Санкт-Петербургский государственный университет

Кафедра системного программирования

Прототип системы ограничений в REAL.NET

Курсовая работа студентки 344 группы

Алымовой Дарьи Андреевны

Научный руководитель: доц. Литвинов Ю. В.

Санкт-Петербург 2018

Оглавление

1) Введение

Для работы над проектом в конкретной предметной области необходимы специалисты, сочетающие как отличные навыки программирования, так и хорошее знание контекста задачи. Однако, чем уже предметная область, тем более редким является такое сочетание. Для упрощения работы специалиста, который, не обладая навыками программирования, сталкивается с задачей создать некую работающую систему, может использоваться визуальное программирование.

Визуальное программирование помогает, не вдаваясь в узкие нюансы классического программирования, задать поведение системы и наглядно его оценить и показать людям, ничего не знающим о деталях проекта. Помимо этого, визуальное программирование является хорошей платформой для обучения людей обычному программированию – можно понять принципы взаимодействия объектов в целом, как они устроены.

Одним из важных инструментов визуального программирования являются CASE-системы (Computer Aided Software Engineering). В целом их определяют как системы, помогающие в проектировании и разработке программ. Для CASE-систем характерно использование графических средств, которые позволяют анализировать различные виды диаграмм, относящихся к предметной области. Основным программным средством системы является графический редактор, за счёт чего CASE-система и относится к визуальному программированию. Пользователь задаёт систему и её поведение в редакторе, после чего код генерируется автоматически. Таким образом, знание программирования совсем необязательно для создания самой программы.

В качестве попытки обобщения процесса создания CASE-системы, привязанной к предметной области, были созданы meta-CASE системы, позволяющие применить подход визуального

программирования самим к себе. Такая система позволяет создавать метамодели – модель, лежащая на один уровень абстракции выше, задающая некие правила для построения моделей по своему подобию. Meta-CASE система позволяет создать абстрактную CASE-систему, которую, в свою очередь, можно адаптировать для конкретного контекста задачи.

В этапы создания метамодели входит также задание ограничений на её элементы – пользователь должен иметь возможность контролировать их для создания более корректной системы. Тема данной работы – создание прототипа системы ограничений для meta-CASE системы.

2) Постановка задачи

На кафедре системного программирования в течение нескольких лет велась разработка над meta-CASE системой QReal [1]. По мере роста популярности платформы .NET было решено перенести на неё QReal. Так начал развиваться проект REAL.NET [2]. Он представляет из себя набор графических редакторов (метаредакторов) с возможностью создавать метамодели и переключаться между ними (структура REAL.NET приведена в пункте 4.1).

Но важно не только создать модель, но и наложить на неё необходимые ограничения, чтобы она работала правильно. Так как проект находится в стадии разработки, встал вопрос о создании прототипа системы, которая даёт пользователю возможность наложить базовые ограничения на модель.

В итоге было решено осуществить возможность задания ограничений на количество элементов модели и значения атрибутов этих элементов, а также реализовать вывод информации о выполнении заданных ограничений в удобной форме. Таким образом, задача состоит в реализации:

1) GUI для задания ограничений;
2) механизма проверки заданных ограничений на количество элементов и значения их свойств;
3) механизма выделения элементов, не прошедших проверку;
4) вывода сообщений о нарушении ограничений.

3) Обзор существующих решений

2.1) Diagram Predicate Framework

Diagram Predicate Framework (DPF) (далее – перевод оригинальной документации [3]) – исследовательский проект, начатый Бергенским университетским колледжем и Бергенским университетом в Норвегии в начале 2006 года. Целью проекта является формализация концепций MDE (Model Driven Engineering).
Целью MDE, как и любой методологии, является повышение качества и производительности разработки. Это осуществляется с помощью представления моделей как главных сущностей процесса разработки и адаптации трансформаций модели для автоматизации реализации. MDE позволяет разработчикам рассуждать на более высоком уровне абстракции и сосредоточиться на предметной области. Это помогает уделять меньше внимания повторяющейся и подверженной ошибкам работе, например, написанию кода.

В современном MDE модели, как правило, обусловлены средствами языков моделирования, такими, как UML (Unified Modeling Language) Семантика этих языков, в свою очередь, в основном определяется полуформально, с помощью текстового описания на английском языке. Это может не гарантировать степень точности, необходимую MDE. По сути, исследования в этой области говорят, что для раскрытия потенциала MDE необходим формальный подход.
DPF пытается устранить этот недостаток, предоставляя формальный подход к (мета)моделированию, трансформации модели и её управлению на основе теории категории и преобразования графов.
В графическом редакторе DPF действует система задания и проверки ограничений, которая в будущем может быть рассмотрена и использована в проекте.

