

Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных
систем

Системное программирование

Алымова Дарья Андреевна

Визуальный язык задания ограничений на модели в REAL.NET

Бакалаврская работа

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

Рецензент:
Инженер-программист АО «ПФ «СКБ Контур»
Першеина А. О.

Санкт-Петербург
2019

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems
Software Engineering

Daria Alymova

Visual constraints language for models in
REAL.NET

Graduation Thesis

Scientific supervisor:
C.Sc. Docent Yurii Litvinov

Reviewer:
Developer, SKB Kontur, Anna Peresheina

Saint-Petersburg
2019

Введение.....	4
1. Постановка задачи	6
2. Обзор литературы	7
2.1 Существующие решения.....	7
2.1.1 OCL	7
2.1.2 VOCL	8
2.1.3 DPF	9
2.1.4 Визуальный язык задания ограничений на модели в QReal11	
2.1.5 Прототип системы ограничений в REAL.NET	12
2.2 Инструменты.....	12
2.2.1 REAL.NET	12
2.2.2 WPF	14
2.2.3. Язык программирования роботов	14
3. Требования к языку.....	16
4. Описание языка ограничений	17
4.1 Синтаксис	17
4.2 Семантика	19
5. Прототип модуля проверки ограничений.....	20
5.1 Интеграция.....	20
5.2 Архитектура классов.....	20
6. Тестирование и апробация.....	22
6.1 Тестирование	22
6.2 Апробация	23
7. Заключение	26
Список литературы	27

Введение

Хотя компьютеры уже очень давно вошли в нашу жизнь, до сих пор находятся люди, не обладающие даже базовой технической грамотностью. Для них представляет сложность любой контакт с компьютером, не говоря уже о владении навыком программирования. Между тем информационные технологии развиваются и умение обращаться с техникой становится необходимостью. Сегодня навык программирования полезен не только в работе с компьютером, но и при любом взаимодействии с окружающей действительностью, на бытовом уровне. Современный человек сталкивается с программированием чаще, чем он мог бы подумать. Запрограммировать робота или дрона, решить логическую задачу в игре можно с помощью визуальных языков программирования. В отличие от текстовых языков, они могут быть понятны даже ребёнку благодаря интуитивному интерфейсу [1]. Низкий входной порог требуемых навыков пользователя делает это направление привлекательным для изучения. Визуальное программирование могло бы сделать гибче использование повседневных предметов – например, позволить пользователю самому настроить духовой шкаф или мультиварку, не прибегая к использованию стандартных, не всегда подходящих, режимов.

В области визуального программирования важное место занимают системы метамоделирования, основной функциональностью которых считается упрощение процессов создания систем моделирования. Модель – это некий язык, тогда как метамодель – это язык, описывающий модель. Как правило, метамодели являются визуальными, реже – текстовыми. Визуальное метамоделирование удобно, например, для создания предметно-ориентированных языков.

Неотъемлемой частью языка программирования являются ограничения. Например, система типов является набором семантических правил, которое не следует из грамматики, но тем не менее важно. Визуальные языки, как и прочие языки программирования, не являются исключением; для них тоже необходимы ограничения. Они дают автору языка возможность описать более узкое множество допустимых моделей, а пользователям – создавать более корректные программы. Многие современные UML-системы, например, предоставляют пользователям редактор с элементами языка, но не проверяют выполнение ограничений, из-за чего не всякая созданная в таком редакторе UML-диаграмма может считаться корректной. Например, можно создать два класса, которые наследуются друг от друга. Визуальные языки предназначены для людей, которые могут являться экспертами в предметной области. Как правило, такие пользователи не обладают навыками программирования и ресурсами для ознакомления с ними. Это могут быть как специалисты других, не смежных с IT областей, так, например, и дети. Поэтому корректная работа системы ограничений, которая может направить их в сторону написания

правильной программы без четкого понимания пользователями концепций программирования, критически важна. Система проверки ограничений на визуальный язык для программирования роботов, которому учат школьников, например, помогла бы избежать ошибок и даже реализовать функцию подсказки или автодополнения.

Существует множество различных систем задания ограничений, часть из которых будет рассмотрена в рамках данной работы. На кафедре системного программирования СПбГУ разрабатывается система REAL.NET [2], для которой будет создаваться визуальный язык ограничений.

1. Постановка задачи

Целью данной работы является создание визуального языка для задания ограничений на произвольной модели в REAL.NET. Для достижения цели были выделены следующие основные подзадачи:

1. сформулировать требования к разрабатываемому языку ограничений;
2. рассмотреть существующие подходы к созданию языка или системы ограничений и выбрать один из них, или предложить своё решение;
3. создать визуальный язык ограничений, соответствующий указанным требованиям;
4. разработать прототип модуля проверки ограничений, заданных на созданном визуальном языке в системе REAL.NET;
5. провести тестирование модуля на различных видах ограничений и его апробацию.

2. Обзор литературы

2.1 Существующие решения

2.1.1 OCL

Одним из самых известных языков задания ограничений является Object Constraints Language [4]. Он входит в стандарт UML и позволяет задавать более точные диаграммы, описывая инварианты.

Ниже приведен список элементарных ключевых слов OCL.

- Ключевое слово `context` уточняет элемент, на который накладывается ограничение.
- Ключевое слово `self` обозначает рассматриваемый объект.
- Ключевое слово `result` обозначает возвращаемое значение.

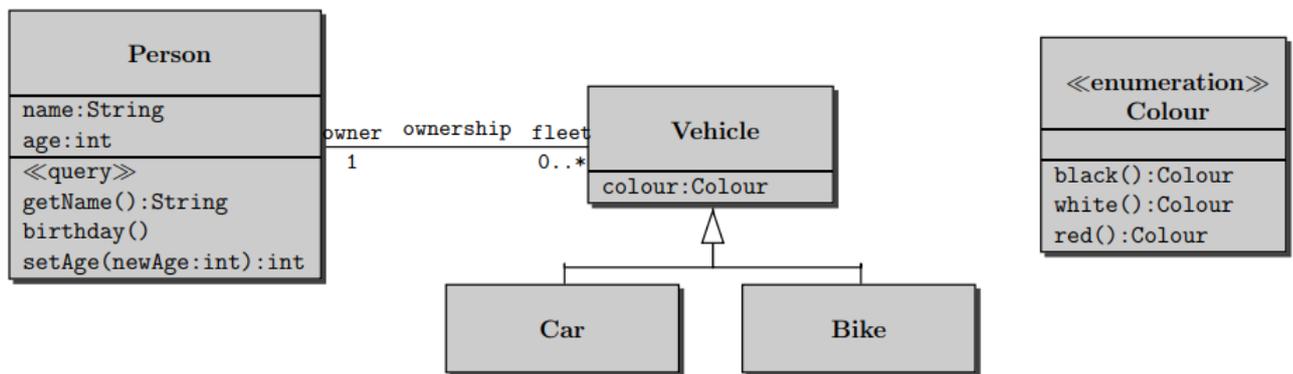


Рис. 1. Пример UML диаграммы из [4]

Для UML диаграммы, представленной на рис. 1, можно привести следующие примеры ограничений на OCL, следуя [4]:

“Владелец транспортного средства должен быть старше 18 лет”:

`context Vehicle`

`inv: self. owner. age >= 18`

“Никто не обладает более чем тремя средствами”:

