

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет

Кафедра системного программирования

Дерипаска Анна Олеговна

Средства задания правил генерации в QReal

Дипломная работа

Допущена к защите.

Зав. кафедрой:

д.ф.-м.н., проф. Терехов А.Н.

Научный руководитель:

ст. преп. Литвинов Ю. В.

Рецензент:

ст. преп. Луцив Д. В.

Санкт-Петербург
2014

SAINT-PETERSBURG STATE UNIVERSITY

Mathematics & Mechanics Faculty

Software Engineering Chair

Anna Deripaska

Definition of generation rules in QReal

Graduation Thesis

Admitted for defence.

Head of the chair:

Professor A. N. Terekhov

Scientific supervisor:

Senior lecturer Y. V. Litvinov

Reviewer:

Senior lecturer D. V. Luciv

Saint-Petersburg
2014

Оглавление

Введение	4
1) Постановка задачи.....	6
2) Обзор	7
2.1) Обзор существующих решений.....	7
2.1.1) MetaEdit+	7
2.1.2) Microsoft Visualization and Modeling SDK	9
2.1.3) Eclipse: Actifsource	11
2.1.4) Средства описания генераторов кода в QReal.....	12
2.2) Обзор используемых решений	15
2.2.1) Среда визуального программирования QReal.....	15
2.2.2) Генерация в QReal:Robots.....	16
2.2.3) Платформа QReal::QUbiq	18
3) Метод задания правил генерации	23
3.1) Общая схема	23
3.2) Описание визуального языка	24
4) Метагенератор	33
4.1) Архитектура решения.....	33
4.2) Обсуждение	34
4.2.1) Другой вариант архитектуры решения	35
4.2.2) Изменение кода генератора	36
5) Апробация	37
Заключение.....	42
Список литературы.....	43

Введение

Программное обеспечение играет в жизни людей всё большую роль, но его разработка является сложным и трудоёмким процессом. При этом довольно часто требуется создание не громоздких сложных систем, а небольших или средних программ, позволяющих решить ту или иную задачу. Поэтому всё более популярными становятся те методы и технологии разработки ПО, которые позволяют в той или иной степени упростить этот процесс.

Одним из таких способов упрощения создания программ является визуальное моделирование [1, 2, 3]. Данный метод основан на том, что программы описываются визуальной моделью. Он предполагает проектирование программ при помощи манипуляции объектами некоторого визуального языка. Визуальный язык содержит набор элементов и набор связей между ними. При этом написание конкретной задачи сводится к созданию визуальной диаграммы в специальном редакторе данного визуального языка. Такой способ увеличивает наглядность и уменьшает сложность разработки программ, что позволяет расширить круг программистов, способных решить поставленные задачи.

Существует два вида визуальных языков: общецелевые (General-Purpose Language, GPL) и предметно-ориентированные (Domain-Specific Language, DSL). Первый вид визуальных языков (GPL) позволяет описать любую программу (или, по крайней мере, очень широкий класс программ), однако из-за этого обладает большой сложностью и громоздкостью итоговых программ, описанных на этих языках. Предметно-ориентированные языки же, в свою очередь, ориентированы на конкретный круг задач в некоторой области. Тем самым DSL содержат только необходимые для конкретной области элементы и связи, что позволяет создавать более компактные и наглядные программы, более приближенные к решаемой задаче, а значит более понятные специалистам в предметной области, не обладающим навыками программирования. Следовательно, данный подход значительно упрощает создание программ по сравнению с GPL. Однако очевидным минусом DSL является то, что для каждой новой предметной области или же задачи требуется новый визуальный язык, что может быть довольно трудозатратно при ручном кодировании.

Для удобного и быстрого создания программ с помощью визуального моделирования используются CASE-системы, которые состоят из редактора одного или нескольких визуальных языков и дополнительных инструментов работы с языком или диаграммой, например, средств генерации кода, визуальной интерпретации программы, методов трансформации моделей и т.д. Такие системы могут

поддерживать несколько визуальных языков, и GPL, и DSL.

Также существуют и DSM-платформы (также называемые metaCASE-системами), которые позволяют создавать сами визуальные языки при помощи парадигмы визуального моделирования. Они особенно полезны для быстрого и удобного создания DSL для решения конкретных задач. Такие системы в целом не отличаются от обычных CASE-систем, однако в качестве поддерживаемого языка metaCASE-системы содержат специальный визуальный язык – метаязык, который позволяет описывать сущности и связи требуемого языка. Тем самым для создания нового визуального языка программисту необходимо формально описать синтаксис этого языка, используя метаредактор, реализующий вышеупомянутый метаязык. После этого можно будет сгенерировать код, реализующий редактор этого языка. При этом некоторые metaCASE-системы также содержат дополнительные специализированные визуальные и/или текстовые языки и различные инструменты, которые, в частности, позволяют для каждого создаваемого визуального языка задавать:

- правила генерации текстового кода по диаграмме, описанной на данном языке;
- формальные правила проверки ограничений на использование языка (а именно: ограничения, проверяемые при создании диаграммы, а также ограничения, проверяемые во время выполнения программы);
- семантику интерпретации данного языка;
- правила трансформации диаграмм на данном языке.

Стоит отметить, что многие из этих дополнительных возможностей metaCASE-систем являются достаточно важными и их наличие может сильно сказываться на удобстве работы с такими системами. В частности, очень часто необходимо не только само формальное описание программы с помощью визуального языка, но и отображение этой диаграммы в некоторый код на текстовом языке, который можно потом запустить и получить результаты. Однако довольно трудоёмко каждый раз заново писать соответствующий генератор кода для каждого нового визуального языка. Поэтому полезно (и, возможно, необходимо) в metaCASE-системе иметь удобный способ формального задания подобных правил генерации кода.

1) Постановка задачи

В данной дипломной работе решается задача разработки и реализации способа задания правил генерации текстового кода по диаграмме на визуальном языке в metaCASE-системе QReal.

Таким образом, в рамках данного диплома надо было сделать следующее.

1). Сделать обзор существующих решений данной задачи в других подобных системах, например, в MetaEdit+, Microsoft Visualization and Modeling SDK и других. Также необходимо рассмотреть существующее решение этой проблемы в самой системе QReal и предложить способы улучшения.

2). Придумать метод задания правил генерации, то есть удобный способ формального описания генераторов, основываясь на рассмотренных существующих решениях. А именно надо разработать визуальный язык, предназначенный для задания правил генерации и использующий текстовые куски целевого кода.

3). Разработать генератор из созданного языка в текстовый код в среде QReal, который мы далее будем называть «метагенератором» (т.е. генератором, позволяющем сгенерировать код конкретного генератора визуального языка по диаграмме на разработанном языке).

4). Апробировать итоговое решение на существующих в QReal визуальных языках. Например, применить решение к структурному языку – QUbiq, позволяющему создавать простые мобильные приложения, а точнее к его подязыку для описания экранных форм создаваемого приложения.

Для этого языка написан вручную свой генератор, однако в данной работе для него надо описать все необходимые правила генерации и сгенерировать соответствующий генератор кода для этого визуального языка.

2) Обзор

2.1) Обзор существующих решений

2.1.1) MetaEdit+

MetaEdit+¹ является metaCASE-системой, которая разрабатывается компанией MetaCase [4]. Являясь metaCASE-системой, она позволяет:

- использовать предметно-ориентированные (DSM) решения, то есть разрабатывать программные системы, используя некоторый DSL;
- создавать собственные DSM-решения, то есть визуальные языки и набор дополнительных инструментов для удобной работы с ними.

MetaEdit+ состоит из двух подсистем:

- MetaEdit+ Workbench – инструмент для разработки языков моделирования и генераторов;
- MetaEdit+ Modeler – среда разработки, предоставляющая полную функциональность инструмента моделирования, при этом позволяя работать с собственными редакторами языков моделирования, генераторами кода и документации, созданных при помощи MetaEdit+ Workbench.

У рассматриваемой системы есть ряд особенностей [11], а именно:

- возможность использования сразу нескольких DSL во время разработки сложных систем;
- многоплатформенность, то есть возможность работать с почти любой операционной системой;
- возможность работать одновременно с несколькими проектами, содержащими несколько моделей сразу;
- поддержана интеграция моделей с другими подобными системами, такими как Eclipse и Microsoft Visual Studio;
- наличие большого набора инструментов для работы с языками моделирования:
 - различные редакторы (для создания, изменения и удаления моделей, а также для указания определённых взаимосвязей между разными моделями);
 - различные браузеры (для отображения структуры моделей);
 - репозиторий (для хранения информации о языках моделирования и

¹ Домашняя страница MetaEdit+, URL: <http://www.metacase.com/> (дата обращения: 18.05.2014)

моделях, в том числе о всех их элементах и свойствах);

- генераторы (для проверки корректности созданных моделей, а также для генерации по моделям соответствующего текстового кода или документации);
- возможность разрабатывать собственные и изменять существующие генераторы;
- возможность импортировать и экспортировать метамодели во внутренний формат системы (MXT), основанный на стандарте XML;
- возможность сгенерировать документацию к проекту;
- и другие (при этом система активно развивается).

Теперь перейдём к возможностям генераторов кода в MetaEdit+. Система содержит набор встроенных генераторов, которые могут быть использованы и которые поддерживают различные текстовые языки программирования, такие как C++, CORBA IDL, Java, Delphi и др. При этом, среда также позволяет разработчикам создавать собственные генераторы для своих языков моделирования.

Для описания генераторов используется специальный текстовый язык, при написании на котором все не изменяющиеся шаблонные строки должны выделяться с обеих сторон одинарными кавычками. Код на этом языке довольно сложен для понимания, однако такой способ задания правил генерации позволяет разрабатывать сколь угодно сложные генераторы.