2.2) Object Constraint Language (OCL)

Object Constraints Language (OCL) - это декларативный язык, описывающий правила, применяемые к UML, разработанный IBM, и являющийся частью стандарта UML. [4]

Для понимания языка разберём простой пример.

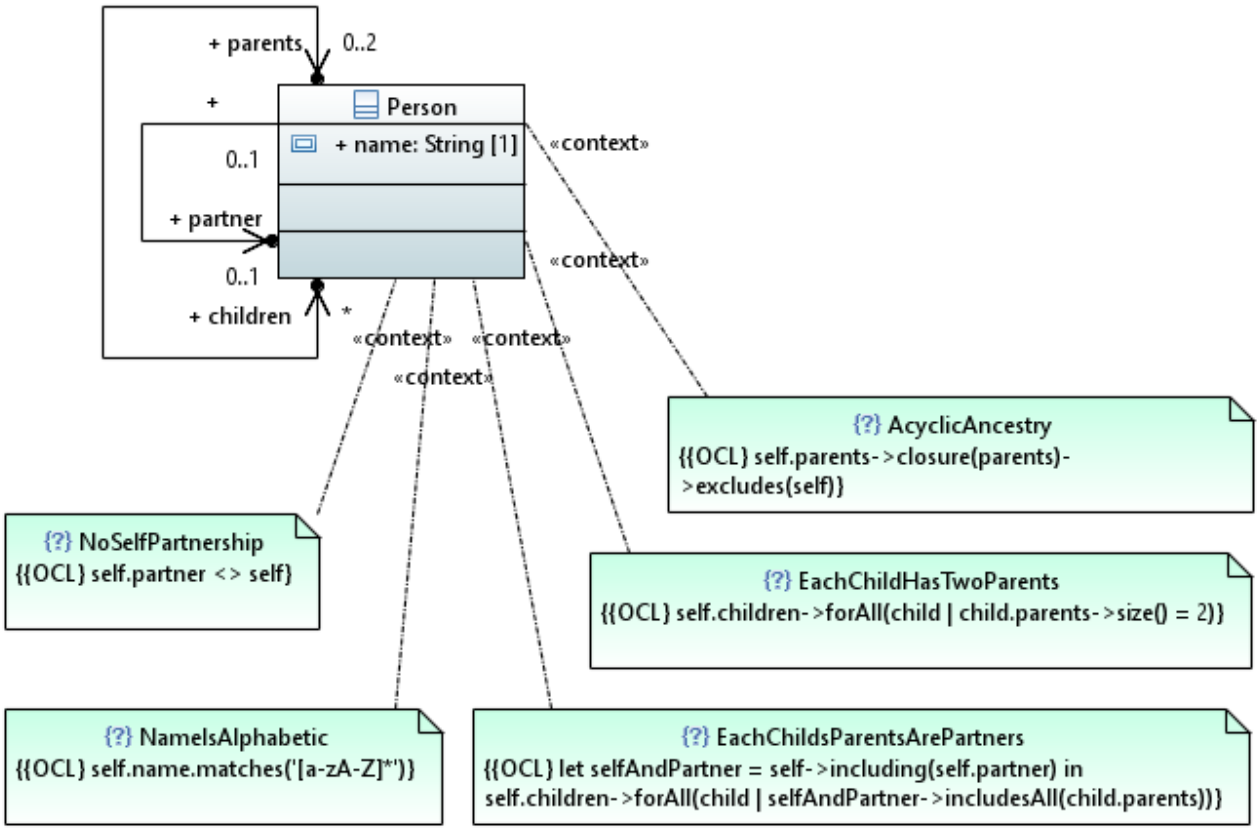

Рис. 1 Пример задания ограничений на OCL [5]

На рис. 1 показан пример задания ограничений на OCL. В качестве модели приведен объект Person со связями партнёр (0..1), родители (0..2) и дети (0..1).

Персона не может являться собственным партнёром:

- NoSelfPartnerShip: self.partner <> self

Имя задано буквами латинского алфавита:

- NameIsAlphabetic: self.matches(‘[a-zA-Z]*’)

Проверка на ацикличность (родители, родители родителей и т.д. не содержат персону):

- AcyclicAncestry: self.parents -> closure(parents) -> excludes(self)

У ребёнка должно быть двое родителей:

- EachChildHasTwoParents: self.children->forAll(child | child.parents->size() = 2)

Родители ребёнка должны быть партнёрами:

- EachChildsParentsArePartners: let selfAndPartner = self.oclAsSet()->including(self.partner) in self.children->forAll(child | selfAndPartner->includesAll(child.parents))

Таким образом, даже на незамысловатом примере можно увидеть основную логику языка. OCL – мощный инструмент задания ограничений, но он не визуален, что делает его менее применимым в среде визуального программирования.