`context Person`

`inv: self.fleet->size <= 3`

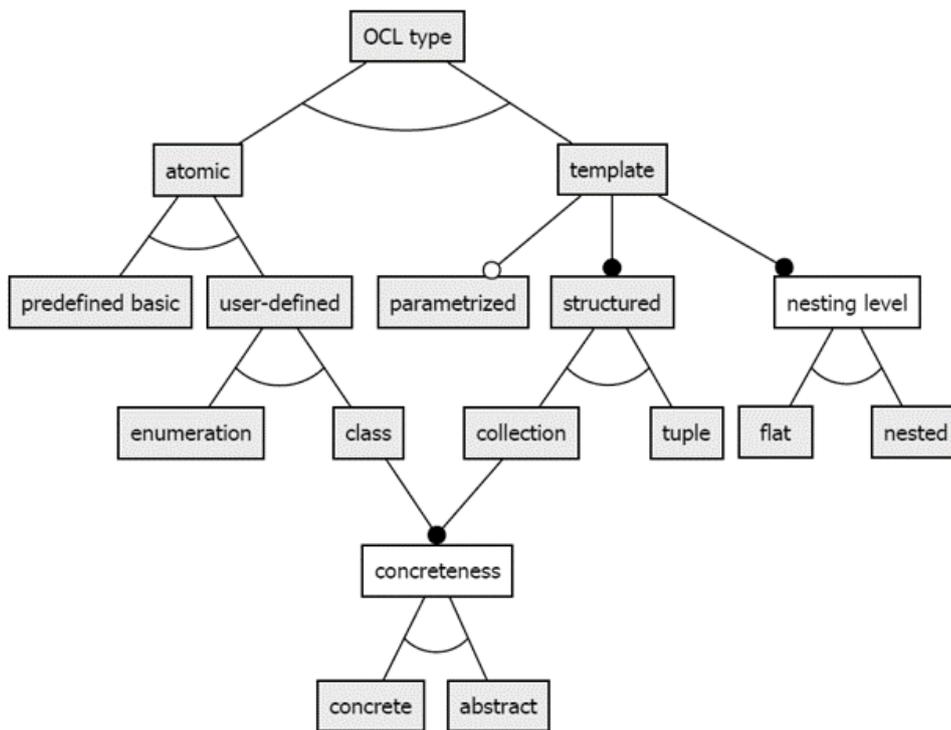


Рис. 2. Система типов OCL [5]

На рис. 2 представлен обзор системы типов OCL. Как видно, эта система довольно сложна, детально её устройство рассмотрено в [5].

За счёт широкой функциональности язык позволяет задавать большой диапазон ограничений. Но именно поэтому он не является интуитивно понятным и требует предварительного изучения.

2.1.2 VOCL

VOCL [6] – графическая интерпретация OCL, которая позволяет этому языку лучше интегрироваться в диаграммы UML. Для составления ограничений с помощью VOCL достаточно знания OCL.

Пример ограничения на OCL и его графическая интерпретация из [6] (рис. 3) представлены ниже.

```

context Company
inv: c.NumberOfEmployees > 50
  
```

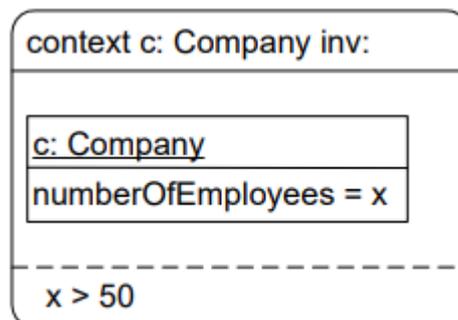


Рис. 3. Пример ограничения на VOCL [6]

Ограничение на рис. 3 представлено графическим элементом, разделённым на три части. В «шапке» указан контекст и класс ограничения. Середина – тело

ограничения – содержит условие («количество сотрудников должно равняться х»). Нижняя часть – текстовая форма для представления предикатов.

VOCL использует основанный на графах синтаксис для визуализации выражений OCL, которые получены из UML [7]. В VOCL есть графические символы для всех ключевых слов OCL [8].

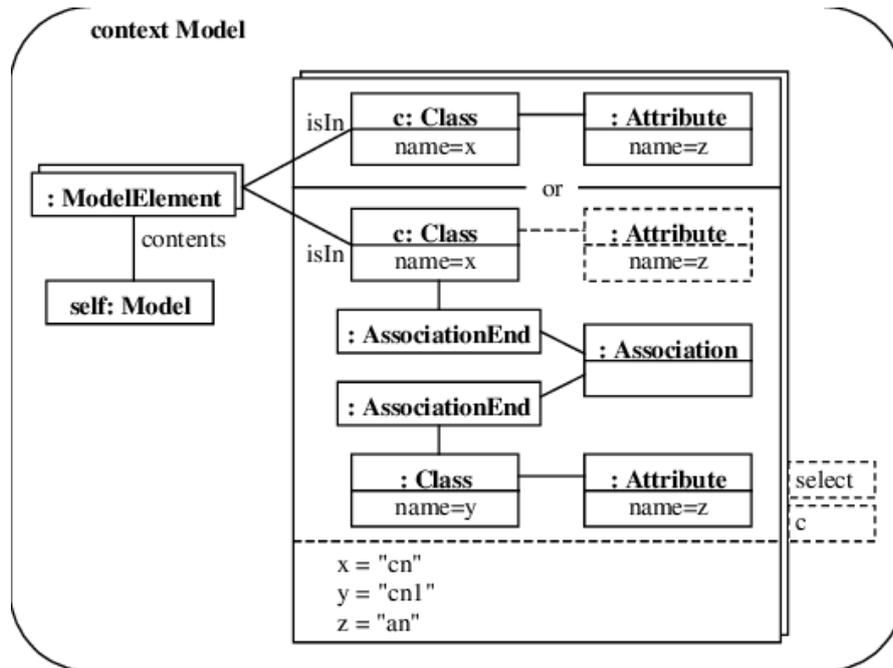


Рис. 4. Пример ограничения на VOCL [9]

На рис. 4 представлен пример более сложного ограничения, описанного на Visual OCL. Это запрос, который выбирает все классы с именем "cn" либо атрибутом "an", либо, в случае отсутствия таковых, те, которые имеют ассоциацию с каким-либо другим классом с именем «cn1», который, в свою очередь, имеет атрибут "an" [9].

Хотя VOCL является визуальным языком, он содержит много текстовых деталей и требует знания OCL, который, в свою очередь, нельзя назвать простым для понимания. Помимо этого, VOCL диаграммы получаются довольно объемными и, возможно, некоторые ограничения разумнее выражать в текстовом виде.

2.1.3 DPF

Diagram Predicate Framework [10] – проект, разработанный Бергенским Университетом с целью формализации предметно-ориентированной разработки (Model Driven Engineering). Основным преимуществом DPF является формальный подход к (мета)моделированию, трансформации модели и её управлению на основе теории категории и преобразования графов [10].

Пользователь вводит переменные, каждая из которых представляется вершиной или дугой графа в графическом редакторе.

