Дерипаска Анна Олеговна

УДК 004.4'22

ЯЗЫКИ ЗАДАНИЯ ОГРАНИЧЕНИЙ

Аннотация

В статье рассматриваются способы задания ограничений на визуальные языки моделирования. Приводится обзор некоторых существующих языков задания ограничений, таких как широко применимый текстовый язык Object Constraints Language и его визуальный аналог Visual OCL, а также более узкоспециализированных языков ограничений. Предлагается новый визуальный язык задания ограничений на модели визуальных языков, разработанный в среде программирования QReal. Описывается апробация предложенного языка применительно к визуальному языку программирования роботов QReal::Robots. Делаются выводы об актуальности предложенного нового языка.

Ключевые слова: визуальное программирование, семантическая корректность программ, задание ограничений, язык ограничений.

ВВЕДЕНИЕ

Разработка сложных программных систем является трудоёмким процессом. Одним из способов упрощения этого процесса является визуальное программирование, в рамках которого программы записываются с помощью набора визуальных моделей (а не на текстовом языке программирования), что гораздо более наглядно.

Для того чтобы можно было создавать такие визуальные модели, были разработаны CASE-системы, которые содержат в себе редакторы одного или нескольких визуальных языков, средства генерации кода по диаграммам на этих языках и другие инструменты разработки программного обеспечения (ПО). При этом универсальные CASE-системы в качестве реализуемого ими языка, как правило, имеют визуальный язык общего назначения, который является довольно мощным и поэтому 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 и увеличивает производительность труда программистов. Но создание CASE-систем, в свою очередь, может быть довольно сложной задачей, так как создание графического редактора визуального языка вручную трудоёмко. Поэтому может быть не оправдано каждый раз при разработке программной системы сначала создавать свою узкоспециализированную CASE-систему, а потом собственно решать требуемую задачу.

Как следствие, появились metaCASE-системы как способ автоматизировать процесс создания своей CASE-системы. Они позволяют достаточно легко и быстро создавать предметно-ориентированные языки и соответствующие инструменты их поддержки. Программист должен сначала создать метамодель своего языка, то есть описать синтаксис своего визуального языка на специальном формальном визуальном языке (метаязыке), предназначенном для этого, в специальном редакторе (называемом метаредактором). Далее по этому описанию автоматически генерируется код, реализующий редактор для определяемого языка. Метамодель представляет собой описание языка на уровне абстракций предметной области и позволяет удобно задать синтаксис визуального языка и некоторые семантические особенности.

При этом, создавая свой визуальный язык для решения определенной задачи,

нужно минимизировать возможность написания некорректных программ. Для этого требуется не только синтаксическая корректность будущих программ, что вполне гарантируется заданной метамоделью языка, но и семантическая корректность, которую в полной мере нельзя описать в метамодели. Поэтому появилась идея задать какие-либо правила семантики создаваемого языка, например ограничения, то есть некоторые логические условия, выполнение которых в той или иной степени гарантирует корректность программы. Существует два типа основных ограничений. Во-первых, это ограничения на состояния системы во время выполнения программы, написанной на данном языке. Во-вторых, ограничения на сам язык, то есть 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 (то есть на сам язык) задаются на том же уровне, что и описываемый визуальный язык, то есть на уровне метамодели, в то время как ограничения первого типа (то есть на состояние системы) должны задаваться уже при написании конкретной программы. При этом очевидно, что для описания ограничений обоих типов нужен специальный язык их задания, так называемый язык задания ограничений.

ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

Существуют два основных способа задания ограничений: на естественном языке и на формальном языке. Утверждения и условия, записанные на естественном языке, то есть неформально, 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 некоторые языки задания ограничений, в том числе визуальные языки.

1. ЯЗЫК ОПИСАНИЯ КОНТЕКСТНЫХ ОГРАНИЧЕНИЙ ССЫЛОЧНОЙ ЦЕЛОСТНОСТИ

1.1. Технология Real-IT

На кафедре системного программирования математико-механического факультета СПбГУ была разработана оригинальная технология Real-IT [7], которая основана на использовании универсального объектно-ориентированного CASE-пакета Real и позволяет разрабатывать информационные системы (ИС), ориентированные на данные. Данная технология представляет собой набор различных методик и инструментальных средств, которые предназначены для решения конкретных задач в рамках разработки рассматриваемых информационных систем. В частности, Real-IT включает в себя средства визуального моделирования, набор генераторов, позволяющих генерировать программный код по формальному описанию системы и набор специальных средств, помогающих поддерживать итеративный характер процесса разработки [7].

Разработка ИС в среде программирования Real-IT состоит из трёх основных этапов:

1) визуальное моделирование баз данных и пользовательского интерфейса,

2) автоматическая генерация приложения по диаграммам,

3) корректирование «вручную» сгенерированного кода программистами при необходимости (например в случае наличия нетривиальной логики).

