parametrizados**
    - Abstract supertyp: Collection(T)
    - Set(T) – sem duplicações
    - Bag(T) – duplicações permitidas
    - Sequence(T) – Bag de elementos ordenados, extremidade de associação {*ordenado*}
    - OrderedSet(T) – Set de elementos ordenados, extremidade de associação {*ordenado, único*}

- **Tipos definidos pelo usuário**
  - Instâncias de *Classes* em MOF e instâncias indiretas de *Classificadores* em UML são tipos
  - *EnumerationType* – conjunto de valores definidos pelo usuário para definir constantes

Tipos

Exemplos

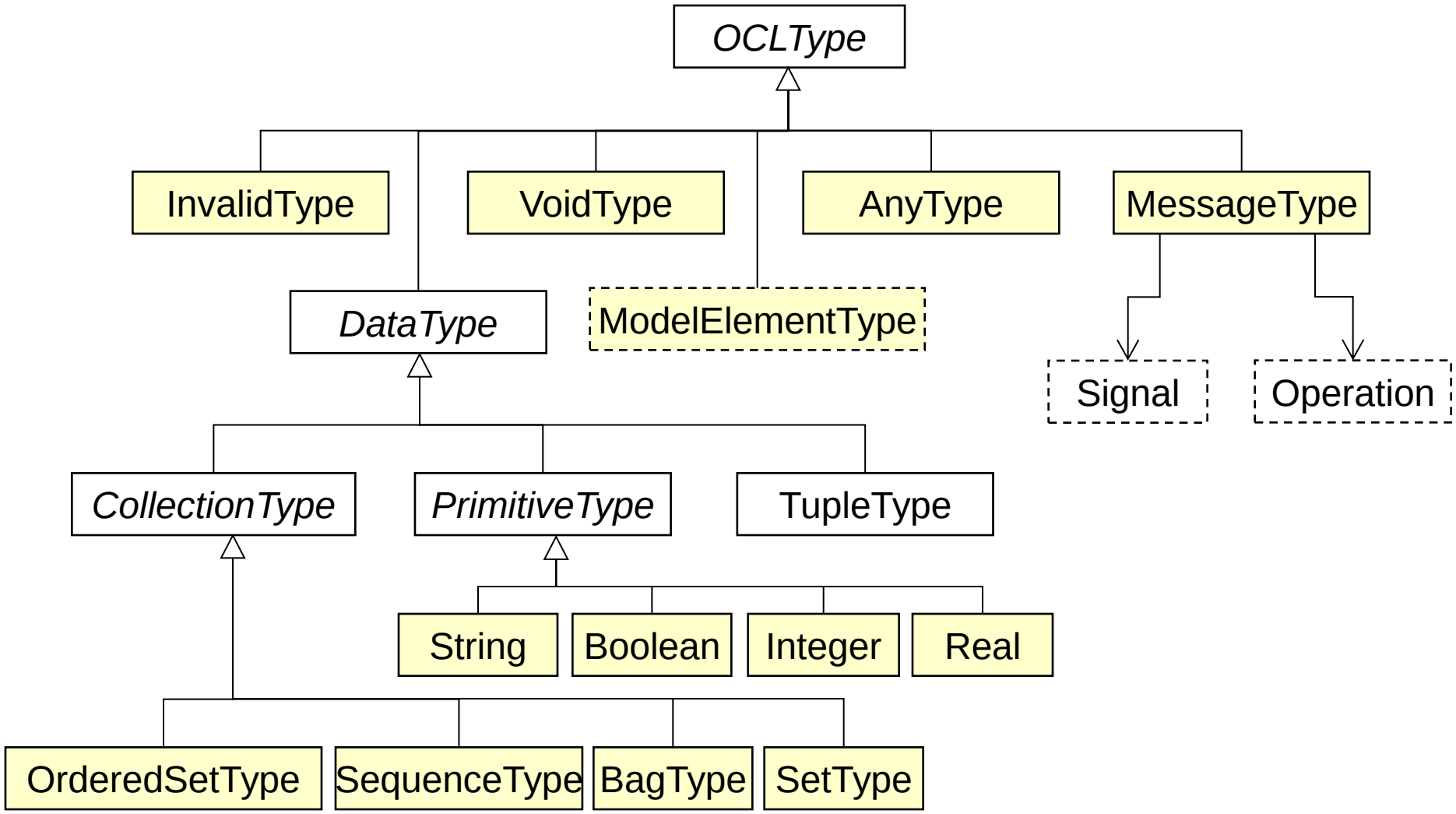
- **Básicos**
  - true, false : *Boolean*
  - -17, 0, 1, 2 : *Integer*
  - -17.89, 0.01, 3.14 : *Real*
  - “Hello World” : *String*

- **Valores definidos, tipos parametrizados**
  - Set{ Set{1}, Set{2, 3} } : *Set(Set(Integer))*
  - Bag{ 1, 2.0, 2, 3.0, 3.0, 3 } : *Bag(Real)*
  - Tuple{ x = 5, y = false } : *Tuple{x: Integer, y : Boolean}*

- **Tipos definidos pelo usuário**
  - Passenger : *Class*, Flight : *Class*, Provider : *Interface*
  - Status::started - enum Status {started, landed}

Tipos

Meta modelo OCL (extração)



Expressões

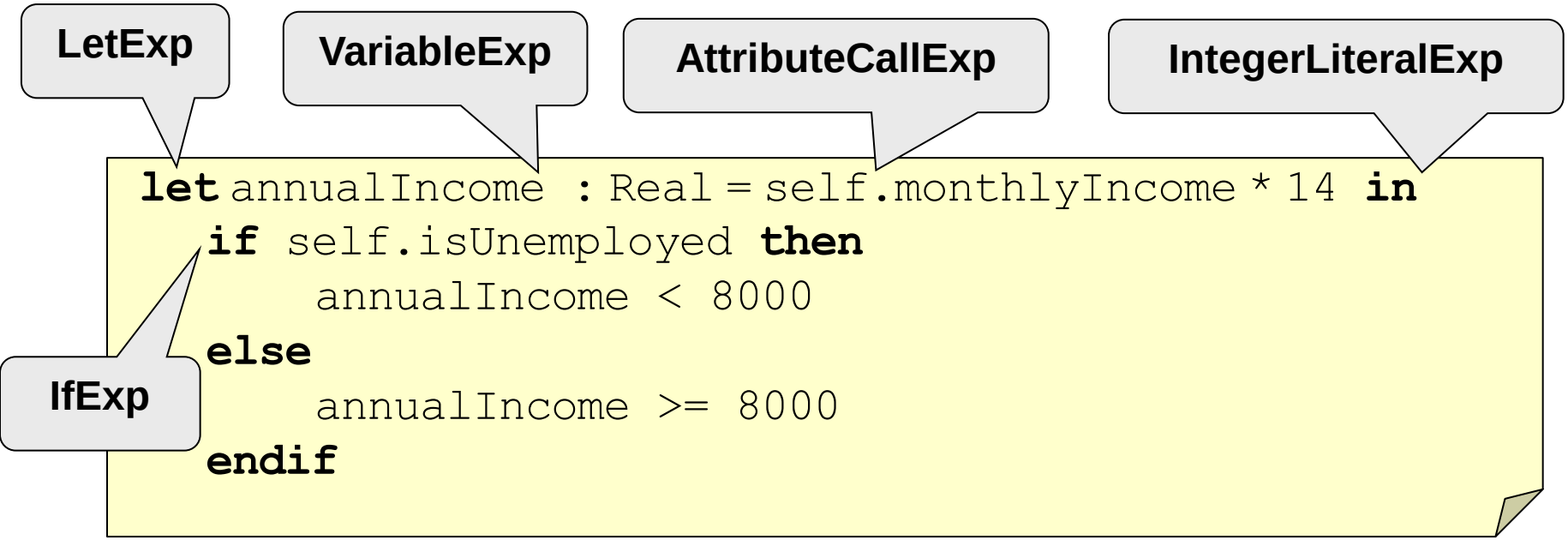
- Cada expressão OCL é uma instância indireta de *OCLExpression*
  - Calculada em um certo ambiente – cf. context
  - Cada expressão OCL possui um **tipo de valor de retorno**
  - **Restrição OCL é uma expressão OCL com um valor *Boolean* de retorno**

- **Expressões OCL simples**
  - *LiteralExp, IfExp, LetExp, VariableExp, LoopExp*

- **Expressões OCL para consulta de informações de modelo**
  - *FeatureCallExp* – Superclasse abstrata
  - *AttributeCallExp* – Atributos da consulta
  - *AssociationEndCallExp* – extremidades de associações de consulta
    - Usando nomes de funções; se nenhum padrão foi especificado, nomes de classes minúsculos devem ser utilizados (se únicos)
  - *AssociationClassCallExp* – classes de associação de consulta (somente em UML)
  - *OperationCallExp* – Chamada de operações de consulta
    - Calcular um valor, mas **não** alterar o estado do sistema!