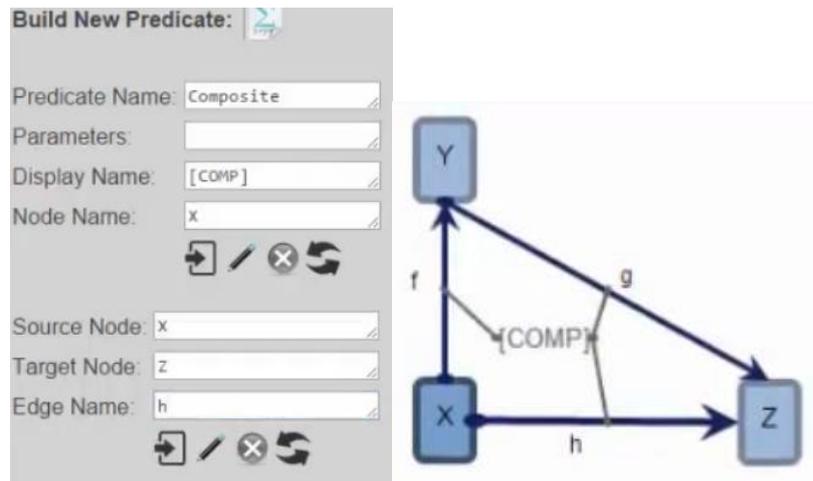


Рис. 5. Пример задания ограничения в DPF [10]

На рис. 5 показана модель в DPF и форма ввода данных для её создания. Заданные элементы относятся к предикату Composite.

```

1  //@@begin auto-generated code
2  function _Pred5(){
3      this.x = [];
4      this.y = [];
5      this.z = [];
6      this.f = [];
7      this.g = [];
8      this.h = [];
9  }
10 //@@end auto-generated code
11 _Pred5.prototype.validate = function(){
12     var errObj = [];
13     for(var i = 0; i < this.f.length; ++i){
14         |
15     }
16
17     return []; // return all the elements that violated the constraint...
18 };

```

Рис. 6. Текстовый редактор DPF [10]

Тело предиката указывается на языке JavaScript в текстовом редакторе, находящемся рядом с графическим. На рис. 6 изображён текстовый редактор DPF, здесь видно объявление элементов диаграммы с рис. 5.

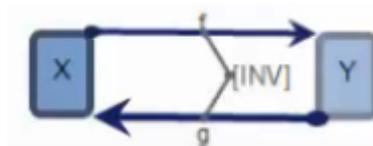


Рис. 7. Пример задания ограничения в DPF [10]

Ограничения в DPF – это, в основном, ограничения на части экземпляра модели. Они делятся на структурные и прикреплённые ограничения. Первые отвечают за свойства классов в метамодели, ограничения на типизацию (какие типы может содержать модель и как они связаны друг с другом) и задаются визуально (рис. 7). Прикреплённые ограничения описывают более

сложные конструкции, невыразимые структурными ограничениями, и задаются на OCL.

Структурные ограничения являются важным примером, так как они визуальны, просты и функциональны, они позволяют задавать фундаментальные ограничения, упрощать работу создания метамоделей и даже разработать систему автодополнения модели.

2.1.4 Визуальный язык задания ограничений на модели в QReal

На кафедре системного программирования СПбГУ велась разработка проекта QReal [11], являющегося предшественником REAL.NET. И в QReal уже были проведены исследования возможных решений для системы ограничений. В частности, визуальному языку задания ограничений на модели в QReal посвящена курсовая работа студентки кафедры системного программирования СПбГУ А.О. Дерипаска в 2012г.

Модель языка ограничений, представленного в данной работе, состоит из одной или нескольких диаграмм, каждая из которых позволяет задавать ограничение на один визуальный язык. Диаграмма основана на элементарных ограничениях, каждое из которых позволяет задавать поведение на элемент или множество элементов языка [12].

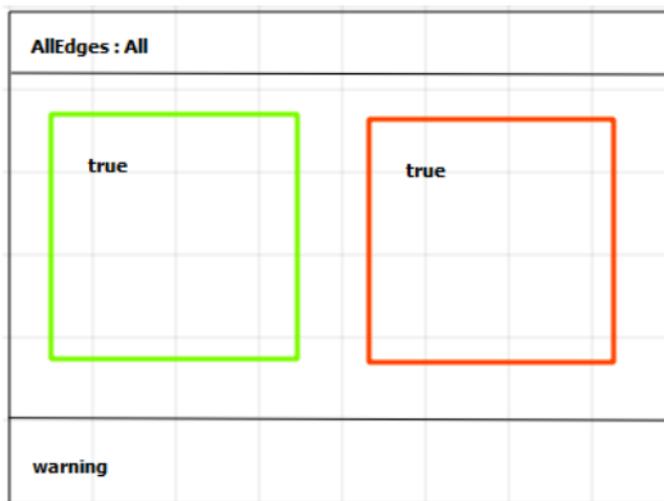


Рис. 8. Пример задания ограничения в визуальном языке задания ограничений на модели в QReal [12]

На рис. 8 представлен пример задания ограничения на этом языке. Он читается как «Для всех вершин всех языков связь должна иметь узел в начале и в конце». Тип, для которого определяется ограничение, и язык вводятся в текстовом виде.

Так как приведенный в данной главе язык был реализован в старой версии проекта и в любом случае нуждался в обновлении, было принято решение создать новый модуль ограничений, основанный на новом, ещё более простом языке.

2.1.5 Прототип системы ограничений в REAL.NET

Проект REAL.NET содержит прототип системы ограничений, описанный в курсовой работе автора [13].

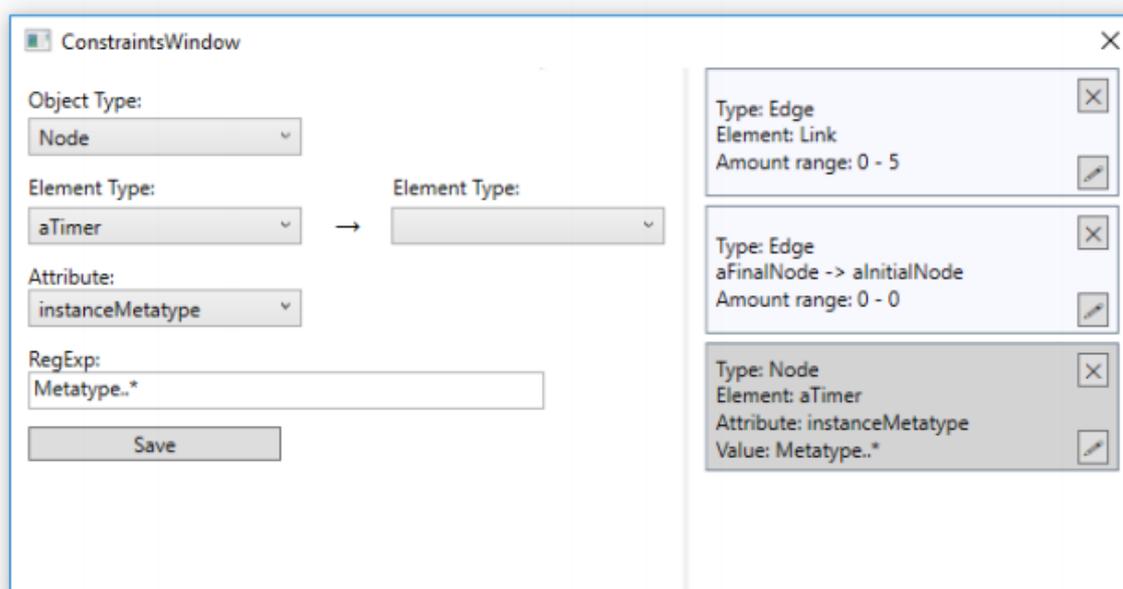


Рис. 9. Форма для работы с ограничениями из работы Алымовой Д.А. [13]

Система представляет из себя форму (рис. 9) для работы с базовыми ограничениями на модель и механизм проверки введенных ограничений. Интерфейс позволял задавать ограничения на количество элементов указанного класса в виде диапазона значений и на атрибуты элементов (через регулярные выражения).

