

Создание интерпретатора OCL для глубокого метамоделирования в REAL.NET

Шигаров Никита Алексеевич

научный руководитель: к.т.н., доц. Литвинов Ю. В.

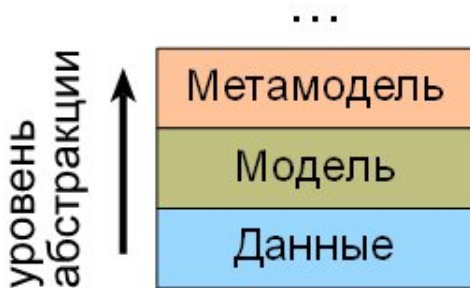
рецензент: инженер-программист АО «ПФ «СКБ Контур»

Перешеина А. О.

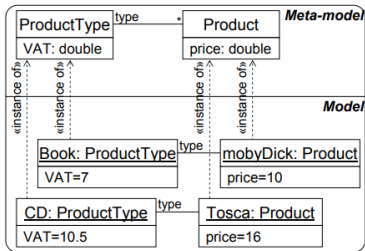
СПбГУ

12 июня 2019 г.

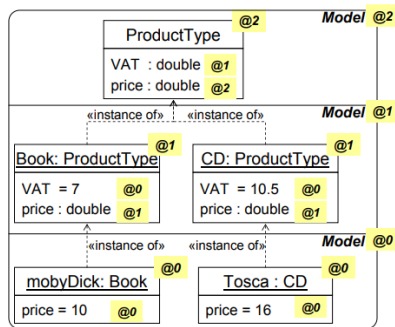
- данные — это множество простейших единиц информации, которые касаются не абстрактных, а конкретных сущностей (например, значение определённого свойства определённого элемента)
- модели — это описание структуры данных (например, информация о том какие есть свойства и связи между объектами)
- метамодели — это средства построения моделей (например, формальные языки или графические нотации для описания структуры классов, свойств и связей)



Типы метамоделирования



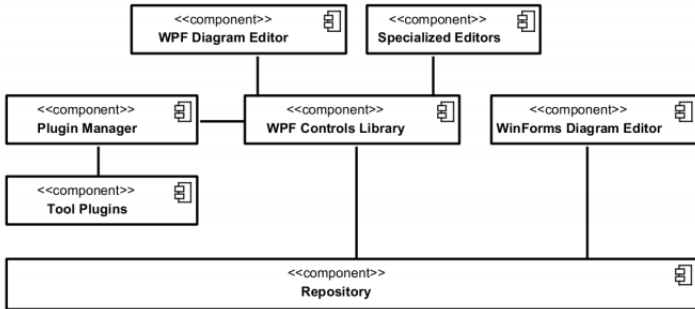
Объект-тип



Глубокое
метамоделирование

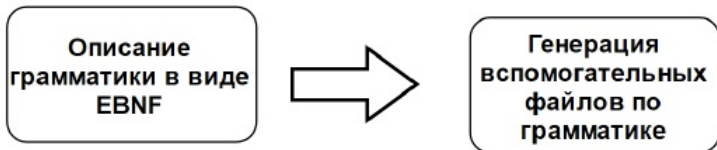
- Object Constraint Language (часть стандарта UML)
- декларативный язык описания ограничений
- **context** Person **inv**: self.age >= 0
- 2 **context** Person **inv**: self.parents->forall(p | p.age > self.age)
- нет современных open source реализаций OCL-интерпретаторов на .NET

- репозиторий для хранения моделей и метамodelей
- возможность написания плагинов
- редактор для визуализирования метаслоев



- определить требования к интерпретатору
- выбрать подмножество языка OCL и модифицировать его в соответствии с требованиями
- реализовать интерпретатор OCL
- выполнить интеграцию с REAL.NET
- выполнить апробацию

- ANTLR 4 – генерация лексического и синтаксического анализатора по грамматике (на языки C#, Java, C++ и др.)
- модифицирована грамматика OCL для поддержки нескольких метаслов