Expressões

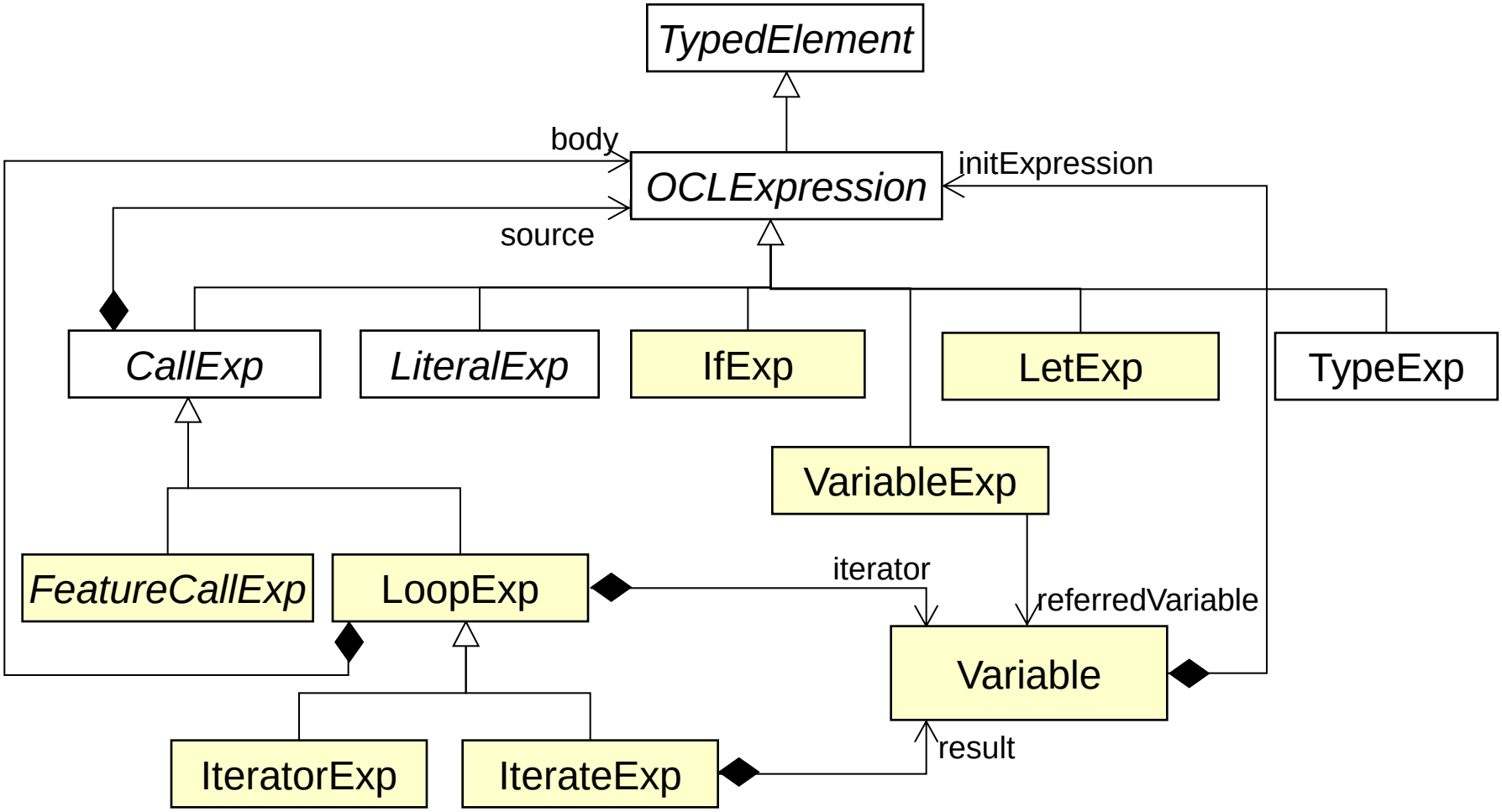
- Exemplos para *LiteralExp, IfExp, VariableExp, AttributeCallExp*



- A **Sintaxe Abstrata** do OCL está descrita como **meta model**
- **Mapeando da sintaxe abstrata para a sintaxe concreta**
  - *IfExp -> **if** Expression **then** Expression **else** Expression **endif***

Expressões

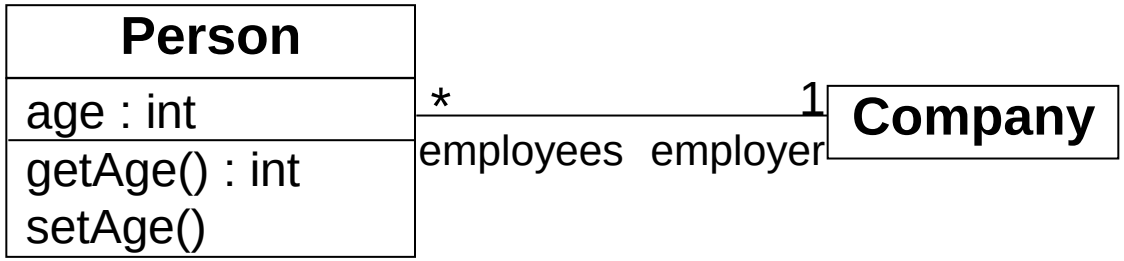
Meta modelo OCL (extração)



LiteralExp: *CollectionLiteralExp*, *PrimitiveLiteralExp*, *TupleLiteralExp, EnumLiteralExp*

Consulta de informações do modelo

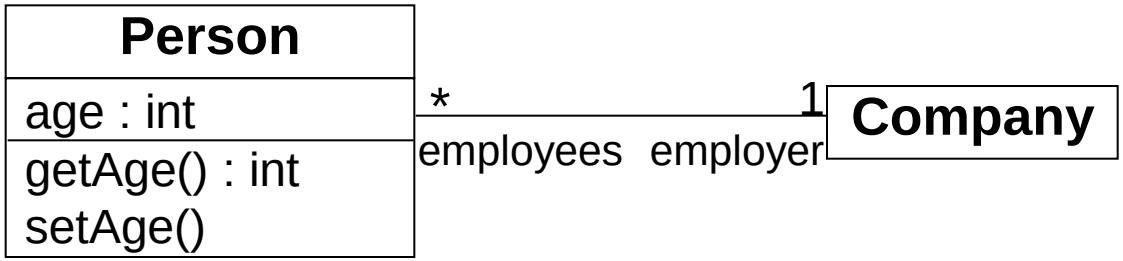
- Instância do contexto
  - `context` **`Person`**
- AttributeCallExp
  - `self.age : int`