Рассмотренные варианты для задания ограничений являются хорошим средством для уже устоявшейся системы, однако задачей текущей работы в условиях развивающегося проекта было создание лишь прототипа системы ограничений для задания и проверки самых основных правил.

4) Описание решения

4.1) Среда визуального моделирования REAL.NET

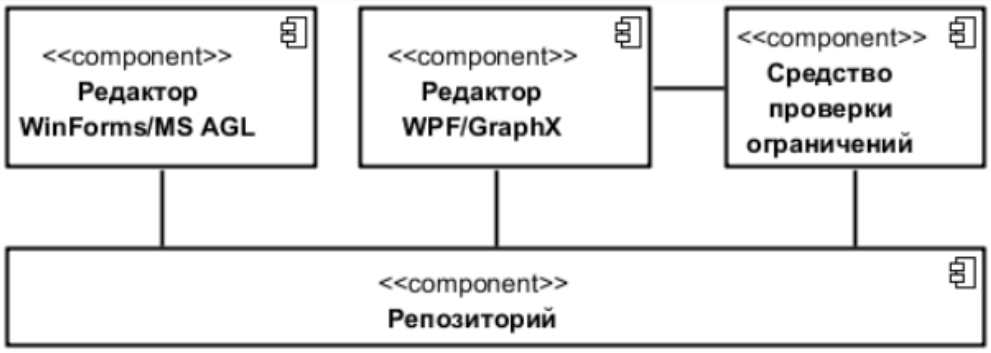

Рис. 2 Общая структура REAL.NET[2]

Общая структура REAL.NET представлена на рис. 2. В контексте данной задачи имеет смысл рассматривать только редактор WPF/GraphX, так как именно для него разрабатывается система ограничений.

