Санкт-Петербургский государственный университет

Математико-механический факультет

Кафедра системного программирования

Визуальный язык задания ограничений на модели в QReal

Курсовая работа студентки 345 группы

Дерипаска Анны Олеговны

Научный руководитель : ст. преп. Литвинов Ю. В.

Санкт-Петербург

2012

Оглавление

1). Введение.

Разработка сложных программных систем является довольно трудоёмким процессом и требует немалое количество высококвалифицированных программистов. Но вполне очевидно, что в мире их недостаточно много. Поэтому уже давно стали разрабатываться различные способы упрощения этого процесса, чтобы гораздо большее число программистов могло бы решить возможно более сложные задачи, что увеличивает общую производительность труда всей команды. Одним из таких упрощений является визуальное программирование. Оно основано на том, что программы записываются с помощью набора визуальных моделей, а не на текстовом языке программирования, что является гораздо более наглядным и на порядок упрощает понимание программы. Тем более эти же самые модели могут быть полезны и специалистам предметной области разрабатываемой программной системы, кто не имеет достаточных навыков программирования, и различным руководителям, заинтересованным прежде всего в общем понимании разрабатываемой системы и контроле за ходом выполнения работы.

Для работы с такими визуальными моделями используются CASE-системы, которые содержат в себе редакторы одного или нескольких визуальных языков, средства генерации кода по диаграммам на этих языках и другие инструменты разработки ПО, которые для этого языка можно сделать. Они сильно помогают программистам при создании ПО. При этом универсальные CASE-системы в качестве реализуемого ими языка, как правило, имеют визуальный язык общего назначения, который, как правило, является очень мощным и тем самым сложным, что может усложнить написание программной системы. Поэтому стал набирать популярность DSM-подход (Domain Specific Modelling, предметно-ориентированное моделирование), при котором визуальные языки и соответствующие редакторы для них создаются под конкретную задачу. Такой подход значительно упрощает процесс разработки ПО, что еще больше увеличивает производительность труда программистов. Но заметим, что создание CASE-систем может быть довольно сложной задачей, т.к. создание графического редактора визуального языка вручную трудоёмко. При этом каждый раз при разработке программной системы необходимо сначала создать свою узкоспециализированную CASE-систему, а потом собственно решать требуемую задачу. И это может быть экономически неоправданно, если в дальнейшем у нас не будет или будет мало выгоды от переиспользования этого средства по сравнению с затратами на его создание.

Как следствие, появилась идея как-то автоматизировать процесс создания своей CASE-системы. И для этого решили применить подход визуального моделирования к самим CASE-системам, так появились metaCASE-системы. Они позволяют достаточно легко и быстро создавать предметно-ориентированные языки и соответствующие инструменты их поддержки. Программист должен сначала только описать синтаксис своего визуального языка на специальном формальном визуальном языке, предназначенном для этого, в специальном редакторе (называемом метаредактором). Далее по этому описанию автоматически генерируется код, реализующий редактор для этого языка. Помимо этого дополнительно можно задавать правила для последующей генерации визуальной модели на создаваемом языке в некоторый текстовый язык, по которым при генерации из описания этого визуального языка генерируется соответствующий генератор для нашего языка. Так же можно задавать семантику интерпретации нашего визуального языка, которая впоследствии позволит отлаживать визуальные модели на этом языке, и .т.п. При описании синтаксиса визуального языка программист, как правило, должен создать метамодель своего языка. Она представляет собой описание языка на уровне абстракций предметной области, для задания которого используется специальный метаязык, позволяющий это делать. Такая метамодель позволяет удобно задать синтаксис визуального языка и некоторые семантические особенности, но далеко не все, что требуются для языка.

При этом, создавая свой визуальный язык для решения определенной задачи, мы всё же хотим сделать его не только работоспособным, но и достаточно удобным для работы. А именно, в частности, сделать язык таким, чтобы минимизировать возможность написания некорректных программ. Для этого требуется не только синтаксическая корректность будущих

программ, что вполне гарантируется заданной метамоделью языка, но и семантическая корректность, которою в полной мере нельзя описать в метамодели. Поэтому появилась идея также иметь возможность задания каких-либо правил семантики создаваемого языка, например ограничений, т.е. некоторых логических условий. Существует два типа основных ограничений. Во-первых, это ограничения на состояние системы во время выполнения программы, написанной на данном языке. И во-вторых, ограничения на сам язык, т.е. ограничения, задаваемые на модель создаваемой программы и проверяемые только во время её написания, что обеспечивает корректность написанной программы в данной предметной области. Нетрудно заметить, что ограничения второго типа (т.е. на сам язык) задаются на том же уровне, что и описывается визуальный язык, т.е. на уровне метамодели, в то время как ограничения первого типа (т.е. на состояние системы) должны задаваться уже при написании конкретной программы.

Для описания ограничений очевидно тоже нужен специальный язык их задания, так называемый язык задания ограничений. А также нужен генератор, который по формальному описанию ограничений сгенерирует код, реализующий эти ограничения, и возможность проверять их. И вот как раз создание такого языка ограничений с необходимыми инструментами и было целью данной работы. При этом отметим, что описывать и проверять ограничения на язык проще, чем на состояние системы. Поэтому в данной работе будут пока рассмотрено задание ограничений именно на сам язык.

2). Обзор существующих решений.

Задавать ограничения можно двумя способами: или на естественном языке, или на формальном языке. Но утверждения и условия, записанные неформально, т.е. на естественном языке, во-первых, могут трактоваться неоднозначно, а во-вторых, сложны для интерпретации вычислительной системой. Формальный же язык ограничений не допускает вольности в толковании высказываний и имеет стандартный синтаксис и семантику, что позволяет и пользователю, и вычислительной системе легко интерпретировать его. Далее будут рассмотрены самые известные языки задания ограничений, OCL и VOCL, а также визуальный язык контекстных ограничений для баз данных, сделанный одним из студентов Мат-Меха СПбГУ в качестве дипломной работы.

2.1). Контекстные ограничения для баз данных [6].

На кафедре системного программирования математико-механического факультета СПбГУ в качестве дипломной работы были создан визуальный язык задания контекстных ограничений для баз данных и разработано средство их генерации по визуальному представлению. Эта дипломная работа была сделана Жолудевым В. В. применительно к технологии Real-IT, разрабатываемой тоже на нашей кафедре. Хотя, как утверждается в отчете дипломной работы, его генерация может быть использована и отдельно, так как нету особой привязки к конкретным средам моделирования.

Среда программирования Real-IT позволяет автоматически генерировать приложения, тесно связанные с базами данных и вообще обработкой данных. Тем самым для создателей этой среды было важно поддержать возможность задания и генерации ограничений на данные, что и было сделано. При этом ограничения используются на уровне баз данных и реализуется с помощью триггеров. Ограничения задаются на основе диаграмм классов и кооперaций UML и представляются при помощи языка XMI (XML Metadata Interchange, унифицированное представление UML в виде XML). Затем по этим входным данным генерируются контекстные ограничения в виде триггеров баз данных.

Стоит пояснить, что такое контекстные ограничения. Это понятие было введено А.Н. Ивановым в своей диссертации. Если рассмотреть два экземпляра классов, связанных между собой ассоциацией, то можно выделить все объекты и ассоциации, связанные с этими двумя объектами, которые и называются контекстом исходной ассоциации. При этом если действительно существуют объекты, связанные с выделенными объектами при помощи ассоциаций, то считается, что существует контекст. А если для ассоциации можно построить контекст, то она называется допустимой. Соответствующие ограничения и называются контекстными ограничениями.

Итак, в рассматриваемой дипломной работе была сделана генерация контекстных ограничений для баз данных по представлениям в виде диаграмм кооперaций UML. При этом была поддержана генерация и для множественности, и для наследования, и других не самых простых деталей.

