Specyfikacje formalne

Wojciech Łobacz

Na podstawie:
Formal specification, Andreas Roth, Peter H. Schmitt

Plan

- Dlaczego potrzebujemy specyfikacji formalnych?
- Idealna operacja - jaka?
- Bankomat – przykład idealny
- (Nie)konsekwencje wynikających ze specyfikacji
- OCL (notacja, metamodel)
- Przykładowe gramatyki dla wyrażeń OCL
- OCL, a logika pierwszego rzędu
- Zaawansowany OCL

- JML – już za tydzień!
- Porównanie z OCL – już za tydzień!

Specyfikacja (formalna)

1)Specyfikacja - zbiór wymagań
2)Specyfikacja formalna
3) matematyczny opis
4) opisuje „co”, nie „jak”
5) możliwa formalna weryfikacja
6) pomaga stwierdzić poprawność systemu
7) bardzo solidny fundament dla implementacji
8) można określić poprawność (zawsze względną)

Specyfikacje. Po co? Na co?

UML – to za mało.

Przydałyby się jakieś zastrzeżenia....

Czego potrzebujemy?

Możliwości opisania zmian w wyniku wykonania operacji

- warunki wstępne ⇒ warunki końcowe

Możliwości określenia/opisania stanu w dowolnym momencie wykonywania programu.

Języka zapisu specyfikacji.

Bankomat – Przykład: wprowadzPIN

Warunek wstępny:

karta włożona do bankomatu,

użytkownik nie autoryzował się

Warunek końcowy:

jeśli pin poprawny

autoryzacja zakończona sukcesem

else

jeśli liczba_prob < 2

liczba_prob = liczba_prob + 1,

autoryzacja zakończona porażką

else

karta zablokowana, autoryzacja zakończona porażką

Operacja spełniająca umowę

Spełnienie umowy operacji op z warunkami początkowymi i końcowymi zachodzi gdy:

jeśli operacja op jest wywoływana w dowolnym stanie spełniającym warunki początkowe, to po jej wykonaniu przejdzie do stanu spełniającego wszystkie warunki końcowe operacji

Czy to jest wystarczające?

- Nie wiadomo, co w przypadku gdy warunki wstępne nie są spełnione
- Zakładamy zakończenie wykonania operacji
- Zakładamy prawidłowe wykonanie operacji (nie np. poprzez rzucenie wyjątku)

NIE

Warunki wstępne, końcowe

Warunki wstępne	Warunki końcowe
włożona karta, prawidłowy PIN, użytkownik nieautoryzowany	autoryzacja użytkownika zakończona sukcesem
włożona karta, użytkownik nieautoryzowany, błędny PIN, próby < 2	autoryzacja użytkownika zakończona niepowodzeniem
włożona karta, użytkownik nieautoryzowany, błędny PIN, próby >=2	autoryzacja użytkownika zakończona niepowodzeniem, zablokowanie karty

Obserwacje

Warunki wstępne się wzajemnie wykluczają

Znajdźmy to, co się nie zmienia

PROBLEM RAMY

Nie, przeciwnie. Znajdźmy to, co się zmienia.

np. modyfikowane:

licznikProb, wszystkie zmienne potrzebne do sprawdzenia autentyczności danych oraz do zablokowania karty

kolejne problemy...

Problem ramy

Wnioskujemy o zmianach zachodzących w świecie, jak efektywnie reprezentować fakty, które się nie zmieniają?

Stan s0:

ma(s0, Jaś, 23 000)

ma(s0, Jaś, przyjaciółka)

Akcja: kupno samochodu s0 → s1

ma(s1, Jaś, 500)

ma(s1, Jaś, daewoo_tico)

Czy Jaś ma nadal przyjaciółkę?

Problem ramy

Reprezentacyjny problem ramy

Inferencyjny problem ramy

Problem kwalifikacji

Problem ramifikacji

Potrzebne dodatkowe informacje

Całkowita poprawność czy częściowa?

Wykonanie:

wymagane, niewymagane, normalne wykonanie

Domyślne wartości umowy

warunki wstępne	--	spełnione
warunki końcowe	--	spełnione
modyfikacje	--	wszystko
zakończenie op.	--	wymagane

Niezmienniki

Niezmiennik z przykładu:

licznikProb zawsze >= 0 oraz <= 2

nie ma 2 takich samych kart

context: CentralHost

validCardsCount – liczba poprawnych kart

Niezmienniki - ograniczeniem?

- Kiedy powinny być zachowywane?
  - Niezmienniki silne
  - Niezmienniki

- ATM.confiscateCard()
  - insertedCard = null; (...)
  - ! validCardsCount !

Potrzebujemy definicji

Klasa C spełnia niezmiennik n, jeśli:

1. Dla każdej operacji op i każdego stanu s spełniającego warunek wstępny op i n, niezmiennik n jest zachowany w każdym stanie zakończonym
2. n jest spełnione po wykonaniu każdego konstruktora