Однако весь процесс разработки начинается именно с моделирования схем баз данных, для этого используется модель классов Real. Модель классов Real основана на модели классов UML, но в неё добавлены некоторые специальные конструкции, полезные для разработки ИС. В частности, для моделирования систем реального времени были добавлены таймеры, сообщения, порты и двунаправленные интерфейсы, а для моделирования баз данных были добавлены индексы и представления. При этом основные термины модели классов UML, такие как классы, атрибуты и ассоциации, переносятся без изменений в модель классов Real.

1.2. Описание контекстных ограничений целостности

Модель классов, которая, в частности, используется и в Real-IT, позволяет определять множества наборов объектов и связей между ними. При этом модель классов в качестве связи допускает любую пару экземпляров двух классов, если между ними существует ассоциация, то есть формально связи являются независимыми друг от друга. Однако в рамках предметной области могут быть определённые ограничения на возможные связи между объектами, которые при помощи модели классов невозможно выразить. По этой причине для технологии Real-IT был разработан специальный визуальный язык ограничений ссылочной целостности, позволяющий описывать подобные ограничения [6, 7].

Разработанный для Real-IT язык ограничений основан на диаграммах кооперации UML, что позволяет использовать этот язык и в других CASE-пакетах, поддерживающих UML. Однако этот язык позволяет описывать ограничения только определённого вида, а именно контекстные ограничения, что делает его уже не универсальным языком задания ограничений. Отметим, что диаграммы, описывающие ограничения на этом языке, по виду полностью повторяют структуру того фрагмента исходной диаграммы классов, на который накладывается это ограничение. А так как диаграммы с ограничениями похожи на соответствующие диаграммы классов, то такое описание ограничений становится «узнаваемым» и более наглядным для программистов при разработке ИС.

Следует пояснить, что такое контекстные ограничения. Это понятие было введено А.Н. Ивановым в своей диссертации [7]. 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 контекст существует. И если для ассоциации можно построить контекст, то она называется допустимой ассоциацией. Тогда контекстными ограничениями ссылочной целостности называются соответствующие ограничения на допустимые ассоциации, то есть ограничения, которые запрещают недопустимые ссылки. Если же у ассоциации не существует контекста, то для неё нет смысла вводить понятие контекстных ограничений, а следовательно, такие ассоциации не будут рассматриваться. При этом видно, что понятие контекстных ограничений совпадает с понятием традиционных ограничений ссылочной целостности с точностью до определения допустимости, для определения которого в данном случае используется контекст.

1.3. Нотация

Теперь следует описать, что представляет собой диаграмма ограничений на языке контекстных ограничений ссылочной целостности, разработанном в рамках рассматриваемой технологии Real-IT. Как было сказано ранее, данный язык ограничений основан на диаграммах кооперации языка UML, но при этом используются не все типы элементов диаграммы кооперации, а именно в рассматриваемом языке есть объекты и связи, но нет сообщений. Объектами на диаграмме ограничений являются объекты из модели классов (с указанием их класса), на которые накладывается ограничение. При этом одному объекту из диаграммы классов может соответствовать несколько объектов на диаграмме ограничений. Связями же между объектами на диаграмме ограничений являются соответствующие связи из диаграммы классов. При этом для связей вводятся стереотипы «Limited» для обозначения связи, на которую накладывается ограничение, и «Absent» для описания отрицаний. Отметим, что диаграмма кооперации будет считаться диаграммой ограничений только в том случае, если на ней будет присутствовать хотя бы одна связь со стереотипом «Limited», при этом для рассматриваемого языка ограничений требуется, чтобы такая связь была ровно одна.

В качестве примера можем рассмотреть некоторые контекстные ограничения, накла-

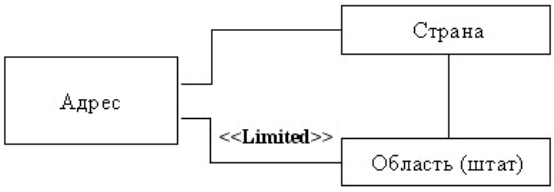

Рис. 1.1. Диаграмма ограничений для ограничения 1)

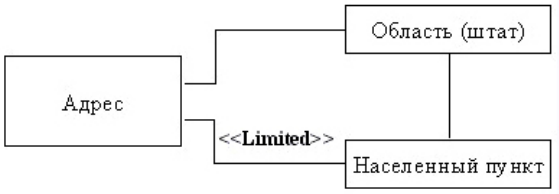

Рис. 1.2. Первая диаграмма ограничений для ограничения 2)