Но рассматриваемая работа не подходит нам, так как это решение сделано именно для баз данных, а в качестве языка задания используется подобие общецелевого языка UML, что может быть излишне для нашей задачи в рамках этой курсовой.

Далее рассмотрим наиболее известные языки задания ограничений.

2.2). Object Constraint Language (OCL)

Одним из самых распространенных формальных языков задания ограничений является объектный язык ограничений OCL (Object Constraint Language) [4], который разрабатывался как встроенный механизм задания ограничений в самом известном общецелевом визуальном языке UML. OCL является текстовым языком для описания дополнительных условий и ограничений [10]. При этом выражения на OCL могут иметь смысл как запросов на объекты из модели некоторого визуального языка, так и инвариантных логических условий на состояние системы

во время выполнения программы. Это в основном обеспечивается тем, что тут есть возможность описывать различные инварианты, задавать пред- и пост- условия на методы и операции и т.д. Задание ограничения на языке OCL представляет собой описание контекста, т.е. указание типа некоторого объекта или его метода, на которое[-ые] мы хотим наложить ограничение, и выражения, обозначающего собственно логику требуемого ограничения. Логическим выражением как раз чаще всего являются инварианты (для классов) и пред-/пост-условия (для методов). Инвариантом типа является логическое условие (скорее всего, довольно сложное), которое должно быть истинно в любой момент времени во время выполнения программы для всех экземпляров класса, указанного в контексте. При этом для одного контекста может задавать сразу несколько инвариантов. Пред- условие для какой-то функции или метода означает тоже некоторое логическое условие, которое обязательно должно выполняться до того, как выполнить данный метод. Если же это условие будет ложным перед вызовом этого метода, то считается, что ограничение не выполнено, т.е. выдается сообщение об ошибке, прерывается программа или вызывается какая-то другая обработка невыполнения ограничений. Пост- условие же для метода должно быть истинным сразу после завершения работы данного метода, иначе, как и с пред- условиями, ограничение не будет считаться выполненным.

Для большей наглядности ниже приведены конкретные примеры задания ограничений на OCL [3].

1). **context** Person **inv** :

self.wife->notEmpty() implies self.wife.age >=18 and
self.husband->notEmpty() implies self.husband.age >=18

2). **context** Адрес **inv**:

self.Населенный пункт→notEmpty() implies
(*self*.Область→notEmpty() and
self.Область.Населенный пункт→includes(self.Населенный пункт))
or
(*self*.Область→isEmpty() and
self.Страна.Населенный пункт→includes(self.Населенный пункт))

3). **context** Job

inv :self.employer.numberOfEmployees >=1
inv :self.employee.age >21

4). **context** Person **inv**:

let income : Integer = *self*.job.salary->sum()
let hasTitle(t: String) : Boolean = *self*.job->exists(title = t) **in**
if isUnemployed then income < 100
else income >= 100 and hasTitle('manager') endif

5). **context** Person::income(d :Date):Integer

pre : d.value >= *self*.job.startDate.value
post :*result* =5000

Как видно из примеров, для указания контекста применения ограничения (т.е. объекта некоторого типа или метода этого класса) используется ключевое слово context. В приведенных выше примерах контекстами являются классы Person (1, 4, 5), Адрес (2) и Job (3), что означает, что заданное после этого контекста ограничение будет действовать на все экземпляры соответствующих классов. Для обращения к методам и полям этого класса, указанного в контексте, используется ключевое слово self. В качестве самих ограничений в данных примерах задаются инварианты (1, 2, 3, 4) и пред-/пост- условия (5). Для описания инварианта используется ключевое слово inv, после которого ставится двоеточие и пишется одно логическое выражение. В качестве логических операций используются одни из стандартных

ключевых слов типа and (конъюнкция), or (дизъюнкция), implies (импликация), not (отрицание) и т.п. При этом для одного контекста можно задавать сразу несколько инвариантов, но каждый из них нужно начинать ключевым словом. В качестве логического выражения могут выступать также и конструкции с if /then/[else]/endif (4). Все части этого оператора тоже состоят из логических выражений. Для описания пред- и пост- условий для методов класса (5) используются ключевые слова pre и post соответственно, после которых тоже ставится двоеточие и указывается логическое выражение. При описании постусловия может быть еще использовано ключевое слово result, которое означает возвращаемое значение функции, указанной в контексте с полным именем (т.е. с именем класса, к которому принадлежит этот метод) и типами входных и выходных данных. Помимо этого есть возможность при задании ограничения после ключевого слова с двоеточием и до самого логического выражения описывать переменные при помощи ключевого слова let (4). Можно вводить сразу несколько переменных (но перед каждой пишется ключевое слово), а конец блока переменных и начало самого логического выражения связывается ключевым словом in. Эти переменные можно будет использовать при написании этого логического выражения.

В целом, OCL интуитивно понятен программисту, а задание логических выражений чем-то похоже на задание их на языке Паскаль. Этот язык считается одним из самых удобных формальных текстовых языков задания ограничений [5]. Но задание не самых простых ограничений может быть уже довольно сложно, при этом некоторые его конструкции могут быть неочевидны. Тем самым OCL является очень мощным, но сложным языком[5]. А главное, OCL не нагляден и является только текстовым языком задания ограничений, что противоречит концепции визуальности того же самого языка программирования UML и других визуальных языков. Именно по этим причинам этот язык не подошел для поставленной задачи.

2.3). Visual Object Constraint Language (VOCL) [1]

Недавно появился еще один формальный язык ограничений, основанный на OCL и на диаграммах кооперaций UML --- это VOCL, основным отличием от OCL которого является его визуальность. Этот язык ограничений разрабатывался именно как полный аналог OCL, но при этом более простой и наглядный. Тем самым VOCL тоже является очень мощным, позволяя выразить почти любое ограничение.

Ограничение на Visual OCL --- это контейнер, в котором записывается VOCL-выражение. Контейнер представляется в виде прямоугольника с закругленными вершинами, разделенного на три части, как показано на рис. 1. [2]

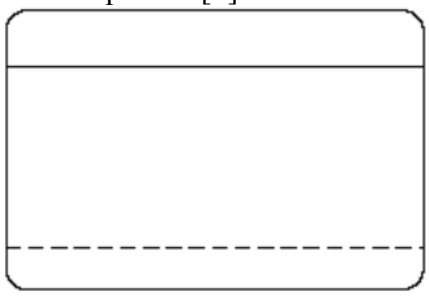

Рис. 1. Ограничение Visual OCL

При этом средняя часть значительно больше, чем остальные, и называется телом ограничения. В верхней части записывается контекст ограничения, т.е. как и в OCL тип класса или полное имя метода, на которое налагается ограничение, и тип накладываемого условия, т.е. инвариант или пред-/пост- условие. Средняя часть содержит собственно само логическое выражение требуемого ограничения, но записывающееся графически специальным образом на VOCL и называющееся выражением Visual OCL [2]. А в нижней части указываются условия на переменные, использованные в теле ограничения, если они вообще были.

Контекст записывается при помощи того же самого ключевого слова, что и в OCL, context, затем пишется идентификатор экземпляра объекта, двоеточие и тип этого объекта (имя класса) или же просто имя класса. Каждые объекты, нужные для описания логического выражения, представляются в виде прямоугольников и кладутся внутрь средней части

ограничения. В этих прямоугольниках для объектов, как и в основном прямоугольнике ограничения, записывается идентификатор объекта и его тип через двоеточие, а ниже этого, под чертой, в текстовом виде уже записывается условие, которое должно быть выполнено. При этом прямоугольник может и не иметь нижней части с чертой, если в этом нет необходимости. Для обращения к экземпляру объекта, на который пишется ограничение, при написании логического выражения можно использовать либо имя его идентификатора, либо ключевое слово self, которые пишутся внутри каждого прямоугольника (для объектов), где это надо, и после которых ставится двоеточие, а не точка, как и в OCL.