Хотя она позволяла задавать нужные ограничения в рамках проекта, их диапазон был небольшим, и система не была основана на визуальном языке.

2.2 Инструменты

2.2.1 REAL.NET

REAL.NET – проект, разрабатываемый на кафедре системного программирования СПбГУ. Он является реализацией DSM (domain specific modeling) платформы, поддерживающей глубокое метамоделирование [3]. С помощью REAL.NET можно создавать визуальные языки и программы на них.

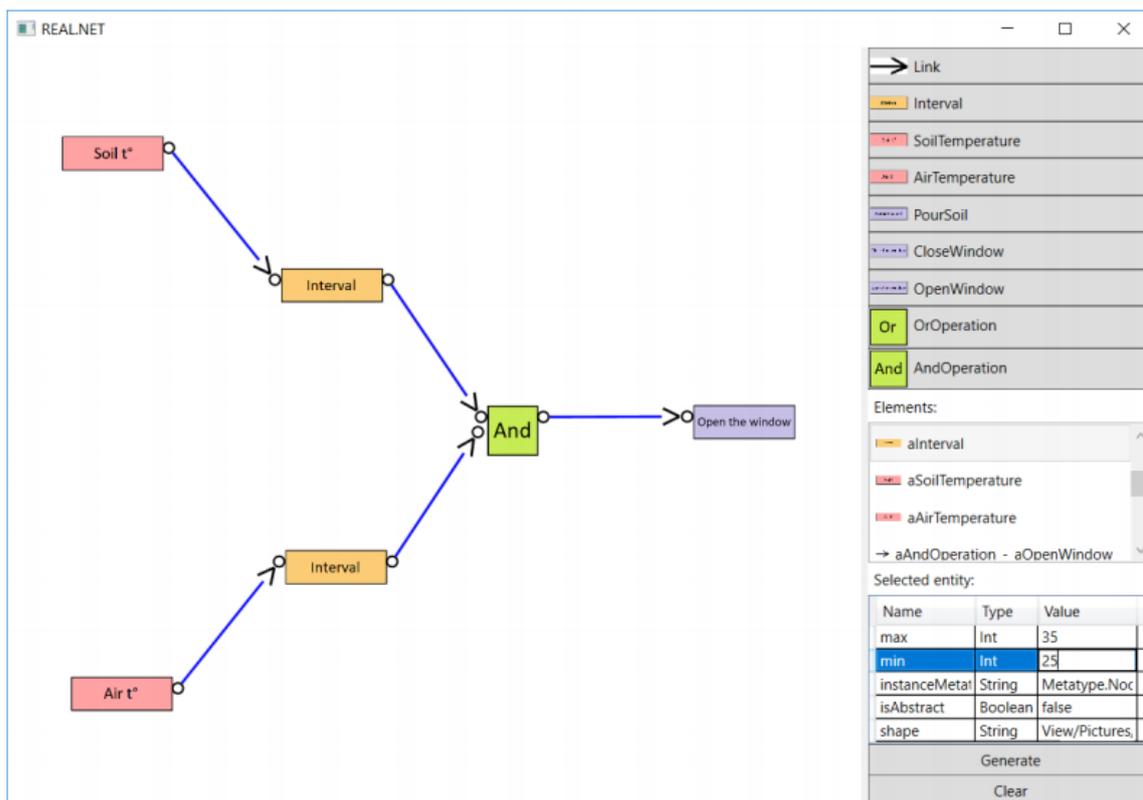


Рис. 10. Редактор REAL.NET на примере модели для умной теплицы [3]

Метамодел и модели создаются с помощью редактора, представленного на рис. 10. Модели представляются в виде графа на сцене (области, в которой рисуется модель), возможные элементы для которого располагаются в правом верхнем углу на палитре. Ниже находится список элементов рассматриваемой модели, а под этим списком выведен редактор свойств.

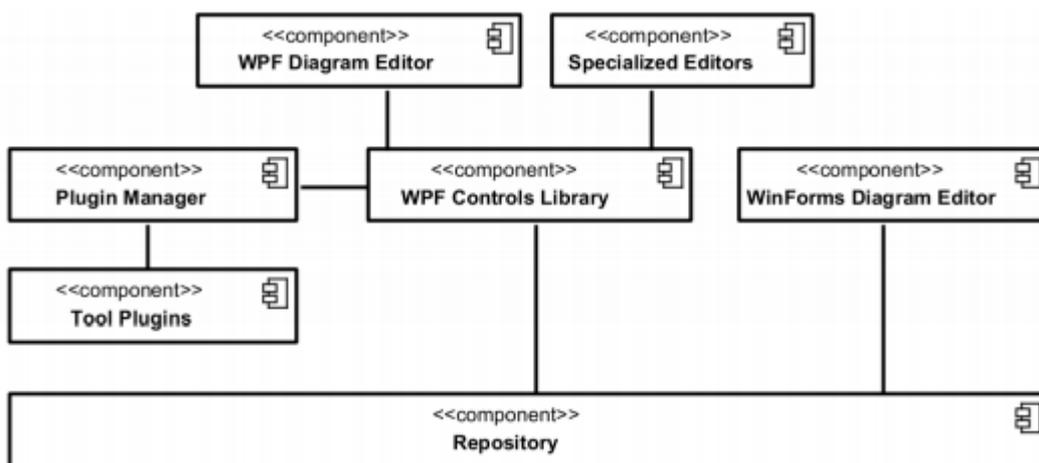


Рис. 11. Диаграмма компонентов REAL.NET [3]

На рис. 11 изображена диаграмма компонентов, отображающая архитектуру проекта. Особый интерес в рамках данной работы представляют следующие компоненты:

- репозиторий (Repository), который является, по сути, хранилищем данных, он содержит информацию обо всех моделях, элементах и свойствах;
- редактор (WPF Diagram Editor), используемый для работы с визуальными языками;
- библиотека компонентов (WPF Controls Library), которая подробно описывает графические элементы, в том числе сцену, палитру и редактор свойств;
- менеджер плагинов (Plugin Manager), являющийся соединяющим звеном между проектом и его плагинами.

Менеджер плагинов подключает все плагины, которые находит в соответствующей директории через конфигурационный класс. Через этот класс плагин может получать все необходимые параметры проекта.

Для визуализации графов используется библиотека с открытым исходным кодом GraphX [14], с её использованием работают сцена и графическое представление модели.

Подсветка элементов, необходимая для работы системы ограничений, осуществляется через классы модели, через свойство, хранящее информацию о том, выполнены ли ограничения для выбранного элемента.

