Санкт-Петербургский Государственный Университет
Математико-Механический факультет

Кафедра системного программирования

Курсовая работа
Тихоновой Марии Валерьевны, студентки 344 гр.
“Метаязык в QReal”

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

Введение

Использование визуальных языков для создания программного обеспечения существенно облегчает разработку, снижая вероятность появления ошибок и делая процесс разработки более наглядным. При этом использование языков общего назначения приводит к тому, что диаграммы теряют наглядность. Зачастую проще создать отдельный язык для решения конкретной задачи и с его помощью решить ее, чем пытаться сделать это с помощью языков общего назначения. Такой подход называется предметно-ориентированным моделированием. Создание специальных языков под каждую задачу отнимет слишком много времени и сил, поэтому для создания этих языков используются DSM-платформы. Примерами DSM-платформ могут послужить такие средства, как MetaEdit+, Eclipse Modeling Framework, Microsoft DSL Tools. Еще одним примером служит DSM-платформа QReal, разрабатываемая на кафедре системного программирования.

Одной из главных составляющих DSM-платформы является метаязык --- язык для описания других языков. Для создания нового языка в QReal сначала абстрактный и конкретный синтаксисы языка описываются с помощью метаязыка в xml-файле, по которому потом генерируется C++ код, реализующий специфику конкретных редакторов. Сгенерированный код компилируется в динамическую библиотеку, которая грузится при запуске QReal. Поскольку xml-описания можно генерировать по визуальным моделям, в QReal были добавлены визуальные средства поддержки метамоделирования, для чего был разработан визуальный метаязык. Но поскольку этот метаязык получился путем визуализации xml-формата метаязыка, то он несколько запутанный, например, некоторые сущности, которые были нужны для описания языка в xml-формате, в графическом языке неуместны.

Целью данной курсовой работы было улучшить метаязык. Исходя из этого, были поставлены конкретные задачи:

- исследовать метаязыки в существующих DSM-платформах
- на основе полученных знаний выявить недостатки метаязыка в QReal
- сформулировать требования к метаязыку

Обзор существующих метаязыков

В рамках данной курсовой работы были рассмотрены метаязыки в следующих DSM-платформах: MetaEdit+, AToM3, Eclipse Modeling Framework, MetaLanguage.

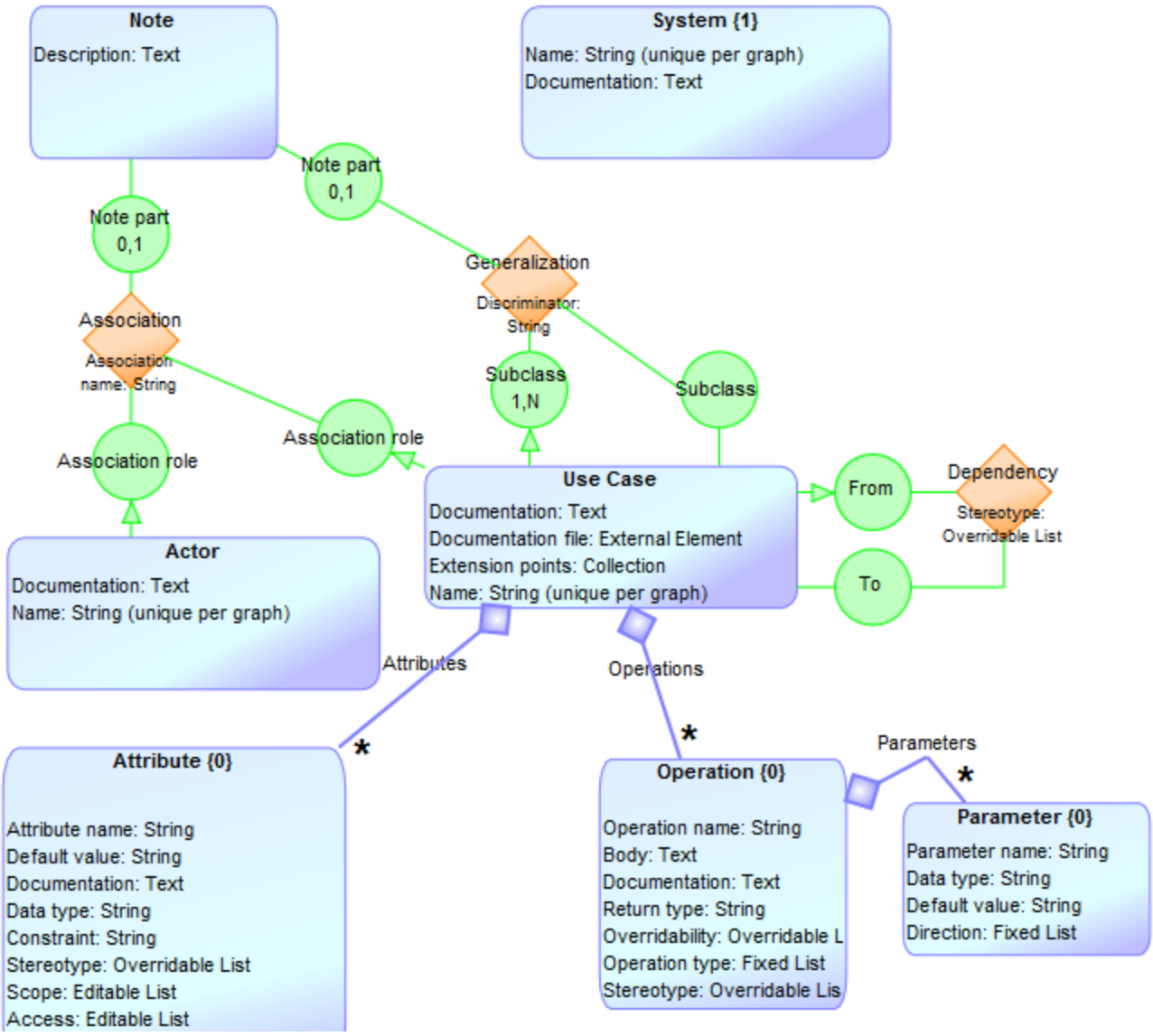
MetaEdit+

MetaEdit+ --- DSM-платформа, разрабатываемая финской компанией MetaCase. MetaEdit+ предназначен для создания DSM-решений. DSM-платформа MetaEdit+ состоит из следующих подсистем:

- MetaEdit+ Workbench – средство создания языков моделирования и генераторов
- MetaEdit+ – полнофункциональная среда для разработки систем с поддержкой возможности использования собственных языков моделирования, генераторов кода и документации, созданных с помощью MetaEdit+ Workbench

В MetaEdit+ используется метаязык GOPRR, получивший свое название от основных объектов, которыми он оперирует: Graph --- граф, Object --- объект, Property --- свойство, Role --- роль, Relationship --- отношение. Граф --- это одна отдельная модель, объекты --- основные элементы графа. С помощью отношений объекты можно связывать между собой, а роль определяет, каким образом объект участвует в связи. Свойство --- это атрибут, который характеризует какой-либо из вышеперечисленных элементов.