Между прямоугольниками для объектов рисуются определенные связи, в зависимости от того, что нам нужно. Логическое умножение (конъюнкция) представляется как последовательность прямоугольников, в которых записываются нужные операнды, располагающиеся друг под другом по вертикали. Логическое сложение (дизъюнкция) изображается вертикальной линией между требуемыми операндами с надписью OR. Импликация же --- сплошной горизонтальной линией с надписью implies.

Помимо просто объектов можно задавать коллекции объектов, если это требуется для логического выражения. В VOCL есть три типа коллекций : набор (set), цепочка (sequence) и сумка (bag), как показано на рис. 2

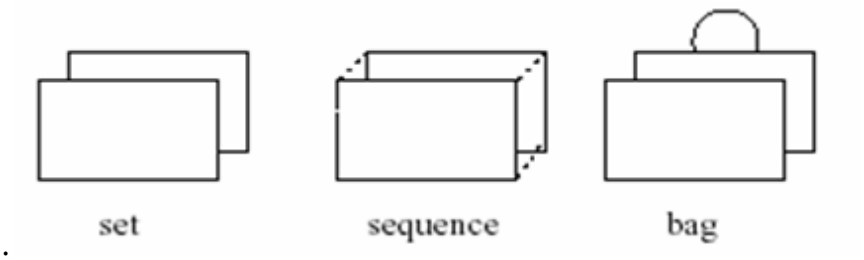

Рис. 2. Операции навигаций в VOCL

Набор предполагает некоторое множество уникальных объектов. Сумка же может содержать повторяющиеся объекты. А в последовательности тоже могут быть повторения, но при этом объекты упорядочены по некоторому принципу. Для работы с этими коллекциями есть свои операции, которые показаны в качестве примера на рис. 3 и интуитивно понятны, особенно тем, кто имел дело с SQL.

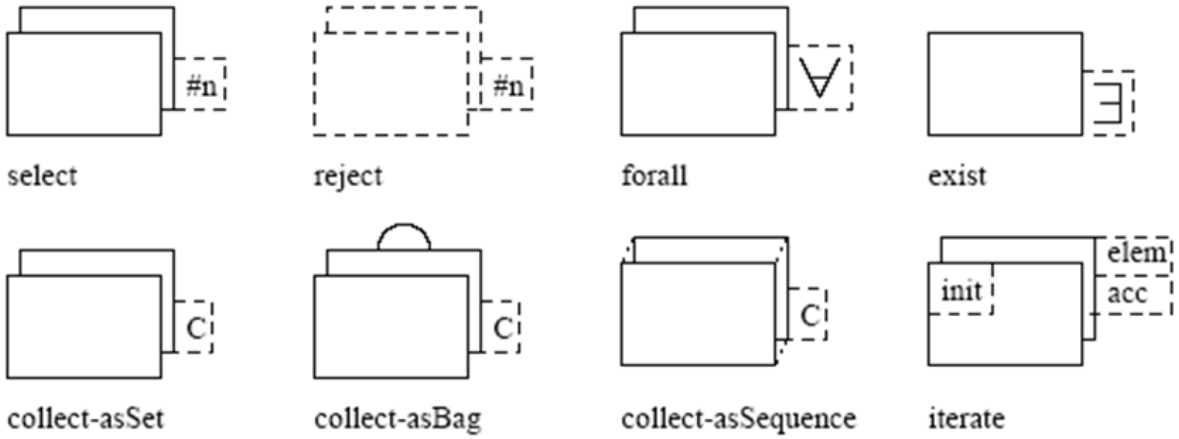

Рис. 3. Операции над коллекциями в VOCL

Далее рассмотрим некоторые примеры [2] тех же самых ограничений, что были приведены для языка OCL, но уже записанные на VOCL.

Сначала на рис. 4 приведем первый пример из примеров OCL.

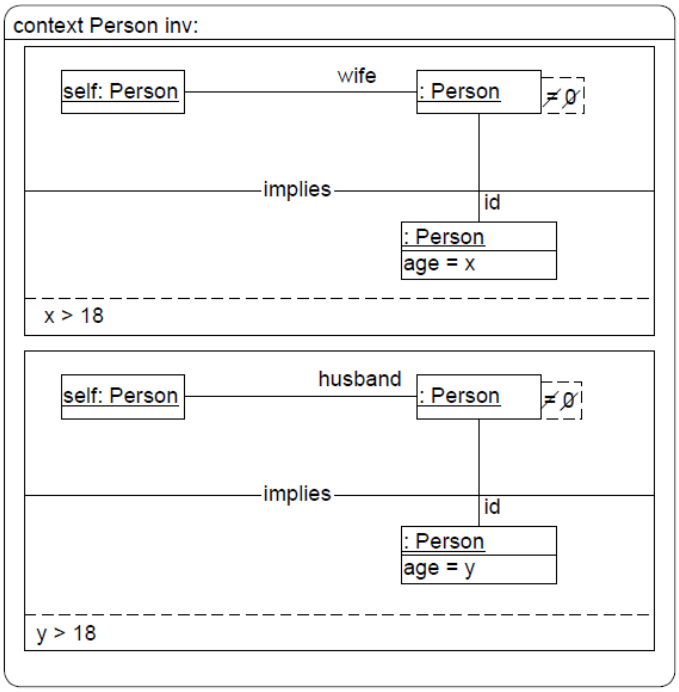

Рис. 4. Пример ограничения 1) на VOCL

В данном примере, помимо уже сказанного, можно заметить, что обращение к некоторому полю класса визуализируется связью между двумя прямоугольниками с именем этого поля над линией. Этими прямоугольниками являются справа --- объект, у которого мы вызываем требуемое поля, а слева --- объект(ы), которые мы получаем, т.е. собственно значение требуемого поля.

Далее наиболее интересен пример 4 (см. рис. 5) из ранее приведенных примеров OCL.

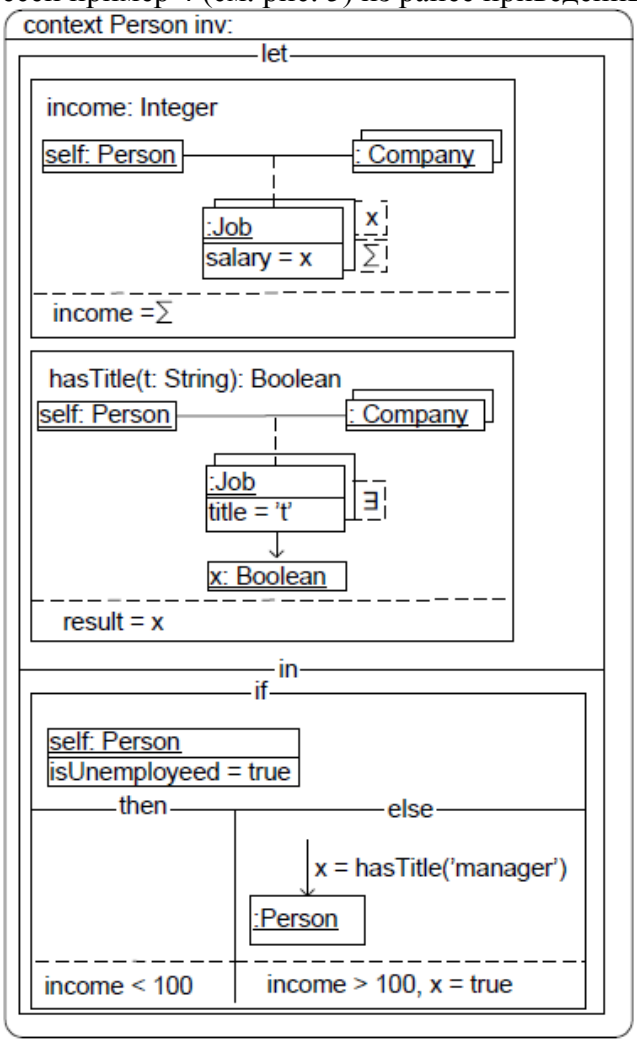

Рис. 5 Пример ограничения 4) на VOCL

В данном примере показано, как задаются отдельные переменные, которые можно будет потом использовать в самом логическом выражении. Как видно, для описания каждой переменной требуется отдельный прямоугольник, в котором записывается нужное выражение VOCL в аналогичном стиле. Также тут показана визуализация оператора if, который представляется в виде прямоугольника, разделенного на три основные части : логическое условие, следствие и альтернатива. При этом оба следствия имеют отделенную пунктиром нижнюю часть для описания условий на переменные, как и в основном прямоугольнике для ограничения. И каждая из этих частей является тоже выражением VOCL.