2.2.2 WPF

Windows Presentation Foundation (WPF) [15] – платформа пользовательского интерфейса для создания приложений. Она является частью платформы .NET и использует язык XAML [16]. XAML – это основанный на XML язык разметки, позволяющий задать графическое представление страницы или приложения без программной реализации.

Связь разметки с кодом осуществляется через привязку (binding) [17]. Она устанавливает связь между пользовательским интерфейсом и логикой приложения. Если привязка и оповещения об изменении привязываемого значения правильно настроены, то элемент представления изменяется вместе с привязанной логической частью. Например, когда пользователь редактирует значение текстового поля, сопоставляемое ему значение в коде автоматически обновляется, и наоборот – при изменении какого-либо значения «изнутри» меняется и его представление.

С помощью WPF выполнена графическая часть проекта REAL.NET.

2.2.3. Язык программирования роботов

Основным визуальным языком, с которым сейчас ведётся работа в REAL.NET, является язык программирования роботов [18]. Это визуальный язык, основанный на графах, позволяющий задавать поведение роботов. Он используется в первую очередь для обучения детей. В REAL.NET представлены не все его элементы.

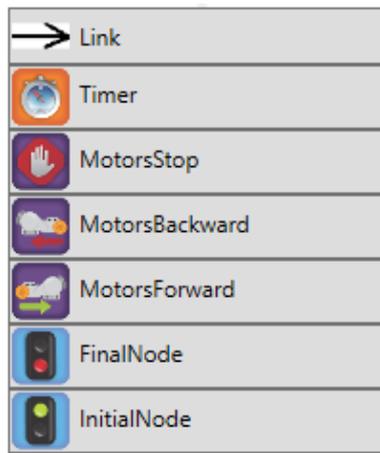


Рис. 12. Элементы языка программирования роботов, представленные в REAL.NET

Элементы языка (рис. 12) позволяют задать поведение робота в виде графа последовательных действий.

- InitialNode и FinalNode – вершины начала и конца программы.
- MotorsForward и MotorsBackward – вершины движения вперёд или назад.
- MotorsStop – вершина, обозначающая остановку.
- Timer – вершина таймера (например, использование вершины Timer с атрибутом delay на 1000 означает, что действие, указанное предыдущей вершиной, будет продолжаться 1000мс).
- Link – связь между действиями.



Рис. 13. Пример диаграммы поведения на языке программирования роботов

На рис. 13 показан пример программы на языке программирования роботов. Согласно этой программе, робот должен включиться, затем двигаться вперёд количество миллисекунд, указанное в свойстве вершины Timer, а после этого закончить работу.

3. Требования к языку

В первую очередь мы хотим добиться легкого создания визуального языка, чтобы человек, не владеющий навыками программирования, мог легко создать метамодель. Требования к метаязыку (языку, на котором описывается метамодель), порождают требования к языку ограничений, так как работать с ними будут одни и те же люди с одними и теми же наборами навыков. Иными словами, если метаязык понятен большинству пользователей, то и язык ограничений должен быть ориентирован на аудиторию такого же уровня. Отсюда следуют требования визуальности и интуитивности. Так как создание визуальных языков происходит в рамках проекта REAL.NET, то язык ограничений должен также быть интегрирован в него. Таким образом, были сформированы три требования, приведенные ниже.

1. Визуальность.
2. Интуитивность: язык понятен и не требует долгого изучения.
3. Интегрированность: хотелось бы, чтобы язык был стилистически похож на визуальные языки, ныне используемые в проекте.

Среди требований к языку ограничений не указана полнота, так как мы согласны уменьшить выразительность за счёт увеличения простоты. К тому же, в данный момент ведётся работа над реализацией задания ограничений на OCL в REAL.NET, которая позволит описывать более сложные ограничения.

4. Описание языка ограничений

4.1 Синтаксис

Язык ограничений состоит из семи типов элементов, логически разделённых на две группы. Первая группа – это элементы самого визуального языка, на который задаётся ограничение, но с отличающимися атрибутами. Вторая группа – логические вершины и дуги, специфические для данного языка ограничений.

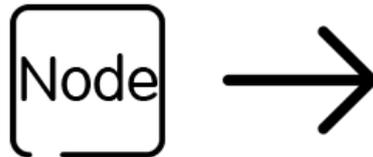


Рис. 14. Элементы визуального языка, на который задаётся ограничение

Хотя элементы первой группы (рис. 14) имеют тот же класс, что и элементы «оригинального» языка, все атрибуты этих вершин и дуг в языке ограничений имеют текстовый тип для того, чтобы пользователь мог задать свойство регулярным выражением или неравенством. Например, если вершина класса Timer не должна иметь свойство delay больше, чем 1000, то в ограничении это прописывается напрямую: свойство delay будет иметь значение <1000.

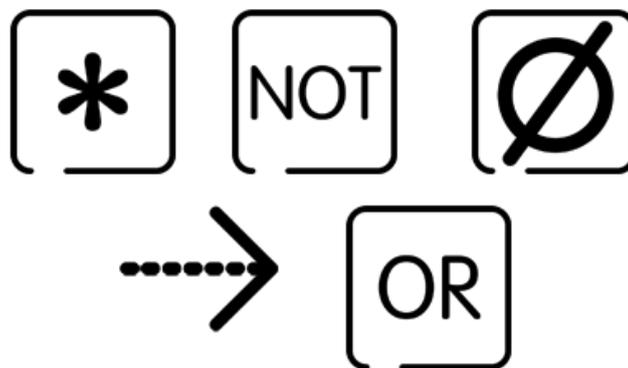


Рис. 15. Логические элементы

Внешний вид элементов второй группы представлен на рис. 15. Их стоит рассмотреть подробнее.



Рис. 16. Логическая дуга

- Логическая дуга (рис. 16) не имеет свойств и нужна для соединения обычных и логических вершин.



Рис. 17. Логическая вершина «любая»

- Вершина «любая» (рис. 17) заменяет вершину, вне зависимости от её класса. Важно пояснить, что, хотя класс вершины может быть любым, это не распространяется на её атрибуты. То есть вершина модели совпадёт с вершиной типа «любая» только при совпадении всех атрибутов. Если пользователь хочет в ограничении описать вершину, которая совпадала бы с абсолютно любой вершиной языка, он должен отдельно прописать это в атрибутах данной вершины.



Рис. 18. Логическая вершина «отрицание»

- Вершина «отрицание» (рис. 18) служит для проверки отсутствия участка модели, следующего за ней. То есть она используется в случаях, когда заданного далее подграфа быть не должно.

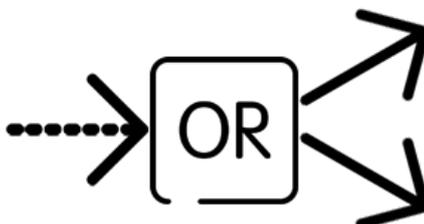


Рис. 19. Логическая вершина «или»

- Вершина «или» (рис. 19) используется для проверки наличия хотя бы одного из указанных подграфов.



Рис. 20. Логическая вершина «пустое множество»

- Вершина «пустое множество» (рис. 20) описывает отсутствие исходящих дуг.

Внешний вид элементов выбран с учётом того, что язык должен использоваться для разных метамоделей (следовательно, иметь нейтральный вид), но, одновременно с этим, вписываться в имеющиеся в проекте языки (чтобы соответствовать требованию интегрированности).