Konsekwencje

Niezmienniki są dziedziczone

Przedefiniowana operacja w podklasie już nie musi spełniać umowy

-- dalsze konsekwencje

Dygresja:

niezmienniki przy pętlach

OCL - wstęp

Object Constraint Language

Uzupełnienie UML, zgodność z UML

OCL – postrzegany jako język formalny

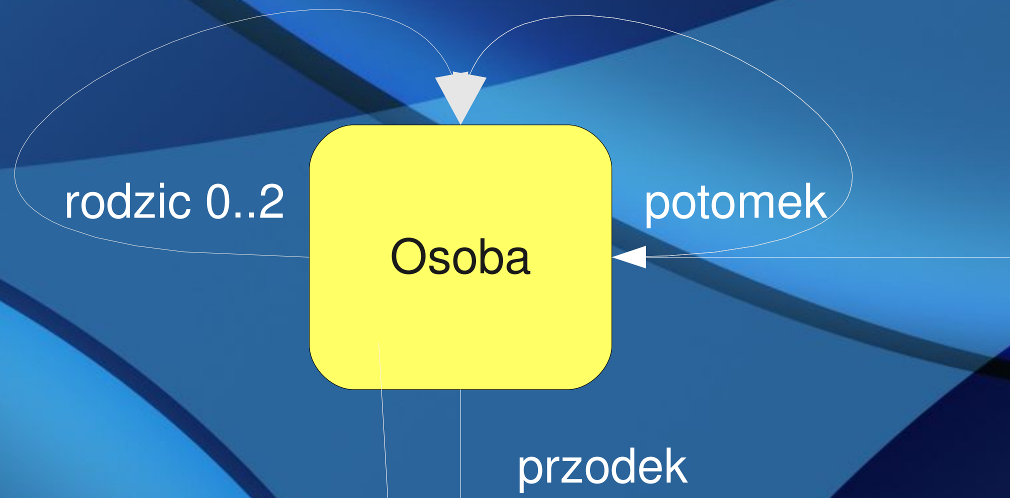
brak efektów ubocznych

OCL – nie został zaprojektowany dla matematyków

użytkownicy OCL ≈ użytkownicy UML

OCL niezmiennik

1. Niezmiennik zawsze wylicza się do wartości logicznej



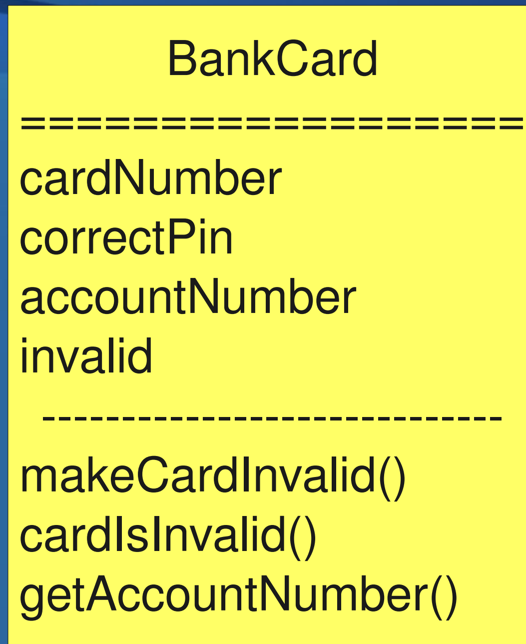
Do czego właściwie się odnosi?
context Person

Kiedy on ma zachodzić?
inv: {rodzic->excludes(self) and
potomek->excludes(self)}
inv = invariant = always

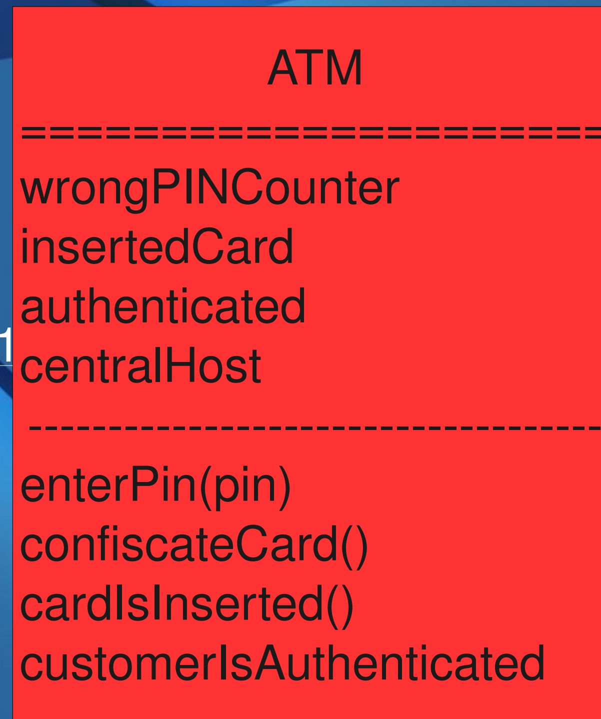
{rodzic->excludes(self) and
potomek->excludes(self)}

OCL - przykłady

Schemat

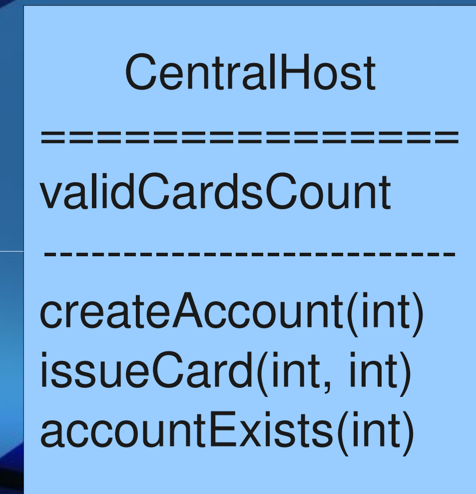


0..1 0..1



0

1 centralHost



OCL przykład

```
context ATM::enterPIN(pin: Integer)
modifies:     customerAuthenticated, wrongPINCounter,
              insertedCard, insertedCard.invalid
pre:          insertedCard <> null and not customerAuthenticated
post:   if pin = insertedCard@pre.correctPIN
        then customerAuthenticated
        else
              if wrongPINCounter@pre < 2
            then wrongPINCounter = wrongPINCounter@pre + 1
                  and not customerAuthenticated
              else
                  insertedCard = null
                  and insertedCard@pre.invalid
                  and not customerAuthenticated
          endif
      endif
```