Для примера визуализации ограничения с пред-/пост- условиями (рис. 6) приведем аналог части примера 5) с пост- условием на VOCL.

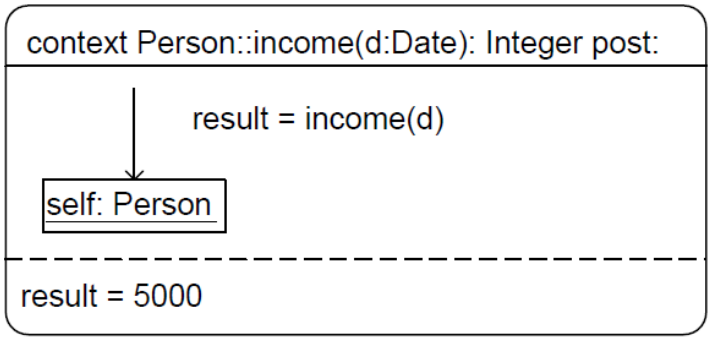

Рис. 6. Пример части ограничения 5) с пост- условием на VOCL

В данном примере обращение к методу визуализируется прямоугольником для объекта, у которого вызывается метод, с входящей стрелкой. Рядом с этой стрелкой можно запоминать результат операции в переменную, в данном случае названную result.

Итак, VOCL с одной стороны является полным аналогом OCL и тем самым достаточно полный и мощный. При этом он визуальный, в отличие от OCL, что устраняет один из главных недостатков того языка. Но с другой стороны задание ограничений на Visual OCL очень сложно, а соответствующие диаграммы с ограничениями получаются очень громоздкими, даже для несложных ограничений (см. рис. 4, рис. 5). При этом многие конструкции и способы визуализации не очень очевидны и понятны, что потребует много времени на обучение или же часто заглядывать в описание этого языка. К тому же, в большинстве случаях совершенно не нужно иметь такой мощный и сложный язык задания ограничений, а вполне достаточно и более простого языка.

3). Постановка задачи.

На кафедре системного программирования математико-механического факультета СПбГУ в течение нескольких лет разрабатывается среда визуального программирования QReal [7]. QReal является CASE-системой и, тем самым, позволяет пользователям проектировать и разрабатывать свое программное обеспечение при помощи набора визуальных моделей, которые описываются на некотором визуальном языке программирования. Каждую конкретную задачу удобнее решать на своем предметно-ориентированном языке, но каждый раз вручную создавать все необходимые редакторы диаграмм является весьма трудоемким занятием. Поэтому решено сделать QReal еще и metaCASE – системой. Т.е. в этой технологии есть возможность быстро и удобно создавать свои собственные визуальные языки, редакторы и другие инструменты поддержки для них. Для этого был разработан специальный формальный язык задания модели будущего визуального языка, т.е. метаязык, и соответствующий редактор для него для создания этой модели, т.е. метаредактор.

Таким образом в QReal тоже необходимо иметь возможность задавать ограничения на создаваемый язык. При этом для начала необходимо поддержать возможность задавать ограничения именно на сам визуальный язык, а не на состояние системы во время выполнения программы, написанной на этом языке. Сейчас в метаредакторе можно задавать только самые простые ограничения, такие как задание элементам их возможных связей и задание для контейнера типов элементов, которые он может содержать. При этом задавать часть ограничений при помощи метаязыка, а остальное при помощи специального языка было бы неправильно. Также, т.к. QReal является визуальной системой программирования, то и задавать ограничения хотелось бы тоже при помощи визуальных, наглядных моделей. Поэтому-то передо мной была поставлена задача придумать и реализовать язык ограничений.

Итак, целью данной курсовой работы было :

1) Рассмотреть различные уже существующие языки задания ограничений.
2) Придумать формальный специализированный для QReal визуальный язык задания ограничений, на котором удобно и понятно задавать ограничения, и создать соответствующий редактор для него в метаредакторе.
3) Написать генератор, который по диаграммам описания ограничений, написанным на языке ограничений, будет генерировать специальный код, реализующий эти ограничения (плагины).
4) Сделать возможность после генерации автоматически подключать плагины с ограничениями, не выходя из системы.
5) Поддержать проверку ограничений при каждом изменении диаграмм, написанных на некотором визуальном языке, на который накладываются ограничения.
6) Выдавать сообщения об ошибках, если какое-то ограничение не выполняется.

4). Описание решения.

Сначала были рассмотрены некоторые существующие языки задания ограничений, такие как самый известный и мощный подобный текстовый язык OCL и его визуальный аналог VOCL. После подробного рассмотрения этих языков было решено не использовать их в качестве языка задания ограничений для QReal: текстовый OCL не подходит ввиду недостаточной наглядности, а VOCL --- ввиду своей громоздкости. Кроме того, оба эти языка, как и другие универсальные языки задания ограничений, являются очень мощными и тем самым излишне сложными, в то время как можно обойтись более простым и приближенным к нашей задаче языку. Поэтому было решено придумать свой узкоспециализированный визуальный язык задания ограничений на создаваемый в метаредакторе визуальный язык в QReal.

Далее надо было проследить, в каких местах происходит изменение диаграммы, написанной на DSL, и вызвать в этих местах проверку ограничений. Потом сделать генератор, который по формальному описанию сгенерирует плагины с ограничениями, вызываемые каждый раз при изменении диаграммы и проверяющие ограничения.

4.1). Описание технологии

Технология QReal является metaCASE-системой, поэтому есть возможность быстро создать редактор придуманного языка ограничений. Для этого был использован метаредактор, в котором можно при помощи специального метаязыка создавать модели (диаграммы), описывающие синтаксис визуальных языков, так называемую метамодель. Суть метаязыка заключается в том, что можно описывать язык на более высоком уровне абстракции, где в качестве основных сущностей выступают «элемент» и «связь». Каждой сущности нужно задать имя и какие-то свойства. Также в метаязыке есть такой элемент, как «свойство», позволяющее добавлять то или иное свойство описываемому элементу будущего языка. При этом в метаредакторе одна сущность может наследовать свойства другой сущности. А также в метаредакторе контейнерам можно задавать, элементы какого типа они могут содержать. Благодаря этим, и некоторым другим возможностям, можно несложно описать метамодель создаваемого языка, в т.ч. языка ограничений. После того, как модель будет задана, можно сгенерировать и автоматически подгрузить редактор для этого языка, предварительно указав имя созданного языка.

Этим способом и был создан редактор языка, в котором можно задавать ограничения на некоторый визуальный язык.

4.2). Описание языка задания ограничений.