4.2 Семантика

Выражение языка ограничений является деревом и состоит из «триггерного элемента» и тела ограничения (рис. 21).

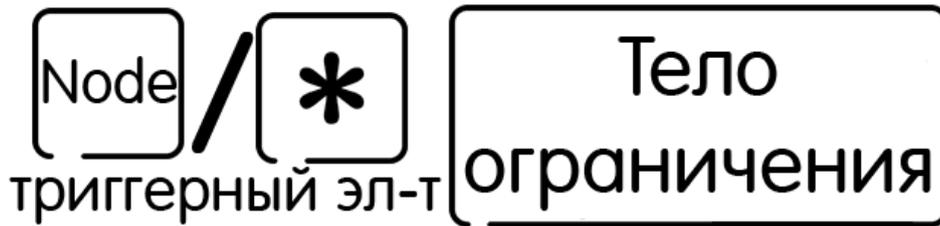


Рис. 21. Выражение языка ограничений

Триггерный элемент – это корень дерева, вершина, для которой выполняется проверка ограничения. Для того, чтобы сработал «триггер» (то есть чтобы началась проверка ограничения для этого элемента), необходимо совпадение классов и атрибутов. В случае, если «триггерным элементом» является логическая вершина «любая», класс не важен.

Тело ограничения состоит из любых элементов языка. Тело может быть не указано, в таком случае выражение является ограничением на атрибут, то есть все элементы указанного в качестве «триггерного элемента» класса обязаны иметь атрибуты указанного значения. Например, если мы зададим ограничение вершиной класса «Рыба» с атрибутом «живая» равным false, то при проверке все вершины класса «Рыба» будут проверяться на значение атрибута «живая».

Всё выражение задаёт подграф ограничения. Как только при проверке находится триггерная вершина в модели, для неё ищется заданный подграф. Отсутствие такого подграфа означает, что проверка не выполнена.

Проверка выполняется рекурсивно – внутри подграфа ограничения каждая вершина из списка вершин модели рассматривается как «триггерная» и сравнивается с соответствующей вершиной модели.

Безусловно, язык, описанный в данной главе, может быть усовершенствован. Уже в процессе работы над ним обсуждалась такая функциональность как дополнительные логические вершины, уточнение количества элементов, описание общих для всего выражения переменных и обязательность проверки ограничения («если тело ограничения существует, то должно быть именно такое»), но она не была реализована, чтобы не нарушить простоту и интуитивность языка.

5. Прототип модуля проверки ограничений

5.1 Интеграция

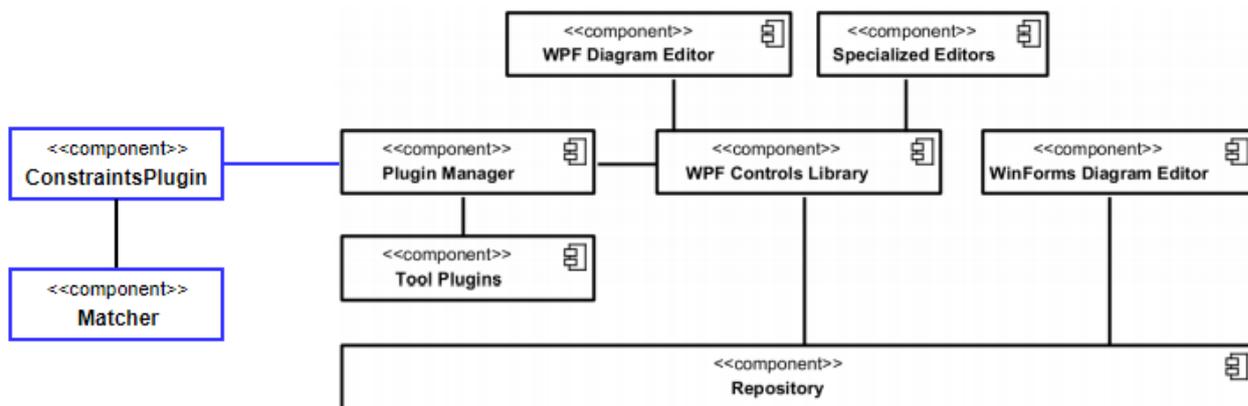


Рис. 23. Архитектура задействованных модулем проверки ограничений проектов в REAL.NET

Задействованные в работе модуля ограничений компоненты системы REAL.NET представлены на рис. 23.

Черным показаны компоненты, имевшиеся в проекте до начала работы над модулем ограничений, синим – использующиеся для этого модуля. Задействованные в рамках данной работы компоненты описаны в разделе «Инструменты» в «Обзоре литературы».

ConstraintsPlugin работает с репозиторием, редактором и библиотекой компонентов через менеджер плагинов. Обработка задач, связанных с ограничениями, их хранением и визуальной частью, происходит именно в ConstraintsPlugin.

Компонента Matcher отвечает за работу над графами. Классы этой компоненты не связаны с ограничениями, они получают на вход модели (модель языка ограничения и модель языка, на который это ограничение накладывается) и проверяют существование подграфа, указанного в модели языка ограничений. В этой же компоненте происходит проверка ограничения перед его сохранением (соответствует ли введенный граф описанию выражения языка ограничений).

5.2 Архитектура классов

Модуль проверки ограничений состоит из двух логических компонент: графической части (интерфейс для работы с ограничениями) и логической (проверка ограничения на принадлежность языку, проверка выполнения ограничения). Диаграмма классов модуля представлена ниже на рис. 22.