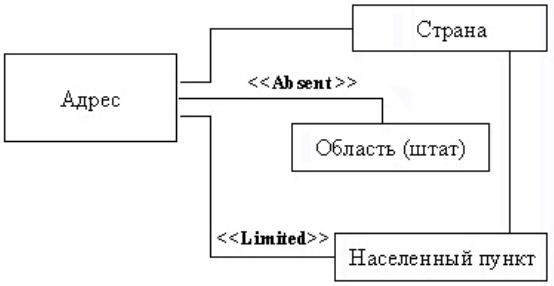

Рис. 1.3. Вторая диаграмма ограничений для ограничения 2)

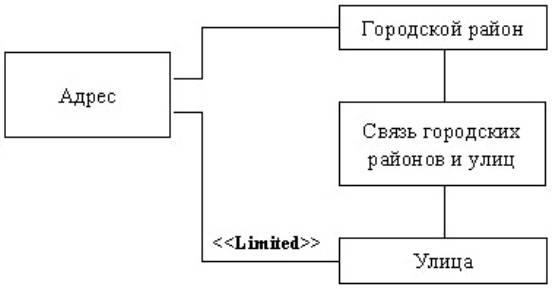

Рис. 1.4. Первая диаграмма ограничений для ограничений 3), 4)

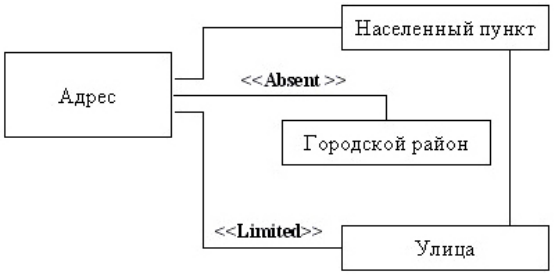

Рис. 1.5. Вторая диаграмма ограничений для ограничений 3), 4)

дываемые на «адрес» в некоторой схеме базы данных:

1) если у «адреса» указана «область», то она должна принадлежать той же «стране», что и «адрес»;

2) если у «адреса» указан «населённый пункт», то он должен принадлежать той же «области», что и «адрес», если же «область» не указана, то «населённый пункт» должен принадлежать той же «стране», что и «адрес»;

3) если указаны «улица» и «район», то они должны быть связаны отношением «связь городских районов и улиц»;

4) если указана «улица», то она должна принадлежать «населённому пункту».

На рис. 1.1–1.5 приведены несколько примеров задания таких ограничений на рассматриваемом языке.

1.4. Выводы

Рассмотренный язык задания ограничений, разработанный в рамках Real-IT, позволяет задавать контекстные ограничения на данные в наглядной форме при помощи некоторого подобия диаграмм кооперации UML. При этом отметим, что задание ограничений на данные при разработке информационных систем используется, во первых, на уровне баз данных и , во-вторых, на уровне приложений. Первое используется для того, чтобы запретить некорректное изменение данных, и реализуется в триггерах баз данных. Второе же используется для того, чтобы разрешить пользователям вводить только корректные данные, то есть данные, удовлетворяющие ограничениям. Для технологии Real-IT были реализованы оба этих способа использования [6, 7]. В частности, в первом случае ограничения, описанные на рассматриваемой диаграмме ограничений, представляются при помощи языка XMI (XML Metadata Interchange, унифицированное представление UML в виде XML), а затем по этим входным данным генерируются контекстные ограничения в виде триггеров баз данных.

2. OBJECT CONSTRAINT LANGUAGE (OCL)

Одним из самых распространенных формальных языков задания ограничений явля-

ется объектный язык ограничений OCL (Object Constraint Language) [4], который разрабатывался как встроенный механизм задания ограничений в самом популярном общецелевом визуальном языке UML. OCL является текстовым языком для описания дополнительных условий и ограничений [11]. 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 есть возможность описывать различные инварианты, задавать пред/постусловия на методы и операции и т. д. Задание ограничения на языке OCL представляет собой описание контекста, то есть указание типа некоторого объекта или его метода, на который мы хотим наложить ограничение, и выражения, обозначающего собственно логику требуемого ограничения. Логическим выражением чаще всего являются инварианты (для классов) и пред/постусловия (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. Предусловие для какой-то функции или метода означает 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, то есть выдается сообщение об ошибке, программа прерывается или вызывается какая-то другая обработка невыполнения ограничений. Постусловие же для метода должно быть истинным сразу после завершения работы данного метода, иначе, как и с предусловиями, ограничение не будет считаться выполненным.

В листинге 1 приведены примеры задания ограничений на OCL [3].

Как видно из примеров, для указания контекста применения ограничения (то есть объекта некоторого типа или метода этого класса) используется ключевое слово `context`. В приведенных выше примерах контекстами являются классы `Person` (1, 4,

Листинг 1

```
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
```

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/постусловия (5). Для описания инварианта используется ключевое слово inv, после которого ставится двоеточие и пишется одно логическое выражение. В качестве логических операций используются стандартные ключевые слова 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/постусловий 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 использовано ключевое слово result, которое означает возвращаемое значение функции, указанной в контексте с полным именем (то есть 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 логического выражения.