OCL - notacja

context ATM::enterPIN(pin: Integer)

pre: self.insertedCard <> null **and**

not self.customerAuthenticated

context atm1:ATM::enterPIN(pin: Integer)

pre: atm1.insertedCard <> null **and**

not atm1.customerAuthenticated

OCL – różne notacje

@pre – przed wykonaniem operacj

-- – jakiś komentarz

context ATM

inv: **0 <=** wrongPINCounter **&&** wrongPINCounter <= 2

inv: 0 <= self.wrongPINCounter && self.wrongPINCounter <= 2

context BankCard

inv: BankCard::allInstances() ->

forall(p1,p2 | p1 <> p2 **implies**
p1.cardNumber <> p2.cardNumber)

OCL - zapytania

context CentralHost

inv: validCardsCount =

BankCard::**allInstances**() -> **select**(not invalid) -> size()

inv: invalidCardsCount =

BankCard::**allInstances**() -> **select**(c | c.invalid) -> size()

inv: invalidCardsCount =

BankCard::**allInstances**() -> **select**(c:BankCard | c.invalid) -> size()

OCL - notacja

kolekcja -> operacja (element:Typ | <wyrażenie>)

kolekcja -> operacja (element | <wyrażenie>)

kolekcja -> operacja (<wyrażenie>)

Odwołanie poprzez ciąg nawigacji

context BankCard inv:

bankCards.transactions.points -> sum() > 100

Suma punktów ze wszystkich transakcji dla wszystkich kart klienta przekracza 100.

Ograniczenia wyprowadzone dla atrybutów

context Account::points : Integer

derive: transactions.points -> sum()

Dla atrybutów klas ograniczenia mogą określać wartości wyprowadzane.

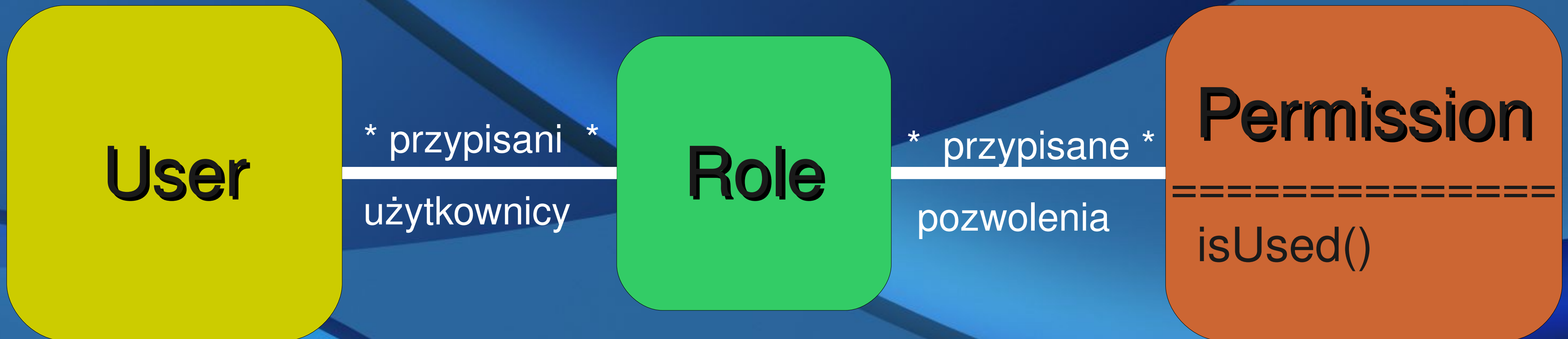
Ograniczenia początkowe dla atrybutów

context: Account::points : Integer

init: **if** client.age > 60 **then** 100 **else** 0 **endif**

Dla atrybutów klas ograniczenia mogą określać wartości początkowe atrybutu (init:).

OCL - relacje



context Permission::isUsed():Boolean

post: result = role.assigned_users -> notEmpty()