Ниже на рисунке (Рис. 1) представлен редактор генератора, в нижней части которого находится пример на текстовом языке задания правил генерации:

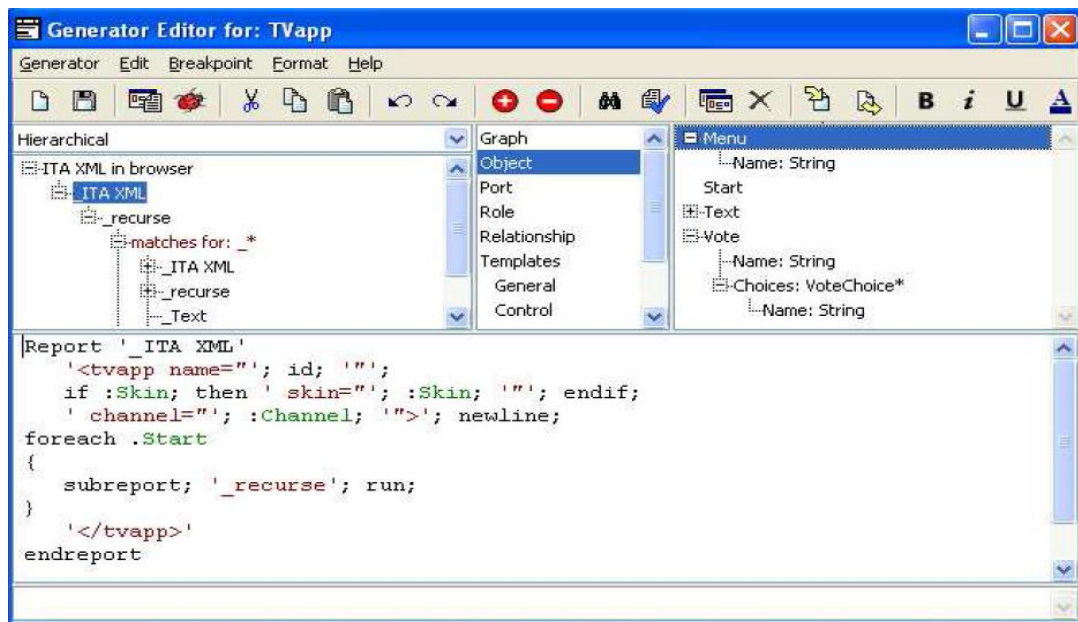


Рис. 1. Редактор генератора (Generator Editor) в MetaEdit+ [5]

При этом система MetaEdit+ позволяет отлаживать разрабатываемые или изменяемые генераторы в специальном редакторе (Рис. 2).

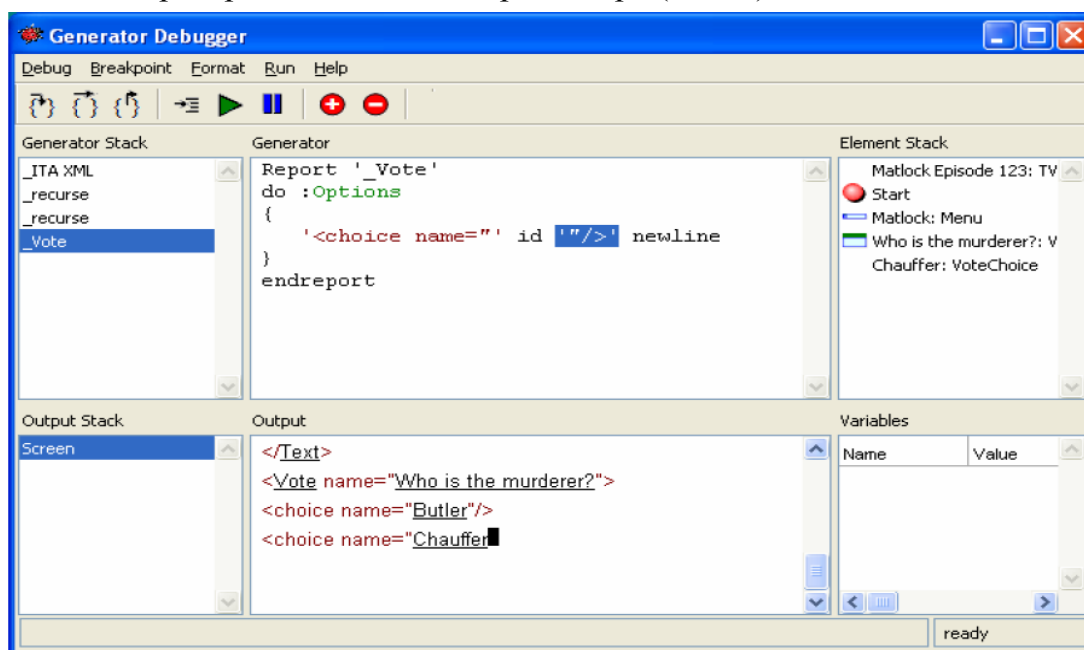


Рис 2. Отладчик генератора (Generator Debugger) в MetaEdit+ [5]

2.1.2) Microsoft Visualization and Modeling SDK

Microsoft Visualization and Modeling SDK (VMSDK)² – это набор средств, позволяющий создавать собственные визуальные языки моделирования, соответствующие им графические редакторы, а также дополнительные вспомогательные инструменты для удобной работы с ними. Тем самым, VMSDK является DSM-платформой, разработанной как часть одного из подходов к созданию программного обеспечения, а именно Software Factories³, который предназначен для увеличения скорости разработки ПО при минимальных затратах.

VMSDK состоит из трёх основных компонентов:

- «мастер проектов», который позволяет создавать новые проекты для работы с существующими предметно-ориентированными языками (DSL);
- специальный редактор («дизайнер»), который позволяет формально описывать метамодель нового визуального языка;
- генераторы кода, которые по модели программы на визуальном языке и по заранее описанному разработчиком шаблону правил генерации генерируют соответствующий текстовый код.

² Visualization and Modeling SDK – Domain-Specific Languages, URL: [http://msdn.microsoft.com/ru-ru/library/bb126259\(v=vs.100\).aspx](http://msdn.microsoft.com/ru-ru/library/bb126259(v=vs.100).aspx) (дата обращения: 18.05.2014)

³ Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools, URL: <http://msdn.microsoft.com/en-us/library/ms954811.aspx> (дата обращения: 18.05.2014)

Ниже приведём некоторые характеристики рассматриваемой платформы [11]:

- для создания метамodelей DSL необходимо описать и абстрактный (при помощи диаграмм классов UML), и конкретный синтаксис языка, при этом разработчику предоставляется широкий набор графических инструкций;
- существует широкий набор различных шаблонов DSL, которые можно переиспользовать при разработке собственных языков;
- поддержана возможность задавать ограничения (то есть некоторые правила, которым должны удовлетворять модели программ и/или редакторы языков) при помощи фрагментов кода на языке программирования C#;
- возможность задавать правила генерации кода для визуальных языков при помощи специального текстового языка;
- возможность генерации кода и XML-файлов согласно описанным правилам;
- разработчик обладает большими возможностями для ручной настройки моделей и редакторов;
- вся дополнительная функциональность, которую разработчик указывает для моделей программ или языков вручную на текстовом языке, создаётся в виде частичных классов, что гарантирует сохранение этого кода при последующих изменениях моделей (с регенерацией кода);
- разработанный при помощи VMSDK редактор языка (вместе с дополнительными инструментами) можно открывать в MS Visual Studio и полноценно работать с ним;

Как говорилось ранее, VMSDK содержит особые генераторы кода, которые по заданным правилам могут генерировать исходный код программы, описанной на некотором DSL. Правила генерации задаются в виде отдельного фрагмента кода (шаблона) на текстовом языке T4⁴, представляющем собой некоторую комбинацию статических текстовых блоков и управляющей логики, согласно которой обрабатываются эти блоки (например, «извлечение значения свойства модели»). Логика описывается при помощи языков программирования Visual Basic или C#, а сами управляющие конструкции выделяются символами <#, #>. При этом в шаблоне необходимо явно указать, какой язык для описания логики будет использован. Например, при выборе языка C# надо в начале шаблона написать следующую строку:

```
<#@ template language="C#" #>
```

Ниже представлен пример шаблона генератора на языке T4 (Рис. 3).

⁴ Code Generation and T4 Text Templates, URL: <http://msdn.microsoft.com/en-us/library/bb126445.aspx> (дата обращения: 18.05.2014)

```

<#@ template debug="false" hostspecific="false" language="C#" #>
<#@ output extension=".cs" #>
<# var properties = new string [] {"P1", "P2", "P3"}; #>
class MyGeneratedClass {
<#
    foreach (string propertyName in properties)
    { #>
        private int <#= propertyName #> = 0;
    } #>
}

```

Рис. 3. Пример шаблона, описывающего генератор, на языке T4 [10]

Однако отметим, что при таком способе задания правил генерации по модели программы будет генерироваться соответствующий целевой код только на языках Visual Basic или C#, а не на любом текстовом языке программирования.

2.1.3) Eclipse: Actifsource

Eclipse⁵ – свободная кроссплатформенная интегрированная среда разработки программного обеспечения, которая предоставляет широкий набор возможностей для разработчика:

- визуальное моделирование;
- генерация текстового кода по моделям программ;
- рефакторинг описанных диаграмм и моделей;
- и многое другое.

Actifsource⁶ является предметно-ориентированным инструментом, реализованным в качестве подключаемого дополнительного модуля к среде программирования Eclipse. Рассматриваемая среда позволяет формальным образом описывать правила генерации для разрабатываемых визуальных языков. На основании таких правил среда позволяет генерировать текстовый код программы, описанной на некотором визуальном языке.

В Actifsource, аналогично Microsoft Visualization and Modeling SDK, в качестве описания генератора выступает некоторый шаблон, написанный в редакторе. Для написания шаблона также используется особый текстовый язык, в котором комбинируются фрагменты целевого кода и логика, выделенная управляющими конструкциями. Однако используемый редактор умеет анализировать вводимый текст, что позволяет полностью убирать управляющие символы. Это значительно упрощает написание генератора, а также увеличивает общую наглядность кода.

⁵ Main Page Eclipse, URL: http://wiki.eclipse.org/Main_Page (дата обращения: 18.05.2014)

⁶ Домашняя страница Actifsource, URL: <http://www.actifsource.com/> (дата обращения: 18.05.2014)

Однако отметим, что такой способ задания генераторов позволяет описывать только создание прототипов программного обеспечения, а не полную генерацию кода.