Ниже приведена метамодель для диаграмм случаев использования, описанная с помощью языка MetaEdit+:



Рассмотрим подробнее элементы языка, их назначение и свойства.

Graph (граф)	Под графом понимается создаваемый язык, то есть совокупность объектов, связей, ролей. Примером графа может служить диаграмма случаев использования. Каждый язык определяется отдельной метамоделью.
Object (объект)	Представляет основные элементы языка моделирования. Примером может служить класс в диаграммах классов UML. В метамодели диаграммы случаев использования объектами являются Актер, Случай использования, Система.
Relationship	Используется для соединения двух и более объектов. Примером

(отношение)	может послужить наследование в диаграммах классов UML.
Role (роль)	Определяет, каким образом объект участвует в связи. Например, объекты, участвующие в связи наследования, имеют роли соответственно “родитель” и “потомок”. В диаграмме случаев использования примерами роли могут послужить роли From и To в отношении Dependency.
Property (свойство)	Определяет характеристики какого-либо объекта графа. Свойства могут иметь простой тип (string, text, bool, number, collection, creation timestamp, editable list, fixed list) или являться ссылкой на другой объект (object) или на внешний ресурс (файл или программу).
Inheritance (наследование)	Вид связи; позволяет создавать подтипы других элементов языка.
Binding (привязка)	Связывает объекты друг с другом, определяя отношение между ними и роль, которую каждый объект в этом отношении играет. Пример привязки в метамодели диаграмм случаев использования - отношение Association: мы создаем привязку между объектами Use Case и Actor, определяем имя отношения (Association) и роли которые играют оба элемента в отношении (Association role).
Object Set (множество объектов)	Коллекция объектов, которые играют в привязке одну и ту же роль.
Decomposition (декомпозиция)	Вид связи; позволяет объектам иметь подграфы (каждый элемент может быть описан с помощью одного подграфа). Например элемент “Процесс” может быть соединен этой связью с диаграммой потока данных. Еще один пример декомпозиции -- связь элемента System и диаграммы Use Case.
Explosion (эксплозия)	Позволяет объектам, связям и ролям цепляться к другим графам (пример: детальная структура Store в диаграмме потока данных может быть описана в диаграмме entity relationship (сущность-связь)).

AToM3

Atom3 -- средство для метамоделирования и работы с мульти-формализмами. Примером формализма могут служить диаграммы сущность-связь (Entity-Relationship), диаграммы состояний (State charts), диаграммы активностей (Activity diagrams). Эти формализмы (их можно считать метаязыками) используются для описания других формализмов (конечных автоматов, обыкновенных дифференциальных уравнений).

Уровень	**Описание**	**Пример**
Метаметамодель	Модель, описывающая формализм, который будет использован для описания других формализмов	Описание диаграмм сущность-связь, диаграмм классов UML
Метамодель	Модель, которая описывает формализм в соответствии с правилами метаметамодели.	Описание конечного автомата, обыкновенных дифференциальных уравнений
Модель	Описание объекта в соответствии с правилами метамодели.	f’(x) = -sinx f(0) = 0 (в примере с дифференциальными уравнениями)

Пример: построение формализма Causal Block Diagrams с помощью метаформализма Entity Relationship. Основной элемент формализма -- элемент Блок, представляющий функции передачи (transfer functions), например, арифметические операторы или оператор интегрирования. Блоки могут соединяться с другими блоками, соединения передают сигналы между блоками. Сигналы -- функции от времени.

Eclipse Modeling Framework

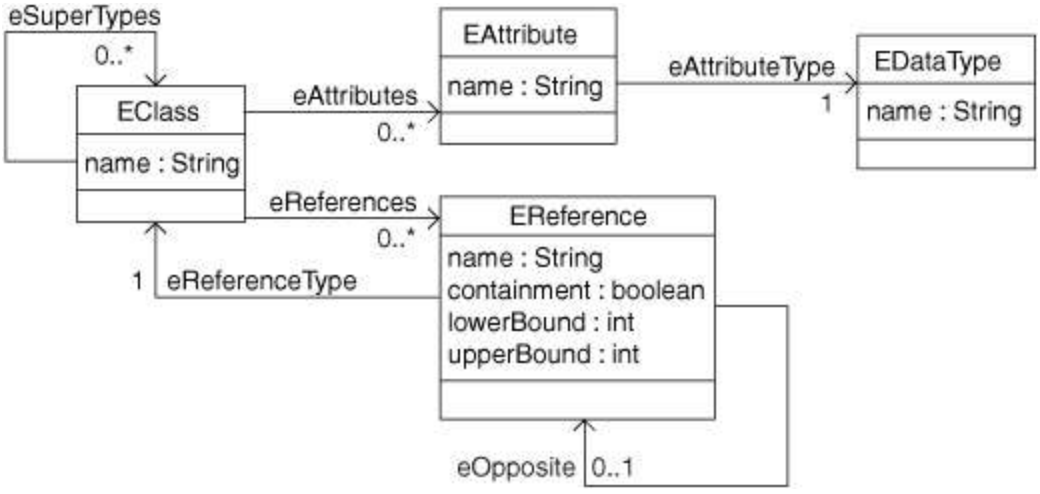
Среда Eclipse --- это многоплатформенная интегрированная среда разработки программного обеспечения. На базе этой среды создаются различные дополнительные технологии, одна из которых – Eclipse Graphical Modeling Framework. Технология GMF предназначена для быстрой разработки графических средств. GMF интегрирует две широко используемые и известные Eclipse-библиотеки --- Eclipse Modeling Framework (EMF) и Graphical Editing Framework (GEF).