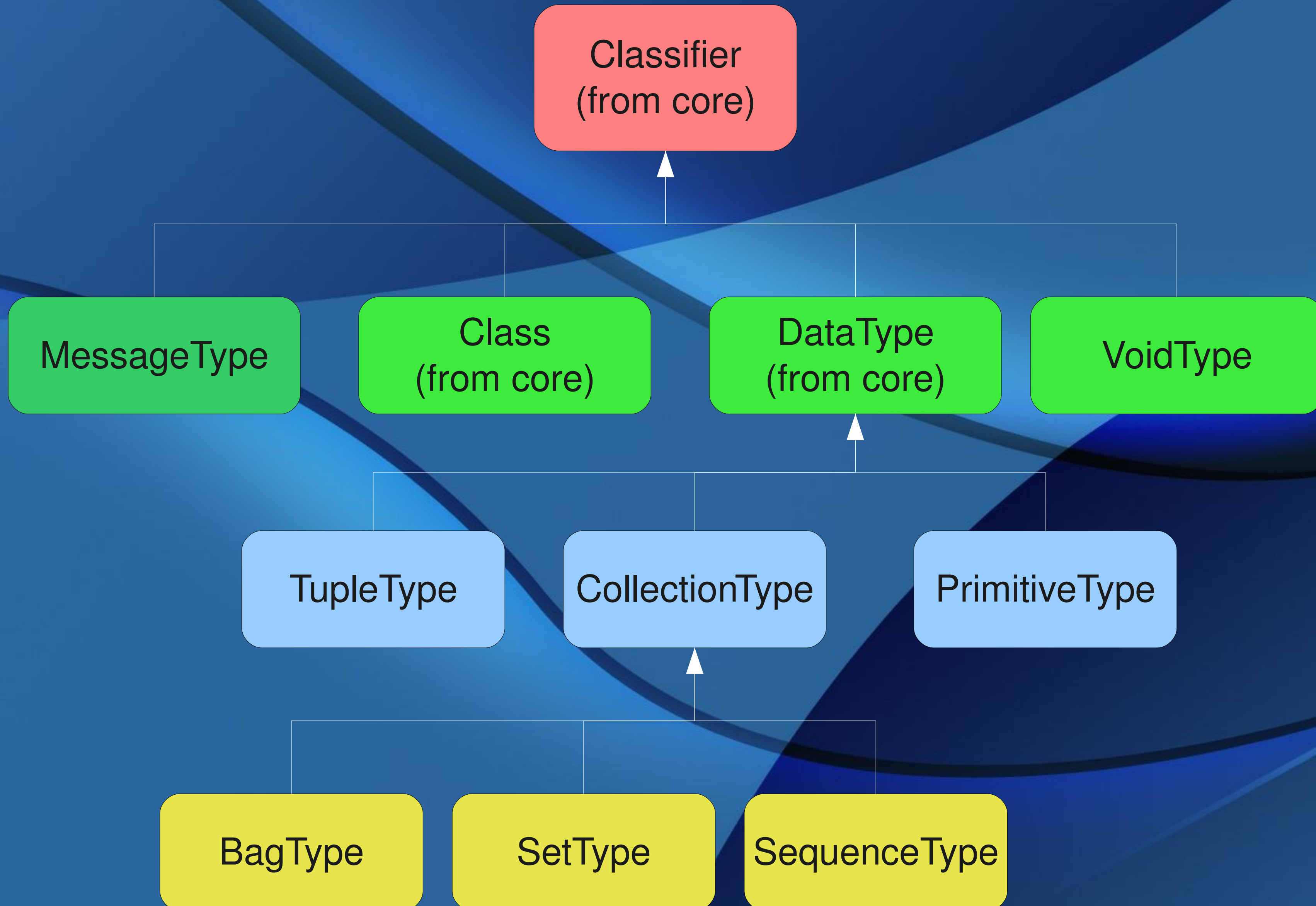
post: self.result = self.role.assigned_users -> notEmpty()

Skrót notacyjny

post: self.role = collect(r | r.assigned_users) → notEmpty()

post: self.role.assigned_users → notEmpty()

Hierarchia



Typy kolekcyjne

Kolekcja (Collection) - abstrakcyjny nadtyp

Zbiór (Set) - nie zawiera duplikatów elementów

Wielozbiór (Bag) - może zawierać duplikaty elementów, wynik kilku nawigacji

Uporządkowany zbiór (OrderedSet)

Ciąg (Sequence) - uporządkowany wielozbiór, elementy mają przypisane numery

Uporządkowany zbiór i ciąg mogą być rezultatem nawigacji po powiązaniu z własnością {ordered}.

Hierarchie

- PrimitiveType
  - OclBoolean
  - OclReal
  - OclInteger
  - OclString
  - AnyType

Relacja zgodny_z

Relacja zgodny_z jest najmniejszą relacją zwrotną i przechodnią na zbiorze wszystkich typów OCL spełniającą następujace warunki:

1. Integer jest zgodny z Real
2. C1 zgodny_z C2 dla wszystkich instancji klasy C1, o ile C1 jest podtypem C2 (diagram UML)
3. S(T) zgodny_z S(D), S to Bag, Set, Collection lub Sequence, o ile T zgodne_z D
4. T zgodne_z OclAny dla każdego typu T, o ile nie jest kolekcją lub krotką
5. OclVoid zgodny_z każdnym innym typem
6. Dla krotek: Krotka(n1:T1, ... nk:Tk) zgodne_z Krotka(n'1:S1...n'k:Sn), gdy {n1..nk) = {n'1,...,n'k} i dla elementów o tych samych nazwach mamy Ti zgodny_z Si

Przykład

Krotka(first: Integer, second: Integer, node: C)

Krotka(second: Integer, node: D, first: Real)

Kiedy są w relacji?

gdy C zgodne_z D

Operacje na typach kolekcyjnych

size() - liczba elementów w kolekcji

count(obiekt) - liczba wystąpień obiektu

includes(obiekt) - True, jeśli obiekt jest elementem

isEmpty() - True, jeśli brak elementów

sum() - suma wszystkich elementów (np. Real lub Integer)

exists(wyrażenie) - True, jeśli wyrażenie jest spełnione przynajmniej przez jeden element

forAll(wyrażenie) - True, jeśli wyrażenie jest spełnione przez wszystkie elementy