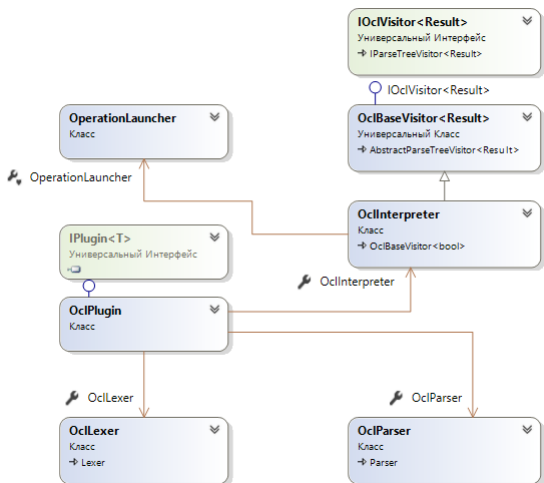
- использован паттерн Visitor



- возможность задавать ограничения на элементы более глубоких метамоделей, соответствующим данному
- возможность обращаться к полям из более глубоких метамоделей
- возможность комбинировать обращения к полям из разных метамоделей

- помечать каждое ограничение номером самой глубокой метамодели относительно данного элемента через символ @
- помечать обращения к полям из более глубоких метамodelей номерами через символ @
- ```
package P
context Function
inv@2:
 self.callParams@2.size() == self.params@1.size()
endpackage
```

# Диаграмма классов решения

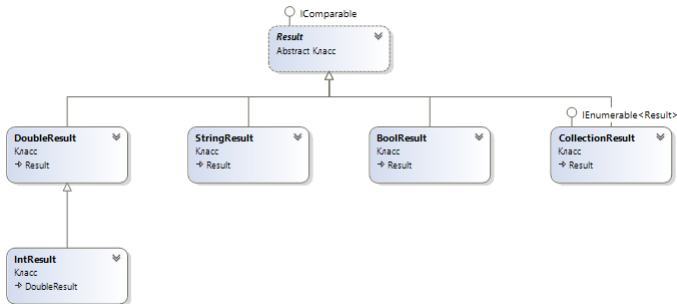


- операции работы с числами (max, abs, round...)
- операции работы со строками (toReal, substring, concat...)
- операции работы с коллекциями (select, iterate, any...)
- возможность определять свои рекурсивные функции

- определение и вызов рекурсивных функций

```
package P
context A
inv@0:
 let fact(n: var) =
 if n = 1 then n else n * fact(n - 1) endif
 let n = 5
 fact(n+1) = 720
endpackage
```

- ДИНАМИЧЕСКАЯ СИСТЕМА ТИПОВ
- `Sequence{Bag{1}, Set{'a'}, Set{1.0}}->first()->asSequence()->first()+1`



# Окно редактора

The screenshot shows the REAL.NET editor interface. The main workspace displays a state transition diagram with three nodes: aFinalNode (top, blue square with a red button), aTimer (middle, orange square with a clock face), and aMotorsForward (bottom, purple square with a robot and a green arrow). Blue arrows indicate transitions: one from aFinalNode to aTimer, and another from aMotorsForward to aTimer.

The right-hand panel shows the 'RobotsTestModel' dropdown menu. Below it is a 'Link' section with a list of elements: Timer, MotorsStop, MotorsBackward, MotorsForward, and FinalNode. The 'Elements:' section lists: aTimer, aMotorsForward, and aFinalNode. Below that, the 'Selected entity:' section shows a table of properties for the selected entity.

| Name             | Type    | Value                        |
|------------------|---------|------------------------------|
| delay            | Int     | 3000                         |
| isValid          | Boolean | true                         |
| instanceMetatype | String  | Metatype.Node                |
| isAbstract       | Boolean | false                        |
| shape            | String  | View/Pictures/timerBlock.png |

Below the table is a 'Constraints' section.

The bottom-left code editor shows the following OCL code:

```
package RobotsTestModel
context aTimer
inv@0:
self.delay = 3001
endpackage
```

An 'Execute' button is located to the right of the code editor.

At the bottom of the window, there are tabs for 'Messages', 'Errors', and 'Ocl'.



- Node -> Function -> Rotate -> FunctionCall

```
package Model
context Function
inv@2:
 self.callParams@2.size() == self.params@1.size()
endpackage
```

- Node -> Function -> Rotate -> Type -> FunctionDefine

```
package Model
context Function
inv@2:
 self.allInstances@2->select(x.type == "Call")->
 forAll(true implies self.allInstances@2->
 select(y.type == "Def")->exists(z.name == x.name))
endpackage
```

- тест на 5 пользователях
- ограничение из последнего примера
- System Usability Scale
- 90, 70, 57.5, 77.5 и 85 баллов

- модифицирована грамматика OCL для поддержки нескольких метаслоев в соответствии с составленными требованиями
- реализован интерпретатор модифицированного языка OCL на языке C#
- выполнена интеграция с REAL.NET с помощью плагина и создания редактора
- для проверки корректности проведены опросы пользователей с оценкой по метрике System Usability Scale