Ниже на рисунке (Рис. 4) представлен пример описания генератора:

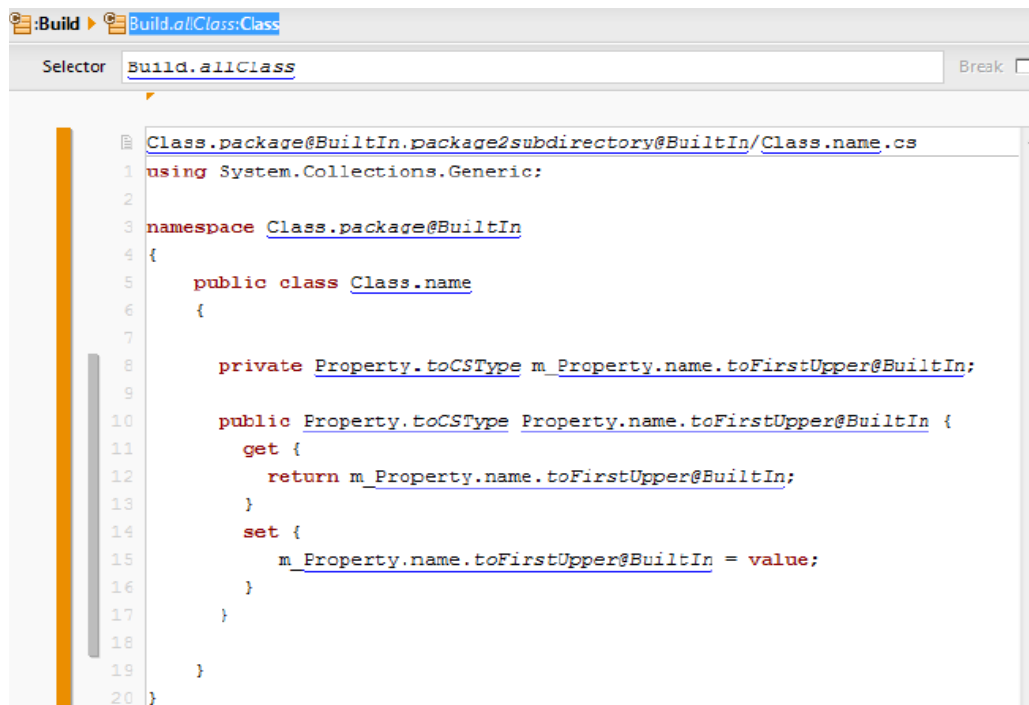


Рис. 4. Пример описания генератора на текстовом языке в Actifsource [10]

2.1.4) Средства описания генераторов кода в QReal

Несколько лет назад на кафедре системного программирования был разработан некоторый способ описания генераторов кода в системе QReal [10]. Для этого был создан текстовый язык описания генераторов Genу и реализован соответствующий интерпретатор и редактор этого языка.

Данное решение является усреднённым вариантом между аналогичными решениями других metaCASE-систем. С одной стороны, предложенное решение явно содержит скелет итоговой программы, что увеличивает наглядность и понятность генерируемого кода. А с другой стороны, оно позволяет использовать управляющие конструкции, что расширяет класс визуальных языков, для которых можно будет задать таким способом правила генерации.

Очевидно, что основой данного решения является сам язык, который позволяет описывать генераторы визуальных языков. Поэтому опишем именно его. Предложенный текстовый язык Genу предполагает ряд основных концепций:

- принцип прямой передачи кода

Этот принцип означает, что любой текст, не обрамлённый управляющими конструкциями, сразу передаётся без изменений в результирующий поток, то есть в

итоговый код программы по описанной диаграмме. Это позволяет увеличить наглядность генерируемого кода, а тем самым и прозрачность разработки генераторов в целом.

- принцип текущего объекта

Данный принцип означает, что при работе с «текущим объектом» в генераторе не требуется явно указывать его имя. При этом «текущим объектом» является текущий узел при обходе графа, построенного по диаграмме, которую мы хотим отобразить в текстовый код.

- система заданий

Этот принцип означает, что каждый файл, написанный на языке Geny, считается именованным заданием, который в свою очередь может содержать внутри себя другие задания. Такой подход позволяет создавать многомодульные программы, что еще больше упрощает описание генераторов.

- управляющие строки (начинающиеся со специальных символов #!)

Эти строки не передаются без изменений в результирующий поток, а служат для того, что выполнить ту или иную служебную операцию:

- `toFile fileName` — перенаправить генерацию в файл с именем `fileName`;
- `foreach` — аналогична принятому во многих других языках смыслу этой конструкции `foreach`, а именно обойти все объекты из переданного списка и выполнить тело цикла;
- `saveObj markName` — сохранить «текущий объект» по имени `markName`;
- `switch/case` — аналогична стандартному `switch`, однако позволяет сравнивать некоторые значения с атрибутами текущего объекта;
- `if %propertyName% operation %propertyValue%` — сравнить значение поля `propertyName` у «текущего объекта» со значением `propertyValue`. При этом в качестве `operation` могут выступать только `=`, `!=` и `contains`;
- `{, }` — отделить блоки кода.

- управляющая конструкция внутри контекста

Этот принцип позволяет при помощи символа `@` обращаться к значениям атрибутов «текущего объекта» и выводить их в результирующий поток. Например, строка `@@name@@` означает, что необходимо обратиться к полю `name` «текущего объекта» и взять его значение.

- система меток объектов

Данный принцип является расширением предыдущего, используя управляющую конструкцию `saveObj` для сохранения «текущего объекта» и последующего обращения к его полям.

Впоследствии был также разработан редактор языка Geny, упрощающий написание генератора при помощи замены ряда управляющих конструкций и меток на особый цвет и шрифт. Например:

- управляющие строки в данном редакторе выделяются не при помощи #!, а при помощи фонового цвета, что позволяет немного уменьшить код, увеличить наглядность и визуально отделить код «прямой» передачи кода (т.е. целевой код) от «непрямой» (т.е. код описания генератора);
- выделение имён полей объектов теперь происходит не при помощи @@@, а при помощи жирного шрифта;
- автоматическое форматирование кода на языке Geny;
- автоматическое распознавание блоков кода, что позволяет больше не писать строки «начала» (#!{) и «конца» (#!}) блоков;
- возможность скрывать управляющие строки, что опять же повышает наглядность генерируемого кода.

Ниже на рисунке (Рис. 5) представлен пример описания генератора на текстовом языке Geny в соответствующем редакторе:

```
1 Task JavaClass
2 / Produce Java class from repo
3
4 foreach Class in elementsByType(Class)
5     toFile name.java
6 class name {
7     foreach MethodsContainer in children
8         foreach Method in children
9             methodVisibility methodReturnType
10            methodName( @#!task MethodParameters@@ ) {
11            }
12        }
13    }
14
15    foreach FieldsContainer in children
16        foreach Field in children
17            fieldVisibility fieldType fieldName;
18        }
19    }
20 }
21 }
22 }
```

Рис. 5. Пример описания генератора на языке Geny в редакторе [10]

Однако у данного решения есть ряд недостатков, ввиду которых и была поставлена задача реализовать немного другой способ задания правил генерации в среде QReal. А именно:

- основой этого решения является текстовый язык. Хотя в нём и используются более-менее привычные элементы, для написания кода на этом языке требуется, естественно, знание этого языка. Однако какого-либо более подробного описания на данный момент нет и не предполагается. Тем более язык этот – текстовый, что менее наглядно и понятно, чем некоторый визуальный аналог;
- данное решение не было до конца реализовано. При этом это решение не было перенесено в итоговый вариант системы QReal, а дальнейшее развитие этого решения не планируется.

2.2) Обзор используемых решений

2.2.1) Среда визуального программирования QReal

На кафедре системного программирования Математико-Механического факультета СПбГУ уже несколько лет разрабатывается metaCASE-система QReal⁷.

С одной стороны, данная система позволяет на некотором визуальном языке из существующего в среде набора языков создавать диаграммы в соответствующем редакторе, тем самым разрабатывать программное обеспечение, описывать архитектуру решений и другое [6, 7]. При этом в QReal существуют дополнительные возможности для работы с языками и диаграммами. Например, среда позволяет:

- задавать и применять рефакторинги к описанным программам (т.е. к диаграммам) на некотором визуальном языке;
- визуально отлаживать написанную программу;
- и др.

С другой стороны, QReal также позволяет самостоятельно создавать новые визуальные языки и соответствующие редакторы [6, 7]. Сначала необходимо описать модель этого языка в специальном редакторе (метаредакторе) при помощи предназначенного для этого визуального языка (метаязыка). При этом метаязык позволяет формально описывать синтаксис создаваемого языка при помощи основных элементов «узел» и «связь». А вот по такому описанию уже можно сгенерировать соответствующий редактор и автоматически подгрузить его в среду. Далее можно работать с новым визуальным языком, как и с другими из набора. При этом в системе QReal для этого случая тоже есть свои дополнительные полезные инструменты. Например, среда предоставляет возможность задавать ограничения на создаваемый язык. Это достигается наличием специального визуального языка и соответствующего

⁷ Домашняя страница QReal, URL: <http://qreal.ru/> (дата обращения: 18.05.2014)

редактора, позволяющего формально описывать необходимые ограничения наглядным образом, а также особого механизма проверки этих ограничений при создании диаграммы на некотором визуальном языке (для которого эти ограничения и создавались).

Помимо этого для ряда визуальных языков системы QReal существуют свои генераторы, которые по диаграмме на соответствующем визуальном языке позволяют сгенерировать код на некотором целевом текстовом языке. На данный момент для создания генератора для нового языка необходимо вручную написать код этого генератора с использованием исходных кодов системы QReal. При этом было замечено, что многие генераторы имеют схожие куски кода, что требует постоянного раскопирования кода. Также на данный момент в среде QReal используется система шаблонов для написания генераторов кода. Это позволяет в разных файлах (которые и называются шаблонами) писать отдельные куски кода на целевом текстовом языке, помечая маркерами места для вставки кода, полученного по диаграмме. Далее рукописный генератор просто использует эти шаблоны, подставляя вместо маркеров необходимый текст. Такой подход более нагляден и позволяет «видеть» структуру программы, которую будет генерировать создаваемый генератор. Однако самостоятельная разработка генераторов еще остается сложной задачей. Именно поэтому в рамках данной работы необходимо поддержать возможность более наглядного и простого создания генераторов кода по диаграмме.

2.2.2) Генерация в QReal:Robots

QReal:Robots

На кафедре системного программирования Математико-Механического факультета СПбГУ на базе metaCASE-системы QReal разрабатывается система визуального программирования роботов QReal:Robots [8].

Рассматриваемая среда позволяет описывать программы поведения роботов. При этом программа представляет собой набор визуальных блоков, соединённых друг с другом связью, показывающей передачу потока управления. Блоки, в свою очередь, выбираются из существующего набора элементов, каждый из которых означает некоторую элементарную команду (например, «включить моторы вперёд/назад», «ждать срабатывание датчика касания», «ждать по таймеру» и т.п.).

Платформа также поддерживает дополнительные возможности, необходимые для разработки программ для роботов, такие как:

- генерация целевого текстового кода по визуальному описанию программы (при этом поддерживает несколько текстовых языков);

- управление роботом с компьютера согласно описанной программе по Bluetooth и USB-интерфейсу;
- исполнение программы без использования реального робота, а именно на двумерной модели робота, отображаемой на экране компьютера в специальном окне;
- возможность использования математических выражений, переменных и показаний датчиков во время вычислений;
- визуальная отладка программы, то есть подсветка исполняемого на данный момент времени блока;

Отметим, что рассматриваемая среда может поддерживать различные конструкторы роботов и различные текстовые языки, в которые необходимо генерировать программу. Однако для подобного расширения необходимо каждый раз дополнительно создавать соответствующий генератор.

Генераторы в QReal:Robots

В рамках платформы QReal:Robots был разработан особый механизм генерации кода [9], который позволяет гораздо быстрее создавать новые генераторы, чем при обычно используемом в QReal подходе. Отметим только, что данный способ всё равно предполагает написание генераторов вручную на текстовом языке C++ в рамках системы QReal.