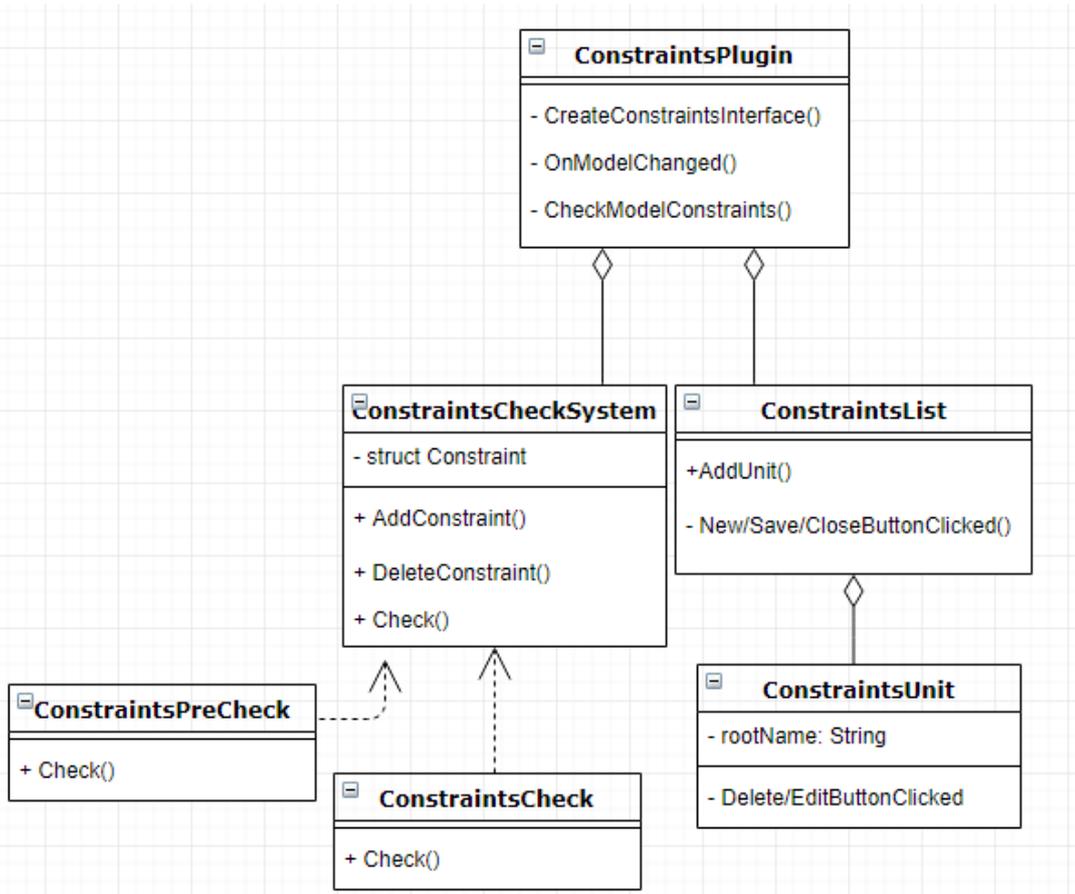


Рис. 22. Архитектура модуля проверки ограничений

В качестве главного класса для управления как запуском механизма проверки, так и визуальной частью является `ConstraintsPlugin`. Он инициализирует нужные системе ограничения области интерфейса, обрабатывает действия пользователя, следит за изменением действующей модели в основном редакторе и запускает проверку ограничения.

Класс `ConstraintsCheckSystem` реализует работу с ограничениями и их хранение. При создании ограничения вызывается проверка из класса `ConstraintsPreCheck`, во время которой отсеиваются некорректно заданные ограничения. При вызове пользователем проверки модели вызывается класс `ConstraintsCheck`, в котором реализован уже сам механизм проверки.

Визуальная часть реализована классом `ConstraintsList`, который представляет из себя панель со списком созданных ограничений и классом `ConstraintsUnit`, который, в свою очередь, является графическим представлением ограничения. Интерфейс позволяет не только создавать, но и редактировать и удалять ограничения.

6. Тестирование и апробация

6.1 Тестирование

В качестве тестовой модели для задания ограничений выступила модель программирования роботов. Далее приведены примеры ограничений, задаваемых на неё на созданном языке ограничений, и результат работы модуля.

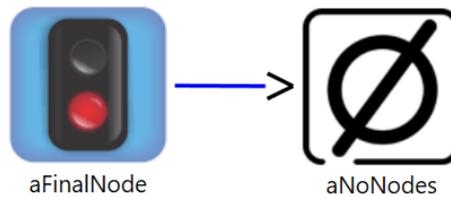


Рис. 24. Пример ограничения

На рис. 24 представлен пример ограничения на вершину модели. Оно трактуется как «ни одна дуга не должна выходить из вершины типа FinalNode». Ниже представлен пример программы, в которой ограничение с рис. 25 не выполнено.

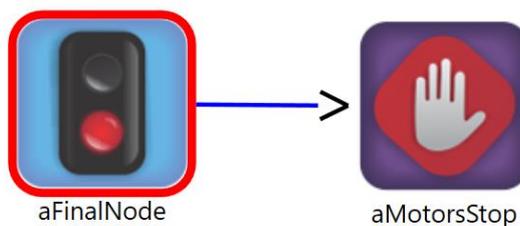


Рис. 25. Пример оповещения пользователя о наличии невыполненного ограничения

В случае наличия в модели вершины, для которой не выполняется заданное ограничение, такой элемент подкрашивается. В данном примере из вершины типа FinalNode выходит дуга, что противоречит ограничению, приведённому на рис. 25, о чём пользователь оповещается при проверке. Если же все ограничения для элемента выполнены (или их нет), то окраска снимается.



Name	Type	Value
delay	Int	>0
instanceMetatype	String	Metatype.Node
isAbstract	Boolean	false
shape	String	View/Pictures/timerBlock.png

Рис. 26. Пример ограничения на атрибут

На рис. 26 представлен пример ограничения атрибут вершины. Он обязывает вершину класса Timer иметь значение атрибута delay выше, чем 0.



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

aTimer

Рис. 27. Пример положительного результата проверки

Рис. 27 показывает положительный результат проверки, так как атрибут delay имеет значение выше, чем 0 и ограничения для вершины класса Timer выполнены.



Name	Type	Value
delay	Int	0
instanceMetatype	String	Metatype.Node
isAbstract	Boolean	false
shape	String	View/Pictures/timerBlock.png

aTimer

Рис. 28. Пример отрицательного результата проверки

При изменении значения свойства delay у рассматриваемой вершины и повторном вызове проверки пользователь получит отрицательный результат – ограничение не выполняется, вершина подкрашивается красным цветом.

6.2 Апробация

Для оценки соответствия языка списку требований, составленному ранее, были опрошены четырнадцать человек разной степени технической грамотности и уровня знакомства с проектом или визуальным программированием в целом.

В качестве материала для ознакомления людям предоставлялся документ с разделом описания языка роботов, языка ограничений и тестирования. После прочтения каждый участник апробации имел порядка двадцати минут чтобы подумать, поработать с графическим редактором, придумать собственное ограничение и ответить на вопросы по удобству использования.

Опрошенные, работающие в прошлом или в данный момент с языком программирования роботов, ответили, что им было бы интересно применить этот язык ограничений.

В свою очередь участники, не имеющие никакого отношения к программированию, не испытали затруднений при работе с проектом и процессом задания ограничений. Студентка экономического факультета, в частности, была воодушевлена наглядностью языка и легкостью его использования.

Основным замечанием, прозвучавшим четыре раза из четырнадцати, была необходимость наглядной инструкции для работы с языком, так как опрошенным было представлено только формальное описание и несколько

примеров. При ознакомлении с описанием у людей возникали вопросы, но они пропадали после устных неформальных пояснений. Также в контексте работы с языком роботов некоторым респондентам был непонятен выбор цветовой гаммы (чёрно-белые элементы, тогда как элементы языка роботов цветные), но этот пункт поясняется в разделе описания языка.

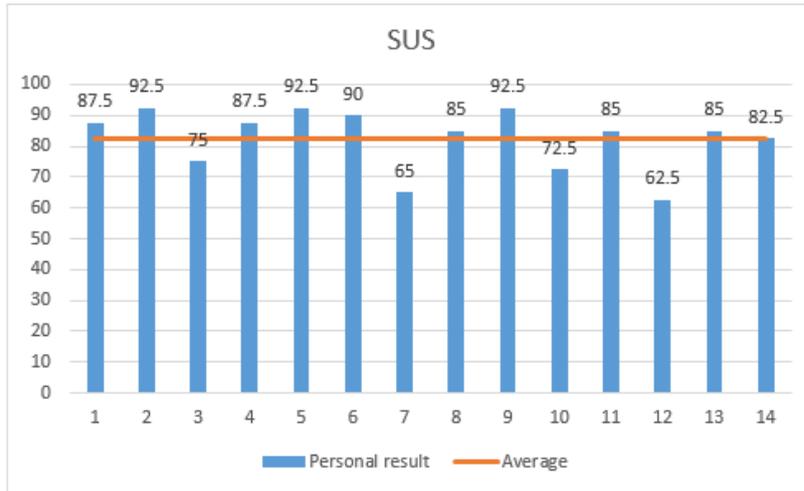


Рис. 29. Результаты SUS

По шкале удобства использования системы (system usability scale) созданный язык ограничений был оценён на 82.5 балла, что равносильно оценке «людям нравится ваша работа и они рекомендовали бы её» [19]. На рис. 29 показаны результаты теста для разных респондентов и среднее значение.