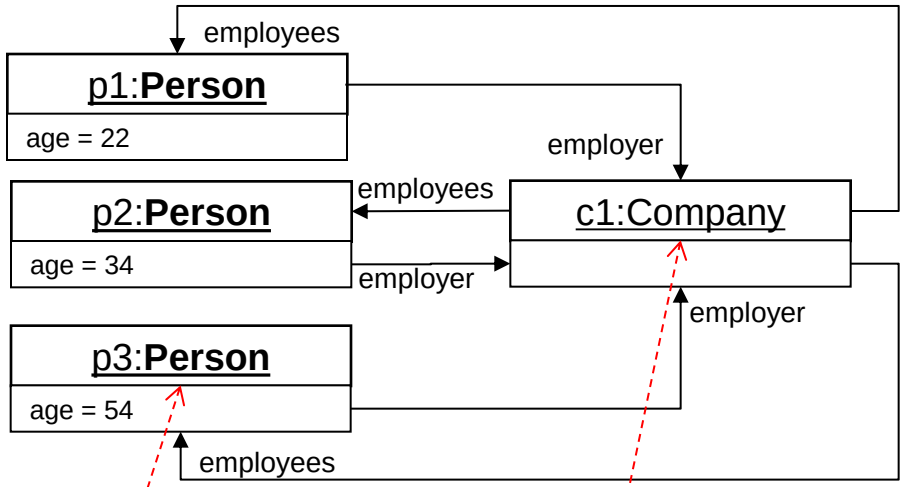


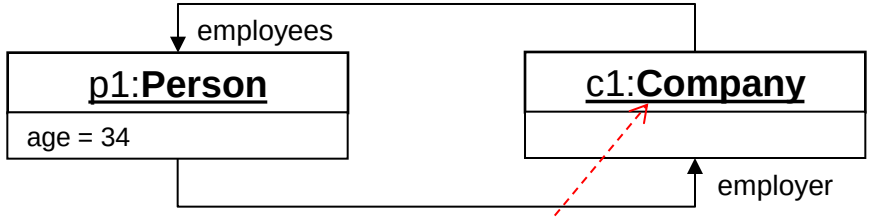
- OperationCallExp
  - Operações não devem possuir **efeitos colaterais**
  - Permitidos: `self.getAge() : int`
  - **Não permitidos**: `self.setAge()`
- AssociationEndCallExp
  - Navegar até o final da associação oposta usando nomes de funções
    `self.employer` – Valor de retorno do tipo ***Company***
  - Navegação geralmente resulta em uma coleção de objetos – Exemplo
    `context` **`Company`**
    `self.employees` – Valor de retorno do tipo **Set (Person)**

Consulta de informações do modelo

Exemplos







context Person
self.employer

`c1 : Company`

context Company
self.employees

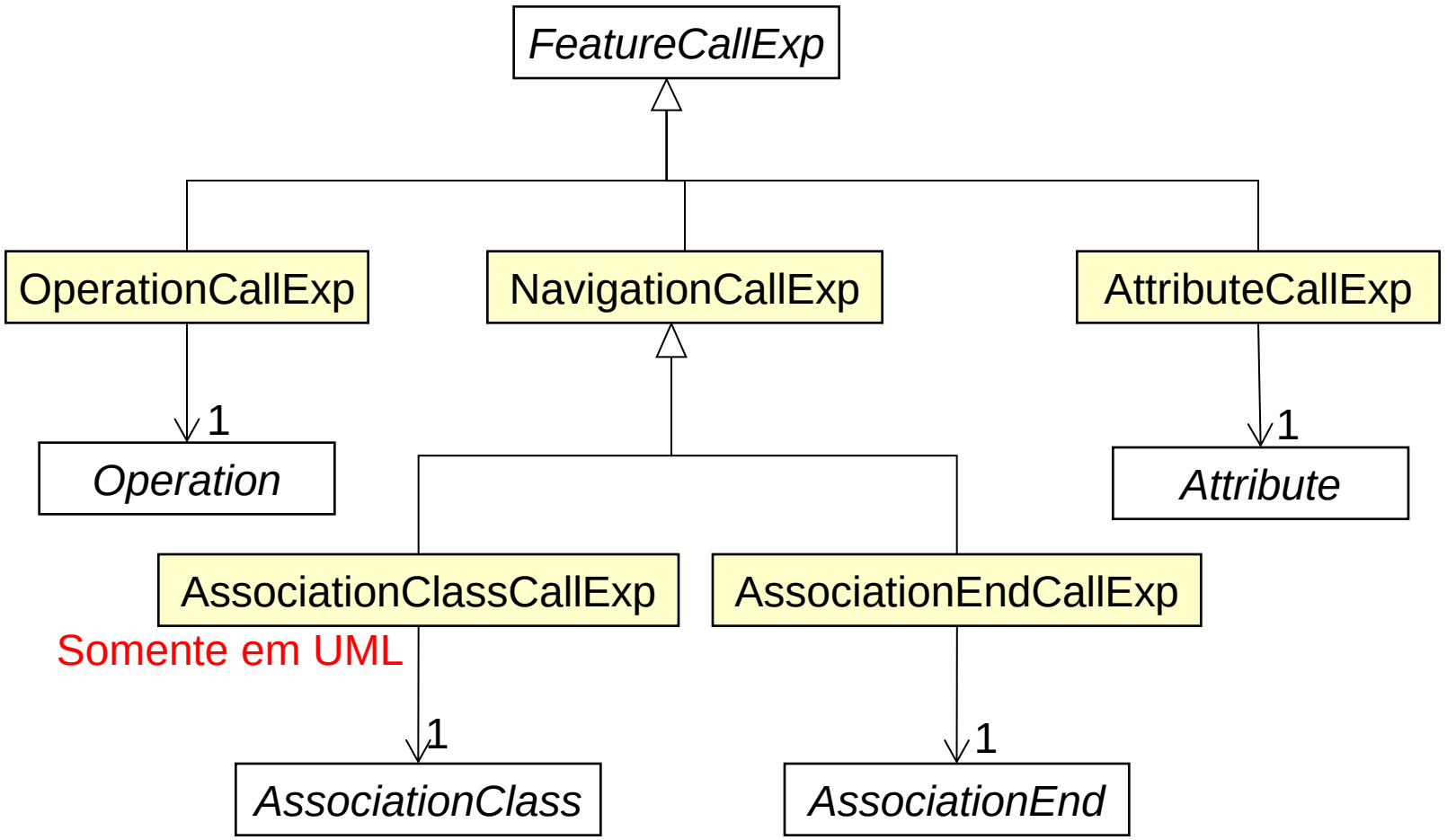
`Set{p1,p2,p3} : Set(Person)`

context Company
self.employees

`Set{p1} : Set(Person)`

Consulta de informações do modelo

Meta modelo OCL (extração)