select(wyrażenie) - kolekcja elementów spełniających wyrażenie

union(kolekcja) - łączy zbiór ze zbiorem lub wielozbiorem, ciąg z ciągiem

Metamodel

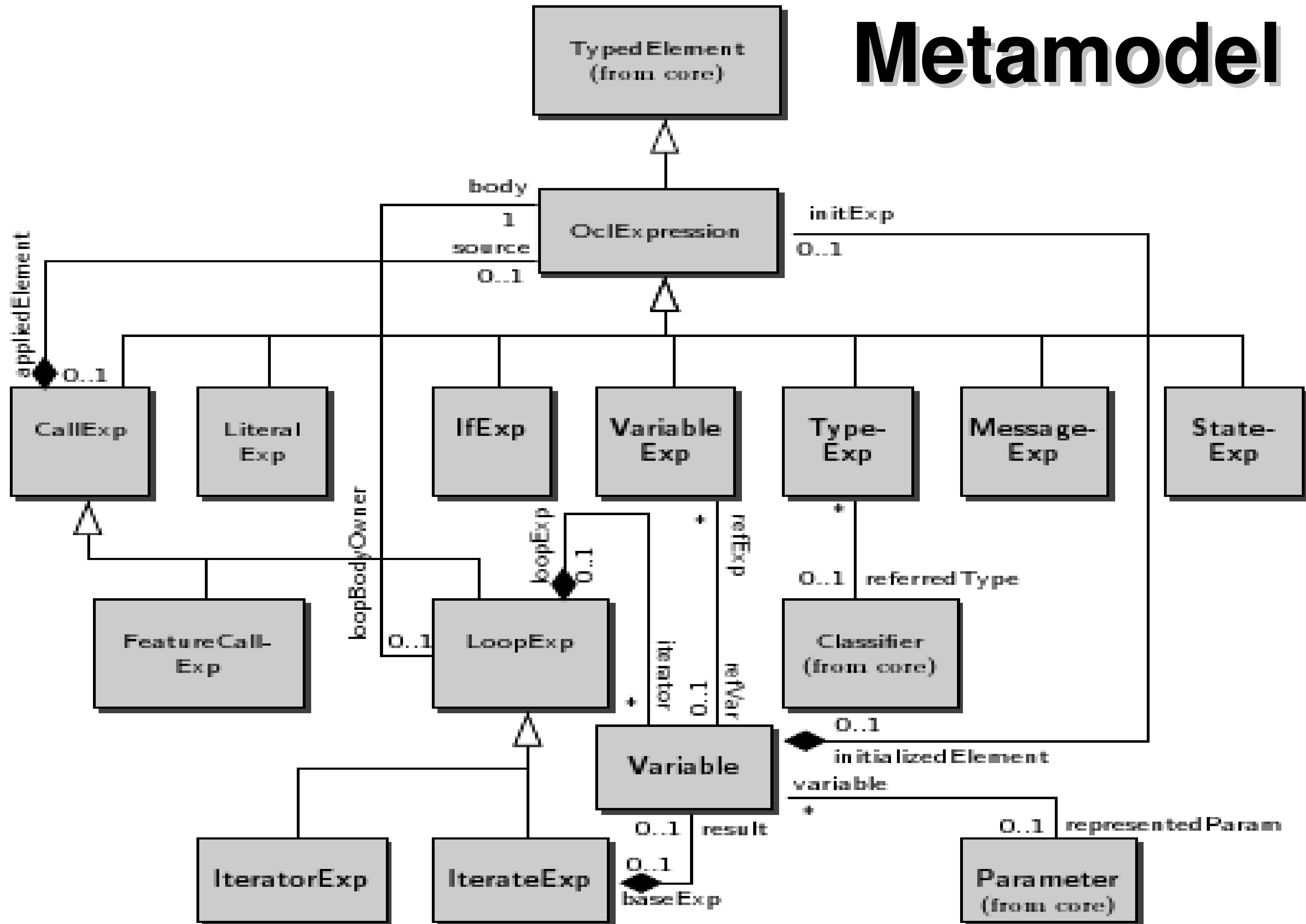
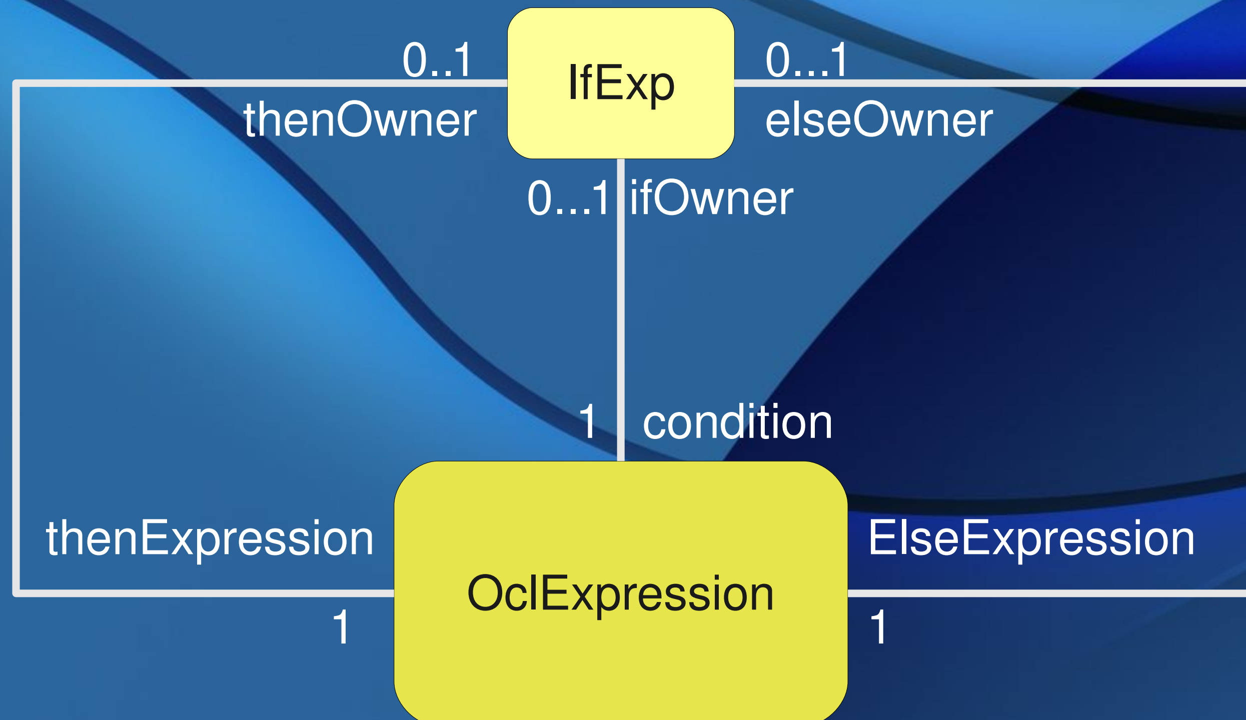