Ранее все генераторы разрабатывались каждый раз «с нуля», что не очень удобно, так как много частей кода во многих случаях либо отличается не сильно, либо не отличается вовсе. Рассматриваемый же механизм предлагает большой набор готовых вспомогательных классов, которые можно переиспользовать. Следовательно, разработчику необходимо написать только определённые компоненты разрабатываемого генератора, а именно:

- главный класс плагина

В нём описывается только внешняя часть генератора (например, пункты меню, некоторая метайнформация и т.д.). Также он может наследоваться от существующего класса, в котором реализована большая часть взаимодействия со средой QReal. В этом случае почти ничего дописывать не придётся.

- генераторы «потока управления» (то есть семантического дерева, «скелета» программы)

Данная часть отвечает за генерацию не готового кода программы, а только объектного дерева, которое только соответствует коду и отражает его структуру. В наиболее типичных случаях эта компонента будет просто наследоваться от

стандартного класса библиотеки, предлагаемой в рамках данного механизма, без каких-либо дополнений. Далее при обходе получаемого дерева вызываются «простые» генераторы.

- «простые» генераторы

Они отвечают уже за конкретный готовый код, который необходимо генерировать для некоторого одного блока или одной конструкции в семантическом дереве. Опять же, для всех существующих блоков уже реализованы соответствующие «простые» генераторы, которые можно просто переопределить или переиспользовать. При этом генераторы на самом деле используют шаблоны, поэтому при реализации генерации для другого текстового языка достаточно только изменить эти шаблоны, не меняя при этом соответствующие «простые» генераторы. Для новых блоков языка необходимо самостоятельно написать код генерации.

- конвертеры

Компоненты, отвечающие за преобразование одного строкового представления данных в другое. В рассматриваемой библиотеке также есть набор стандартных конвертеров, которые можно использовать.

- другие вспомогательные компоненты

Эта часть предполагает написание дополнительных компонентов, реализующих специфичную логику, которую не удалось покрыть «простыми» генераторами и конвертерами.

Более подробно о создании новых генераторов в среде программирования QReal:Robots можно прочитать в соответствующей инструкции [9].

2.2.3) Платформа QReal::QUbiq

На кафедре системного программирования в качестве курсовой работы в рамках среды программирования QReal была разработана платформа QReal::QUbiq [13], предназначенная для создания мобильных приложений. Рассматриваемая платформа содержит:

- редактор для формального описания разрабатываемой программы при помощи соответствующего визуального языка;
- генератор, который по описанной диаграмме генерирует текстовый код мобильного приложения, написанного под платформу Ubiq Mobile⁸;

⁸ Домашняя страница UbiqMobile: <http://ubiqmobile.com/> (дата обращения: 18.05.2014)

Платформа Ubiq Mobile

На кафедре системного программирования, но при участии студентов и аспирантов из СПбГУ и СПбУ ИТМО также разрабатывается платформа Ubiq Mobile [12], предназначенная для разработки кросс-платформенных распределенных мобильных сервисов. При этом одной из основных целей является расширение круга программистов, способных самостоятельно быстро и эффективно создавать программные системы для мобильных устройств, что достигается путём сокрытия от разработчика определённой специфики при проектировании подобных систем. В то же время платформа обеспечивает работу созданных на её основе приложений в условиях любых мобильных сетей.

Также платформа поддерживает широкий постоянно пополняющийся круг разнообразных мобильных устройств, не требуя при этом от программиста специфичных знаний работы с ними, предоставляя ему возможность прозрачной разработки. Однако данная платформа ориентирована не на всевозможные программы, работающие с Интернетом, а только на приложения, являющиеся компонентами больших распределенных систем, то есть которые характеризуются сложной бизнес-логикой сервера и мультиплатформенностью [13].

Для платформы Ubiq Mobile была реализована архитектура, включающая в себя следующие компоненты:

- сервер, на котором выполняется вся бизнес-логика сервисов;
- набор мобильных клиентов, выступающих простейшими графическими терминалами (то есть это экземпляры клиентской программы, которые запущены на мобильном устройстве).

При этом вся работа сервисов происходит именно на сервере, в то время как мобильные устройства только получают результаты, которые надо отобразить. Программист, в свою очередь, работает не с самим мобильным устройством, а с некоторым виртуальным хостом на сервере. А мобильные терминалы воспринимаются только как некоторые источники внешних событий, которые обрабатываются приложением.

Отметим, что одновременно на сервере могут работать несколько мобильных приложений, а значит, конечному пользователю предоставляется сразу список сервисов, с которыми он может работать через свой идентификационный номер, полученный им после регистрации на сервере.

Тем самым, для разработки современных мобильных систем со сложной логикой поведения при помощи платформы Ubiq Mobile программисту требуется обладать только базовыми знаниями языка C# и библиотек API самой платформы.

Визуальный язык QUbiq

Для разработки мобильных приложений со сложной логикой поведения в среде QReal был разработан визуальный язык [13], который позволяет формально описывать пользовательский интерфейс и нетривиальную логику поведения программы. Таким образом, разработанная модель такого визуального языка (имя модели: «qUbiqMetamodel») состоит из трёх основных частей (Табл. 1.):

Имя	Описание
QUbiq Presentation Editor	<ul style="list-style-type: none">• «Язык задания представлений», который позволяет описывать экранные формы приложения, а также правила перехода между ними;• На основании этого языка можно описывать рабочие прототипы мобильных приложений;• Корневой диаграммой является «диаграмма представлений» (то есть «qUbiqPresentationDiagram»).
QUbiq Logic Editor	<ul style="list-style-type: none">• «Язык задания логики», который позволяет описывать логику поведения обработчиков различных событий (например, переход на другую экранную форму, нажатие кнопки и т.п.);• Корневой диаграммой является «диаграмма обработчика» (то есть «qUbiqLogicDiagram»).
QUbiq Condition Editor	<ul style="list-style-type: none">• «Язык задания условий», который является подмножеством языка «qUbiqLogicEditor» и позволяет в отдельных диаграммах описывать более сложные логические условия;• Корневой диаграммой является «диаграмма условия» (то есть «qUbiqConditionDiagram»);• Описанные «диаграммы условий» могут быть использованы в одной или нескольких диаграммах обработчиков, что повышает наглядность разрабатываемого приложения.

Таблица 1. Основные части визуального языка QUbiq

Новое мобильное приложение на рассматриваемом визуальном языке описывается следующим образом:

1). В первую очередь необходимо создать одну «диаграмму представлений», которая выступает в качестве основы программы. В свойствах этой диаграммы указываются основные параметры разрабатываемого приложения, в том числе его имя и абсолютный путь до папки, в которую необходимо его сгенерировать. При этом

наличие этой диаграммы уже обеспечивает генерацию прототипа искомого приложения.

2). Далее необходимо на созданной «диаграмме представлений» описать набор экранных форм, используя предназначенные для этого элементы визуального языка «QUbiq Presentation Editor». Для каждой экранной формы задаётся её внешний вид, а именно то, какие в ней расположены элементы, то есть кнопки, списки, картинки, сетки или тексты. У каждого элемента пользовательского интерфейса (т.е. кнопки, списка или сетки) и у каждой формы, в свою очередь, можно задать имя обработчика, который будет вызываться при нажатии на него или при переходе на форму соответственно.

3). На той же «диаграмме представлений» можно указать простые правила перехода между формами. Такие правила задаются при помощи специального элемента-связи визуального языка (а именно «transitionLink»). Для того чтобы указать, что при нажатии на некоторый контрол (то есть на элемент пользовательского интерфейса) необходимо сменить отображаемую экранную форму, необходимо этой связью соединить данный контрол с формой, на которую требуется перейти.

4). Создать все необходимые «диаграммы обработчиков», имена которых были указаны в свойствах элементов на «диаграмме представлений». Также, возможно, потребуется создание и нескольких «диаграмм условий», которые будут использовать конкретные обработчики.

5). На «диаграмме представления» можно также описать все необходимые переменные, которые по соглашению являются глобальными и которые, следовательно, можно будет использовать во всех диаграммах разрабатываемого приложения. При этом типом переменной может являться целое число, строка, список или сетка (аналог двумерного массива).

Генератор QUbiq

После того, как разработчик в редакторе формально описал новое мобильное приложение на визуальном языке QUbiq, у него есть возможность сгенерировать текстовый код программы в рамках платформы Ubiq Mobile. Однако фактически генерируется не всё приложение целиком, а только его прототип на основании «диаграммы представлений» (то есть по сути генератор для «QUbiq Presentation Editor»), который можно собрать, запустить и проверить работоспособность.

Таким образом, при помощи разработанного генератора по описанию программы можно получить «скелет» искомого приложения вместе с кодом следующих частей:

- описание всех экранных форм с корректным их заполнением;
- правила перехода между описанными экранными формами;
- заглушки для всех обработчиков, указанных в контролах и экранных формах;
- описание всех переменных.

3) Метод задания правил генерации

В качестве метода задания правил генерации в среде QReal было выбрано использование специального визуального языка. В рамках данной дипломной работы был разработан и реализован такой визуальный язык с использованием средств metaCASE-системы QReal. То есть разработанный язык был формально описан на метаязыке в метаредакторе, а далее по данной модели был автоматически сгенерирован редактор для него.

При этом отметим, что на данный момент разработанный язык позволяет создавать генератор не для всевозможных визуальных языков, а только для «структурных» (т.е. для тех, которые только описывают структуру программы и не содержат работу с потоками данных или управления). Для описания же генераторов для «поведенческих» языков (т.е. для тех, которые содержат работу с потоками данных и потоками управления) необходимо данный язык несколько дополнить, чтобы можно было задавать правильный порядок обхода модели на визуальном языке.

3.1) Общая схема

Язык задания правил генерации позволяет описывать формальным образом генератор текстового языка для некоторого визуального языка. Если метамодель состоит из нескольких разных языков, то правила генерации описываются отдельно для каждого языка. Описываемая модель языка должна состоять из «корня», содержащего одну или несколько диаграмм. Каждая диаграмма в свою очередь строится из визуальных элементов языка, которые позволяют задавать определенные правила генерации. Основным элементом для задания правил генерации является элемент, позволяющий задавать тип элемента визуального языка (для которого создается генератор) и соответствующий ему текстовый код, который должен генерироваться описываемым генератором при обходе элемента указанного типа. Также данный язык позволяет задавать:

- правила генерации для всех «детей» (в смысле вложенности) элемента некоторого типа в цикле, для связей, а также для всего приложения в целом;
- текстовые шаблоны, генерирующиеся в файл или же в метку с определенным именем;
- конвертеры, которые позволяют перевести одно строковое представление данных в другое.