Язык задания ограничений позволяет задавать ограничения на различные визуальные языки, которые будут проверяться уже во время создания пользователем программ (диаграмм), написанных на каком-либо из этих языков. Модель языка позволяет задавать ограничения на некоторую метамодель визуального (-ых) языка (-ов). Эта модель состоит из одной или нескольких диаграмм, каждая из которых позволяет задавать ограничение ровно на один визуальный язык в рамках данной метамодели. Причем можно описывать несколько диаграмм ограничений для одного визуального языка. Диаграмма же строится из элементарных ограничений, каждое из которых позволяет задавать ровно одно ограничение на какой-либо элемент (т.е. узел или связь) или даже множество элементов визуального языка, на которые накладываем ограничения. Внутри каждого из этих элементарных ограничений указываются имя типа элемента или выборка из элементов какого-то типа, на которые мы накладываем ограничения, и сами логические условия, которые должны быть истинны для любого указанного элемента в любой момент во время работы над диаграммой, чтобы ограничение считалось выполненным. Имя типа элемента и выборка задаются в специальном текстовом виде. Логическое условие задается графическим образом при помощи специальных элементов языка ограничений.

Подробнее опишем элементы языка ограничений. В нижеприведенной таблице для каждого элемента приводится его имя (name) , отображаемое пользователю имя (displayedName) и общее описание, в т.ч. различные свойства и назначение. Заметим, что поля name и displayedName есть почти у всех элементов всех визуальных языков, которые создаются при помощи метаредактора в QReal. Значение поля displayedName элемента --- это имя элемента, которое видит пользователь при обычной работе с данным языком и используется только им же. А значение поля name --- это фактическое имя элемента, которое используется внутри самой программы и внутри редактора и всех различных инструментов поддержки данного языка. Тем самым, для того, чтобы задать ограничение на какой-либо элемент некоторого языка, надо знать и указывать его обычное name. Поэтому-то для языка ограничений тоже было решено привести оба имени.

<table>
<tr><th>№</th><th>displayedName / name</th><th colspan="3">Описание элемента</th></tr>
<tr><td>1</td><td>Metamodel Constraints / metamodelConstraints</td><td colspan="3">Корневой элемент модели задания ограничений. Позволяет задавать ограничения на некоторую метамодель визуальных языков.
Содержит поля :
• Name --- собственное имя модели ограничений
• Metamodel Name --- имя метамодели, на которую хотим наложить ограничения
В качестве этого имени может выступать ключевое слово All, которое означает, что написанные в этой модели ограничения будут проверяться для всех метамоделей.
• Output Dir Path --- абсолютный путь до папки, в которую будет генерироваться код с ограничениями
• Dir Path to QReal --- абсолютный либо относительный (отн-но Output Dir Path) путь до папки с исходниками QReal
Все эти поля обязательно должны быть заполнены.</td></tr>
<tr><td>2</td><td>Constraints Diagram / constraintsDiagram</td><td colspan="3">Основная диаграмма, на которую непосредственно будут кидаться элементы для задания ограничений. Позволяет описывать ограничения на конкретный визуальный язык из уже рассматриваемой метамодели.
Содержит поля :
• Name --- собственное имя диаграммы (задавать не обязательно)
• Language Name --- имя визуального языка, на которое накладываем ограничения (обязательно)
В качестве этого имени тут тоже может выступать ключевое слово All, которое означает, что написанные на этой диаграмме ограничения будут проверяться для всех языков рассматриваемой метамодели.</td></tr>
<tr><td>3</td><td>Node Constraint / nodeConstraint</td><td rowspan="2">Основные элементы задания ограничений.
Содержат поля :
• Error Type --- тип ошибки, который зависит от важности ограничения. Это нужно для корректного информирования об ошибке, если данное ограничение не будет</td><td rowspan="2">для одного типа соответствующего элемента рассматриваемого визуального языка.
Поля :
• Name --- имя типа элемента, на которое накладывается ограничение</td><td>для узла.</td></tr>
<tr><td>4</td><td>Edge Constraint / edgeConstraint</td><td>для связи.</td></tr>
</table>

5	Nodes Constraint / nodesConstraint	выполнено. Есть три вида ошибок : ➢ warning (по умолчанию; элемент, для которого не выполняется ограничение, будет подсвечен красным цветом) ➢ critical (аналогично, но ко всему прочему выдается текстовое сообщение) ➢ verification (поведение этого типа ошибки будет либо warning, либо critical в зависимости настроек во время написания диаграмм на рассматриваемом визуальном языке) • **Text of error** --- текст выдаваемого сообщения об ошибке при невыполнении ограничения. Причем этот текст будет выдаваться только в случае соответствующего типа ошибки.	для некоторого подмножества элементов визуального языка одного и того же соответствующего метатипа (узел / связь). Поля : • **Name** --- имя некоторого типа элемента, на подмножество которого накладывается ограничение. Также есть возможность в качестве значения этого поля записать ключевое слово All (или AllNodes / AllEdges), что означает, что ограничение задается на все элементы соответствующего метатипа. • **Selection** --- «выборка» из элементов, принадлежащих типу, указанному в Name. Это обычное текстовое выражение вида : «свойство элемента» «знак сравнения» «значение свойства». Тут так же можно написать ключевое слово All	для узлов
6	Edges Constraint / edgesConstraint			для связей
7	Property / propertyNode	Элемент языка ограничений для задания условия на некоторое свойство элемента, на которое накладывается ограничение. Поля : • Property --- имя некоторого свойства элемента, на которое накладывается ограничение. • Sign --- знак сравнения (т.е. “>”, “ <”, “ >=”, “<=”, “!=”, “==”) • Value --- значение свойства, с которым хотим сравнивать. Отметим, что Value будет иметь тип «число», если введенное пользователем похоже на число, иначе тип «строка». Это верно и для «выборок» для всех остальных элементов языка ограничений.		
8	Begin Node / beginNode	Для задания ограничений на начальный узел рассматриваемой связи.	Содержат поля : • **Name** ---- имя нашего элемента ограничения. (не обязательно;	

9	End Node / endNode	Для задания ограничений на конечный узел рассматриваемой связи.	нужно только для наглядности) • **Exists** --- логическое условие на необходимость существования рассматриваемого элемента. Возможны три варианта : ➢ True --- обязательное существование элемента ➢ False --- обязательное отсутствие этого элемента ➢ Doesn't matter --- не имеет значения его статус существования.
10	Parent / parent	Для задания ограничений на «родителя» (в смысле контейнера) рассматриваемого узла.	
11	Childrens / childrens	Элемент для задания ограничений на подмножество всех «детей» (в смысле контейнеров) рассматриваемого узла.	Содержат поля : • **Name** --- Здесь можно либо оставить строчку по умолчанию, либо записать ключевое слово All. (не обязательное поле, нужно только для наглядности). • **Selection** --- «выборка» из соответсвующих элементов. Это так же обычное текстовое выражение вида : «свойство элемента» «знак сравнения» «значение свойства». Если выборка пуста, то ограничение проверяется для всех рассматриваемых элементов. • **Count** --- логическое условие на количество рассматриваемых элементов. Задается в текстовом виде : «знак сравнения» «число». Если поле пустое, то это условие на элементы и не проверяется.
12	Incoming Links / incomingLinks	Элемент для задания ограничений на подмножество входящих в рассматриваемый узел связей.	
13	Outgoing Links / outgoingLinks	Элемент для задания ограничений на подмножество выходящих из рассматриваемого узла связей.	
14	Incoming Nodes / incomingNodes	Элемент для задания ограничений на подмножество входящих в рассматриваемый узел узлов, т.е. узлов на концах входящих связей.	
15	Outgoing Nodes/ outgoingNodes	Элемент для задания ограничений на подмножество выходящих из рассматриваемого узла узлов, т.е. узлов на концах выходящих связей.	

Тем самым, очевидно, что элементами языка ограничений, задающими ограничения на узлы, являются : nodeConstraint, nodesConstraint, parent, childrens, incomingNodes, outgoingNodes, beginNode, endNode. А элементами задания ограничений на связи являются : edgeConstraint, edgesConstraint, incomingLinks, incomingNodes.