Рассмотрим подробнее язык ECore --- язык, с помощью которого задаются метамодели в EMF. Ниже приведен список элементов языка:

EClass	Элемент, который представляет класс (элемент нового языка). Характеризуется именем и может иметь атрибуты и ссылки Для поддержки наследования класс может ссылаться на другие классы как на надклассы.
EAttribute	Элемент, представляющий атрибут. Имеет имя и тип.

EDataType	Представляет собой простые типы, используется как тип EAttribute, характеризуется именем. Может быть как простым типом (string или int), так и каким-нибудь объектным типом, например, java.util.Date
EReference	Элемент для моделирования ассоциаций между классами. Моделирует один конец ассоциации, характеризуется именем и типом. На другом конце ассоциации должен быть тип EClass. Если ассоциация в обе стороны, то должен быть еще одна ссылка (ссылка на другой конец ассоциации -- eOpposite). Иначе для ссылки на другой конец ассоциации используется атрибут eReferenceType. Также ссылка имеет атрибуты множественности (верхнюю и нижнюю границы множественности).

Ниже представлена метамодель языка ECore.



MetaLanguage

MetaLanguage --- языковой инструментарий, созданный для разработки визуальных предметно-ориентированных языков. Отличительной особенностью системы разработчики считают возможность задать метамодель языка как новый метаязык и использовать для задания языков, таким образом, сделав этот процесс бесконечным.

Рассмотрим элементы метаязыка в MetaLanguage. Основными сущностями метаязыка являются Сущность, Связь и Ограничение.

Сущность	Сущность --- любая конструкция нового языка. Характеристики:

	● имя, которое определяет сущность в метамодели единственным образом ● число, которое характеризует количество данных сущностей которые можно создать в метамодели ● атрибуты ● операции ● накладываемые ограничения ● флаг единственности Атрибут --- именованное свойство сущности, включает в себя возможное множество значений. Характеристики: ● имя, единственным образом определяющее атрибут для сущности ● тип, определяющий возможное множество значений для данного атрибута и операции, которые могут над ним совершаться ● значение по умолчанию ● описание, содержащее дополнительную информацию сущности Сущность может иметь произвольное число атрибутов или не иметь их вовсе. Операция --- действия, которые можно производить над сущностью. Применение операции обычно приводит к тому, что сущность меняет свое состояние. Характеристики: ● имя, которое единственным образом определяет операцию ● параметры операции ● значения параметров по умолчанию, которые будут использованы, если параметры не были определены ● тип возвращаемого значения ● описание, содержащее дополнительную информацию о операции
Связь	Связь используется для описания некоторого отношения между сущностями (физического или концептуального) . Характеристики: ● имя, единственным образом определяющее связь в метамодели

	● тип, который определяет семантику связи ● множество ограничений ● множество атрибутов ● множественность, определяющая, сколько сущностей может участвовать в связи ● флаг единственности Связи бывают трех типов: наследование, ассоциация, агрегация. Наследование --- отношение между обобщенной сущностью (родителем, надклассом) и частной (потомком, подклассом). Потомок наследует все свойства родителя, его операции и отношения и может также доопределять свои свойства, операции и отношения. Потомок может использоваться везде, где используется родитель, обратное неверно. Сущность может иметь одного родителя и неограниченное число потомков, таким образом множественность отношения 1 : M Ассоциация --- отношение, которое определяет, что сущности одного типа связаны с сущностями другого типа. Ассоциация может быть однонаправленной или двунаправленной. Помимо упомянутых общи свойств имеет свое уникальное свойство “роль” --- имя, определяющее один конец ассоциации. Множественность отношения ассоциации --- M : M Агрегация --- своего рода вид ассоциации, связь, моделирующая отношение “часть - целое”. Основное отличие агрегации от ассоциации состоит в том, что ассоциация моделирует отношение между равноправными сущностями, а агрегация --- между сущностями, одна из которых является главной, а другая --- зависимой. Связь агрегации всегда имеет направление, множественность данного отношения --- 1 : M
Ограничение	Ограничения задают семантику визуального языка. Часть ограничений задается при описании структуры метамодели, часть --- через непосредственное определение на некотором языке. Все ограничения можно поделить на два типа: ● ограничения, накладываемые на сущности ● ограничения, накладываемые на связи Ограничения на сущности, в свою очередь, можно поделить на следующие типы: ● ограничения, накладываемые на уникальность имени

	экземпляра сущности ● ограничения на количество экземпляров сущности в модели ● ограничения, накладываемые на значения атрибутов сущности Ограничения, накладываемые на отношения, могут быть поделены на следующие типы: ● ограничения, накладываемые на уникальность имени экземпляра отношения ● ограничения, накладываемые на типы сущностей, участвующих в отношении ● ограничения, накладываемые на множественность ● ограничения, накладываемые на значения атрибутов сущностей, участвующих в отношении

Требования к новому метаязыку

На основании проведенного анализа метаязыков и работы с метаязыком в QReal были сформулированы следующие требования к метаязыку:

Требования, относящиеся к инфраструктуре

1. Разделение абстрактного синтаксиса и конкретного

Абстрактный синтаксис --- описание сущностей разрабатываемого визуального языка, конкретный синтаксис --- описание графического представления этих сущностей. Абстрактный синтаксис описывает модель знаний, которые система имеет о разрабатываемом языке, а конкретный синтаксис описывает то, как пользователь видит язык и то, с чем он работает.

В большинстве рассмотренных метаязыков абстрактный и конкретный синтаксис разделены. В QReal такого разделения нет, и графическое представление элемента является просто одним из его свойств. Предлагается разделить абстрактный и конкретный синтаксис.