Стоит отдельно уточнить, что мы понимаем под «текстовыми шаблонами». В данной работе шаблоном является некоторый конкретный код, написанный на

целевом текстовом языке программирования (программу на котором и должен генерировать по диаграмме разрабатываемый генератор визуального языка). Такой код в шаблоне может содержать набор меток, выделенных специальными символами (например, «@@» или «##»), которые впоследствии при генерации будут заменены на сгенерированный код в соответствии с этими метками. При этом метка, выделенная с обеих сторон символом «@@», означает, что вместо неё в текст просто будет вставлен некоторый кусок кода. Метка же, выделенная с обеих сторон символом «##», означает, что вместо неё в код будет вставлено обращение к значению поля с указанным названием текущего элемента, т.е. элемента, для которого этот текстовый шаблон написан. Например, метка вида «##name##» будет заменена на значение поля «name» текущего элемента. Если шаблон описан не в рамках некоторого элемента, а в рамках всего приложения в целом, тогда текущим элементом по умолчанию является корневой (т.е. основной) элемент визуального языка, для которого разрабатывается генератор.

3.2) Описание визуального языка

Теперь подробно опишем каждый элемент языка. В приведённой ниже таблице для каждого элемента указаны его внутреннее имя (name), имя, отображаемое пользователю (displayName), и его общее описание, то есть назначение и различные свойства элемента. Отметим, что упомянутыми полями name и displayName обладают почти все элементы всех визуальных языков, созданных с использованием встроенного в QReal метаредактора. Значение поля name – это логическое имя элемента, которое не отображается пользователю, но фактически используется внутри самой системы во всех инструментах и редакторах. В отличие от него, значение поля displayName – это физическое имя элемента, которое видно пользователю и используется только им при обычной работе с данным элементом визуального языка (то есть при создании программ на визуальном языке, а не при создании дополнительных инструментов для самого языка).

Таким образом, ясно, что и при задании правил генерации для некоторого визуального языка необходимо использовать фактические имена элементов, т.е. значения полей name, хотя они явно не отображаются пользователю. Ввиду этого для данного языка было решено указать в таблице значения обоих полей с именем. Также стоит отдельно упомянуть, что метамодель нашего визуального языка задания правил генерации называется «GeneratorsMetamodel», а сам язык – «GeneratorEditor» (отсюда и название соответствующего редактора «Generator Editor»).

Ниже приводится таблица (Таблица 2) с описанием элементов визуального

языка задания правил генерации:

№	displayedName / name	Описание элемента	Свойства элемента	Примечания
Root Diagrams (т.е. корневые элементы)				
1	Generator Root / GeneratorRoot	Корневой элемент модели задания правил генерации. Позволяет задавать правила генерации на некоторый визуальный язык некоторой метамодели визуальных языков.	<ul style="list-style-type: none"> • Name – собственное имя модели генератора; • Metamodel Name – имя метамодели, в которой находится визуальный язык, для которого создаются правила генерации; • Language Name – имя визуального языка, для которого разрабатывается генератор; • Language Type – тип указываемого визуального языка. Есть три варианта: <ul style="list-style-type: none"> ➤ <u>structural</u> («структурный» язык) ➤ <u>behavioral</u> («поведенческий» язык) ➤ <u>other</u> (особый тип языка, который нельзя однозначно отнести к двум другим) • Node Name – имя корневого элемента 	<ul style="list-style-type: none"> • «Корневой» элемент – это элемент, который указывается в свойстве «nodeName» элемента «MetamodelDiagram» в метаредакторе при создании визуального языка; • Для задания правил генерации для некоторого визуального языка необходимо, чтобы его «корневой» элемент содержал свойства, отвечающие за «имя создаваемой программы» и «путь до папки, в которую будет сгенерирована программа»; • Все поля данного элемента должны быть обязательно заполнены.

			<p>указываемого визуального языка;</p> <ul style="list-style-type: none"> • programName Property name – имя свойства корневого элемента, в котором указывается имя генерируемой программы по диаграмме визуального языка; • toGeneratePath Property name – имя свойства корневого элемента, в котором указывается абсолютный путь до папки, в которую будет генерироваться программа; • Directory to generate – абсолютный путь до папки, в которую будет генерироваться код генератора; • Path to QReal Source – абсолютный путь до папки с исходниками QReal. 	
--	--	--	---	--

2	Generator Diagram / GeneratorDiagram	Основная диаграмма, на которую непосредственно будут добавляться элементы для задания правил генерации.	<ul style="list-style-type: none"> Name – собственное имя диаграммы. 	Для одного визуального языка может быть описано несколько таких диаграмм.
Basic Elements (т.е базовые элементы задания правил генерации)				
3	Application / ApplicationNode	Элемент для задания общего кода генерируемой программы.		<ul style="list-style-type: none"> Элемент позволяет задавать основные шаблоны описываемого генератора, а также все используемые свойства корневого элемента визуального языка; Текущим элементом данного узла является «корневой» элемент (это нужно для описания шаблонов).
4	Semantic / SemanticNode	Элемент, позволяющий задать, какой код должен генерироваться при обходе элемента указываемого типа	<ul style="list-style-type: none"> Name of Element – тип узла, для которого задается правило генерации; text code – шаблон генерируемого кода; 	<ul style="list-style-type: none"> Для текстового шаблона текущим элементом является элемент, указанный в свойстве «Name of Element»; <u>По умолчанию</u>: код, генерируемый для всех элементов указанного типа сохраняется в метку с именем «@@elements_name@@», где «name» – это значение поля «Name of Element».
5	Semantic For Edge / SemanticForEdge	Элемент, позволяющий задать, какой	<ul style="list-style-type: none"> Name of Element – тип связи, для которой задается 	<ul style="list-style-type: none"> См. «Примечания» у SemanticNode (№ 4); В свойствах «Types of Src

		код должен генерироваться при наличии связи указываемого типа	<p>правило генерации</p> <ul style="list-style-type: none"> • text code – текстовый шаблон генерируемого кода; • Types of Src element – типы элементов-узлов (через запятую), которые должны находится в начале указываемой связи для выполнения данного правила генерации; • Types of Dst element --- аналогично «Types of Src element», но рассматриваются элементы в конце связи; 	<p>element» и «Types of Dst element» можно использовать специальное слово «ANY», означающее любой тип элемента-узла визуального языка</p> <ul style="list-style-type: none"> • В текстовом шаблоне также можно обращаться к свойствам узлов, указанных в свойствах «Types of Src element» и «Types of Dst element», при помощи меток вида «###src.property##» и «###dst. property ##» соответственно, где «property» – это имя некоторого требуемого свойства. • <u>По умолчанию:</u> код, генерируемый для всех элементов указанного типа сохраняется в метку с именем «@ @elements_name_Src Name_DstName@ @», где «name» – это значение поля «Name of Element», а «SrcName» и «DstName» – это значения полей «Types of Src element» и «Types of Dst element» соответственно
6	Template / TemplateNode	Элемент для задания текстового шаблона	<ul style="list-style-type: none"> • text code – шаблон генерируемого 	<ul style="list-style-type: none"> • Должно быть обязательно указано свойство либо «fileName», либо

		конкретного генерируемого кода	кода; <ul style="list-style-type: none"> • fileName – имя файла, в который сохранится описываемый шаблон; • markName – имя метки, в которую сохранится описываемый шаблон; 	«markName», но <u>не оба</u> одновременно. Приоритет у поля «fileName»; <ul style="list-style-type: none"> • Для текстового шаблона текущим элементом является соответствующий текущий элемент «отца» данного узла TemplateNode (т.е. контейнера, в котором находится данный элемент)
7	Foreach / ForeachNode	Элемент, позволяющий задать правила генерации для всех «детей» (в смысле вложенности в контейнер) элемента текущего типа.	<ul style="list-style-type: none"> • fileName и markName – см. соответствующие свойства у «TemplateNode» (№6) вместе с примечаниями 	<ul style="list-style-type: none"> • Для всех «детей» «текущего» элемента их сгенерированный код собирается в одну строку (через «перевод строки») и сохраняется в указанный файл или метку; • «Текущим» элементом является соответствующий текущий элемент того узла, с которым данный элемент соединён при помощи связи SimpleEdge.
Elements for Converter (т.е. сущности для задания новых конвертеров данных) <i><не поддерживаются></i>				
8	Converter / ConverterNode	Элемент, позволяющий описать новый конвертер	<ul style="list-style-type: none"> • fileName и markName – см. соответствующие свойства у «TemplateNode» 	<ul style="list-style-type: none"> • Описывает перевод одного типа данных в другой, и сохраняет этот код в указанный файл или метку;

		данных	(№6) вместе с примечаниями	<ul style="list-style-type: none"> Описывается при помощи элементов с № 9 по №12, соединённых специальной связью «поток управления» «Control Flow» (№ 14); «Текущий элемент» аналогичен текущему элементу у «ForeachNode» (№7)
9	Converter Semantic / converterSemantic	Элемент, позволяющий применить существующий конвертер	<ul style="list-style-type: none"> converter file name – имя файла, в котором написан существующий конвертер 	На вход принимает некоторые данные, применяет указанный конвертер, а результат передает на выход.
10	Converter Foreach / converterForeach	Элемент, позволяющий проходить по переданному списку данных		На вход принимает список данных, выполняет нужные операции (указанные в «детях» данного элемента) и выдаёт на выход изменённый список данных.
11	Merge / converterMerge	Элемент, позволяющий объединить список объектов в один	<ul style="list-style-type: none"> file name of special merger – имя файла, в котором реализован специфичный соединитель данных 	На вход принимает список данных, а на выход выдает один объект, полученный соединением данных полученного списка
12	Read / converterRead	Элемент, позволяющий «читать» значение из указанного поля элемента языка	<ul style="list-style-type: none"> property name – название свойства текущего элемента 	<ul style="list-style-type: none"> На вход ничего не даётся, читается значение указанного свойства текущего элемента, а на выход выдаётся это значение; «Текущим элементом» является

				соответствующий текущий элемент «отца»-контейнера типа «ConverterNode».
Relationship (т.е. связи для описания взаимоотношений между элементарными правилами генерации)				
13	To Edge / SimpleEdge	Простая связь между элементами		Соединяет элементы типа «Semantic» с элементами типов «Foreach» или «Converter».
14	Control Flow / ControlFlow	Связь, означающая «поток управления», необходимая для задания конвертера		Соединяет элементы с № 9 по №12, эмулируя поток передачи данных между элементами.

Таблица 2. Перечень элементов визуального языка задания правил генерации

Отметим, что многие элементы языка задания правил генерации являются контейнерами, то есть внутри этих элементов существует возможность добавлять другие элементы языка. Однако не любой элемент языка может быть внутри таких контейнеров. Ниже приведём таблицу (Таблица 3), в которой указано, какие элементы могут быть внутри каких контейнеров. Остальные же элементы, не указанные в этой таблице, не являются контейнерами.