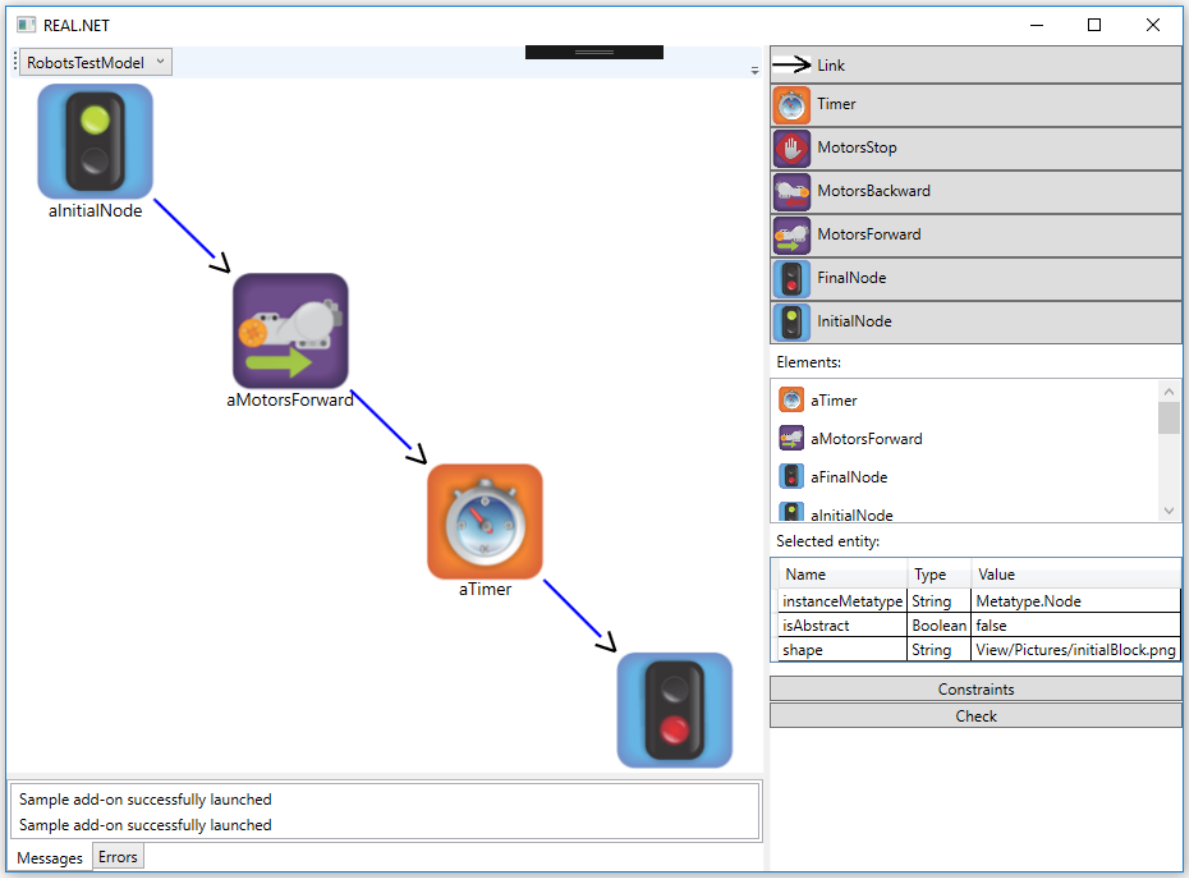

Рис. 3. Редактор WPF/GraphX

На рис. 3 представлен редактор WPF/GraphX. Он хранит действующую модель, которая, в свою очередь, содержит информацию обо всех своих элементах. Вызов окна задания ограничений осуществляется нажатием на кнопку Constraints.

4.2) GUI для задания ограничений

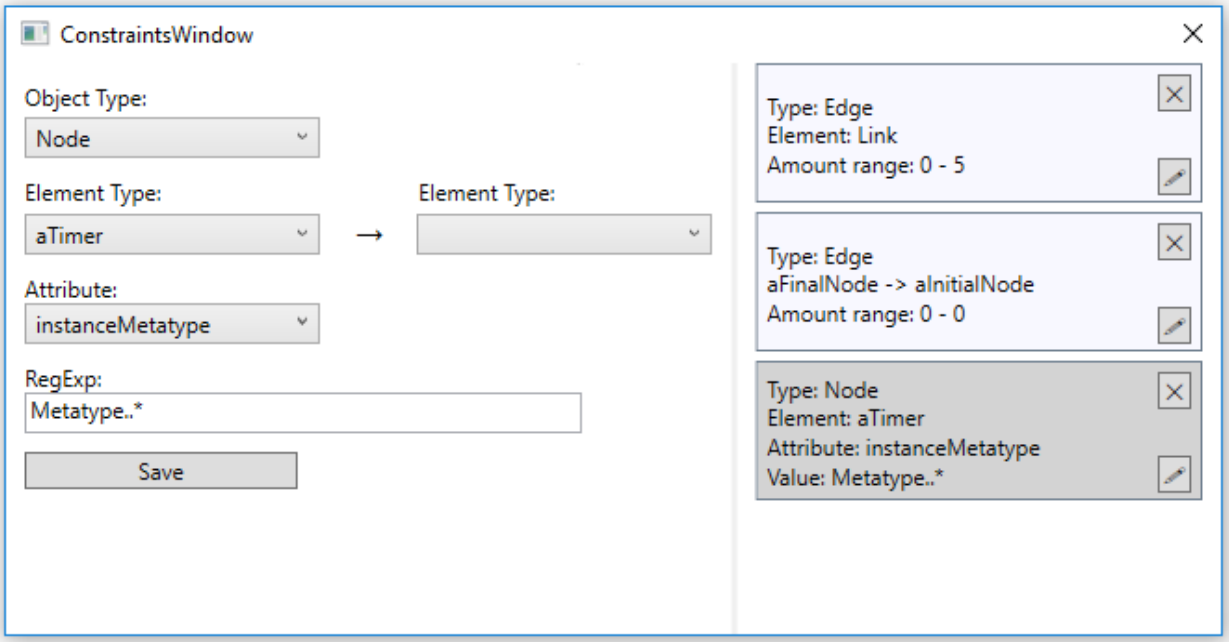

Рис. 4. Окно задания ограничений

В результате работы было создано окно (доступное по нажатию кнопки из основного интерфейса проекта) задания ограничений (рис. 2).

В правой части представлены в виде панели уже существующие ограничения модели (они доступны для редактирования или удаления). В левой части пользователь может выбрать

- тип объекта (вершина или связь)
- тип элемента (один тип для задания ограничений на количество элементов и атрибуты, два типа – для задания ограничений на дугу из типа1 в тип2)
- атрибут
- значение атрибута

Атрибуты могут быть типа Boolean (тогда пользователь выбирает true или false), String (тогда появляется возможность ввода регулярного выражения, которому должно соответствовать значение атрибута) или Int (тогда вводится пара чисел, задающая диапазон возможных значений, края включаются). Аналогично с заданием ограничений на атрибуты типа Int, ограничение на количество элементов задаётся диапазоном. В этом случае, если пользователь хочет ввести конкретное число, он пишет диапазон с одинаковыми значениями краёв (например, 1 – 1).