Можно выделить следующие плюсы такого разделения:

- Для одной и той же абстрактной модели можно будет задавать несколько графических представлений
- В существующей версии метаязыка не видно, у каких объектов уже задано графическое представление и какое оно. Разделение абстрактного и конкретного синтаксисов позволит задавать графическое представление отдельно, и будет видно, для каких элементов оно уже задано, а для каких нет

- Одному элементу можно будет задавать несколько графических представлений
- Можно будет просто задавать View-To-View Transformations

Также у данного разделения есть и свои минусы:

- В большинстве языков такое разделение не нужно. Например, язык QReal::Robots очень простой, каждому элементу соответствует свое графическое представление, и весь язык можно выразить с помощью одной метамодели
- Если разделить абстрактный и конкретный синтаксис, то увеличится и количество моделей, а следовательно, возрастет сложность работы с метаязыком. Может получиться, как с Eclipse Modeling Framework, пользователи которого жалуются на его чрезмерную сложность.

2. Возможность экспортировать метамодель в другие форматы

Хотелось бы иметь возможность взаимодействовать с другими DSM-платформами, например, с MetaEdit+, для чего необходимо уметь экспортировать метамодель в те форматы, которые поддерживают данные DSM-платформы. Например, MetaEdit+ хранит метамодели в формате MXT (MetaEdit+ XML Types), который базируется на стандарте XML.

3. Задание ограничений

Решить, нужно ли задавать ограничения в метамодели или вынести в отдельный язык.

Требования, относящиеся к элементам метаязыка

1. Порт

Порт --- конструкция языка, которая задает место на элементе, к которому можно присоединять связь, и позволяет задавать синтаксические и семантические ограничения на связи между объектами. Если мы хотим, чтобы данная связь могла присоединяться к разным местам на фигуре с разной семантикой, то нужно добавить порт на символ фигуры.

Сейчас порт в метаязыке QReal представляет собой просто точку или множество точек на фигуре, к которым можно присоединять связи. Хочется реализовать порт как сущность в метаредакторе.

Предлагаемое представление порта:

- в метаредакторе: является отдельной сущностью в палитре, потомком фигуры
- в модели: квадратики по сторонам фигуры (с именем порта), к которым можно присоединять связи

- для генератора\интерпретатора: генератор (или интерпретатор) различает порты по именам и берет связи, присоединенные к нужным портам

Атрибуты порта:

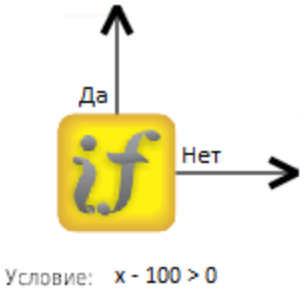
- имя
- кратность (максимальная и минимальная или точное значение)
- ограничения на то, какие связи можно присоединять к каким портам

Для поддержки портов нужно будет модифицировать связи, чтобы можно было определять, к каким портам их можно присоединять, а не к каким фигурам.

Также имеет смысл уметь создавать массивы портов или списки портов, чтобы обращаться к ним одинаково, и сделать порты динамически создаваемыми (т.е. мы создаем порт, вытягиваем из него связь, и второй порт появляется автоматически).

Примером может послужить блок условного оператора в визуальном языке системы QReal:Robots. Сейчас блок условного оператора выглядит так: внутри блока находится условие, и из него выходят две связи: например “условие меньше 0” и связь для оставшегося случая:

Если ввести понятие типизированного порта, то блок может выглядеть так:



У блока будет два порта: “да” и “нет”, из них будут выходить соответствующие связи Такое представление более удобно и с точки зрения пользователя, и для генерации.

2. Общая запутанность метаязыка

Как уже было сказано, метаязык в QReal был получен путем визуализации xml-формата, поэтому некоторые его элементы не нужны в визуальном метаязыке. Например, Association: нужно задавать начало и конец связи, хотя это никак не используется в сгенерированном языке, но без этого связь не создастся. Также в метаязыке нет четкого определения провязки (раскрытия элемента с помощью какой-либо другой диаграммы).

Требования, относящиеся к отношениям между элементами

1. Отношение эксплозии

Предлагается добавить отношение эксплозии. Отношение эксплозии позволяет объектам и связям быть связанными с другими графами (т.е. “раскрываться” в эти графы). Так, в диаграмме потоков данных (Data Flow Diagram) элемент Store может быть связанным отношением эксплозии с диаграммой сущность-связь.

Источником эксплозии может быть любой объект, для которого она задана, а целью --- корневой элемент любой диаграммы.

Предлагаемое представление эксплозии:

- в метаредакторе: как отношение
- в редакторе: как пункт контекстного меню или автоматически

2. Провязка и наследование

Провязка --- бинарное отношение между элементами, определяющее иерархию между ними. Например, алгоритм метода класса можно описать с помощью диаграммы активности. Провязку в QReal раньше добавить можно было с помощью всплывающего меню:

- add connection (добавить провязку)
- disconnect (удалить провязку)

Сейчас и у провязки, и у наследования слишком размытое определение, хочется сделать его более явным. И наследование, и провязку, предлагается сделать типом связи.

Требования, относящиеся к организации взаимодействия между языками

1. Package merge

Опыт работы в создании визуальных языков показал, что довольно часто хочется переиспользовать уже существующий язык, создав новый язык, являющийся его подмножеством или надмножеством. Эти действия можно осуществить с помощью операций package import и package merge соответственно.

Package merge --- прямое отношение между двумя пакетами, когда содержимое обоих пакетов комбинируется в новый пакет. Отношение package merge похоже на связь Generalization между элементами в том смысле, что первый элемент добавляет к своим свойствам свойства второго, и получается новый элемент, который содержит объединение свойств исходных элементов.

Механизм используется, когда элементы двух разных пакетов имеют одинаковые имена и представляют одинаковые сущности (чтобы иметь разные описания одной и той же сущности для разных случаев). Package merge может быть рассмотрен как операция, которая берет содержимое двух пакетов и создает новый пакет, который комбинирует содержимое исходных пакетов.