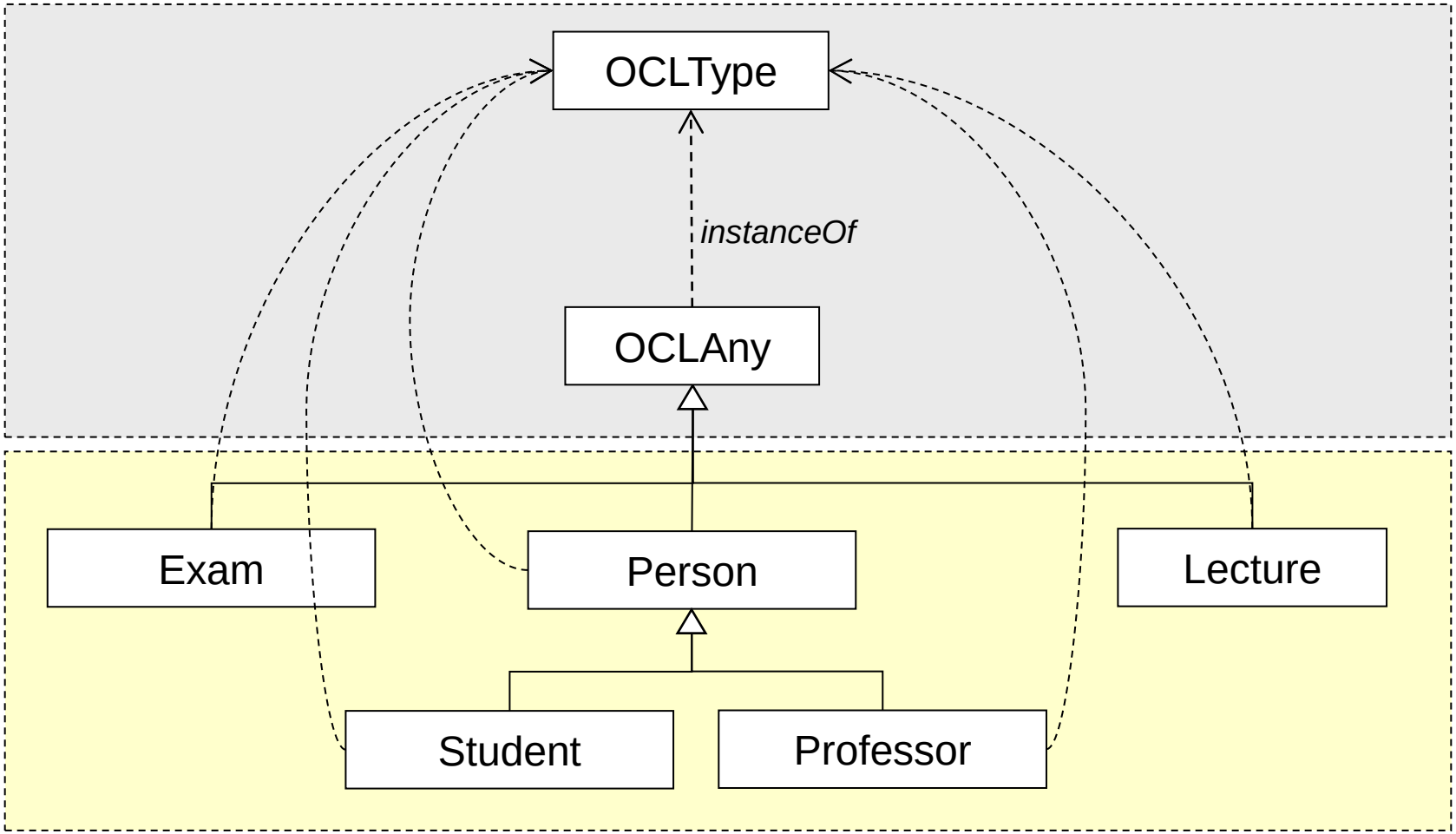
Biblioteca OCL: Operações para OclAny

- *OclAny* - **Supertipo** de todos os outros tipos em OCL
  - **Operações** são **herdadas** por todos os outros tipos.
- **Operações** de *OclAny* (extração)
  - O objeto receptor é denotado por *obj*

Operação	**Resultado**
=(obj2:OclAny):Boolean	True, se *obj2* e *obj* referenciam o mesmo objeto
oclIsTypeOf(type:OclType):Boolean	True, se *type* é o tipo de *obj*
oclIsKindOf(type:OclType): Boolean	True, se *type* é um tipo direto ou indireto do supertipo ou o tipo de *obj*
oclAsType(type:Ocltype): Type	O resultado é um *obj* do tipo *type*, ou *undefined*, se o tipo atual de *obj* não for *type* nem um subtipo direto ou indireto dele (casting)

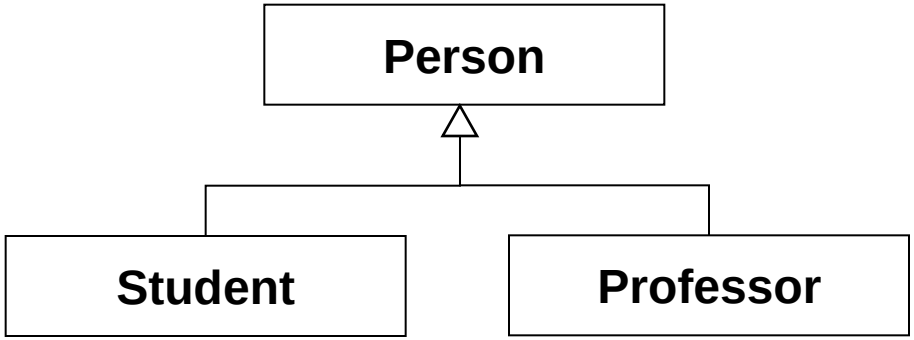
Operações para OclAny

Ambiente pré-definido para tipos de modelo



Operações para OclAny

- ***oclIsKindOf vs. oclIsTypeOf***



context **Person**
self.oclIsKindOf(Person) : *true*
self.oclIsTypeOf(Person) : *true*
self.oclIsKindOf(Student) : *false*
self.oclIsTypeOf(Student) : *false*

context **Student**
self.oclIsKindOf(Person) : *true*
self.oclIsTypeOf(Person) : *false*
self.oclIsKindOf(Student) : *true*
self.oclIsTypeOf(Student) : *true*
self.oclIsKindOf(Professor) : *false*
self.oclIsTypeOf(Professor) : *false*

Operações para Tipos Simples

- Tipos simples **Pré-definidos**
  - Integer {Z}
  - Real {R}
  - Boolean {true, false}
  - String {ASCII, Unicode}

- Cada tipo simples possui operações pré-definidas

Tipo simples	**Operações pré-definidas**
Integer	*, +, -, /, abs(), …
Real	*, +, -, /, floor(), …
Boolean	and, or, xor, not, implies
String	concat(), size(), substring(), …

Operações para Tipos Simples

- Sintaxe
  - v.operation(para1, para2, …)
    - Exemplo: “bla”.**concat**(“bla”)
  - Operações sem parênteses (notação fixa)
    - Exemplo: 1 **+** 2, true **and** false

Assinatura	**Operação**
Integer **X** *Integer* → *Integer*	{+, -, *}
t1 **X** *t2* → *Boolean*	{<,>,≤,≥}, *t1, t2* typeOf {Integer or Real}
Boolean **X** *Boolean* → *Boolean*	{and, or, xor, implies}

Operações para Tipos Simples

Operações Booleanas - semântica

- OCL é baseada em **lógica de três valores (trivalente)**
  - Expressões são mapeadas em três valores {true, false, undefined}
- Semântica das operações
  - $\mathcal{M}$(I, *exp*)= *I(exp*), Se exp ainda não resolvida
  - $\mathcal{M}$(I, **not** *exp*)= ¬$\mathcal{M}$ (I, *exp*)
  - $\mathcal{M}$(I,(*exp1* **and** *exp2*)) = $\mathcal{M}$(I, *exp1*) $\wedge$ $\mathcal{M}$(I, *exp2*)
  - $\mathcal{M}$(I,(*exp1* **or** *exp2*)) = $\mathcal{M}$(I, *exp1*) $\vee$ $\mathcal{M}$(I, *exp2*)
  - $\mathcal{M}$(I,(*exp1* **implies** *exp2*)) = $\mathcal{M}$(I, *exp1*) $\rightarrow$ $\mathcal{M}$(I, *exp2)*
- Tabela vardade: true(1), false (0),undefined (?)

Undefined: Valor de retorno no caso de falha na expressão

1. Acesso aos elementos de um vetor vazio
2. Erro durante *Type Casting*
3. …

¬	
0	1
1	0
?	?

$\wedge$	0	1	?
0	0	0	0
1	0	1	?
?	0	?	?

$\vee$	0	1	?
0	0	1	?
1	1	1	1
?	?	1	?

$\rightarrow$	0	1	?
0	1	1	1
1	0	1	?
?	?	1	?

Operações para Tipos Simples

Operações Booleanas - semântica

- Exemplo simples de uma expressão OCL **undefined**
  - *1/0*
- **Query** se undefined– OCLAny.oclIsUndefined()
  - *(1 / 0).oclIsUndefined() : true*
- Exemplos de expressões booleanas
  - *(1/0 = 0.0)* ***and*** *false : false*
  - *(1/0 = 0.0)* ***or*** *true : true*
  - *false* ***implies*** *(1.0 = 0.0) : true*
  - *(1/0 = 0.0) implies true : true*

Operações para coleções

- Coleção é um **supertipo abstrato** para todos tipos de conjuntos
  - Especificação de operações **mútuas**
  - *Vetor, Conjunto, Sequencia, ListaOrdenada* herdam essas operações
- **Atenção:** Operações com um valor de retorno pré-definidos criam uma nova coleção (sem efeitos colaterais)
- Sintaxe: v -> op(…) – Exemplo: {1, 2, 3} -> size()

- Operações de coleções (extração)
  - O recebimento de um objeto é denotado por *coll*

Operação	**Resultado**
size():Integer	Número de elementos no *coll*
includes(obj:OclAny):Boolean	True, se *obj* existe no *coll*
isEmpty:Boolean	True, se *coll* não contem elementos
sum:T	Soma de todos os elementos em *coll* Os elementos devem ser do tipo Integer ou Real

