Правительство Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Санкт-Петербургский государственный университет»

Системного программирования

Клепач Богдан Валентинович

Развитие графической подсистемы DSM-платформы QReal

Бакалаврская работа

Допущен к защите.

Зав. кафедрой:

д. ф.-м. н., профессор Терехов А.Н.

Научный руководитель:

ст. преп. Кириленко Я.А.

Рецензент:

ст.преп. Брыксин Т.А.

Санкт-Петербург, 2015

SAINT-PETERSBURG STATE UNIVERSITY

Faculty of Mathematics and Mechanics

Software Engineering Chair

Bogdan Klepach

Developing graphic subsystem in QReal DSM platform

Bachelor's Thesis

Admitted for defence.

Head of the chair:

Professor Terekhov A.N.

Scientific supervisor:

Senior Lect. Kirilenko J.A.

Reviewer:

Senior Lect. Bryksin T.A.

Saint-Petersburg

2015

Оглавление

Введение

В настоящее время для разработки ПО нередко используются средства визуального программирования. Данные средства позволяют представить разрабатываемую систему в виде визуальной модели, описанной на некотором языке. Такой подход облегчает понимание системы, делая процесс разработки более наглядным. Инструментальные средства, позволяющие быстро описывать визуальные языки и редакторы к ним, называют DSM[1]-платформами.

Важным аспектом данных средств является графическая подсистема, включающая в себя способы задания визуального представления элемента и его отрисовки на диаграмме. В данной работе рассматривается DSM-платформа QReal[2][1, 6], позволяющая генерировать код визуального редактора по заданной метамодели. К сожалению, реализованная графическая подсистема не удовлетворяет всем желаемым требованиям. Первая проблема – отсутствие интерактивных элементов управления, таких как кнопка, флажок и т.д. Добавление таких элементов управления позволит расширить круг возможных задач, для которых применима данная платформа. Вторая проблема – это статичность получаемого визуального представления элемента. То есть отсутствие возможности взаимодействия элементов управления с мышью и клавиатурой и изменения визуального представления элемента при изменении его свойств. При изменении значения любого свойства элемента ожидается, что его внешний облик должен автоматически изменяться согласно измененным значениям. И,

[1] Domain Specific Modelling
[2] http://qreal.ru/

наоборот, при изменении элемента автоматически должны изменяться значения его свойств.

Постановка задачи

Целью данной работы является переработка графической подсистемы DSM-платформы QReal. То есть необходимо модифицировать графическую подсистему так, чтобы в ней поддерживались элементы управления и присутствовала возможность динамического изменения внешнего вида элементов путем изменения его свойств.

Глава 1. Обзор существующих решений

Для создания визуального языка на основе QReal используется метаредактор, являющимся, по сути, визуальным языком со своей метамоделью. На сцене из элементов данного языка составляется диаграмма, по которой можно сгенерировать визуальный редактор и после этого встроить его в систему. Выделяют два типа графических элементов в диаграмме: узловые элементы и связи. Для каждого из этих элементов можно задать визуальное представление. Само же визуальное представление генерировалось редактором форм и передавалось метаредактору в качестве внешнего вида некоторого элемента. В дальнейшем, при генерации редактора для описанного визуального языка, графическое представление сущностей сохранялось в файле формата XML с описанием метамодели. При загрузке в среду сгенерированного редактора из файла метамодели выделяются визуальные описания сущностей и сохраняются в отдельных файлах, пути к которым указаны в файле ресурсов. Механизм отрисовки и поведения элементов (рис. 1) на сцене, в свою очередь реализован с помощью Qt Graphics View Framework.

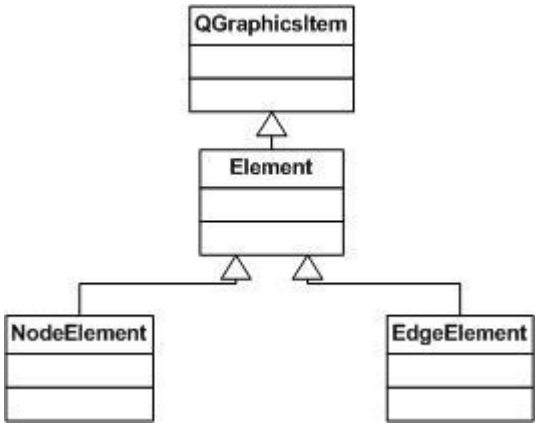

Рис. 1. Иерархия базовых элементов в QReal

Поставленная задача сложнее, чем, кажется, в прошлом были предприняты попытки её решения. Рассмотрим две наиболее результативные попытки.

1.1. SDF[3]

В текущей реализации графическая подсистема системы QReal основывается на внутреннем формате SDF[7]. Для задания графического представления элемента используется встроенный редактор форм. В нем пользователь, перетаскивая нужный элемент на сцену, изменяя размер, затем, изменяя свойства (цвет, стиль) на панели свойств, может задавать внешний вид элемента (рис. 2). По нему редактор форм построит описание в формате SDF.