Участники package merge:

Участниками package merge являются следующие объекты:

- Receiving Package --- пакет, который расширяется с помощью содержимого второго пакета, второй операнд мерджа (до того, как был мердж)

- Merged Package --- пакет, который мерджится в Receiving Package, первый операнд мерджа

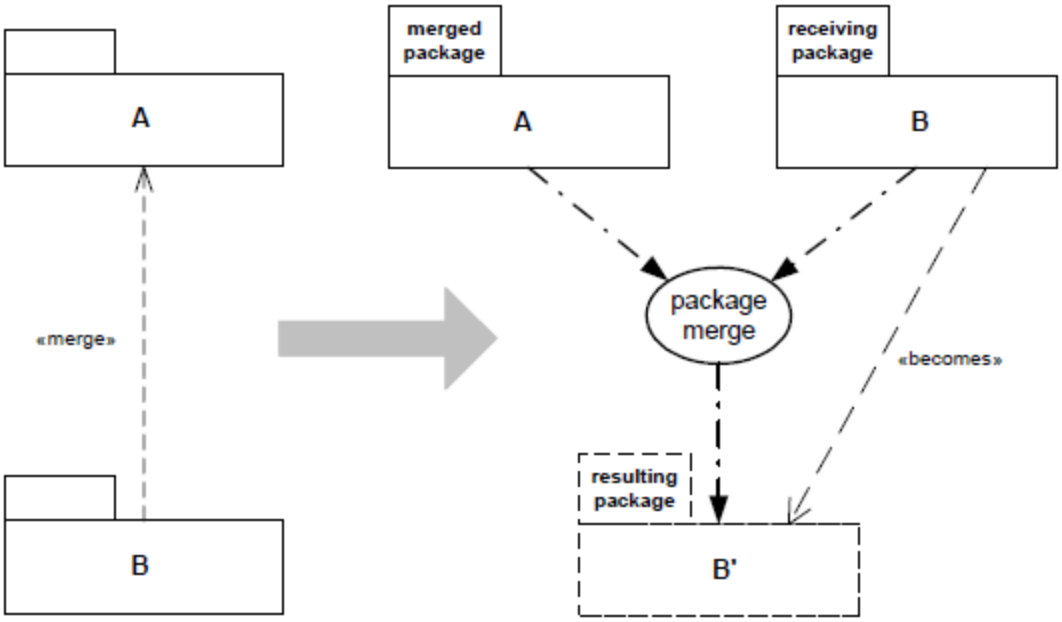
- Resulting Package --- пакет, который получился при мердже (Receiving Package после мерджа)

- Merged Element --- элемент, который присутствует в Merged Package

- Receiving Element --- элемент, который присутствует в Receiving Package. Если такой элемент присутствует еще и в Merged Package, то элементы при мерждe объединяются (как --- показано дальше)

- Resulting Element --- элемент, который присутствует в Resulting Package. Если элемент был и в Merged Package, и в Receiving Package, то результат будет элементом из Receiving Package, к которому применена операция package merge. Если элемент был только в Merged Package или в Receiving Package, то результатом будет элемент из Merged Package или Receiving Package соответственно.

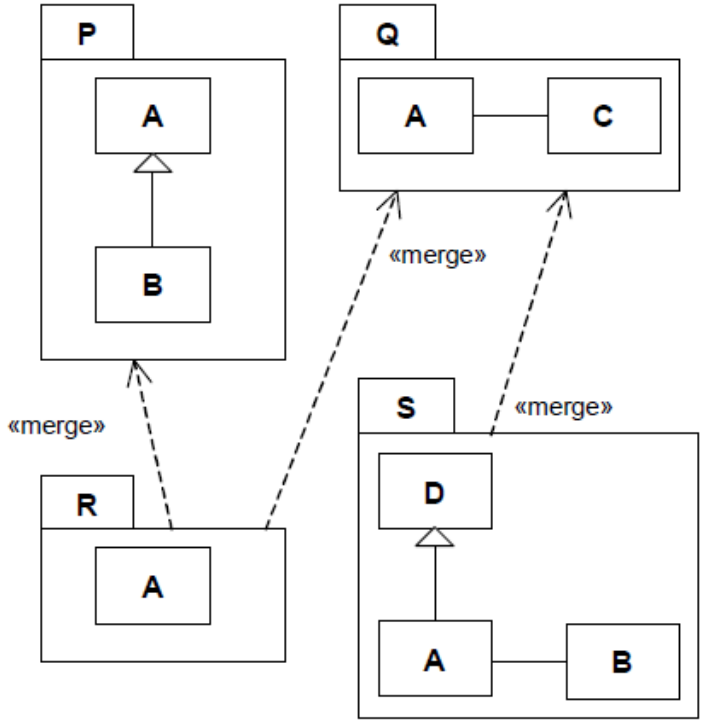
Схема package merge:



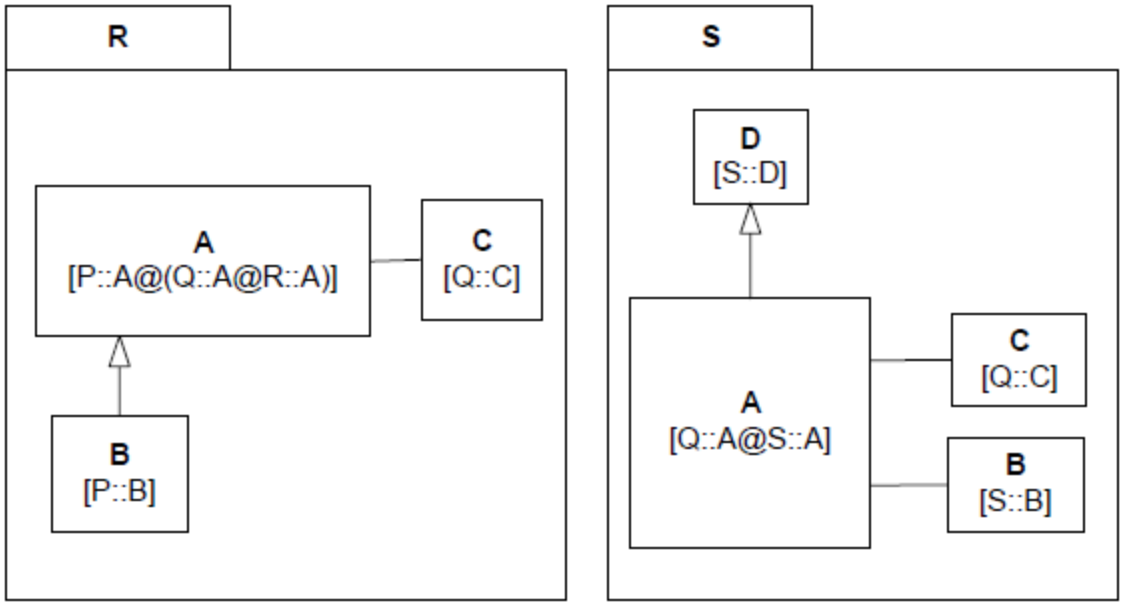
На данной картинке показана схема package merge для пакетов A и B. B --- это Receiving Package, а A --- Merged Package.