Fig. 5.6. Toplevel metaclass diagram for OCL expressions

OCL metamodel

OclExpressionCS ::= CallExpCs | VariableExpCS | LiteralExpCS | LetExpCS | MessageExpCS | IfExpCS



IfExp

context IfExp

inv: self.condition.type.name = ’Boolean’

inv: self.condition.type.oclIsKindOf(PrimitiveType)

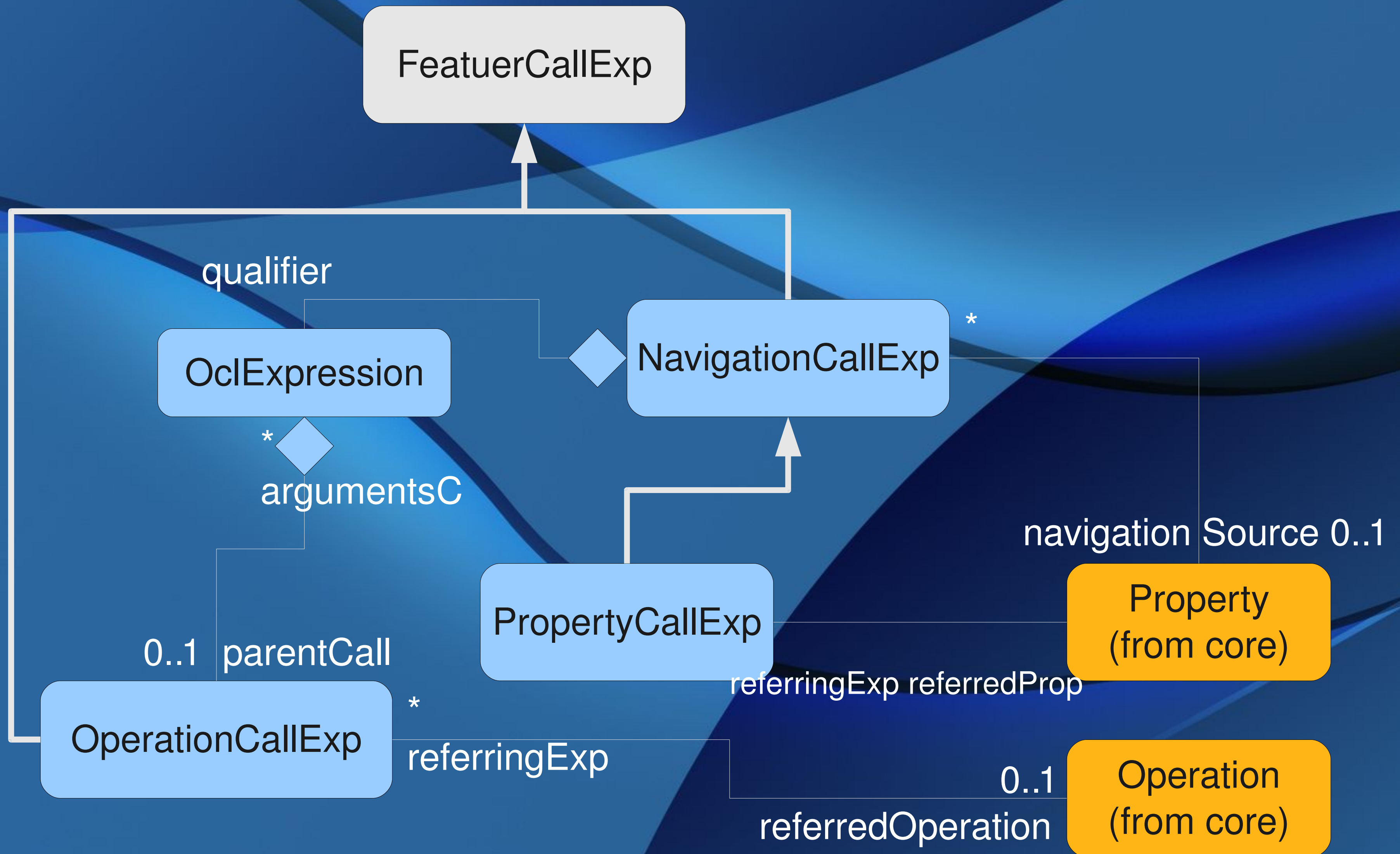
inv: self.type = thenExp.type.commonSuperType(elseExp.type)

if c:Boolean **then** t:Integer **else** s:Real **endif**

IfExpCS ::= 'if' OclExpCS 'then' OclExpCs 'else'

OclExpCs 'endif'

FeatureCall metamodel



Gramatyka 2

FeatureCallExpCs ::= NavigationCallExpCS |
PropertyCallCS | OperationCallExpCS

p1.cardNumber

insertedCard@pre

BankCard::allInstances()

p1.cardNumber <> p2.cardNumber

self.role.assigned_users

Trochę gramatyki

OperationCallExpCS ::=

(A) OclExpCS(1) sNameCS OclExpCS(2) |

(B) OclExpCS '->' sNameCS'(' argumentsCS?')' |

(C) OclExpCS'.'sNameCS ismarkedPreCS?'('argumentsCS?')' |

(D) sNameCS ismarkedPreCS?'('argumentsCS?')' |

(G) pathNameCS'('argumentsCS?')' |

(H) sNameCS OclExpCS

Trochę gramtyki - przykłady

(A) wrongPINCounter + 1

wrongPINCounter < 2

wrongPINCounter = wrongPINCounter + 1

insertedCard <> null

(B) self.role.assigned_users -> notEmpty()

s -> union(s2)

Trochę gramtyki - przykłady

(C) self.insertedCard.pinIsCorrect()

self.insertedCard.pinIsCorrect@pre()

(D) pinIsCorrect()

pinIsCorrect()@pre

(E) BankCard::allInstances()

(F) -wrongPINCounter

not cardIsInserted()

Gramatyki – krok dalej

pathNameCS   (className::opName())

(A)PropertyCallExpCS ::= OclExpressionCS'.' sNamCS isMakredPreCS?