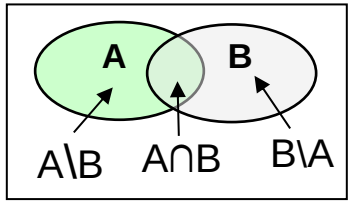
Operações para coleções

- **Operações em modelos vs. Operações OCL**

OCL-Constraint	*Semantic*
context Container inv: self.content -> first().isEmpty()	Operation *isEmpty()* sempre retorna true
context Container inv: self.content -> isEmpty()	Instâncias Container não devem conter elementos

Operações para Set/Bag

- *Set* e *Bag* definem operações adicionais
  - Geralmente baseadas em **conceitos de teoria dos conjuntos**
- **Operações de Set** (extract)
  - Recebendo objetos denotados por set



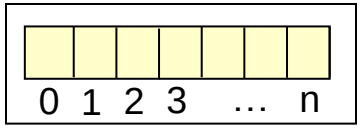
Operação	**Resultado**
union(set2:Set(T)):Set(T)	União de *set* e *set2*
intersection(set2:Set(T)):Set(T)	Interscção de *set* e *set2*
difference(set2:Set(T)):Set()	Diferença de set; elementos de *set que não estão em set2*
symmetricDifference(set2:Set(T)): Set(T)	Conjunto de todos os elementos que estão em set ou set2 mas não existem em ambos

- **Operações de Bag** (extract)
  - O objeto receptor é denotado por bag

Operação	**Resultado**
union(bag2:Bag(T)):Bag(T)	União de *bag* e *bag2*
intersection(bag2:Bag(T)): Bag(T)	Intersecção de *bag* e *bag2*

Operações para OrderedSet/Sequence

- *OrderedSet* e *Sequences* definem operações adicionais
  - Permite acesso e modificação através de um **Índice**



- **Operações de OrderedSet** (extract)
  - O objeto receptor é denotado *orderedSet*

Operação	**Resultado**
first:T	O primeiro elemento de *orderedSet*
last:T	O último elemento de *orderedSet*
at(i:Integer):T	O elemento de índice i de *orderedSet*
subOrderedSet(lower:Integer, upper:Integer):OrderedSet(T)	Um subconjunto de *orderedSet,* todos os elementos de *orderedSet* incluindo os elementos de uma posição abaixo e uma acima
insertAt(index:Integer,object:T) :OrderedSet(T)	O resultado é uma cópia de *orderedSet*, incluindo o elemento *object* na posição do *índice*

- **Operações de Sequence**
  - Análogas às operações de OrderedSet

Operações baseadas em Iteração

- OCL define operações em coleções baseadas em iteração
  - Expression Package: LoopExp
  - **Projeção** de novas Coleções além das existentes
  - Compacta **especificação de declaração** ao invés de algoritmos imperativos

- Operações Pré-definidas
  - select(exp) : *Collection*
  - reject(exp) : *Collection*
  - collect(exp) : *Collection*
  - forAll(exp) : *Boolean*
  - exists(exp) : *Boolean*
  - isUnique(exp) : *Boolean*

- iterate(…) – Iteração sobre todos os elementos de uma coleção
  - Operação genérica
  - Operações pré-definidas são executadas com o iterate(…)

Operações baseadas em Iteração

Operações Select-/Reject

- **Select** e **Reject** retornam subconjuntos de coleções
  - Iteração sobre toda a coleção e seus elementos
- Select
  - **Resultado**: Subconjunto da coleção, incluindo elementos onde *booleanExpr*

```
collection -> select( v : Type | booleanExp(v) )
collection -> select( v | booleanExp(v) )
collection -> select( booleanExp )
```

- Reject
  - **Resultado**: Subconjunto da coleção, incluindo elementos onde *booleanExpr* = **false**

```
collection-> reject(v : Type | booleanExp(v))
```

=

```
collection-> select(v : Type | not (booleanExp(v))
```


Operações baseadas em Iteração

Operações Select-/Reject

- Semântica de *Select-Operation*

OCL

```
context Company inv:
   self.employee -> select(e : Employee | e.age>50)  ->
notEmpty()
```

Java

```
List persons<Person> = new List();
for ( Iterator<Person> iter = comp.getEmployee();
iter.hasNext() ){
     Person p = iter.next();
     if ( p.age > 50 ){
          persons.add(p);
     }
}
```


Operações baseadas em Iteração

Operação de Collect

- *Operação Collect* retorna uma nova coleção a partir de uma existente. Copiando **Propriedades** de objetos e não os próprios objetos.
  - O objeto resultante sempre é **Bag<T>.T** define o tipo da propriedade a ser copiada

```
collection -> collect( v : Type | exp(v) )
collection -> collect( v | exp(v) )
collection -> collect( exp )
```

- Exemplo
  - *self.employees -> collect(age)* – Tipo de retorno: Bag(Integer)
- Notação curta
  - *self.employees.age*

Operações baseadas em Iteração

Operação Collect

- Semântica do operador *Collect*

OCL

```
context Company inv:
   self.employee -> collect(birthdate)  -> size() > 3
```

Java

```
List birthdate<Integer> = new List();
for ( Iterator<Person> iter = comp.getEmployee();
iter.hasNext() ){
     birthdate.add(iter.next().getBirthdate()); }
```

- O uso de *asSet()* elimina duplicações

OCL

```
context Company inv:
   self.employee -> collect(birthdate)  -> asSet()
```

Set (sem duplicações)

Bag (with duplicates)

Operações baseadas em Iteração

Operações ForAll-/Exists

- Checagem **ForAll**, se todos os elementos forem avaliados como true

```
collection -> forAll( v : Type | booleanExp(v) )
collection -> forAll( v | booleanExp(v) )
collection -> forAll( booleanExp )
```

- **Exemplo:** self.employees -> forAll(age > 18)

- **Nesting** de chamadas de forAll (*Produto Cartesiano*)

```
context Company inv:
self.employee->forAll (e1 | self.employee -> forAll (e2 |
        e1 <> e2 implies e1.svnr <> e2.svnr))
```

- ***Alternativa***: Uso de multiplos iteradores

```
context Company inv:
self.employee -> forAll (e1, e2 | e1 <> e2 implies e1.svnr <> e2.svnr))
```

- Checagem **Exists**, se pelo menos um elemento é avaliado como true
  - Beispiel: employees -> exists(e: Employee | e.isManager = true)

Operações baseadas em Iteração

Operador Iterate

- **Iterate** é a forma genérica de todas as operações baseadas em iteração

- **Sintaxe**

  collection -> iterate(**elem** : Typ; **acc** : Typ =
  <initExp> | **exp(elem**, **acc**))

  - Variável **elem** é do tipo ***Iterator***
  - Variável **acc** é do tipo ***Accumulator***
  - Tem um valor inicial definido por initExp
  - **exp(elem, acc)** é uma função para calcular **acc**

- **Exemplo**

  collection -> collect(x : T | x.property)

  -- semantimente equivalente a:

  collection -> iterate(x : T; acc : T2 = Bag{} | acc -> including(x.property))

Operações baseadas em Iteração

Operador Iterate

- Semântica para o operador *Iterate*

OCL

```
collection -> iterate(x : T; acc : T2 = value | acc -> u(acc, x)
```

Java

```
iterate (coll : T, acc : T2 = value){
      acc=value;
      for( Iterator<T> iter =
coll.getElements(); iter.hasNext(); ){
            T elem = iter.next();
            acc = u(elem, acc);
      }
}
```

- Exemplo
  - Set{1, 2, 3} -> iterate(i:Integer, a:Integer=0 | a+i)
  - Result: 6

Ferramentas de apoio

- **Lista de desejos**
  - Análise Sintática: Editor support
  - Validação de consistência lógica (Retirar Ambiguidade)
  - Validação dinâmica de invariantes
  - Validação dinâmica de Pre/Pos condições
  - Geração de código e automação de testes

- **Hoje**
  - Ferramentas UML oferecem editores OCL
  - Ferramentas MDA oferecem geração de código e expressões OCL
  - Meta modelagem de plataformas oferecem a oportunidade de definir Restrições OCL para meta modelos.
    - O editor deveria checar dinamicamente as restrições ou modelagem restrita, respectivamente.