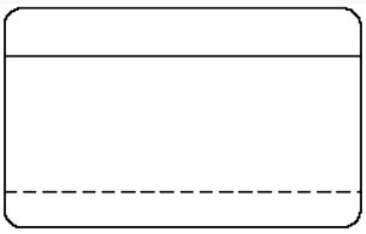

Рис. 2. Ограничение Visual OCL

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.

3. VISUAL OBJECT CONSTRAINT LANGUAGE (VOCL)

Недавно появился еще один формальный язык ограничений, основанный на OCL и на диаграммах кооперации UML, – это VOCL [1], основным отличием которого от OCL 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. 2 [2].

При этом средняя часть значительно больше, чем остальные, и называется телом ограничения. В верхней части записывается контекст ограничения, то есть, как и в OCL, тип класса или полное имя метода, на которое налагается ограничение, и тип накладываемого условия, то есть инвариант или пред/постусловие. 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. Все 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, как и в OCL, ставится двоеточие, а не точка.

Между прямоугольниками для объектов рисуются определенные связи, в зависимости от требуемой операции. Логическое умножение (конъюнкция) представляется как последовательность прямоугольников, в которых записываются нужные операнды,

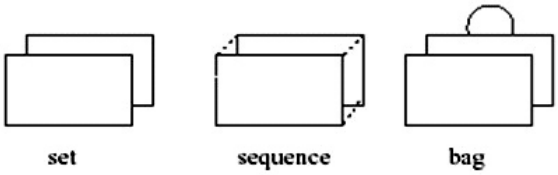

Рис. 3. Операции навигаций в VOCL

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. 3.

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. 4 и интуитивно понятны, особенно тем, кто имел дело с SQL.

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

Сначала на рис. 5 приведем первый пример из примеров OCL.