№	Контейнер	Типы элементов, которые может содержать данный контейнер
1	GeneratorRoot	GeneratorDiagram
2	GeneratorDiagram	TemplateNode ApplicationNode SemanticNode SemanticForEdge ForeachNode ConverterNode
3	ApplicationNode SemanticNode SemanticForEdge converterSemantic converterMerge	TemplateNode

4	ForeachNode	SemanticNode
5	ConverterNode	converterSemantic converterRead converterMerge converterForeach
6	converterForeach	converterSemantic

Таблица 3. Список элементов-контейнеров визуального языка задания правил генерации с указанием тех типов, элементы которых могут находиться внутри контейнера

Таким образом, при помощи данного визуального языка можно в соответствующем редакторе формально описывать генераторы для визуальных языков. Для этого необходимо сначала создать элемент типа «Generator Root» в качестве корневого элемента модели создаваемого генератора и заполнить все требуемые свойства. Затем нужно добавить к корню элементы-детей (в смысле вложенности), относящиеся к типу «Generator Diagram», которые и являются рабочими диаграммами, на которых можно рисовать все необходимые правила генерации, используя остальные элементы нашего визуального языка.

4) Метагенератор

Метагенератор – это генератор, позволяющий генерировать генератор текстового кода по диаграммам на визуальном языке.

4.1) Архитектура решения

Используя разработанный язык задания правил генерации и соответствующий редактор (т.е. в «Generator Editor»), пользователь может формально описывать модель генератора для своего визуального языка. Далее можно по этой модели сгенерировать код. При этом необходимо предварительно убедиться, что графическая (отображающая иерархию элементов такую, что и на диаграмме) и логическая (отображающая другую иерархию элементов, согласно задаваемой пользователем логике) модели в соответствующих обозревателях системы согласованы, так как процесс генерации происходит по логической модели описанной модели генератора, в то время как процесс рисования диаграмм затрагивает графическую модель. Однако стоит отметить, что для элементов-узлов нашего визуального языка была поддержана автоматическая синхронизация логической и графической модели, что упрощает работу с ним. Для более подробной информации о работе в системе QReal, в частности о создании диаграмм и моделей, можно прочитать в тематических статьях об этой среде разработки, например [6].

Сгенерированные файлы кода сохраняются в папке, указанной в свойстве «Directory to generate» корневого элемента модели генератора. Далее система QReal предложит пользователю автоматически собрать этот код и подключить получаемый собранный плагин генератора (то есть подключаемый модуль). При этом стоит отметить, что код генератора можно также собирать и вручную, в результате чего в папке `qreal/bin/plugins` появится соответствующий плагин генератора (.dll) визуального языка. При запуске же системы этот плагин будет автоматически загружен.

Далее можно работать с данным генератором. То есть пользователь может создать любую диаграмму на визуальном языке, для которого был разработан генератор, а потом по этому формальному описанию программы сгенерировать код приложения.

Таким образом, общая архитектура решения изображена на рисунке (Рис. 6):

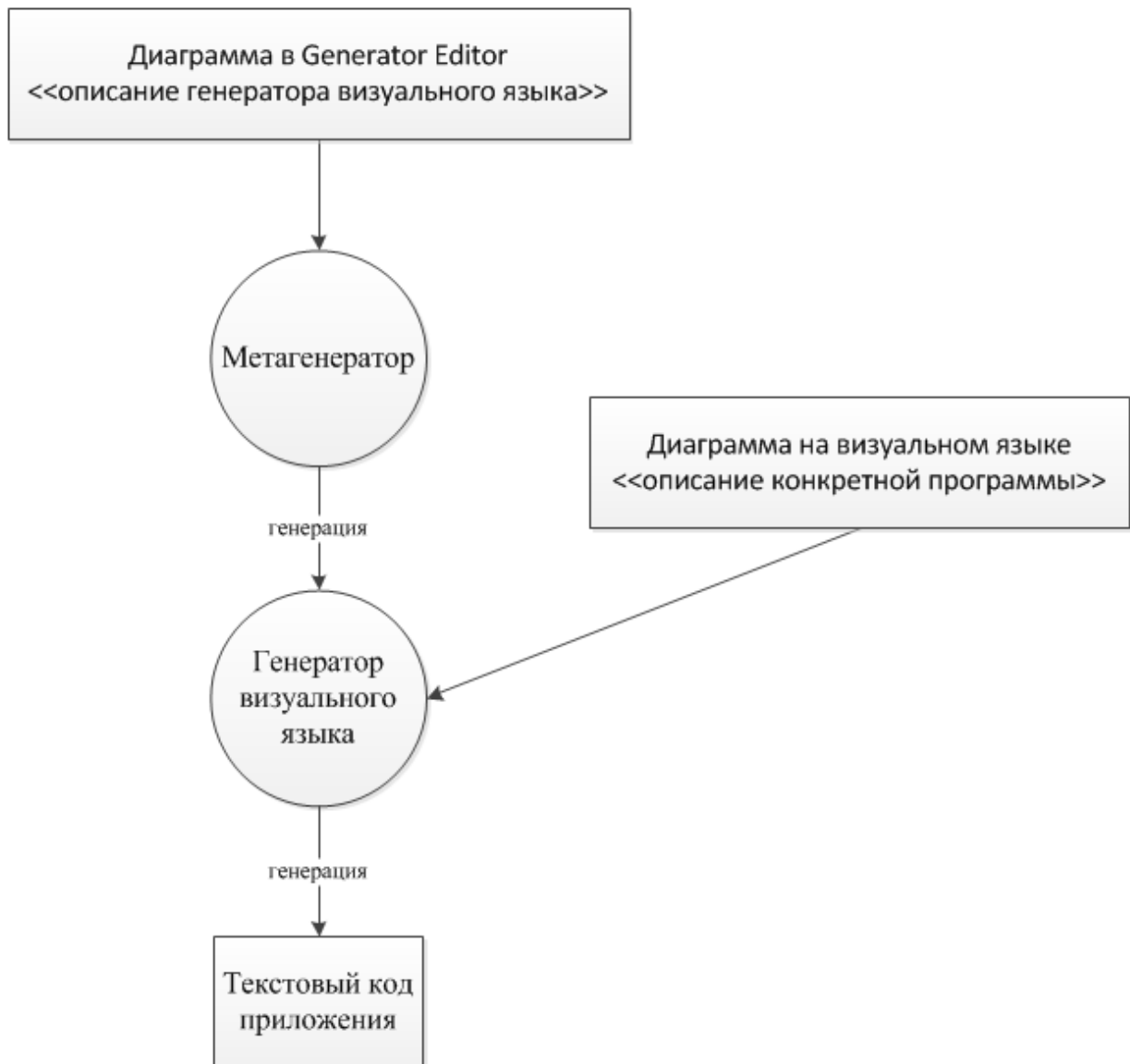


Рис. 6. Общая архитектура решения

Метагенератор получает на вход диаграмму с формальным описанием генератора, а на выход выдаёт сам генератор визуального языка. Этот генератор в свою очередь, принимая на вход диаграмму на визуальном языке, возвращает соответствующую текстовую программу.

4.2) Обсуждение

В данном пункте напишем о том, какие ещё варианты реализации рассматривались, но не были реализованы.

4.2.1) Другой вариант архитектуры решения

В ходе работы на ранних этапах был предложен альтернативный вариант архитектуры решения. В данном варианте, как и в выбранном ранее (Рис. 6), необходимо сначала формально описать генератор визуального языка при помощи разработанного редактора. Далее же предлагается, чтобы метagenератор сразу генерировал код приложения по той формальной модели генератора и соответствующей диаграмме, описывающей программу на визуальном языке.

Этот вариант общей архитектуры решения изображен на рисунке (Рис. 7):

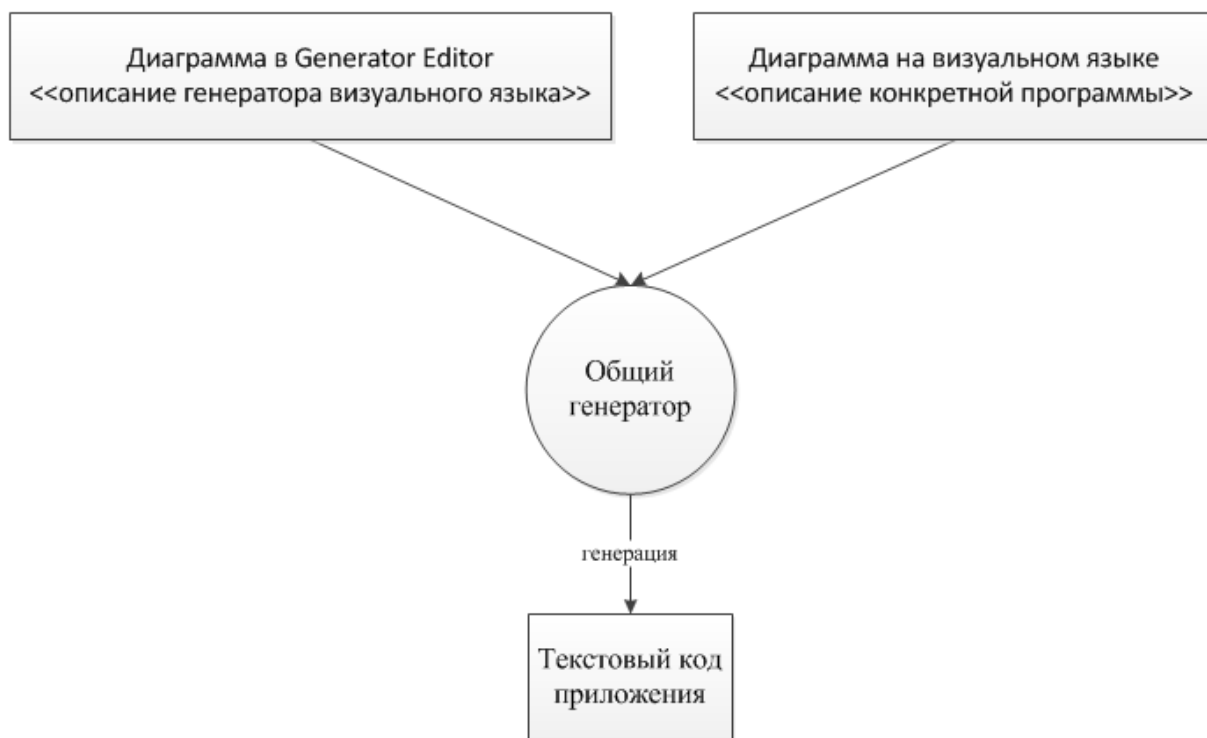


Рис. 7. Альтернативная общая архитектура решения

Метагенератор получает на вход и диаграмму с формальным описанием генератора, и диаграмму на визуальном языке, а на выход выдаёт сразу соответствующую текстовую программу.