(B) PropertyCallExpCS ::= pathNameCS (static)

Jeszcze więcej gramatyki

(A)NavigationCallExpCS ::= AssociationEndCallExpCS

(B)NavigationCallExpCS ::= AssociationClassCallExpCS

(C) AssociationEndCallExpCS ::= OclExpressionCS'.' sNameCS('['argumentCS']')? isMarkedPreCS?

Gramatyki iteratorów

(A) IteratorExpCS ::= OclExpressionCS '→' sNameCS('(VarDecl, (','VarDecl)? ' | ')? OclExpressionCS')'

(B) IteratorExpCS ::= OclExpressionCS'.'sNameCS'('argCS?')'

(C) ItaratorExpCS ::= OclExpressionCS'.'sNameCS

(D) IteratorExpCS ::= OclExpressionCS'.'sNameCS ('['argumentsCS']')?

Iteratory standardowe (A)

exists

forAll

isUnique

sortedBy

any

one

collect

select

reject

collectNested

Iteratory

(A1) source '→ ' 'select' '(' p ' | ' body ')'

(A2) source '→' 'select' '(' body ')'

(B) i (C) inaczej

Podstawowe klocki

context (x1,..xk)?classPath::op(p1:T1,...pn:Tn):T

pre (prename1)?: precondition1

post (postname1)?: postcondition1

...

pre (prenamek)?: preconditionk

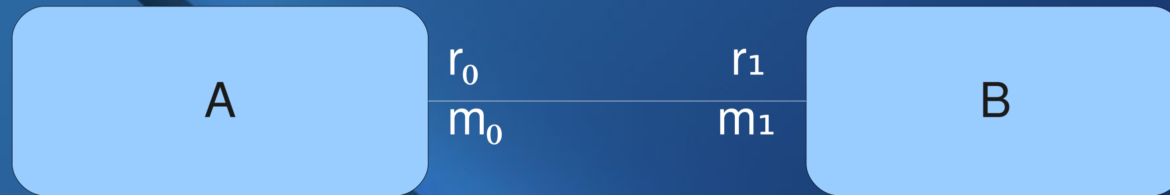
post (postnamek)?: postconditionk

OCL Semantyka

Logika pierwszego rzędu

logika dynamiczna

inne...



Sygnatura

1. Dla każdej relacji binarnej pomiędzy klasami A i B o nazwach ról r_0 i r_1 oraz multiplikatorami m_0 i m_1 R:
   1. r: $A \rightarrow B$, gdy m = 1
   2. r: $A \rightarrow Set(B)$, gdy $m \neq 1$
   3. r: $A \rightarrow Sequence(B)$, gdy $m \neq 1$ oraz przy B: << ordered>>
   4. r: $A \times B$, gdy m = m' = *
2. Symbole relacji złożonych

Sygnatura

3. Funkcje unarne dla atrybutów
4. Metody n-parametrowe
   - statyczne
   - lokalne
5. $\sum$ - wszystkie z biblioteki standardowej
6. Relacje proste w $\sum$

Operatory

OCL	Logic	OCL	Logc
not	!	x.intersection(y)	x ∩ y
and	&	x.union(y)	x ∪ y
or	|	x.includes(y)	x ∈ y
implies	->	x.excludes(y)	x∉ y
x.including(y)	x∪{y}	x.includesAll(y)	x ⊆y
x.excluding(y)	x\{y}	x.isEmpty()	x ≐ Ø
x.excludesAll(y)	x ∩ y	x.notEmpty()	x ≠ Ø
x.equals(y)	x≐ y	x <> y	x ≠ y

Zaczynajmy przekształcenia

```
insertedCard <> null and not customerAuthenticated
```

```
insertedCard (self) = null & !customerAuthenticated(self) .
```

KeY:

```
\forall ATM x;(x.<created> ->
  insertedCard(x) != null & customerAuthenticated(x))
```

Logika pierwszego rzędu, a OCL

OCL e0->forAll(x | e1)

FOL $\forall x.(x \in [e0] \to [e1])$

OCL e0->exists(x | e1)

FOL $\exists x.(x \in [e0] \,\&\, [e1])$

OCL e0->select(x | e1)

FOL se0,e1 (new symbol) with definition

$\forall x.(x \in se0,e1 <> (x \in [e0] \,\&\, [e1]))$

OCL e0->collect(x | e1)

ce0,e1 (new symbol) with definition

Logika pierwszego rzędu, a OCL

FOL

$\forall z.(z \in ce0,e1 <> \exists x.(x \in [e0] \ \& \ z = [e1]))$

OCL e0->isUnique(x | e1)

$\forall x.\forall y.(x \in [e0] \ \& \ y \in [e0] \ \& \ [e1] = \{x/y\}[e1] -> x = y)$

FOL

OCL e0->any(x | e1)

skx,e0,e1 (new symbol) with definition

FOL

$\exists x.(x \in [e0] \ \& \ [e1]) - skx,e0,e1 \in [e0] \ \& \ \{x/skx,e0,e1\}[e1]$

Semantyka iteratorów

union(s:Set(T)):Set(T)

post: result->forAll(elem |

self->includes(elem) or s->includes(elem))

post: self ->forAll(elem | result->includes(elem))

post: s ->forAll(elem | result->includes(elem))

ewentualnie:

self -> union(s : Set(T)) : Set(T) =

self -> iterate(x ; u:Set(T) = s | u->including(x))

Select

```
source -> select(iterator | body)
post: result -> forAll(e | source.includes(e))
post: result -> forAll(e | body)
post: source -> forAll(e | body implies
   result.includes(e))
```