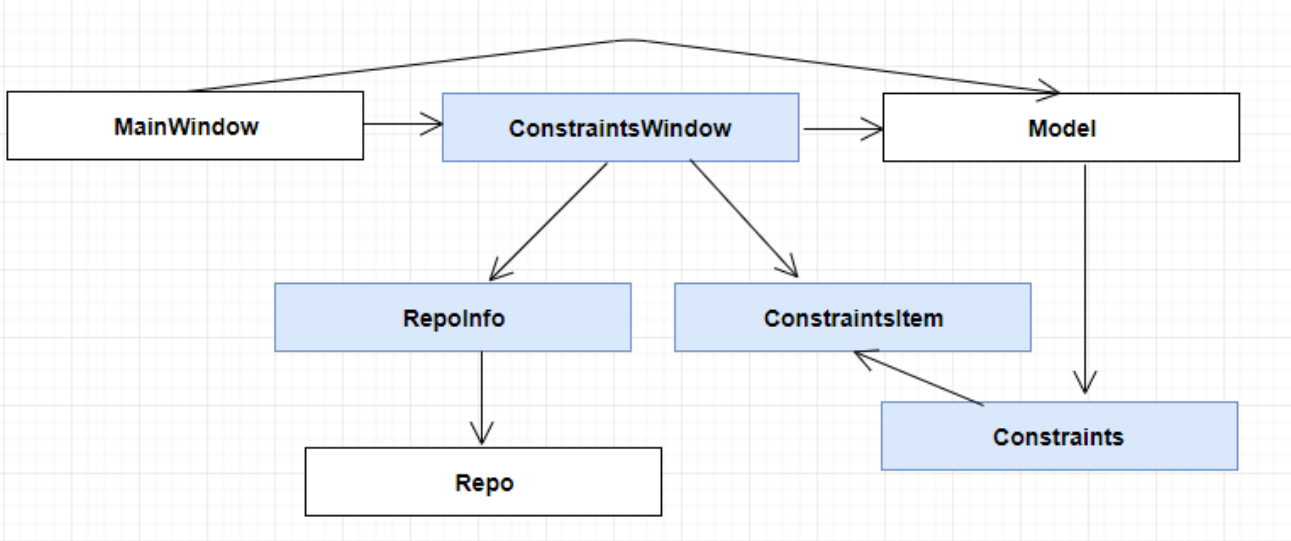

Рис. 5. Диаграмма классов, связанных с интерфейсом системы ограничений

На рис. 3. представлена диаграмма классов, связанных с интерфейсом системы ограничений. Здесь и далее на приведенных диаграммах синим цветом выделены классы, созданные в рамках работы.

ConstraintsWindow – элемент системы, описанный в пункте 4.1), вызывающийся из MainWindow. В качестве аргумента окну ограничений передаётся модель. Класс Constraints хранит список элементов типа ConstraintsItem, который является источником данных для панели ограничений. При удалении или добавлении элемента список обновляется, при попытке добавить ограничение на тип объекта и элемента, ограничение на которые уже существует, имеющееся значение ограничения меняется. Список существует в классе модели,

поэтому ограничения хранятся на протяжении работы программы. При своей инициализации ConstraintsWindow обращается к model.Constraints и заполняет панель элементами списка.

Класс RepoInfo предоставляет классу ConstraintsWindow необходимую информацию из Repo (класс, хранящий данные модели) – список существующих в модели элементов, их типы и атрибуты, типы атрибутов. Эти списки используются как источники элементов типа ComboBox в окне ограничений.