Примеры Package Merge:

Далее рассмотрены несколько ситуаций package merge.



Мы хотим смерджить в пакет R пакеты P и Q, а в пакет S --- только пакет Q.
Результат:



Здесь @ обозначает оператор мерджа.

Требования, не относящиеся ни к какой группе

1. Возможность задания палитр в языках

Хочется уметь объединять элементы новых языков в группы в палитре. Нужно придумать разумное графическое представление задания этих групп и поддержать генерацию для него.

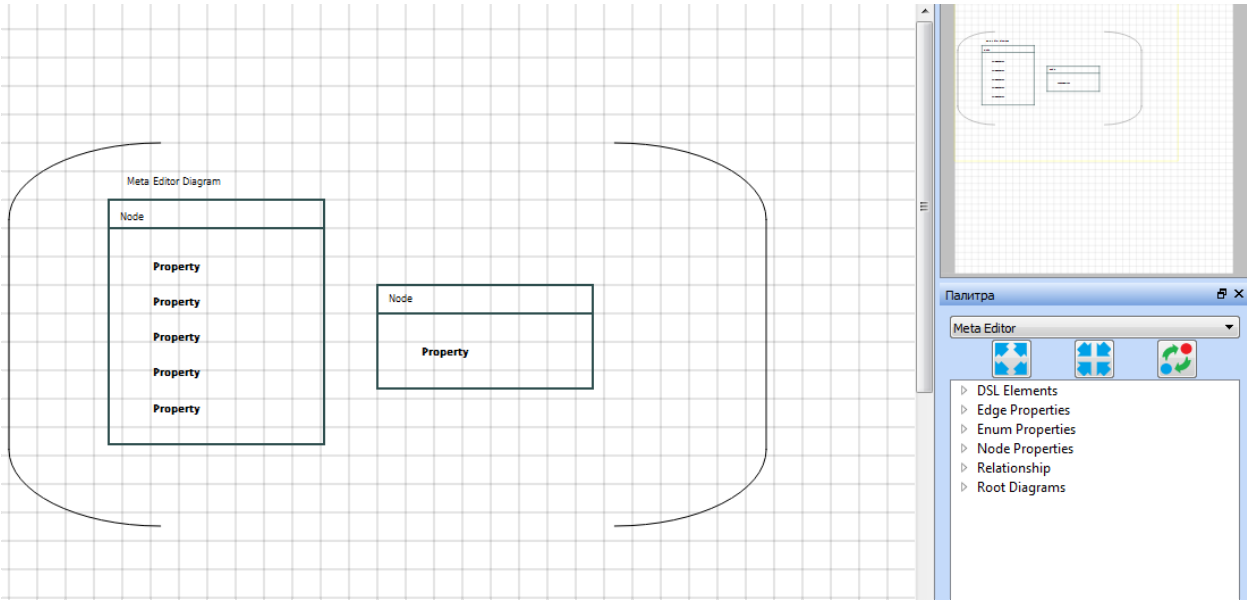
Простые метаязыки

Концепция метамоделирования сама по себе довольно сложна для понимания. К тому же, приходится разбираться с метаязыком, сущностями и отношениями в нем. Хотелось создать метаязык, достаточно простой (с минимально возможным числом элементов) и в то же время достаточно выразительный для решения хотя бы каких-то содержательных задач.

Было проведено дополнительное исследование с целью выяснить, насколько простым может быть метаязык, с помощью которого в то же время уже можно будет решать простые задачи. Было создано семейство метаязыков, причем первый метаязык был создан с помощью метаязыка QReal, а каждый последующий создавался с помощью предыдущего.

Этап 1: Метаязык “Сущность-диаграмма”

Ниже представлена метамодель метаязыка “Сущность-диаграмма”.



На первом этапе создадим простейший метаязык, в котором будет только элемент “Сущность” и элемент “Диаграмма” (который также будет корневым элементом метаязыка). У диаграммы будут следующие свойства:

- имя языка, который мы создаем
- отображаемое в палитре имя нового языка
- имя корневого элемента нового языка