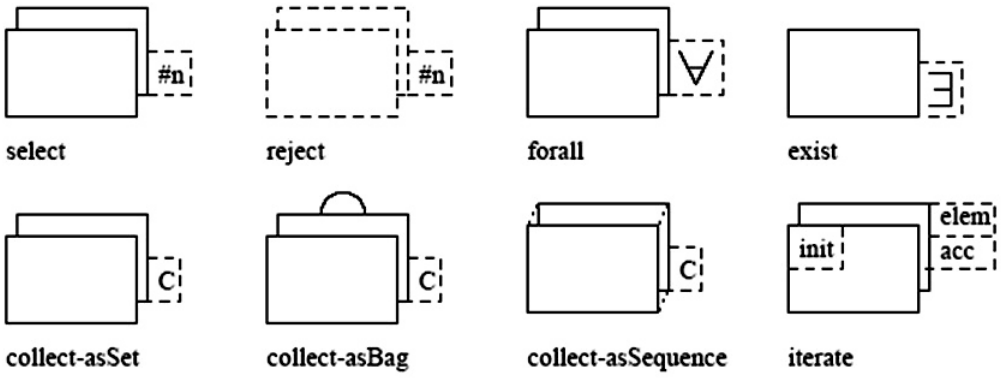

Рис. 4. Операции над коллекциями в VOCL

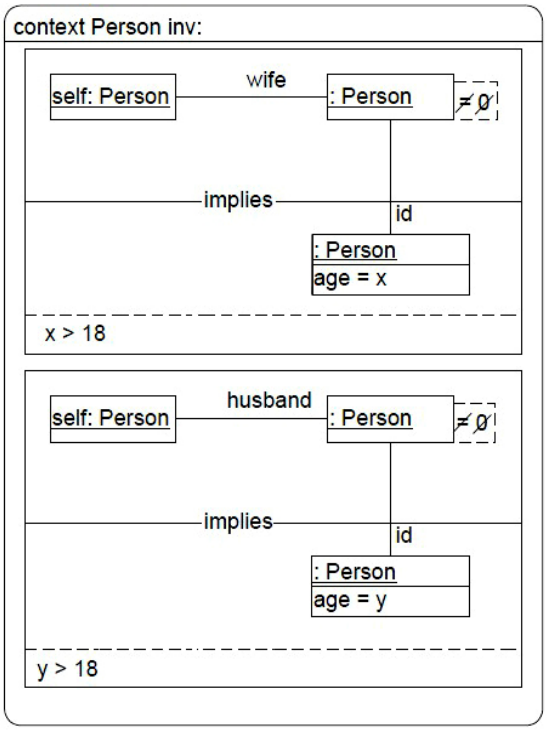

Рис. 5. Пример ограничения 1) на VOCL

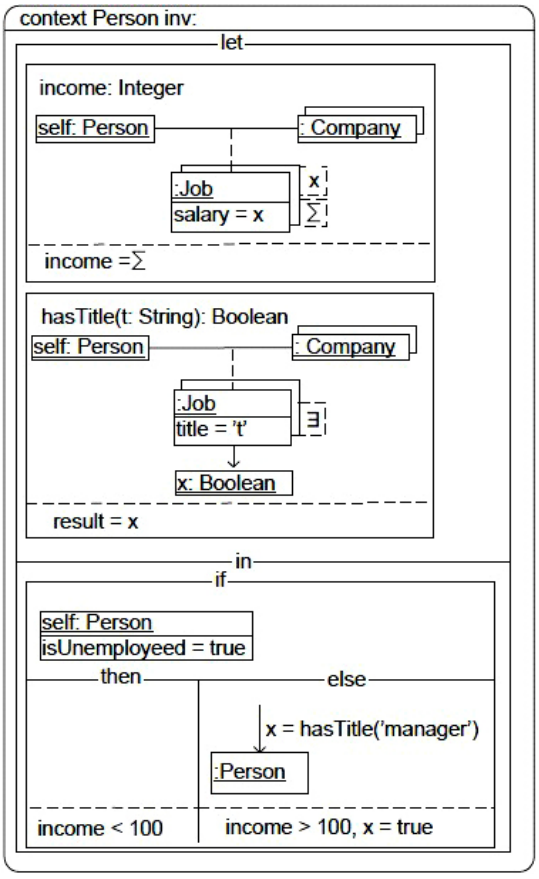

Рис. 6. Пример ограничения 4) на VOCL

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 требуемые поля, а слева – объект(ы), которые мы получаем, то есть собственно значение требуемого поля.

Далее наиболее интересен пример 4 (см. рис. 6) из ранее приведенных примеров OCL.

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/постусловиями (рис. 7) приведем аналог части примера 5) с постусловием 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`.

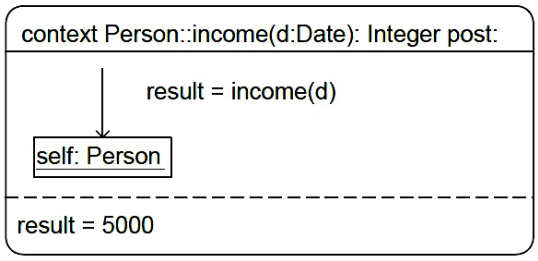

Рис. 7. Пример части ограничения 5) с постусловием на VOCL

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 OCL. Но, с другой стороны, задание ограничений на Visual OCL очень сложно, а соответствующие диаграммы с ограничениями получаются громоздкими даже для несложных ограничений (см. рис. 4, 5). При этом многие конструкции и способы визуализации не очень очевидны и понятны, что потребует много времени на обучение или частые обращения к описанию этого языка. К тому же, в большинстве случаев совершенно не нужно иметь такой мощный и сложный язык задания ограничений, а вполне достаточно и более простого языка.

ЯЗЫК ЗАДАНИЯ ОГРАНИЧЕНИЙ В QREAL

1. Технология QReal

На кафедре системного программирования математико-механического факультета СПбГУ в течение нескольких лет разрабатывается среда визуального программирования QReal [8]. Технология 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. Помимо этого QReal является и metaCASE-системой, то есть в 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, то есть метаязык, и соответствующий редактор для него для создания этой модели, то есть метаредактор [9].

Таким образом, для QReal тоже необходимо иметь возможность задавать ограничения на создаваемый язык. При этом для начала было решено поддержать возможность задавать ограничения именно на сам визуальный язык, а не на состояние системы во время выполнения программы, написанной на этом языке. А так как QReal является визуальной системой программирования, то и задавать ограничения при помощи визуальных, наглядных моделей целесообразней.

Требуемый язык задания ограничений был разработан в самой же системе QReal. По формальному описанию визуального языка в метаредакторе был автоматически сгенерирован соответствующий редактор языка ограничений. Вручную был написан генератор, который по описанию ограничений на визуальный язык генерирует соответствующие плагины для проверки этих ограничений.

2. Описание языка ограничений

Рассматриваемый 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 на этом языке позволяет задавать ограничения на некоторую модель визуального языка. Эта модель ограничений 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 (то есть узел или связь) или даже множество элементов визуального языка, на которые накладываются 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 по-

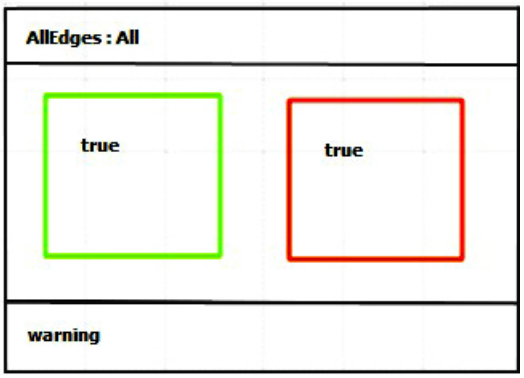

Рис. 8. Пример ограничения на нашем языке ограничений

мощи специальных элементов языка ограничений.