4.3) Механизм хранения и проверки заданных ограничений, вывод сообщений

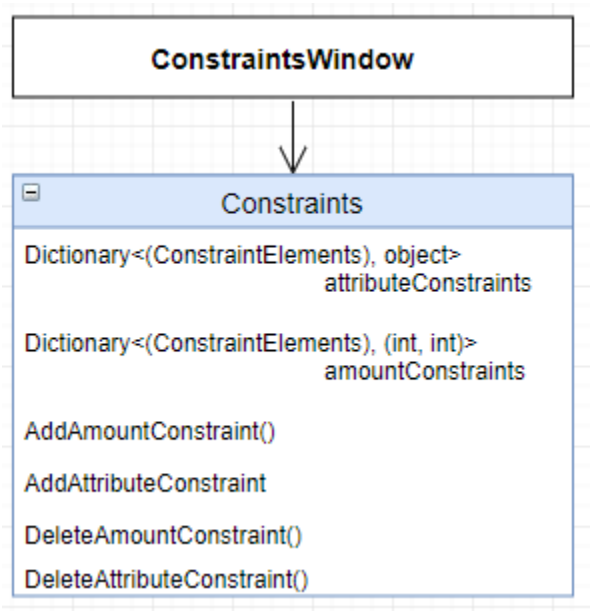

Рис. 6. Управление ограничениями в Constraints

После нажатия на кнопку удаления или сохранения ограничений информация о введённых данных попадает в Constraints (рис. 4). В зависимости от совершенного действия вызывается метод добавления/удаления количественного ограничения или ограничения на атрибуты. Правила хранятся в виде Dictionary в model.Constraints, причём ограничения на атрибуты и ограничения на количество хранятся в разных Dictionary (amountConstraints и attributeConstraints).

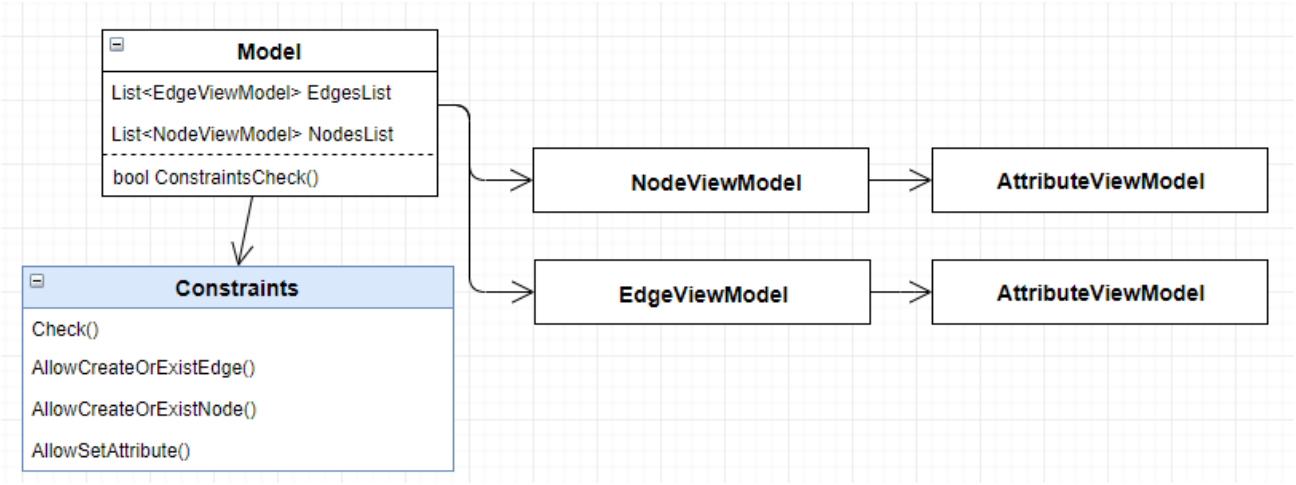

Рис. 7. Взаимодействие ограничений с элементами во время проверки

Когда пользователь закрывает окно ограничений, происходит автоматическая проверка. Помимо этого, основное окно проекта рядом с кнопкой вызова ограничений имеет кнопку Check, которая выполняет аналогичную проверку. Взаимодействие классов во время проверки показано на рис. 5.

Начинается проверка с вызова метода model.ConstraintsCheck(), который возвращает true или false, в зависимости от результата проверки. В случае, если ограничения нарушены, в консоль ошибок программы выведутся все элементы, у которых есть нарушение, иначе пользователь увидит сообщение о том, что проверка прошла успешно.

Model.ConstraintsCheck(), в свою очередь, возвращает результат работы метода Constraints.Check вместе с Constraints.ErrorMessage (сообщение об ошибке для вывода в консоль).

Constraints.Check() принимает на вход список вершин и связей модели, проходит по ним и их атрибутам и проверяет соответствие с системой ограничений с помощью методов CheckEdge/Node() и AllowSetAttribute(). Если правила ограничений нарушены, в сообщение об ошибке добавляется информация об «элементе-нарушителе».

4.3) Визуальная обработка элементов

При проверке ограничений элементы, которые нарушают заданные правила ограничений, подсвечиваются красным. Необязательно проводить проверку, чтобы вызвать подсветку – создаваемые элементы, которые не соответствуют правилам, тоже выделяются во время своей инициализации. Пример того, как это выглядит, представлен на рис. Д и Е.