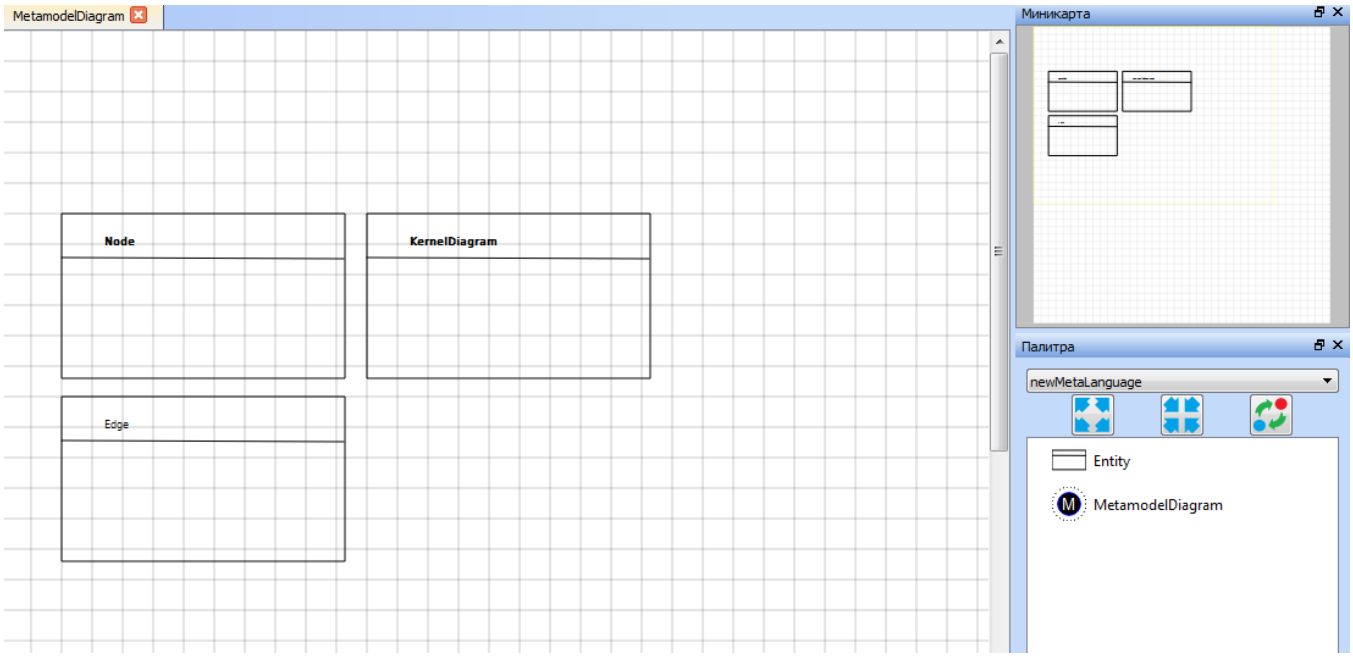
- путь до папки с исходниками QReal
- путь до папки, в которую будет сгенерирована xml-ка и код

У сущности будет только свойство “shape”, позволяющее задавать графическое представление элементам языка.

Такой метаязык, хотя он и очень простой, уже позволяет создавать непустое множество языков. Например, векторный редактор с заданным множеством используемых фигур или язык для отображения размещения кнопочек на форме. Между элементами нельзя задавать связи, но на этом этапе это и не нужно.

Этап 2. Метаязык “Сущность, связь, диаграмма”

Метамодель этого метаязыка, описанная с помощью метаязыка “сущность, диаграмма”:



С помощью метаязыка, созданного на предыдущем этапе, создадим следующий метаязык, в котором будет три элемента: сущность, связь и корневая диаграмма. У элементов не будет никаких свойств, кроме имени.

Генерация для метаязыка будет осуществляться следующим образом:

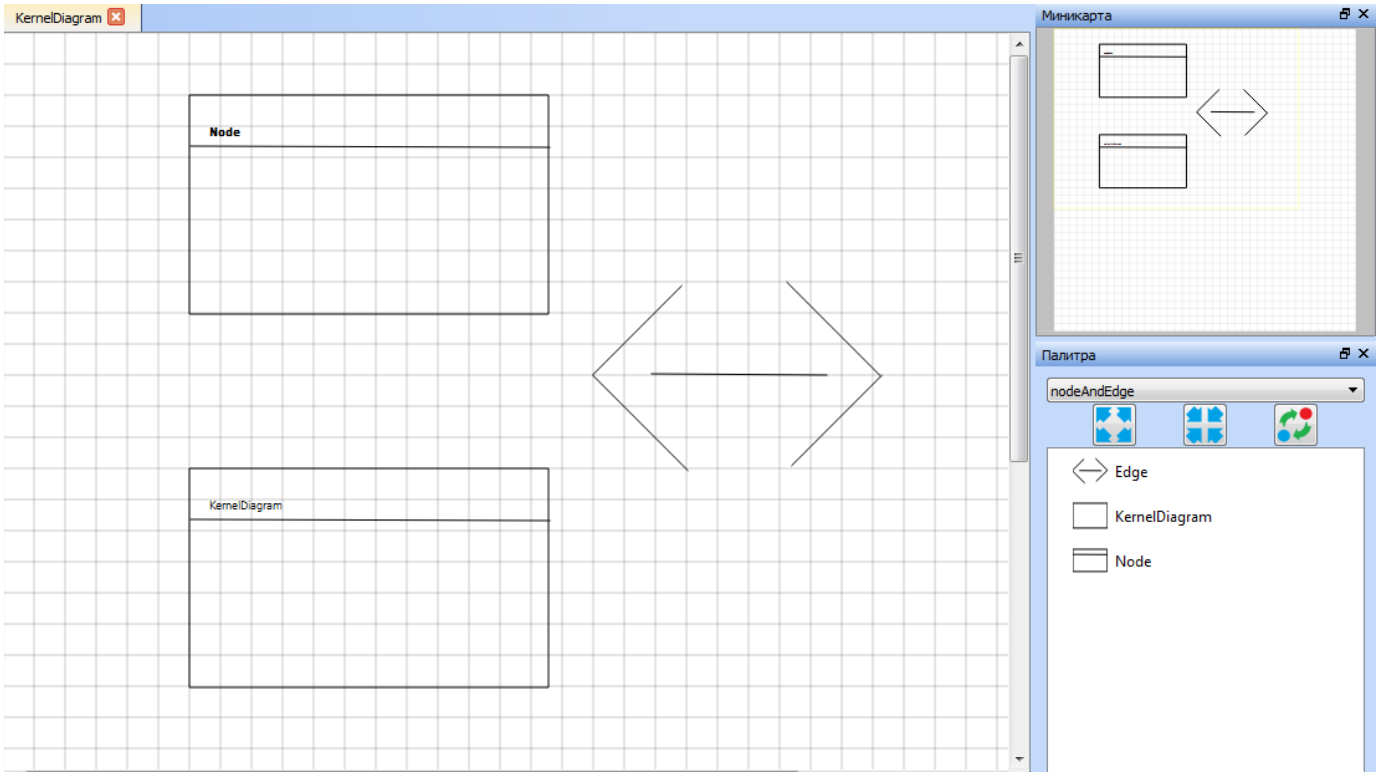
- если тип элемента --- сущность, то генерируем то же самое, что на предыдущем этапе. Причем если имя элемента --- KernelDiagram, то мы хотим, чтобы он был корневым в новом метаязыке (поскольку свойств в метаязыке пока что нет и мы не можем задавать в метамодели имя корневого элемента, то это нужно делать в генераторе по имени элемента). Также нужно в генераторе прописать, куда будет генерироваться код, определить путь до исходников QReal и имя нового метаязыка. Если имя элемента --- Node, то генерируется сущность, как в предыдущем генераторе, причем поскольку возможности задавать форму пока что

нет, то форма этой сущности задается в генераторе xml-описанием

- если тип элемента --- связь, то генерируем направленную стрелочку с портами

С помощью данного метаязыка можно задавать языки для описания графов. Частным случаем такого языка является следующий метаязык.