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

В верхней части прямоугольника указана выборка `AllEdges : All`, что означает, что наше ограничение проверяется для всех визуальных языков всех метамоделей и накладывается на все элементы типа «связь». В нижней части указан тип ограничения `warning`, тем самым при невыполнении этого ограничения нарушившая его связь будет подсвечена красным цветом (рис. 9), но дополнительных сообщений не будет показано пользователю. В средней части расположены зелёный (слева) и красный (справа) квадраты, которые обозначают элементы типа «узел», находящиеся в начале и в конце рассматриваемой связи соответственно. Внутри этих квадратов отображается значение свойства `exists`, в данном случае это `true`, а это означает, что данный узел обязан существовать. Тем самым данное ограничение считается выполненным, если на обоих концах связи есть узлы.

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

И начало, и конец первой слева связи прикреплены к узлам «старт» и «моторы вперед» соответственно, тем самым ограничение выполнено. У второй же слева связи только начало прикреплено к узлу «моторы вперед», а её конец ни к чему не присоединён, то есть ограничение уже не выполняется для этой связи, и она подсвечивается красным цветом.

3. Механизм проверки ограничений

Во время создания диаграмм на некотором визуальном языке, для которого есть ограничения, пользователю удобнее сразу узнать, что 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.

В первую очередь, во время работы пользователя с некоторой диаграммой отслеживаются те места в коде нашей системы, где происходят определенные 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, обрабатывает класс `MainWindow`. Этот класс является основным классом системы QReal, который управляет общей координацией между компонентами. В частности, он является связующим звеном и для механизма проверок ограничений.

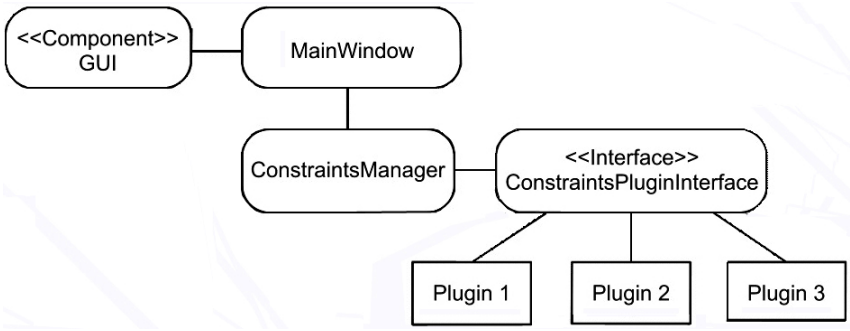

Рис. 10. Общая идея механизма проверки ограничений

Получив сигнал, что какой-то элемент был изменен, `MainWindow` вызывает метод проверки этого элемента у менеджера ограничений `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 окне для ошибок 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.

4. 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 [10]:

а) перед блоком «моторы стоп» не может следовать блок «моторы вперёд» или «моторы назад» (рис. 11, рис. 12).

В верхней части прямоугольника указано, что ограничение задаётся на узлы типа «EngineStop» («моторы стоп»). В нижней части указан тип ограничения `critical`, тем самым при невыполнении ограничения

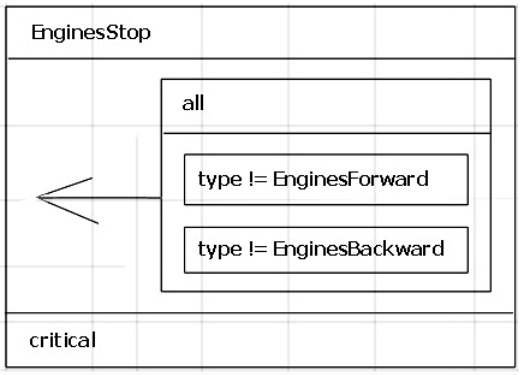

Рис. 11. Задание ограничения 2 а)