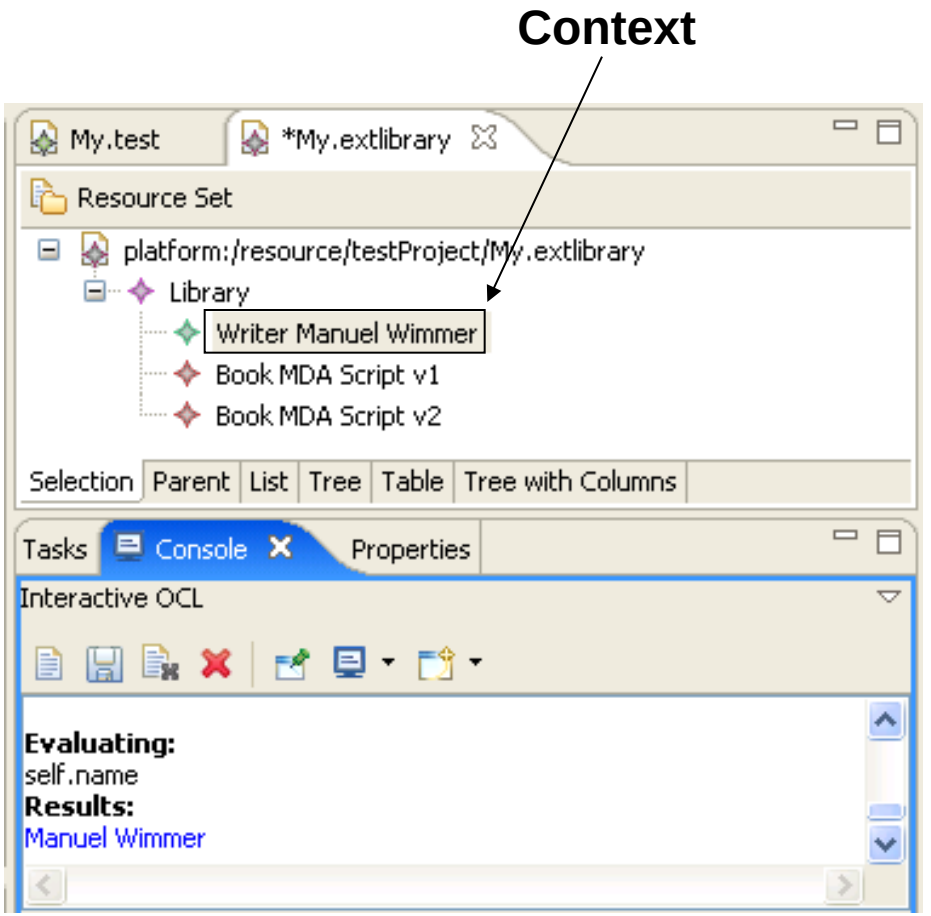
Ferramentas OCL

- Alguns parsers OCL que checam sintaxe e restrições e aplicam em modelos são gratuitos.
  - IBM Parser
- Dresden OCL Toolkit 2.0
  - Geração de código Java sem as restrições
  - Possível integração com ArgoUML
- OCL-frameworks foram originados em áreas EMF e o projeto UML2 do Eclipse
  - Octopus
  - Frauenhofer Toolkit
  - OSLO
  - EMFT OCL-Framework/Query-Framework

Ferramentas OCL

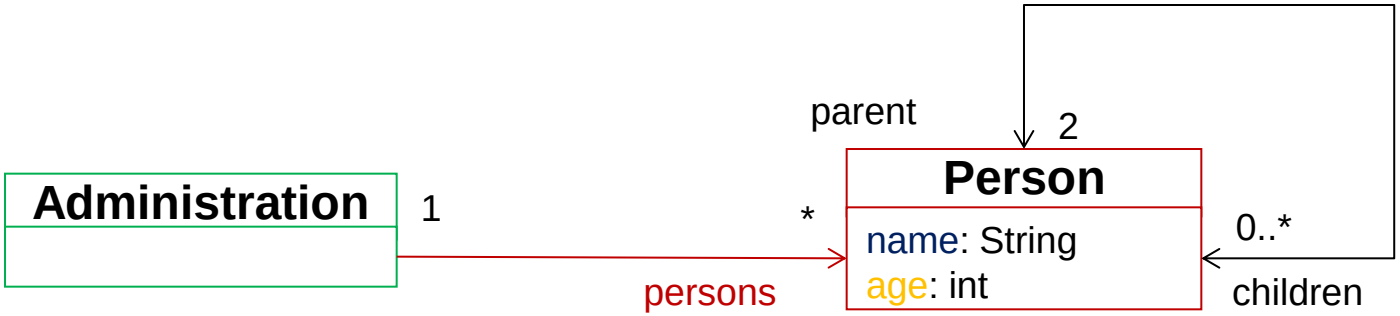
- **EMFT OCL-Framework**
  - Baseado em EMF
  - *OCL-API* – Habilita o uso de OCL em programas Java
  - *Console OCL Interativo* – Habilita a definição e avaliação de restrições OCL

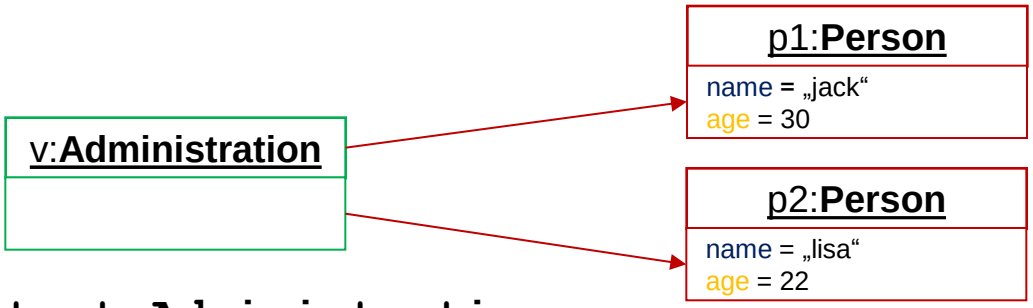
- **EMFT Query-Framework**
  - **Objetivo**: Consulta semelhante a SQL para modelos de informação
  - **select** exp **from** exp **where** *oclExp*



TUWEL: Interactive OCL Console Screencast

Exemplo 1: Navegação (1)