Iterator

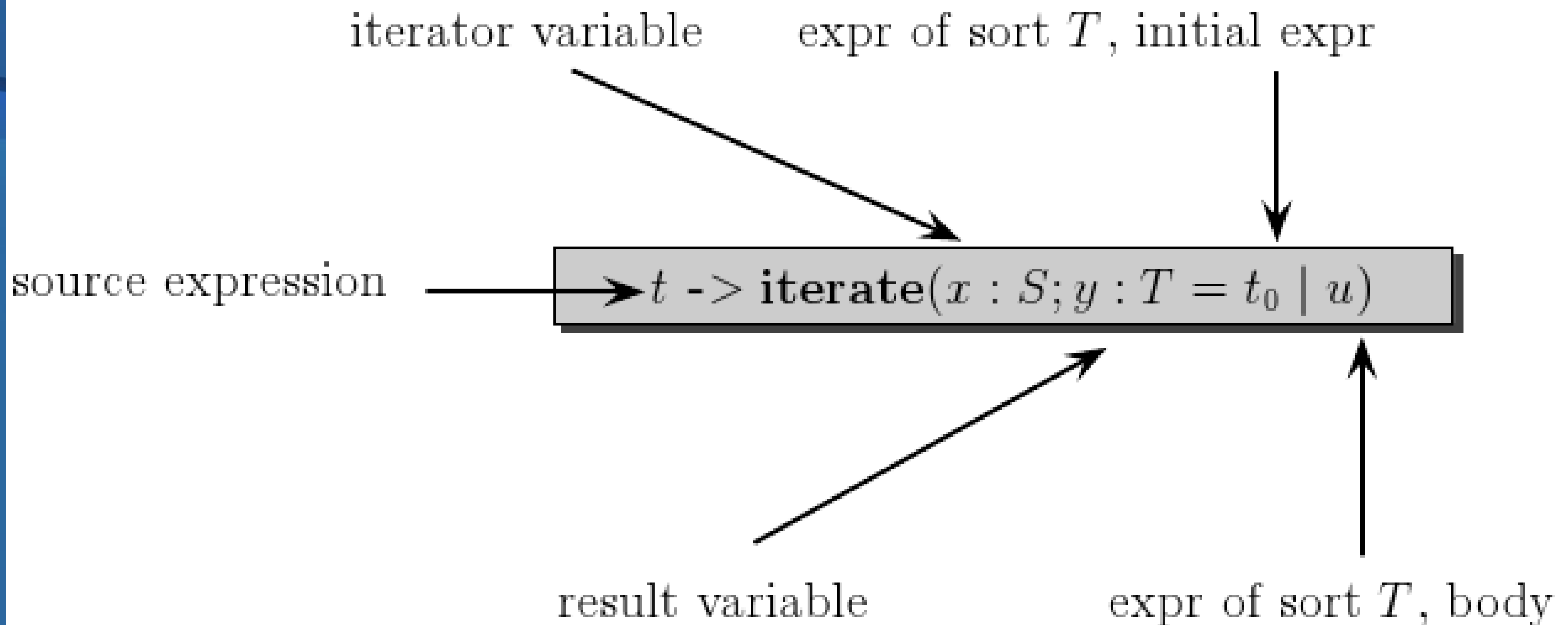


Fig. 5.11. Syntax of the `iterate` construct

Itarator - wymagania

1. Zmienna x różna od y
2. Zmienna y nie występuj w termie t
3. Zmienne x i y nie występują w t0
4. Typy y i u są zgodne
5. Typ t jest typu Kolekcja(S), gdzie S to typ t

Przykłady

```
t->forAll(x | a)          =    t -> iterate(x;y = true |  y and a )
t->exists(x | a)          =    t -> iterate(x;y = false |  or a )
t-> collectNested(x | u)  =    t -> iterate(x;y = Bag{} |  y ->including(u))
t->collect(x | u)         =    t->collectNested(x | u) -> flatten()
t->select(x | a)          =    t-> iterate(x;y = Collection{} |
                                  if a then y.including(x) else y)
t->any(x | e)             =    t->select(x | e)-> asSequence()-> first()
t-> flatten() a           = if t.type.elementType.oclIsKindOf(CollectionType)
                             then   t -> iterate(c;acc:Bag = Bag{} |
                                  acc -> union(c->asBag))
                             else t
                              endif
```

Rozmaitości

Problem z nullem

null.a = null.b

Wyjątki

Plusy OCL

- Język obsługuje pełny model obiektowy UML
- Jest ortogonalny (brak dużych zlepków takich jak select…from…where…group by…having...order by…)
- Jest dość popularny w środowisku UML
- Łatwa intuicyjna semantyka

Minusy OCL

- Fatalna składnia, przerosty składniowe, nieczytelne wyrażenia
- Brak uniwersalności (np. żadnych możliwości rekurencyjnych)
- Niedospecyfikowanie (np.. brak semantyki dla zakresów nazw)
- Redundantny i trochę przypadkowy wybór funkcjonalności
- Brak myślenia o zanurzeniu OCL w uniwersalny język programowania
- Brak perspektyw baz danych i innych abstrakcji programistycznyc
- Brak efektów ubocznych
- Niejasny stosunek do wartości zerowych (podważa przykrycie SQL, gdzie wartości zerowe są ważną funkcjonalnością)
- Niespójność i przecinanie się funkcjonalności OCL z funkcjonalnością UML 2.1
- Nieprzystosowanie do roli języka zapytań (brak optymalizacji)

Bibliografia

„Formal specification” Andreas Roth, Peter H. Schmitt

Źródła ilustracji

„Formal specification” **Andreas Roth, Peter H. Schmitt**

Dziękuję za uwagę