В итоге для решения задачи дипломной работы был выбран первый вариант архитектуры (Рис. 6), представленный ранее, а не альтернативный (Рис. 7), ввиду ряда недостатков второго варианта, таких как:

- более высокая сложность в реализации;
- второй вариант архитектуры не позволяет явно генерировать код генератора визуального языка, а следовательно не разрешает вносить изменения в код вручную, что может быть в некоторых случаях полезно;

- генерация приложения по диаграмме на визуальном языке происходит заведомо дольше, чем при первом варианте архитектуры, так как итоговым генератором для визуального языка во втором варианте архитектуры является сам метagenератор, которому необходимо каждый раз при генерации программ интерпретировать сразу две визуальные программы, одна (модель генератора) из которых скорее всего довольна громоздка, в то время как при первом варианте только одну диаграмму, а именно саму программу, описанную на визуальном языке.

4.2.2) Изменение кода генератора

Ранее отмечалось, что весьма полезно иметь возможность менять вручную код генератора визуального языка. А выбранная и реализованная нами архитектура решения как раз позволяет по формальной модели генератора получать явный его код, который можно при желании изменить и заново пересобрать, получив новый плагин генератора.

Однако на данный момент наше решение не поддерживает возможность «возвращения изменений обратно в модель», то есть все изменения, которые были сделаны вручную в коде генератора, не попадают обратно в саму модель генератора. Хотя необходимость этого при разработке генераторов довольно спорна.

При этом также данная реализация решения не предполагает какую-либо иерархию создаваемых генераторов с выделением в них общих и изменяемых частей для возможности более удобного переиспользования ранее разработанных генераторов, что в целом значительно упростило бы работу с кодом генераторов визуальных языков. Однако это вполне можно исправить, несколько изменив метagenератор, в то время как сам язык не потребует никаких изменений. На данный момент же подобной возможности не потребовалось.

5) Апробация

Предложенное решение было применено к языку задания экранных форм QUbiq Presentation Editor (см. пункт 2.2.3). Для этого в соответствующем редакторе была создана диаграмма на разработанном языке задания правил генерации, которая формально описывает генератор рассматриваемого визуального языка. Ниже приведены обе части этой диаграммы (Рис. 8, Рис. 12).

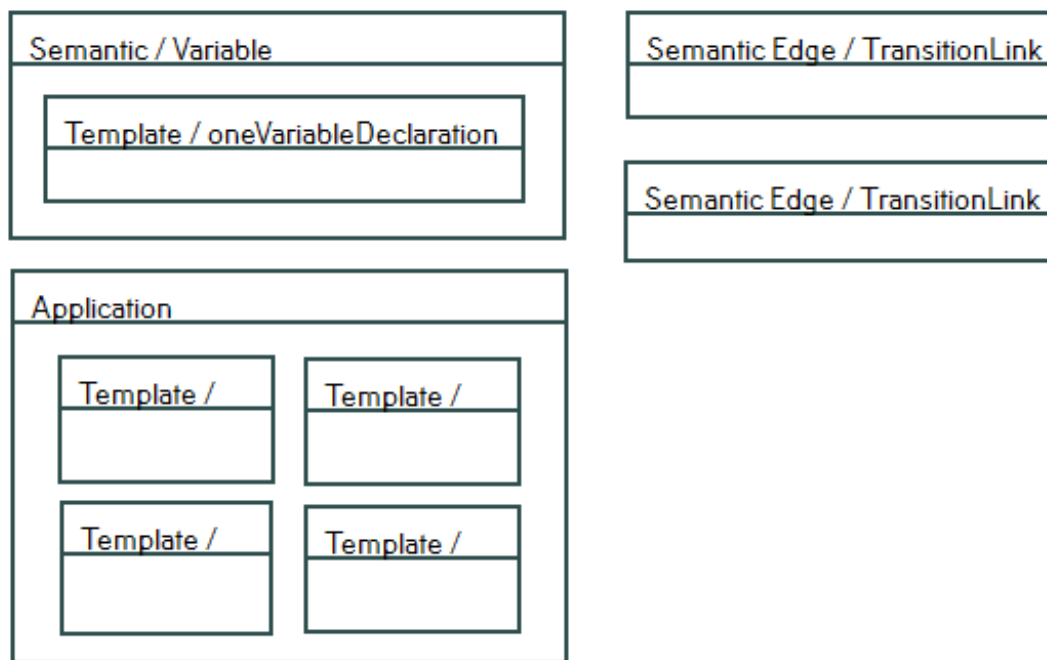
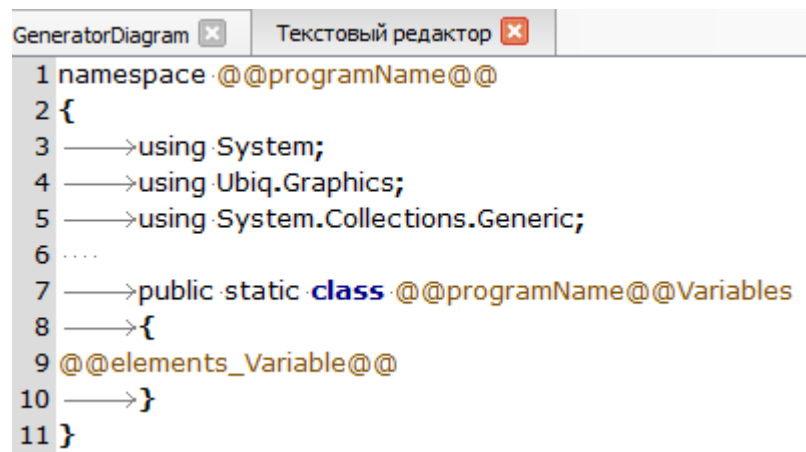


Рис. 8. Первая часть диаграммы генератора для языка
QUbiqPresentationEditor

Эта часть диаграммы (Рис. 8) описывает правила генерации:

- шаблонов, общих для всего приложения (см. элемент Application на Рис. 8);
- для элементов, относящихся к типу Variable (см. элемент Semantic на Рис. 8);
- для элемента-связи TransitionLink и элементов, соединённых ею (см. элементы Semantic Edge на Рис. 8).

Шаблоны, как уже говорилось, описываются текстовым языком с использованием меток. Например, можно посмотреть на файл-шаблон (Рис. 9), в котором должны объявляться все переменные, используемые генерируемым приложением.



```

1 namespace @@programName@@
2 {
3     using System;
4     using Ubiq.Graphics;
5     using System.Collections.Generic;
6     ....
7     public static class @@programName@@Variables
8     {
9         @@elements_Variable@@
10    }
11 }

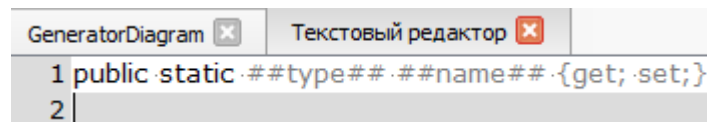
```

Рис. 9. Шаблон генерируемого файла, который объявляет переменные программы

В данном коде используются только две метки:

- «@@programName@@», вместо которой при генерации приложения будет подставляться имя программы;
- «@@elements_Variable@@», вместо которой будет подставлен код, сгенерированный для всех элементов диаграммы типа Variable.

В свою очередь, код, который необходимо генерировать для каждого элемента типа Variable, указан в соответствующем поле элемента Semantic, представленном на первой части диаграммы всего генератора (Рис. 8). Ниже представлен шаблон этого кода (Рис. 10).



```

1 public static ##type## ##name## {get; set;}
2

```

Рис. 10. Шаблон кода, генерируемого для элемента типа Variable

Рассматриваемый шаблон тоже содержит две метки, но уже другого типа, а именно «##type##» и «##name##». При генерации приложения вместо них будут подставляться значения полей «type» и «name» текущего элемента типа Variable соответственно.

Элемент SemanticForEdge, аналогично элементу Semantic, задаёт код, который необходимо сгенерировать при посещении указанного элемента. Однако в шаблоне кода SemanticForEdge, в метках, выделенных символами «##», можно дополнительно использовать ключевые слова «src» и «dst». Ниже представлен пример такого кода для рассматриваемого генератора экранных форм платформы QUbiq (Рис. 11).

```

1 void On##src.name##Clicked(object sender, EventArgs args)
2 {
3     Screen.Content = Create##dst.name##();
4 }

```

Рис. 11. Шаблон кода, генерируемого для каждой связи *TransitionLink*, соединяющей элементы *Button* и *Slide*

Видно, что в данном фрагменте кода использованы следующие две метки:

- «##src.name##», вместо которой при генерации приложения будет подставляться значение поля «name» начального элемента (то есть элемента, из которого выходит связь), в рассматриваемом случае элемента типа *Button*;
- «##dst.name##», вместо которой аналогично будет подставляться значение поля «name», но уже конечного элемента (то есть элемента, в которого входит связь), в рассматриваемом случае элемента типа *Slide*.

Ниже приведём вторую часть диаграммы (Рис. 12), на которой описываются правила генерации для основного элемента рассматриваемого языка, а именно элемента задания экранной формы *Slide*.