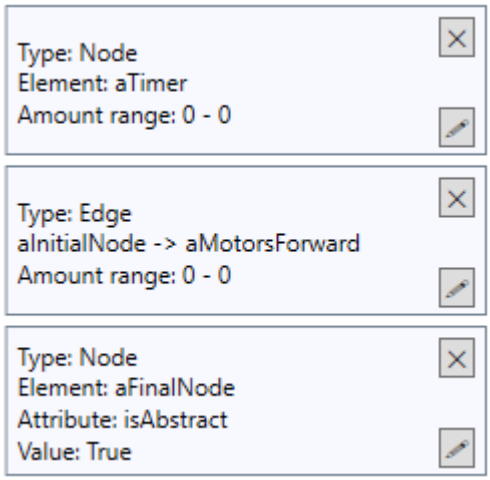

Рис. 8. Заданные правила

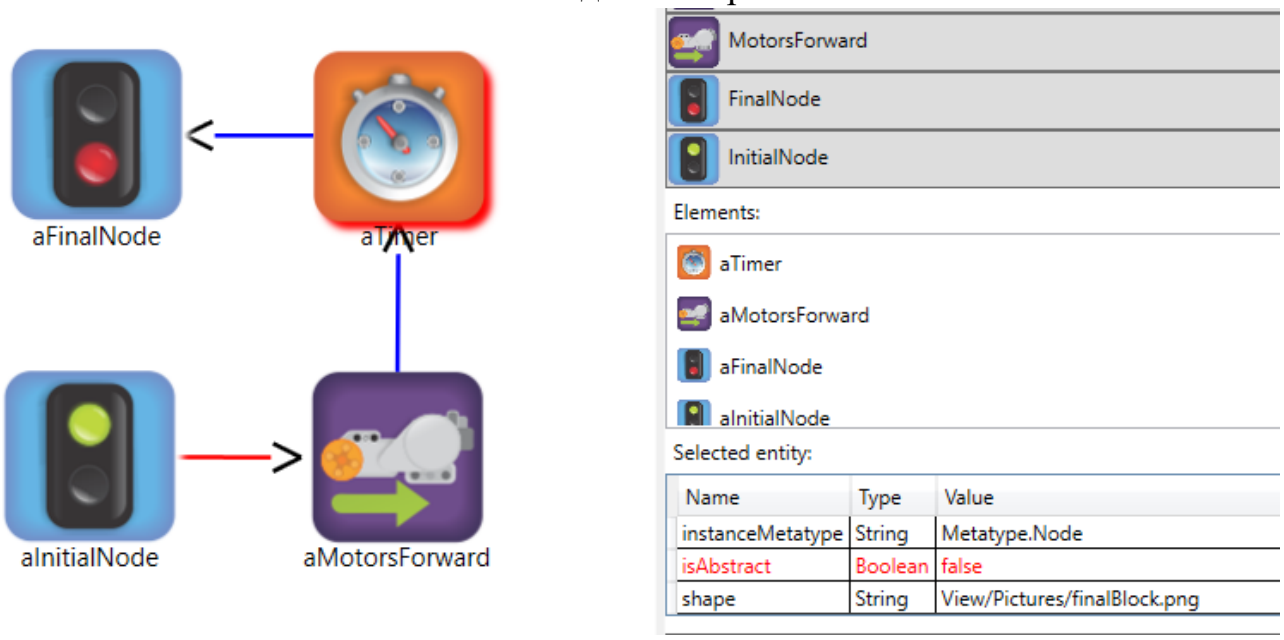

Рис. 9. Подсветка элементов с нарушениями ограничений

Дуга, вершина или атрибут, не соответствующие заданным ограничениям, окрашиваются в красный (в случае вершины красной становится тень). На рис. 6 показаны введенные правила ограничений: должны отсутствовать элементы типа aTimer и дуги из aInitialNode в aMotorsForward, а значение атрибута isAbstract элемента типа aFinalNode должно равняться false. Поэтому у соответствующих элементов появилась красная подсветка (рис. 7).