Заметим также, что все элементы языка ограничений, кроме PropertyNode, являются контейнерами, т.е. есть возможность внутрь этих элементов кидать другие элементы. При этом есть ограничения на то, какие типы элементов могут быть внутри контейнеров. Ниже приведена таблица взаимоотношений элементов в отношении, какие элементы могут быть внутри каких контейнеров :

№	Контейнер	Типы элементов, которые может содержать данный контейнер
1	metamodelConstraints	constraintsDiagram
2	constraintsDiagram	Все элементы языка (для удобства пользователя)

3	Любой элемент, задающий ограничения на узлы	propertyNode parent childrens incomingLinks outgoingLinks incomingNodes outgoingNodes
4	Любой элемент, задающий ограничения на связи	propertyNode beginNode endNode

Теперь пользователь сможет в соответствующем редакторе для языка ограничений описывать различные ограничения на визуальные языки, записывая их формальным образом на вышеописанном языке ограничений. Для этого требуется создать элемент типа Metamodel Constraints, как корневой элемент искомой модели ограничений, и добавить ему в качестве детей элементы типа Constraints Diagram, которые и являются рабочими диаграммами, где пользователь может рисовать правила требуемых ограничений. При этом каждая такая диаграмма отвечает только за один визуальный язык, но для одного и того же языка может быть несколько диаграмм.

Приведем небольшой пример ограничения на нашем языке ограничений (рис.8).

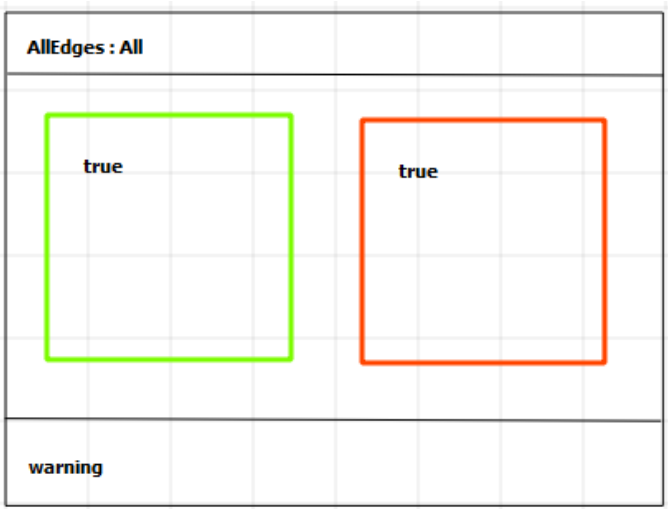

Рис. 8. Пример ограничения на нашем языке ограничений

Это ограничение проверяется для всех визуальных языков всех метамоделей и накладывается на все элементы типа «связь». Оно считается выполненным, если на обоих концах связи есть узлы. Иначе, ограничение считается невыполненным и нарушившая ограничение связь подсвечивается красным цветом (рис. 9).

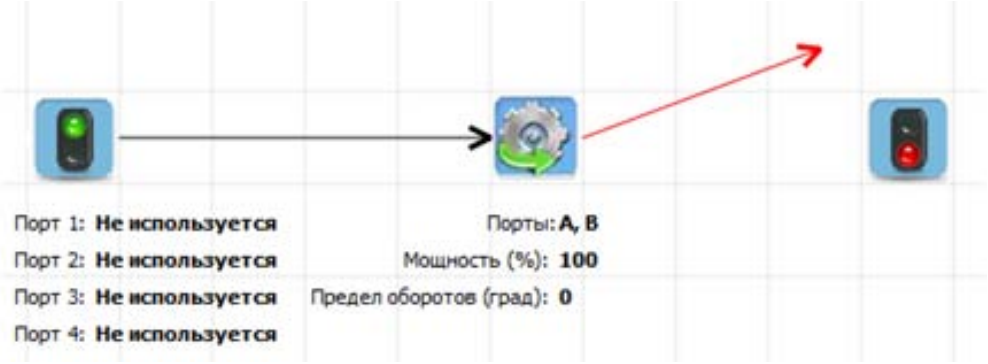

Рис. 9. Пример подсветки связи, нарушившей ограничении из рис.8.

4.3). Генерация ограничений.

После того, как задана модель ограничений для некоторой метамодели (или для всех метамоделей), есть возможность сгенерировать по этому описанию код, проверяющий эти ограничения. Генерация происходит по логической модели нашей модели ограничений, поэтому предварительно необходимо согласовать графическую и логическую модели в соответствующих обозревателях, располагающихся слева от рабочей области рисования диаграмм. Более подробно о создании моделей и диаграмм в QReal написано в тематических статьях об этой системе программирования, например [8].

При генерации в папке, указанной в «Output Dir Path», одном из свойств модели ограничений, создаётся папка constraints, в которую при проходе по логической модели ограничений будут генерироваться соответствующие файлы с кодом. Этот код можно собрать, тем самым получив плагин ограничений (.dll) в папке qreal/bin/plugins.

Для каждой модели ограничений генерируются файлы с классом ConstraintsPlugin, который удовлетворяет общему интерфейсу плагинов ограничений ConstraintsPluginInterface. По этому интерфейсу каждый плагин может вернуть:

1) имя метамодели, для которой были написаны и сгенерированы ограничения, т.е. то же самое значение, что и у поля Metamodel Name элемента Metamodel Constraints
2) уникальный идентификатор, который по сути является просто собственным именем модели ограничений, т.е. значение поля Name элемента Metamodel Constraints
3) результат проверки ограничений для некоторого элемента по его id. Результатом этой проверки является список ошибок, в зависимости от того, каким ограничениям не удовлетворяет элемент, или же пустой список, если элемент удовлетворяет всем ограничениям.

Для каждого элемента типа Constraints Diagram тоже генерируются свои файлы с соответствующим классом, который может возвращать имя языка, для которого описаны ограничения на данной диаграмме , и так же результат проверки элемента на ограничения. Эту проверку и вызывает ConstraintsPlugin, проходя по всем своим диаграммам.

Для каждого элемента типов nodeConstraint, nodesConstraint, edgeConstraint или edgesConstraint генерируется своя аналогичная функция проверки элемента в соответствующем файле для диаграммы. Именно эту проверку для всех своих элементов и вызывает функция проверки диаграммы.

Остальные же элементы генерируются как некоторый код проверки своего ограничения внутри функции проверок соответствующих основных элементов задания ограничений (типов 3 - 6).

После того, как генерация кода проверки ограничений будет успешно завершена, пользователю будет предложено автоматически собрать этот код и подключить собранный плагин ограничений, не выходя из системы.

4.4). Механизм проверки ограничений.

Во время создания диаграмм на некотором визуальном языке, для которого есть ограничения, пользователю удобнее сразу знать, если какое-то ограничение на язык не выполняется. Это позволит пользователю сразу же исправить допущенную им ошибку или же, наоборот, быть уверенным, что описываемая им диаграмма семантически корректна. Для этого был так же реализован механизм проверки ограничений.

В первую очередь надо во время работы пользователя с некоторой диаграммой отслеживать те места в коде нашей системы, где происходят определенные её изменения, например :

1) при изменении имени элемента в логической/графической моделях
2) при изменении любого свойства элемента
3) при изменении родственных отношений в смысле контейнеров в логической модели
4) при удалении элементов из логической модели
5) при подключении связи в другое место

6) при создании нового элемента в логической модели

Во всех этих местах посылаются сигналы, что тот или иной элемент или же группа элементов были изменены. Их в свою очередь ловит класс MainWindow. Этот класс является основным классом системы QReal, который знает почти обо всём и управляет общей координацией между компонентам. В частности, он является связующим звеном и для механизма проверок ограничений.