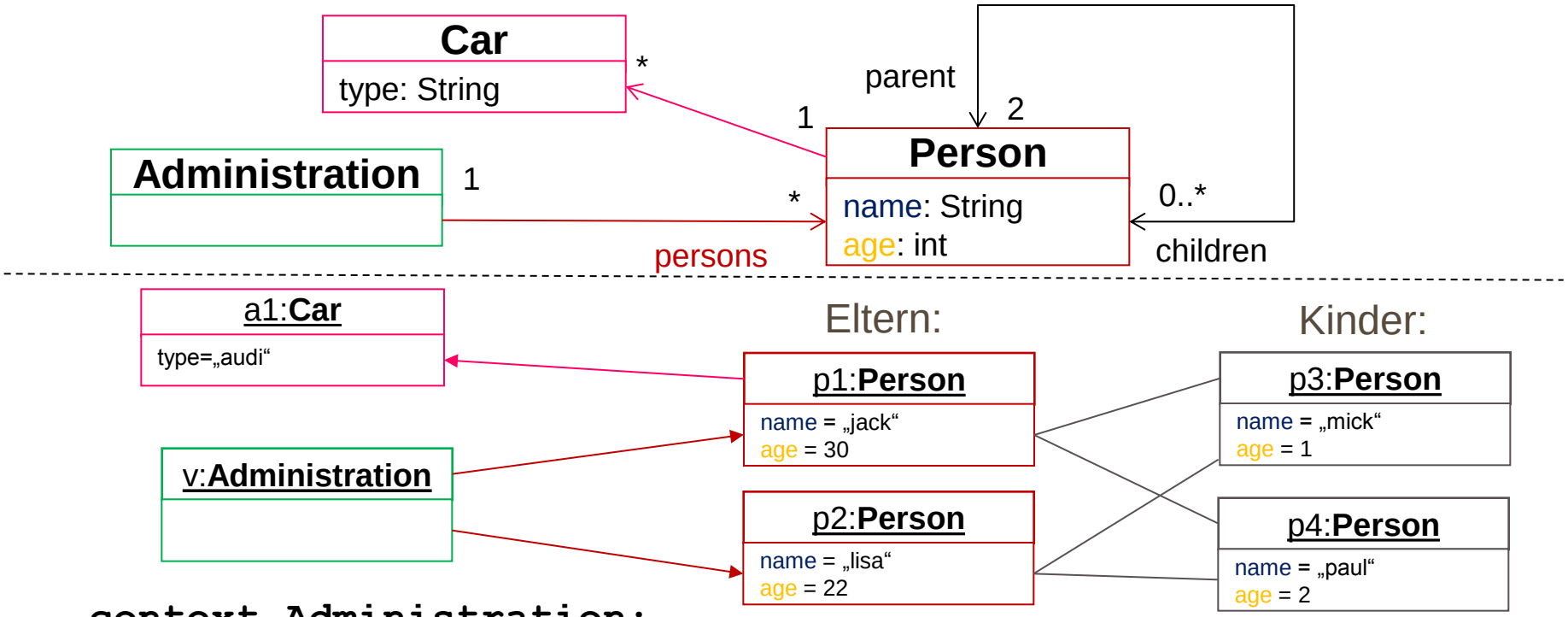





```
context Administration:
```

- `self.persons` → {Person p1, Person p2}
- `self.persons.name` → {jack, lisa}
- `self.persons.alter` → {30, 22}

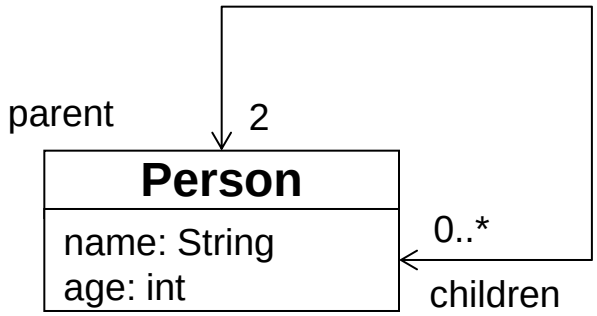
Exemplo 1: Navegação (2)



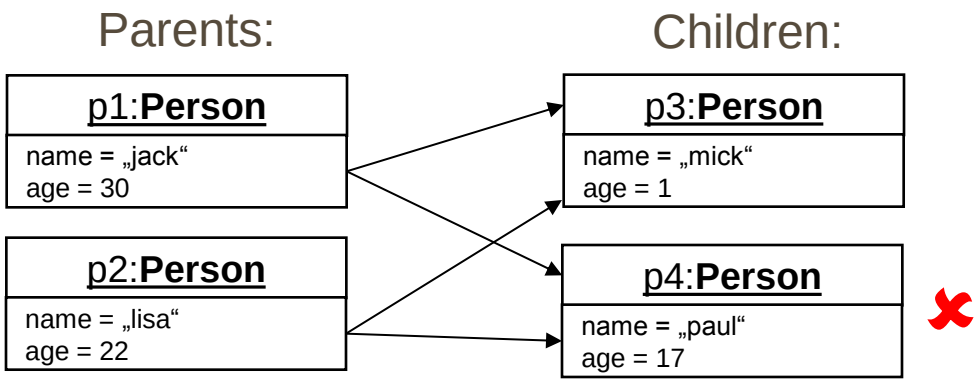

```
context Administration:
```

- `self.persons.children` → {{p3, p4}, {p3, p4}}
- `self.persons.children.parent` → {{{p1, p2}, {p1, p2}}, ...}
- `self.persons.car.type` → {{"audi"}}

Exemplo 2: Invariante (1)

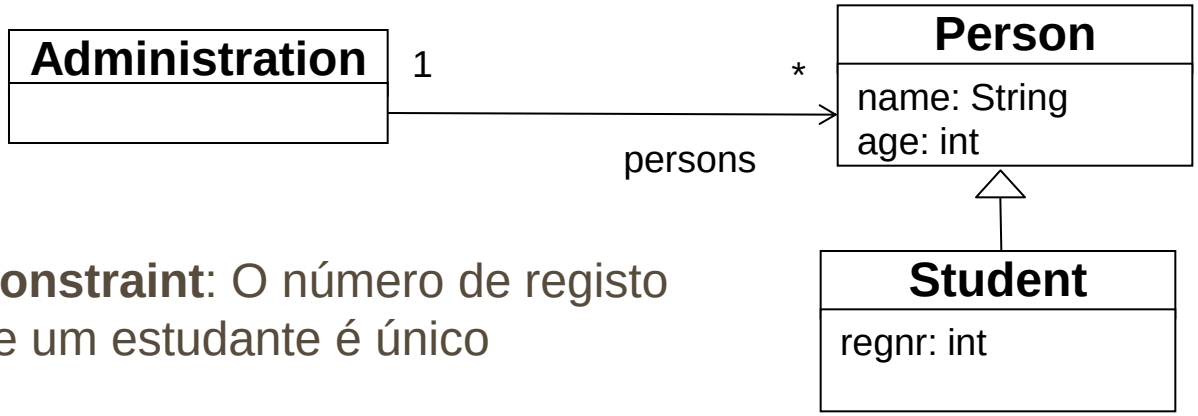


Constraint: Uma criança é pelo menos 15 anos mais nova que seus pais.




```
context Person
inv: self.children->forAll(k : Person | k.age
 < self.age-15)
```

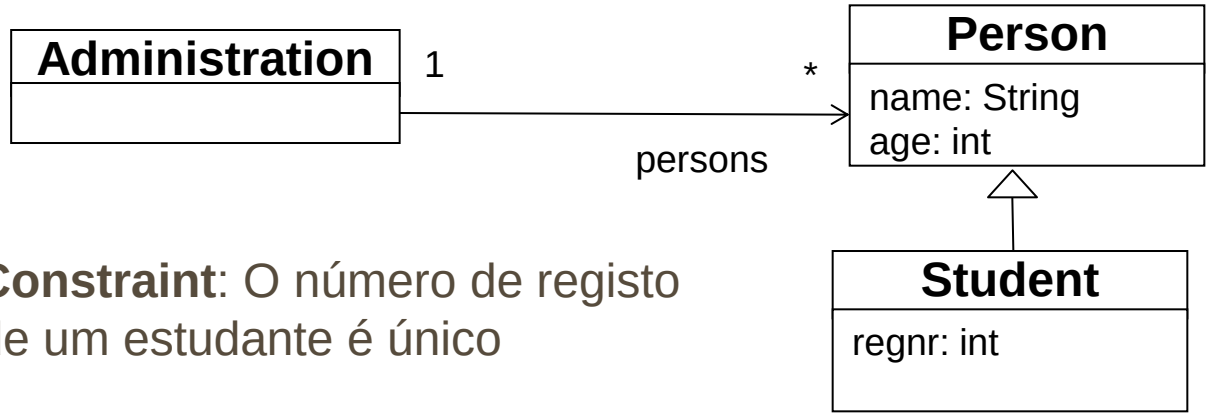

Exemplo 2: Invariante (2)



Constraint: O número de registo de um estudante é único

```
context Administration
inv uniqueRegnr :
  self.persons -> select(e : Person| e.oclIsTypeOf(Student))
                       -> forAll(e1 |
   self.persons -> select(e : Person | e.oclIsTypeOf(Student))
                       -> forAll(e2 |
  e1 <> e2 implies   e1.oclAsType(Student).regnr       <>
                           e2.oclAsType(Student).regnr))
```


Exemplo 2: Invariante (2) cont.