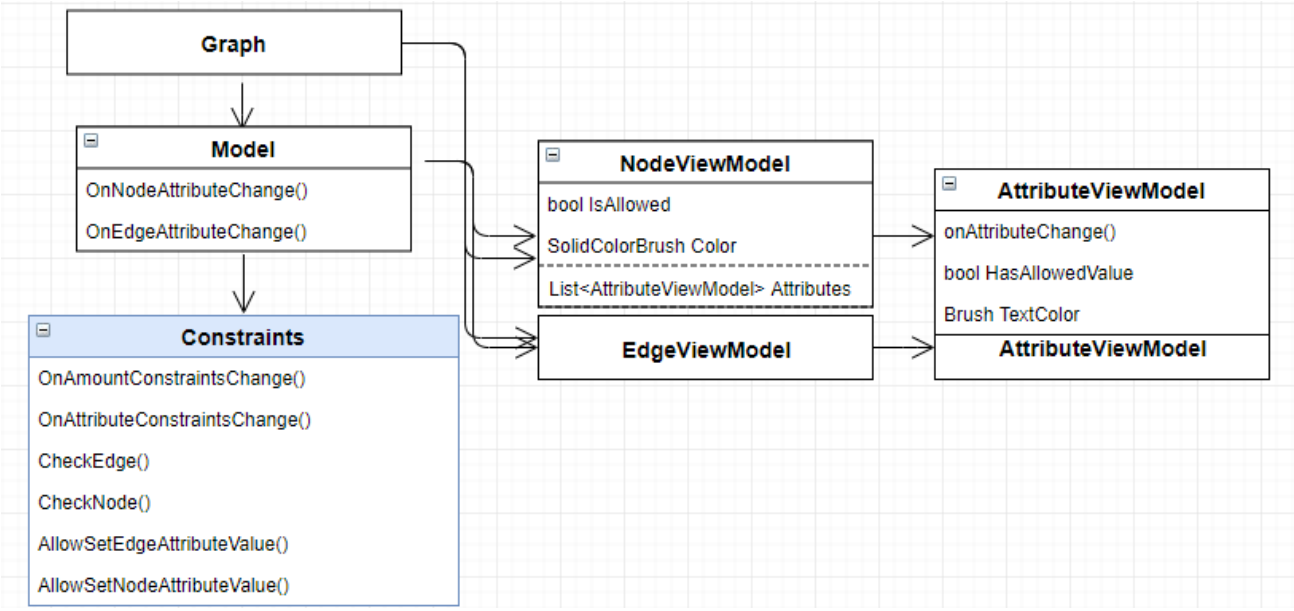

Рис. 10. Взаимодействие классов в ходе подсветки элементов

На рис. 8 представлена диаграмма классов для обработки подсветки элементов.

Подсветка настраивается свойством Color в классах элементов, а это свойство в свою очередь меняется в зависимости от значения свойства IsAllowed. Во время создания связи или вершины в классе Graph значением этого свойства становится результат проверки model.Constraints.AllowCreateOrExistNode/Edge(). Аналогичным образом работает перекрашивание атрибута: цвет текста задаёт TextColor, который зависит от HasAllowedColor. При инициализации элемента и её атрибутов для каждого атрибута создаётся событие

OnAttributeChange(), которое привязывается к методу model.OnEdge/NodeAttributeChanged. В этих методах происходит проверка значения меняемого атрибута.

При изменении ограничений класс Constraints создаёт событие OnAmount/AttributeChange(), которое провоцирует модель на новую проверку и перекрашивание элементов.

5) Заключение

Основными итогами данной курсовой работы стали реализация прототипа системы ограничений в среде Real.NET. Был создан интерфейс, позволяющий пользователю в удобном виде задавать ограничения на количество элементов модели, на их свойства. Таким образом, были реализованы:

1) интерфейс для задания пользователем ограничений;

2) механизм проверки ограничений на количество элементов и их значения свойств;

3) механизм выделения элементов и свойств, не прошедших проверку ограничений;

4) вывод сообщений об ошибках в случае невыполнения ограничений.

6) Список литературы

[1] QReal – URL:

http://qreal.ru/static.php?link=QReal (online; accessed: 10.01.2018).

[2] Литвинов Ю.В., Кузьмина Е.В., Небогатиков И.Ю., Алымова Д.А., Среда предметно-ориентированного визуального моделирования REAL.NET – URL:

http://se.math.spbu.ru/SE/seminar/materials/realnet.pdf(online; accessed: 10.01.2018).

[3]Diagram Predicate Framework – URL:

http://www.uib.no/en/rg/pt/56494/diagram-predicate-framework (online; accessed: 26.02.2018).

[4] Object Constraint Language – URL:

https://en.wikipedia.org/wiki/Object_Constraint_Language (online; accessed: 02.03.2018).

[5]OCL Constraint Examples for UML (using Papyrus) – URL:

https://help.eclipse.org/oxygen/index.jsp?topic=%2Forg.eclipse.ocl.doc%2Fhelp%2FOCLExamplesforUML.html (online; accessed: 02.03.2018).