Этап 3. Метаязык “Сущность, связь, отношение Contains”

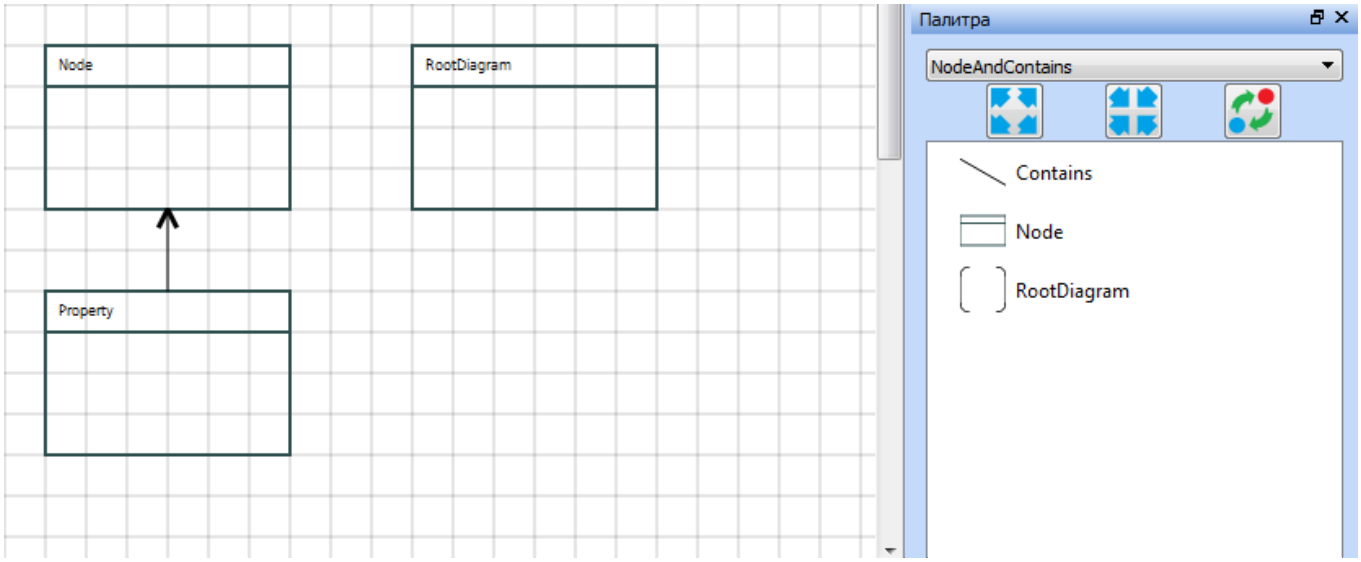


Из предыдущего языка мы хотим получить язык, в котором будут элементы

- сущность
- корневая диаграмма
- направленная связь “Contains”

Метамодель этого языка на предыдущем метаязыке будет содержать две сущности (сущность нового метаязыка и корневую диаграмму) и связь. Связь Contains нужна для того, чтобы ввести на следующем этапе элемент “Свойство”.

Этап 4. Метаязык “Сущность, свойство, диаграмма”



Новый язык будет содержать три элемента:

- сущность
- свойство
- корневая диаграмма

Стоит отметить, что элемент “корневая диаграмма”, который присутствует на каждом этапе, является требованием движка QReal, без него новый язык создать нельзя.

Представленное семейство метаязыков может быть использовано для обучения метамоделированию или для создания простых языков. Можно также продолжить цепочку дальше, выясняя, какие языки можно с помощью этих метаязыков создавать.

Заключение

В рамках курсовой работы были получены следующие результаты:

- проанализировано несколько метаязыков в разных DSM-платформах
- сформулированы недостатки метаязыка в QReal и требования к нему
- создано семейство простых метаязыков, которые могут применяться для создания простых языков и для обучения метамоделированию

Результаты исследования показали, что метаязык QReal не содержит ряда типичных для метаязыков сущностей, полезных для метамоделирования. Поэтому был предложен список требований к метаязыку, реализация которых далеко выходит за рамки данной курсовой работы.

Список литературы

1. Kelly, S., Tolvanen, J. Domain-Specic Modeling: Enabling Full Code Generation // Wiley-IEEE Computer Society Press. 2008. 448 pp.

2. OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2.4.1

3. MetaEdit+. The Graphical Metamodeling Example

4. de Lara, J., Vangheluwe, H. AToM3: A Tool for Multi-formalism and Meta-modelling // Lecture Notes in Computer Science, 2002

5. Vangheluwe, H, Sun, X., Bodden, E. Domain-specific Modelling with AToM // In Proceedings of the th OOPSLA Workshop on Domain-Specific Modeling, 2004

6. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E., Eclipse Modeling Framework // Addison-Wesley Professional. 2008. 744 pp.

7. Sukhov A. O., Lyadova L. N. MetaLanguage: a Tool for Creating Visual Domain-Specific Modeling Languages // Proceedings of the 6th Spring/Summer Young Researchers' Colloquium on Software Engineering, SYRCoSE. – 2012. – C. 42-53.

8. Сухов А. О. Сравнение систем разработки визуальных предметно-ориентированных языков //Математика программных систем: межвузовский сборник научных статей // Перм. гос. нац. исслед. ун-т. – 2012. – С. 84-111.

9. Кузенкова А.С., Литвинов Ю.В., Брыксин Т.А., Метамоделирование: современный подход к созданию средств визуального проектирования // Материалы второй научно-технической конференции молодых специалистов «Старт в будущее», посвященной 50-летию полета Ю.А. Гагарина в космос. СПб. 2011. С. 228-231