Получив сигнал, что какой-то элемент был изменен, MainWindow вызывает сначала функции своей проверки ограничений:

1) для самого элемента (для узлов и связей)
2) для всех его детей и всех его родителей в смысле контейнеров и для всех связей, связанных с этим элементом (только для узлов)

При вызове функции проверки ограничений самого элемента MainWindow вызывает метод проверки этого элемента у ConstraintsManager, который вернёт список состояний для каждого проверенного ограничения, выполнено оно или нет. ConstraintsManager в свою очередь обойдет все подгруженные плагины ограничений и вызовет соответствующий метод проверки ограничений для элемента. Далее MainWindow проходит по этому списку и в случае, если какое-то ограничение не выполняется, информирует об этом пользователя. Если тип ограничения был warning, то элемент, нарушивший ограничение подсвечивается красным цветом. Если же тип --- critical, то, помимо подсветки, в специальном для ошибок окне появляется текст сообщения о том, что ограничение не выполнено (этот текст задавался при создании правила ограничения).

Общая схема данного механизма проверок представлена на рис. 10.

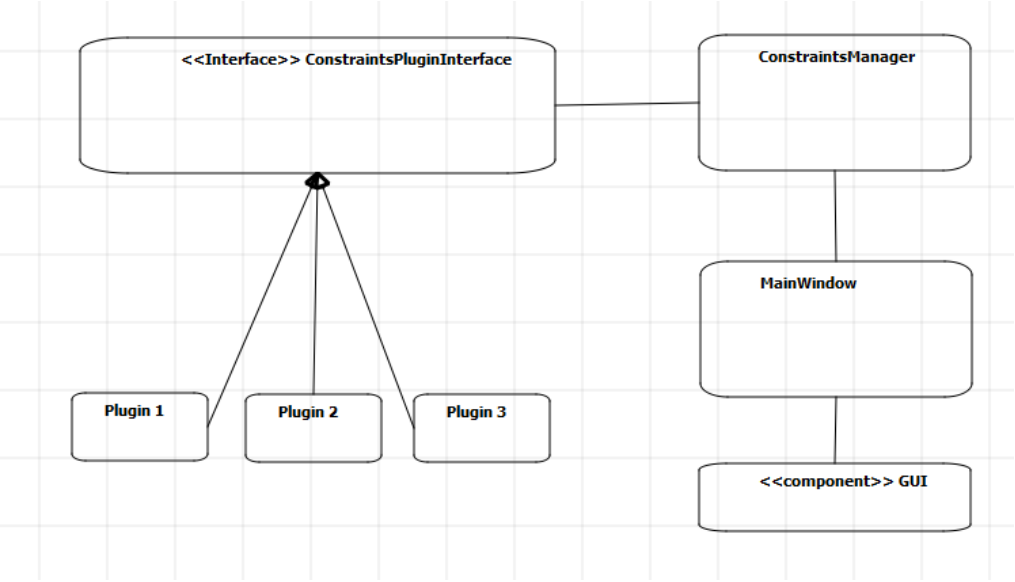

Рис. 10. Общая идея механизма проверки ограничений.

5). Апробация.

В качестве применения были реализованы следующие ограничения :

1) Для всех визуальных языков любой элемент метатипа «связь» должен иметь узлы на обоих концах (см. рис. 8, рис. 9)
2) Ограничения для языка для роботов [9] :
   a) перед блоком «моторы стоп» не может следовать блок «моторы вперёд» или «моторы назад» (рис. 11, рис. 12)

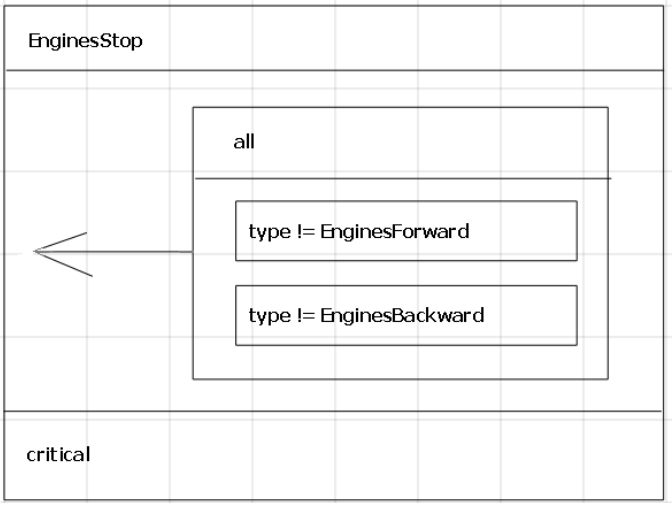

Рис. 11. Задание ограничения 2.а)

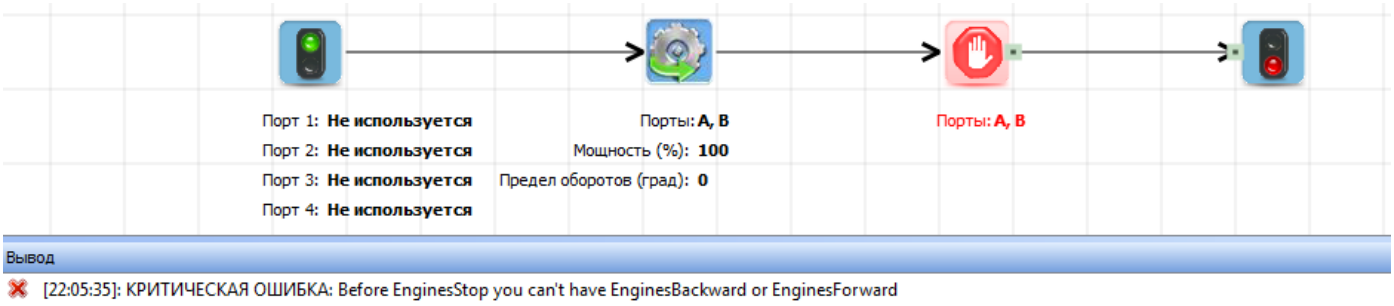

Рис. 12. Пример невыполнения ограничения 2.а)

На рис. 12 видно, что за блоком «моторы вперед» сразу же стоит блок «моторы стоп», что противоречит нашему ограничению 2.а). Поэтому блок «моторы стоп» подсветился красным цветом, а в специальном окне ниже появился текст ошибки, сообщающий пользователю о том, что данное ограничение не выполнено.

b) на диаграмме роботов не должно быть более одного блока инициализации (рис. 13, рис. 14)

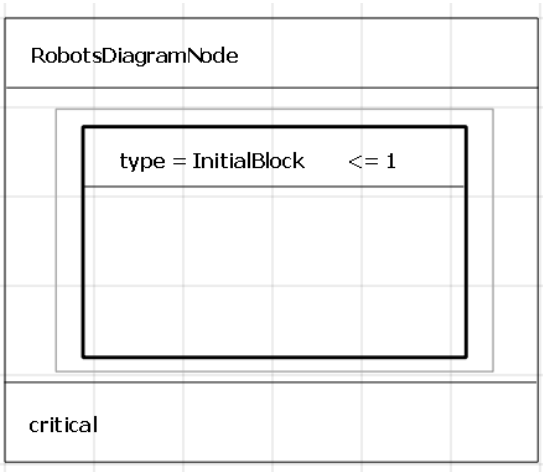

Рис. 13. Задание ограничения 2.б)