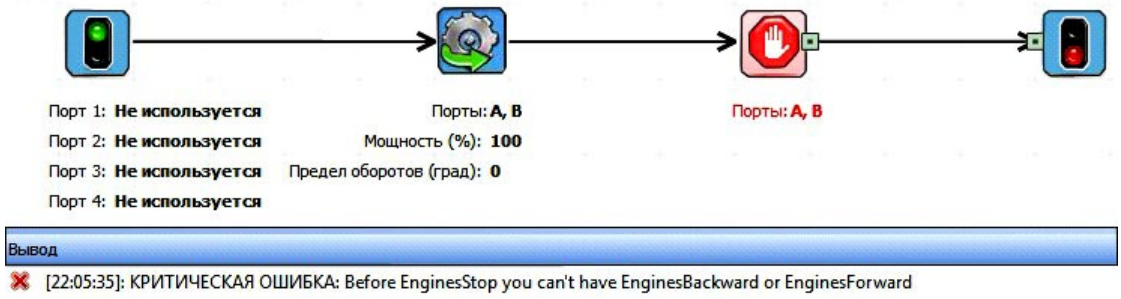

Рис. 12. Пример невыполнения ограничения 2 а)

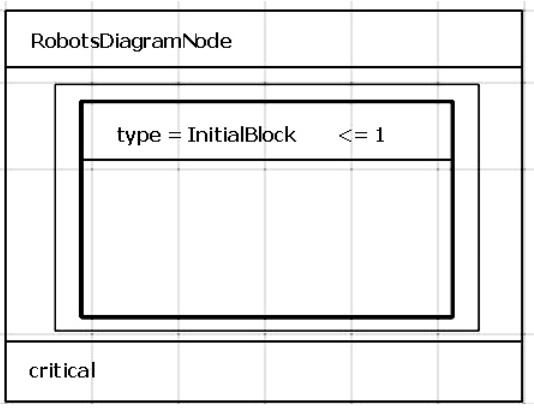

Рис. 13. Задание ограничения 2 б)

нарушивший его узел будет подсвечен красным, а также пользователю будет показано текстовое сообщение об ошибке. В средней части расположен элемент в виде квадрата со стрелкой во вне, который обозначает множество узлов, находящихся в начале всех входящих в `EngineStop` связей. Внутри этого элемента расположены два прямоугольника для задания ограничения на некоторое свойство каждого узла из рассматриваемого множества. В данном случае, это условия `type != EnginesForward` и `type != EnginesBackward`, которые означают, что типом каждого узла из рассматриваемого множества не может являться ни `EnginesForward` («моторы вперед»), ни `EnginesBackward` («моторы назад») соответственно.

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.

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

В верхней части прямоугольника указано, что ограничение задаётся на узлы типа `RobotsdiagramNode` (диаграмма роботов). В нижней части указан тип ограничения `critical`, что означает, что при невыполнении ограничения нарушившая его диаграмма будет подсвечена красным (если данная диаграмма будет узлом другой диаграммы), а пользователю будет показано текстовое сообщение об ошибке. В средней части расположен квадрат с двумя контурами, у которого при этом внутренний контур выделен более темным цветом. Данный элемент обозначает все узлы, находящиеся внутри данной диаграммы и удовлетворяющие условию `type = InitialBlock` (то есть узлы типа «блок инициализации»). А ограничение для этого множества узлов задано в верхней части справа от этого элемента (то есть «<= 1»), что означает, что таких узлов всего должно быть не более одного.

На рис. 14 на рассматриваемой диаграмме находятся два узла типа «блок инициализации», тем самым считается, что ограничение 2 б) не выполнено. Поэтому в специальном окне появился текст об ошибке, но ничего не подсвечено красным, так как эле-

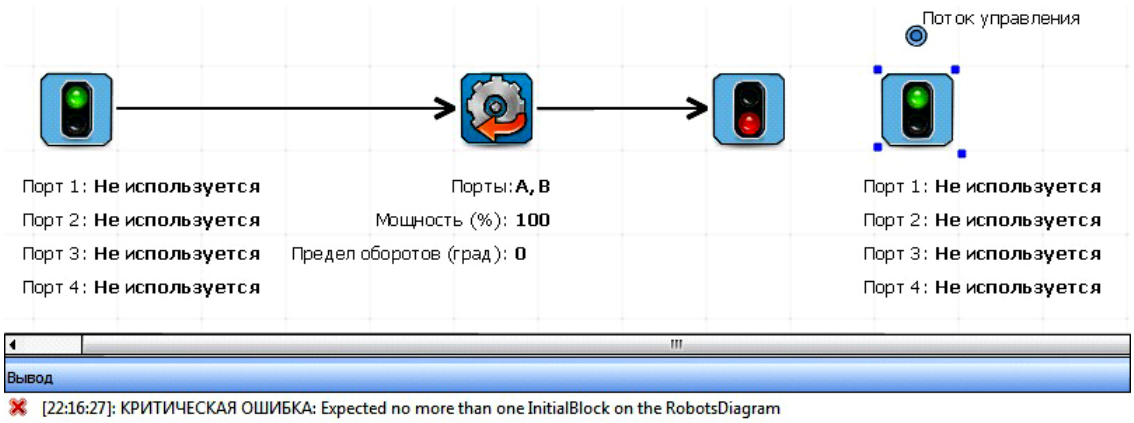

Рис. 14. Пример невыполнения ограничения 2 б)