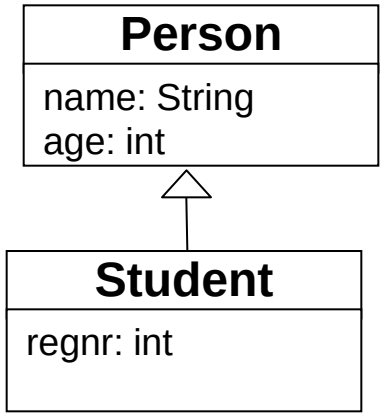


Constraint: O número de registo de um estudante é único

```
context Administration
inv uniqueRegnr :
  self.persons -> select(e : Person| e.oclIsTypeOf(Student))
        -> forAll(e1, e1 | e1 <> e2 implies
        e1.oclAsType(Student).regnr <>
        e2.oclAsType(Student).regnr)
        )
```

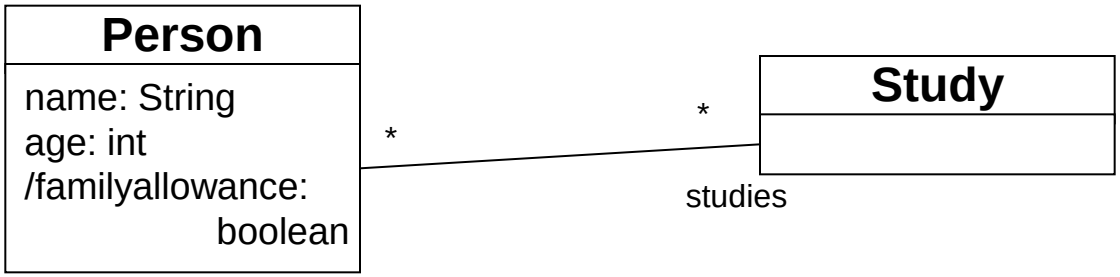

Exemplo 2: Invariante (2) cont.

Constraint: O número de registo de um estudante é único.




```
context Student
inv uniqueRegnr :
  Student.allInstances() -> forAll(e1, e1 | e1 <> e2 implies
        e1.oclAsType(Student).regnr <>
        e2.oclAsType(Student).regnr))
```

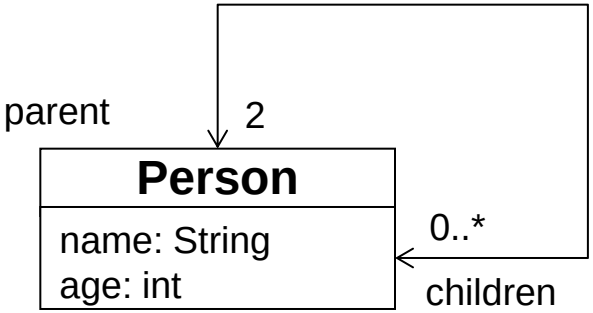

Exemplo 3: Atributos Herdados



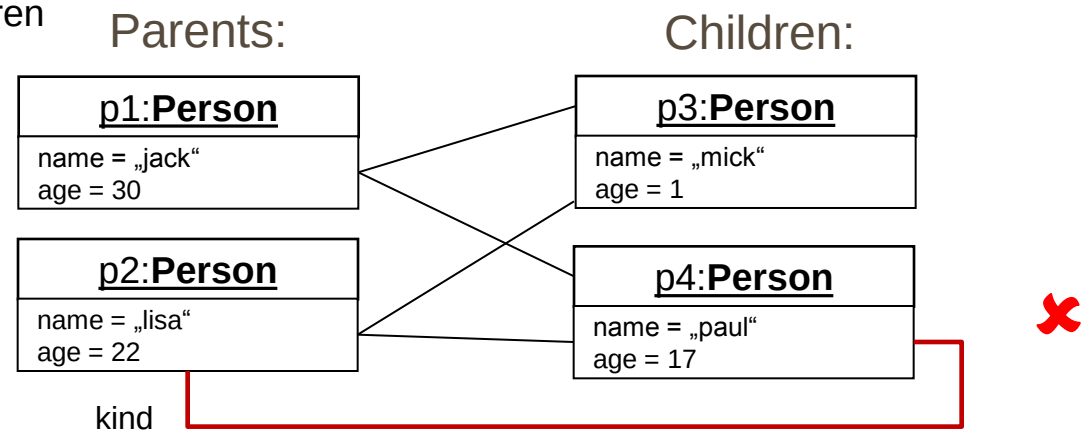
Uma pessoa é dependente da família se for menor de 18 anos, ou é estudante e possui até 27 anos.

```
context Person::familyallowance
derive: self.age < 18 or
        (self.age < 27 and self.studies -> size() > 0)
```


Exemplo 4: Definições



Constraint: Uma pessoa não pode estar relacionada a ela mesma



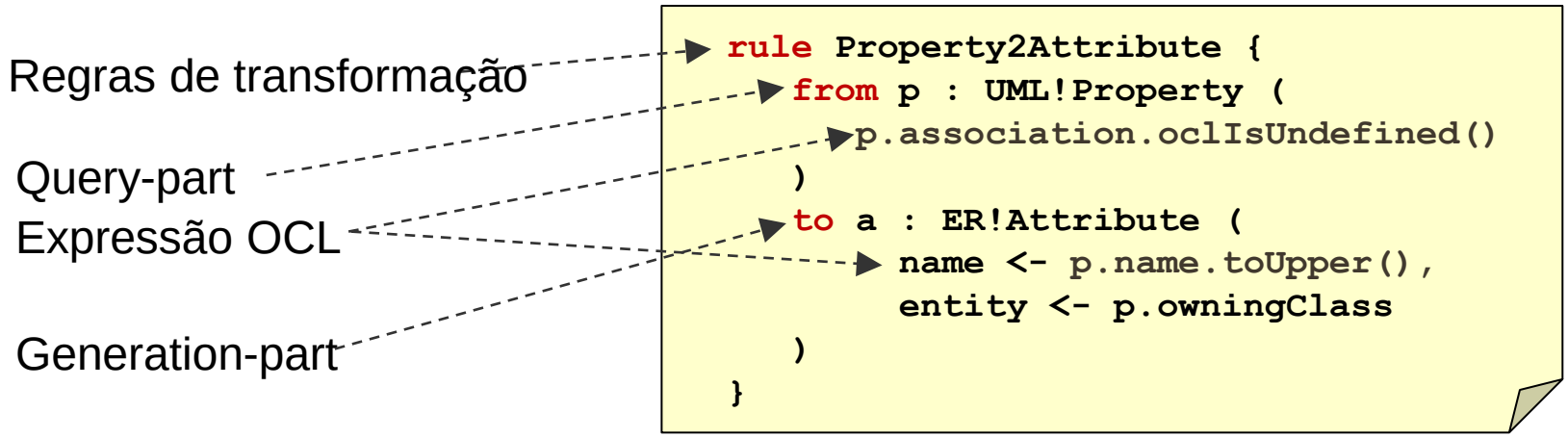

```
context Person
def: relative: Set(Person) = children-> union(relative)
inv: self.relative -> excludes(self)
```

Premissa: Semântica de ponto fixo, senão, necessário if then else

Outlook

Próxima unidade: ATLAS Transformation Language (ATL)

- **Query-part** (`from`) – O que deve ser transformado?
  - Quando **selecionar** a relevância dos elementos do modelo
    - Adicionalmente é necessário indicação de tipos, restrições nos atributos e extremidades de associação, que são especificados em OCL.
- **Generation-part**(`to`) – O que necessita ser criado?
  - Quando **criar** uma estrutura alvo
    - Adicionalmente é necessário informar o tipo, informações derivadas, que são calculadas em OCL.



Referências em OCL

- **Literatura**
  - Object Constraint Language Specification, Version 2.0
    - http://www.omg.org/technology/documents/formal/ocl.htm
  - Jos Warmer, Anneke Kleppe: The Object Constraint Language - Second Edition, Addison Wesley (2003)
  - Martin Hitz et al: UML@Work, d.punkt, 2. Auflage (2003)

- **Ferramentas**
  - OSLO - http://oslo-project.berlios.de
  - Octopus - http://octopus.sourceforge.net
  - Dresden OCL Toolkit - http://dresden-ocl.sourceforge.net
  - EMF OCL - http://www.eclipse.org/modeling/mdt/?project=ocl

MORGAN & CLAYPOOL PUBLISHERS

MODEL-DRIVEN SOFTWARE ENGINEERING IN PRACTICE

Marco Brambilla,
Jordi Cabot,
Manuel Wimmer.
Morgan & Claypool, USA, 2012.

www.mdse-book.com
www.morganclaypool.com
or buy it at: www.amazon.com