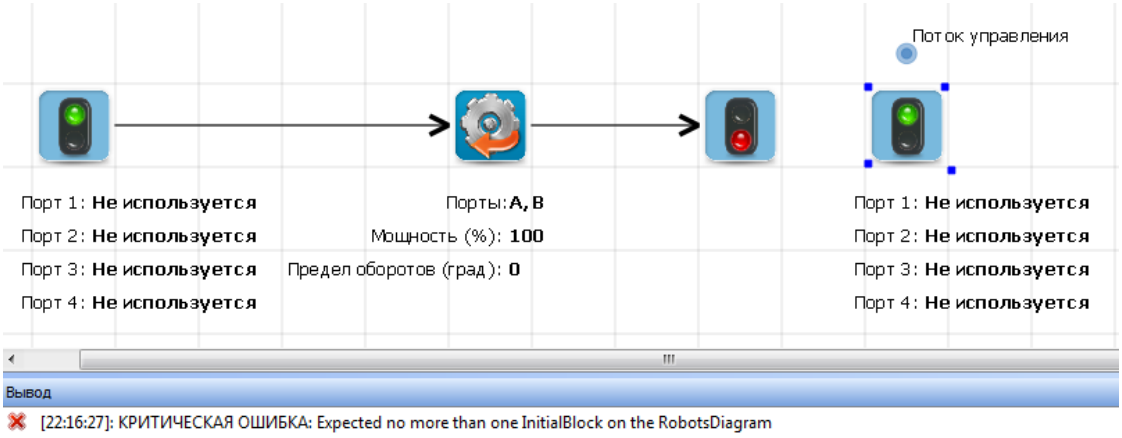

Рис. 14. Пример невыполнения ограничения 2.б)

На рис. 14 при нарушении ограничения появился текст об ошибке, но ничего не подсвечено красным, т.к. элементом, нарушившим данное ограничение, является сама диаграмма.

c) в блоках «моторы вперёд» и «моторы назад» свойство «мощность» должно быть не больше 100, но не меньше -100 (рис. 15, рис. 16)

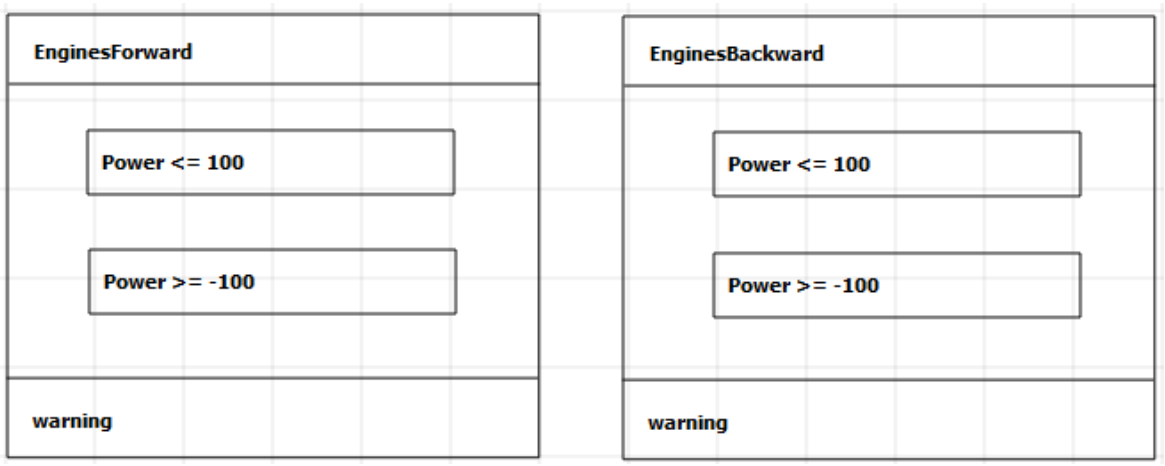

Рис. 15. Задание ограничения 2.в)

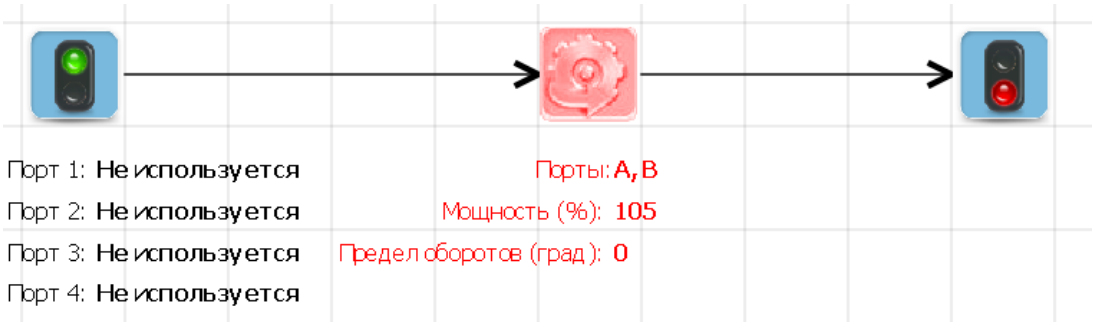

Рис. 16. Пример невыполнения ограничения 2.в)

На рис. 16 при нарушении ограничения никаких ошибок не выдается, но сам блок «моторы вперёд», неудовлетворяющий 2.в), подсвечивается красным цветом, так как тип данного ограничения warning, а не critical.

6). Заключение.

Основным итогом данной курсовой работы стало поддержка в среде программирования QReal возможности быстро и удобно задавать ограничения на визуальные языки, который проверяются во время написания программ на этих языках. Тем самым, получены следующие результаты:

1) изучены одни из самых широко применимых языков задания ограничений, такие как OCL и VOCL.
2) разработан формальный визуальный язык задания ограничений и создан соответствующий ему редактор
3) поддержан механизм проверки ограничений во время создания диаграмм на некотором визуальном языке
4) реализована выдача пользователю сообщений, в случае если некоторое ограничение не выполнено
5) придуман и реализован генератор ограничений, генерирующий по формальному описанию модели ограничений код в виде плагина, проверяющий эти ограничения
6) поддержана возможность автоматически подгружать плагины ограничений после их генерации, не выходя из системы

7). Список литературы :

1) Language elements with examples, Visual OCL, URL: http://tfs.cs.tu-berlin.de/vocl/language/examples.htm (дата обращения: 3.03.2012)
2) Christiane Kiesner, Gabriele Taentzer, Jessica Winkelmann, Visual OCL: A Visual Notationof the Object Constraint Language, URL: http://user.cs.tu-berlin.de/~gabi/gKTW02.pdf (дата обращения: 15.04.2012)
3) Object Constraint Language; The Modelling, Simulation and Design lab (MSDL); URL : http://msdl.cs.mcgill.ca/presentations/02.06.07.OCL/presentation.html (дата обращения: 18.04.2012)
4) Edward Willink, Modeling the OCL Standard Library // Proceedings of the workshop on OCL and Textual Modelling (OCL 2011) // Electronic Communications of the EASST.2011, с. 100 – 120 ; URL : http://journal.ub.tu-berlin.de/eceasst/article/view/663/673 (дата обращения: 23.05.2012)
5) Ограничения целостности и язык OCL, Интернет университет информационных технологий, URL : http://www.intuit.ru/department/database/rdbintro/10/4.html (дата обращения: 12.03.2012)
6) Жолудев В. В.“Генерация контекстных ограничений для баз данных”, Дипломная работа, 2007.
7) 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
8) Кузенкова А.С., Дерипаска А.О., Литвинов Ю.В., Поддержка метамоделирования в среде визуального программирования QReal // Материалы межвузовского конкурса-конференции студентов, аспирантов и молодых ученых Северо-Запада "Технологии Microsoft в теории и практике программирования". Спб.: Изд-во СПбГПУ, 2011. С. 100-101
9) Брыксин Т.А., Литвинов Ю.В., Среда визуального программирования роботов QReal:Robots // Материалы международной конференции "Информационные технологии в образовании и науке". Самара. 2011. С. 332-334
10) Edward Willink, Aligning OCL with UML// Proceedings of the workshop on OCL and Textual Modelling (OCL 2011) // Electronic Communications of the EASST.2011, с. 121 – 141 ; URL : http://journal.ub.tu-berlin.de/eceasst/article/view/664/674 (дата обращения: 23.05.2012)