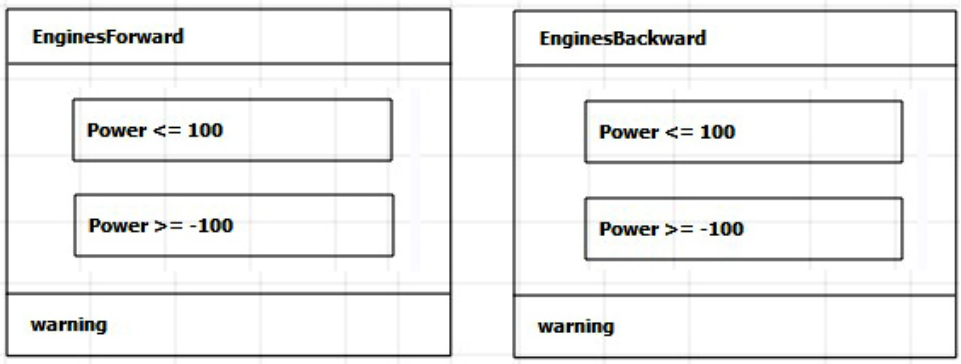

Рис. 15. Задание ограничения 2.в)

ментом, нарушившим данное ограничение, является сама диаграмма.

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

На рисунке изображено два однотипных прямоугольника, которые задают ограничения на узлы типа `EnginesForward` («моторы вперед») и `EnginesBackward` («моторы назад») соответственно. В средней части этих прямоугольников расположены два прямоугольника для задания ограничения на некоторое свойство каждого узла указанных типов. В данном случае, это условия `Power <= 100` и `Power >= -100`, которые означают, что у рассматриваемых узлов свойство `Power` («мощность») должно быть меньше 100, но больше –100 соответсвенно.

На рис. 16 у узла типа `EnginesForward` («моторы вперед») значение свойства «мощность» равно 105, что нарушает наше ограничение 2 в). Поэтому этот блок подсвечивается красным, но никаких ошибок не выдается, так как тип данного ограничения `warning`, а не `critical`.

Рис. 16. Пример невыполнения ограничения 2.в)

ЗАКЛЮЧЕНИЕ

В данной статье были рассмотрены наиболее известные существующие методы задания ограничений на визуальные языки. В частности, были описаны один из самых широко применимых текстовых языков OCL и его визуальный аналог VOCL.

Также был предложен свой формальный визуальный язык задания ограничений, который был реализован в среде визуального программирования QReal. В рамках данной технологии был создан соответствующий редактор, позволяющий описывать различные ограничения на модели визуальных языков, и механизм генерации целевого кода по этому описанию, проверяющий ограничения во время создания диаграмм на этих языках.

В качестве апробации нашего языка были реализованы несколько ограничений для среды визуального программирования роботов Lego Mindstorms NXT QReal:Robots [10]. Описание этих ограничений оказалось действительно наглядным и довольно несложным. А встроенный механизм проверки позволяет пользователю создавать семантически корректные программы, выдавая сообщения об ошибке, если какое-то из ограничений не выполнено.

Литература

1. Language elements with examples, Visual OCL / URL: http://tfs.cs.tu-berlin.de/vocl/language/examples.htm (дата обращения: 18.02.2012).

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: 18.02.2012).

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: 18.02.2012).

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: 18.02.2012).

5. Ограничения целостности и язык OCL, Интернет-университет информационных технологий, URL: http://www.intuit.ru/department/database/rdbintro/10/4.html (дата обращения: 18.02.2012).

6. *Жолудев В.В.* Генерация контекстных ограничений для баз данных / Дипломная работа, 2007.

7. *Иванов А.Н.* Автоматизированная генерация информационных систем, ориентированных на данные / Диссерт. на соиск. ученой степени кандидата физ.-мат. наук, 2005.

8. *Терехов А.Н., Брыксин Т.А., Литвинов Ю.В. и др.* Архитектура среды визуального моделирования QReal // Системное программирование, 2009. Вып. 4. С. 171–196.

9. *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.

10. *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.

11. *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: 18.02.2012).

Abstract

Ways to specify constraints on visual modeling languages are considered in this article. An overview of some existing constraints languages such as the widely applicable text-based language Object Constraints Language, its visual analogue Visual OCL and other more specialized constraints languages is given. New visual language to define constraints on visual modeling languages, developed in the programming environment QReal, is described. Approbation of the proposed language applied to the robots visual programming language QReal::Robots is touched upon. Conclusions about the relevance of the proposed new language are also included.

Keywords: visual programming, semantic correctness of programs, defining of constraints, constraints language.

Дерипаска Анна Олеговна,
лаборант-исследователь СПбГУ,
anna.deripaska@gmail.com