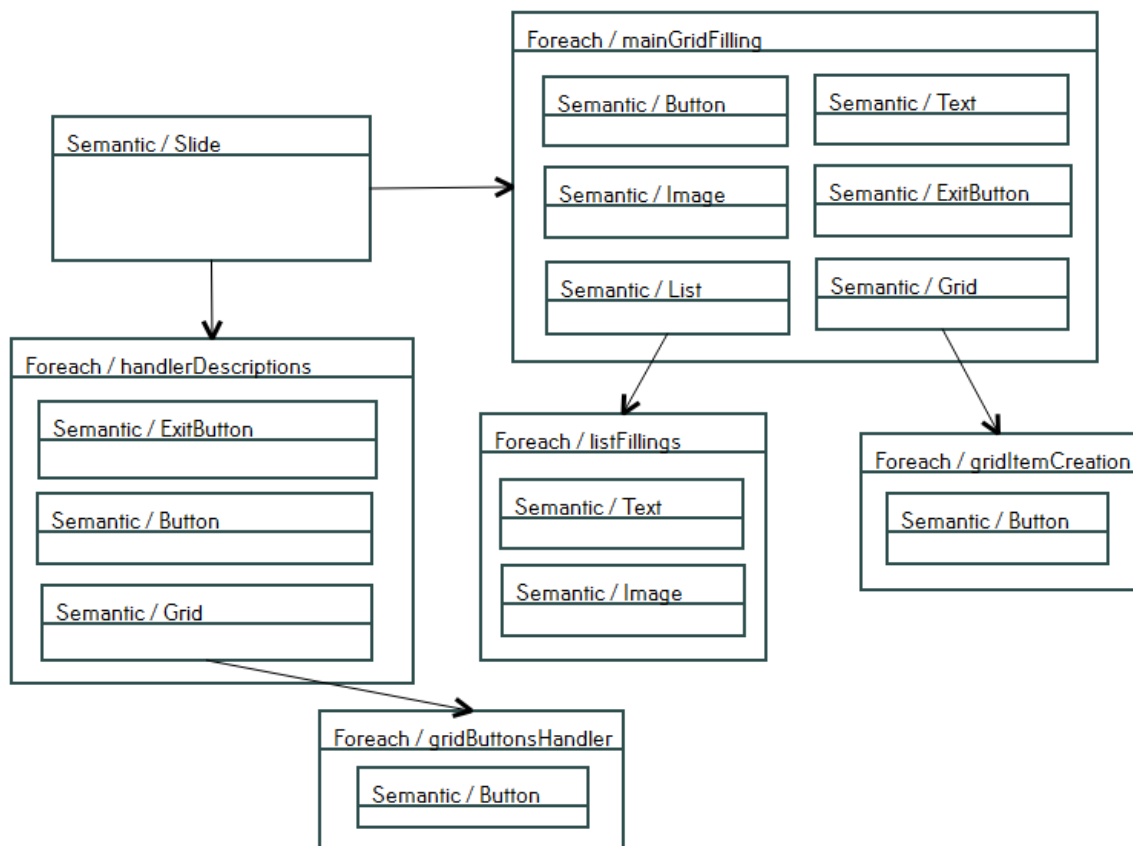


Рис. 12. Вторая часть диаграммы генератора для языка *QUbiqPresentationEditor*

На этой части видно, что элемент Semantic, описывающий генерируемый код для элемента типа Slide, соединён связью с двумя элементами Foreach, которые ассоциированы с метками «mainGridFilling» и «handleDescriptions» соответственно. Это означает, что в шаблоне кода для Slide были использованы соответствующие метки.

Каждый элемент Foreach задаёт, какой код необходимо сгенерировать при обходе всех «детей» (в смысле вложенности в контейнер) элемента Slide, а после этого сохранить в указанную метку. При этом код, генерируемый для каждого отдельного «ребёнка», задаётся уже известным способом, а именно при помощи элемента Semantic. При этом очевидно, что при описании правил генерации для каждого типа «детей» можно также использовать вложенные описания при помощи элемента Foreach, как это и сделано в рассматриваемом случае (см. элементы Semantic для List и Grid на Рис. 12).

Таким образом, при помощи реализованного визуального языка была создана диаграмма, описывающая правила генерации для визуального языка задания экранных форм системы QUbiq. Разработанный метagenератор, в свою очередь, позволил по этому формальному описанию сгенерировать конкретный генератор рассматриваемого языка, который по диаграмме приложения (на рассматриваемом языке) генерирует его текстовый код.

Ниже приведён пример такой диаграммы (Рис. 13) на визуальном языке QUbiq Presentation Editor, по который полученный генератор может сгенерировать код приложения. Подробно об этом описано в соответствующей курсовой работе [13].

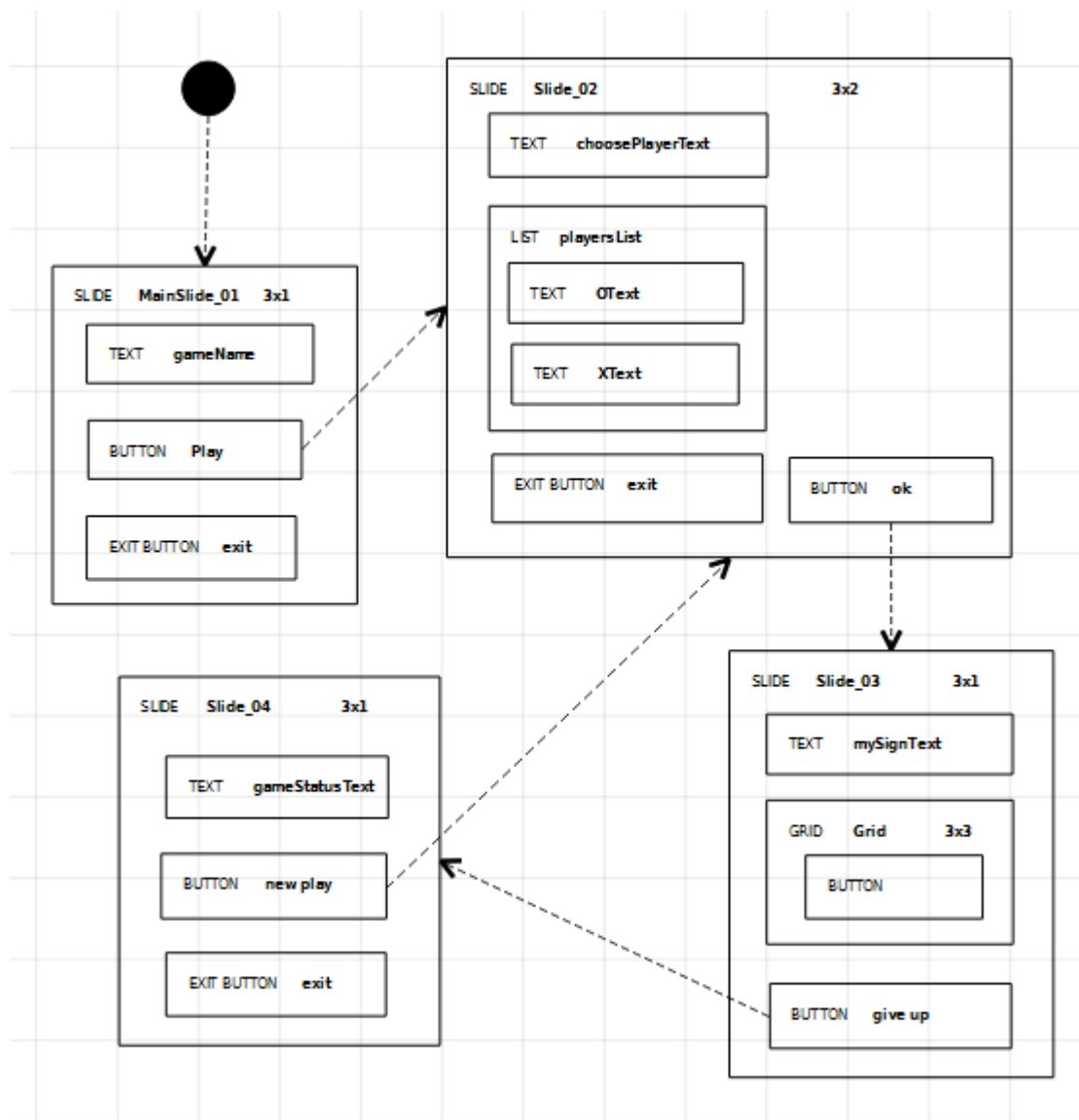


Рис. 13. Пример основной диаграммы экранных форм для мобильного приложения, а именно онлайн-игры в крестики-нолики для одного игрока на мобильном устройстве [13]

Заключение

В ходе данной дипломной работы была поддержана возможность задания правил генерации в среде программирования QReal. Для достижения такого результата были выполнены следующие задачи.

1). Сделан обзор существующих решений данной задачи в других средах программирования, таких как MetaEdit+, Actifsource (Eclipse) и т.п. Также было рассмотрено предыдущее решение создания генераторов в среде QReal.

2). Разработан язык, позволяющий описывать правила генерации некоторого визуального языка в текстовое представление. Также создан редактор, позволяющий создавать диаграммы на этом языке.

3). Реализован генератор, позволяющий по диаграммам на разработанном языке порождать описанный генератор кода. Далее этот генератор можно автоматически подгрузить в среду и уже генерировать код программ, описанных при помощи диаграмм на некотором визуальном языке, для которого данный генератор и создавался.

При этом генератор поддерживает только один класс визуальных языков, а именно структурные, что покрывает достаточно большую часть практически полезных языков.

4). Решение апробировано на существующем языке системы QReal, а именно на языке задания экранных форм технологии QUbiq, являющимся структурным языком. Для этого языка уже существовал «модельный» генератор, написанный заранее вручную. Однако в рамках дипломной работы соответствующий генератор был формально описан на визуальном языке.

Список литературы

- [1] R. B. Kieburtz [и др.]. A software engineering experiment in software component generation // Proceedings of the 18th international conference on Software engineering. – IEEE Computer Society. 1996. – P. 542 – 552.
- [2] Gray J., Karsai G., An examination of DSLs for concisely representing model traversals and transformations // System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on. – IEEE. 2003. – P. 10.
- [3] Kelly S., Tolvanen J.-P., Visual domain-specific modeling: Benefits and experiences of using metaCASE tools // International Workshop on Model Engineering, at ECOOP. – 2000.
- [4] Tolvanen J.-P., Rossi M., MetaEdit+: defining and using domain-specific modeling languages and code generators // Proceeding OOPSLA '03 Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. – New York, 2003. – P. 92-93.
- [5] Tolvanen J.-P., Pohjonen R., Kelly S., Advanced tooling for domain-specific modeling: MetaEdit+. // OOPSLA Workshop on Domain-Specific Modeling. – Montreal, Canada, 2007.
- [6] 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". – Санкт-Петербург: Изд-во СПбГПУ, 2011. – С. 100-101.
- [7] Терехов А.Н., Брыксин Т.А., Литвинов Ю.В. и др., Архитектура среды визуального моделирования QReal. // Системное программирование. Вып. 4. – Санкт-Петербург: Изд-во СПбГУ, 2009. – С. 171-196.
- [8] 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.
- [9] Генераторы в QReal:Robots, URL: <http://github.com/qreal/qreal/wiki> Генераторы-в-QReal:Robots, Дата последнего обращения: 18.05.2014
- [10] Подкопаев А. В., Средства описания генераторов кода для предметноориентированных решений в metaCASEсредстве QReal (курсовая работа). – СПб, 2012
- [11] Сухов А.О., Разработка инструментальных средств создания визуальных предметно-ориентированных языков (кандидатская диссертация). – Пермь, 2013. – С. 29-39

- [12] Onossovski V., Terekhov A., Ubiq Mobile – a New Universal Platform for Mobile Online Services // Proceedings of 6th Seminar of Finish-Russian University Cooperation (FRUCT) Program. – Helsinki, 2009.
- [13] Дерипаска А.О., Визуальный язык для платформы Ubiq Mobile в среде QReal (курсовая работа). – СПб, 2013