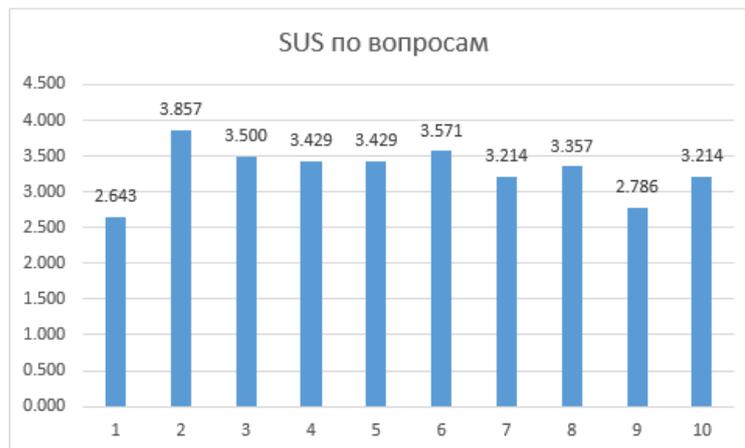


Рис. 30. SUS для каждого вопроса

Как показывает диаграмма на рис. 30, самый высший балл язык получил по пункту номер два, который звучит как «Я считаю систему излишне сложной» (то есть опрошенные не считают систему излишне сложной), а самый низкий балл по пункту один, который звучит как «Я хотел бы постоянно использовать эту систему», что объясняется тем фактом, что абсолютное большинство опрошенных редко сталкиваются с визуальным программированием, следовательно, нечасто испытывают необходимость использовать

визуальный язык ограничений в целом. К тому же, данная работа разрабатывалась как способ задать простейшие ограничения, поэтому она не отвечает абсолютно всем требованиям к системе ограничений.

Язык ограничений, разработанный в рамках данной работы, можно назвать аналогом структурных ограничений в DPF, позволяющим задавать несложные ограничения на свойства элементов или их последовательность, тогда как более сложные содержательные ограничения удобнее выразить через OCL.

В целом результаты апробации показали, что язык соответствует требованиям, сформулированным в данной работе.

7. Заключение

В рамках данной дипломной были получены следующие результаты:

1. были сформулированы требования к создаваемому языку ограничений, а именно: визуальность, интуитивность и интегрируемость;
2. был проведён обзор следующих систем проверки и языков ограничений: OCL, VOCL, DPF;
3. был создан визуальный язык ограничений, соответствующий требованиям, сформулированным в рамках данной работы;
4. был разработан прототип модуля задания ограничений на созданном языке для модели роботов в REAL.NET;
5. модуль был протестирован на различных ограничениях и была проведена апробация на пользователях с разным уровнем технической подготовки, которая показала соответствие разработанного языка поставленным требованиям.

Список литературы

- [1] M. Portsmore, ROBOLAB, Intuitive Robotic Programming Software to Support Life Long Learning // APPLE Learning Technology Review. — 1999
- [2] Литвинов Ю.В., Кузьмина Е.В., Небогатиков И.Ю., Алымова Д.А., Среда предметно-ориентированного визуального моделирования REAL.NET // Всероссийская научная конференция по проблемам информатики СПИСОК. — 2017
- [3] Elizaveta Kuzmina, Yurii Litvinov, Implementation of “smart greenhouse” visual programming tool using deep metamodeling // Готовится к публикации. — 2019
- [4] Bernhard Beckert, Introduction to OCL // UNIVERSITÄT KOBLENZ-LANDAU, URL: <http://www.unikoblenz.de/~beckert/Lehre/Verification/10OCL.pdf> (дата обращения: 10.04.2019)
- [5] Jordi Cabot, Martin Gogolla, Object Constraint Language (OCL): a Definitive Guide // 12th Int. School on Formal Methods: Model-Driven Engineering (SFM'12). — 2012
- [6] Christiane Kiesner, Gabriele Taentzer, Jessica Winkelmann, Visual OCL: A Visual Notation of the Object Constraint Language // Forschungsberichte der Fakultät IV -- Elektrotechnik und Informatik, Bericht-Nr. — 2002
- [7] Christian Heinzemann, Component Story Decision Diagrams // Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn. — 2014
- [8] Manuel Oriol, Bertrand Meyer, Objects, Components, Models and Patterns // 47th International Conference, TOOLS EUROPE 2009, Zurich, Switzerland. — June 29-July 3, 2009
- [9] Dominik Stein, Stefan Hanenberg, Rainer Unland, A Graphical Notation to Specify Model Queries for MDA Transformations on UML Models // University of Duisburg-Essen
- [10] Thomas Calder, A Model Driven Approach to Web Page Evaluation // Master Thesis The Department of Computer Engineering Bergen University College & The Department of Informatic University of Bergen. — 2016
- [11] А.Н. Терехов, Т.А. Брыксин, Ю.В. Литвинов и др., Архитектура среды визуального моделирования QReal. // Системное программирование. Вып. 4. СПб.: Изд-во СПбГУ. 2009, С. 171-196
- [12] Дерипаска Анна, Визуальный язык задания ограничений на модели в QReal // Кафедра системного программирования СПбГУ. — 2012
- [13] Алымова Дарья, Прототип системы ограничений в REAL.NET // Кафедра системного программирования СПбГУ. — 2018
- [14] Alexander Smirnov, George Birbilis, GraphX for .NET (Core Branch) // <https://github.com/panthernet/GraphX> (дата обращения: 08.05.2019)
- [15] Maira Wenzel, Dennis Lee, Petr Kulikov, etc., Introduction to WPF in Visual Studio // URL: <https://docs.microsoft.com/en-us/dotnet/framework/wpf/getting-started/introduction-to-wpf-in-vs> (дата обращения: 10.05.2019)

[16] Genevieve Warren, Gordon Hogenson, Saisang Cai, etc., Get started with WPF // URL: <https://docs.microsoft.com/en-us/visualstudio/designers/getting-started-with-wpf?view=vs-2019> (дата обращения: 10.05.2019)

[17] Genevieve Warren, Gordon Hogenson, Saisang Cai, etc., Data Binding (WPF) // URL: <https://docs.microsoft.com/en-us/dotnet/framework/wpf/data/data-binding-overview> (дата обращения: 10.05.2019)

[18] Dmitry Mordvinov, Yurii Litvinov, Timofey Bryksin, Andrey Terekhov, TRIK Studio: Technical Introduction // Saint Petersburg State University. —2017

[19] Jeff Sauro, MEASURING USABILITY WITH THE SYSTEM USABILITY SCALE (SUS) // URL: <https://measuringu.com/sus/> (дата обращения: 14.05.2019)