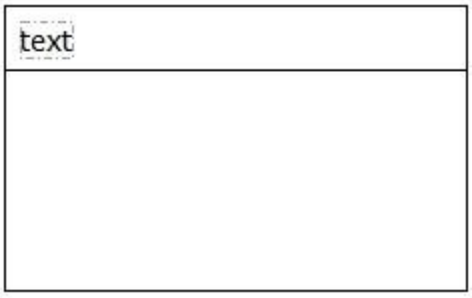

Рис. 2. Пример внешнего вида элемента, заданного посредством редактора форм.

```
<!--Описываем размер изображения-->
<picture sizex="232" sizey="143">
<!-- описываем прямоугольник и его свойства -->
    <rectangle fill-style="none" stroke-style="solid" stroke-width="0"
    fill="#ffffff" y1="0" y2="141" stroke="#000000" x1="0" x2="230"/>
<!-- описываем линию и её свойства -->
    <line fill-style="none" stroke-style="solid" stroke-width="0"
    fill="#ffffff" y1="32" y2="32" stroke="#000000" x1="0" x2="231"/>
<!--описываем текст и его свойства -->
    <text font-size="15" font-name="MS Shell Dlg 2" b="0" u="0"
    y1="8" i="0" font-fill="#000000" x1="8">text</text>
</picture>
```

Листинг 1. Визуальное представление элемента на рис. 1 в формате SDF

[3] Stencil Description Format

Полученное описание будет сохраняться в качестве внешнего вида элемента. Для удобства работы с графическим представлением существует возможность редактирования некоторых свойств на сцене. К сожалению, проблема с настраиваемым внешним видом элемента на диаграмме остается, так как единственной возможностью настроить внешний вид является редактор форм, который на выходе дает статическое изображение из простых примитивов практически без возможности динамического манипулирования свойствами.

1.2. Qt Widgets

Qt Widgets - модуль, предоставляющий набор элементов для создания пользовательского интерфейса. Для каждого типа элементов определен набор свойств. Библиотека Qt Widgets содержит множество различных элементов, в том числе и элементов управления. Расширение графической системы за счет внедрения поддержки виджетов в графическое представление элементов уже проводилось в рамках одного из защищенных дипломов, для этого был разработан формат WTF[4][2, 3] и связанный с ним стек технологий.

WTF

WTF - это формат, основанный на XML, инкапсулирующий SDF. Для этого формата была осуществлена поддержка элементов управления, переработаны механизмы отрисовки, обработки свойств, работы с мышью. Разработан редактор виджетов, генерирующий WTF-описание, интегрированный с репозиторием, где хранится информация об элементах, и редактором форм. Применение данного подхода расширило возможности графического представления элемента, частично разрешило проблему статичности графического представления за счет взаимодействия элементов

[4] Widget Template Format

управления с мышью и клавиатурой. Но получившаяся технология громоздка и требует поддержки. При необходимости реализации сложных виджетов с анимацией возникают проблемы из-за невозможности описания сложной логики в коде.

1.3. QtQuick

Технология описания графического представления, использующая библиотеку QtQuick. Интерфейсы, разработанные в QtQuick, представляются в виде визуального иерархического дерева элементов. В свою очередь, каждый элемент представляет собой совокупность блоков: графических (Rectangle, Image) и поведенческих (state, animation). QtQuick использует концепцию логики моделирования приложений через иерархию состояний. Кроме того, переходы и богатый набор анимаций позволяют покрыть большой объем задач. Для описания используется язык QML[5], являющийся декларативным языком программирования, основанным на JavaScript, с помощью которого можно описывать иерархию объектов и их состояний. Одним из преимуществ QML является механизм связывания свойств (property binding), то есть при изменении свойства в элементе автоматически происходит обновление данных зависящих от этого свойства, что упрощает решение проблемы автоматического изменения отображения элемента при изменении свойств.

QtQuick 1.0 (QtDeclarative)

Библиотека QtQuick 1.0 [8] использует QPainter/QGraphicsView API для отрисовки сцены. Для примера приведем, как будет выглядеть на языке QML элемент (рис. 3), схожий с рассмотренным в обзоре SDF.

[5] Qt Meta-Object Language

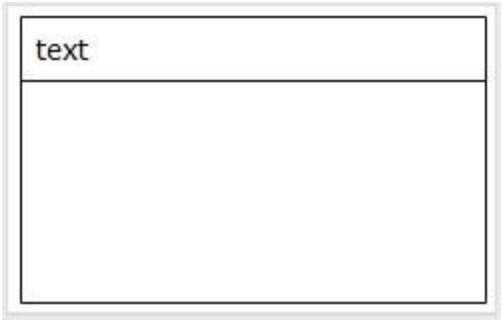

Рис. 3. Пример элемента отрисованный с использованием библиотеки QtQuick 1.0

И его код на языке QML (в данном случае был добавлен пользовательский тип линия).

```
// задаем прямоугольник и его свойства
Rectangle {
    width: 232; height: 143; border.width: 1; border.color: "black";
    // задаем линию и её свойства
    Line {
          x1: 0; y1: 32; x2: 231; y2: 32; style: "solid"; color: "black";
    }
    // задаем текст и его свойства
    Text {
          x: 8; y: 8; color: "#000000"; text: "text";
          font.family: "MS Shell Dlg 2";
          font.pixelSize: 15; font.bold: false;
          font.italic: false; font.underline: false;
    }
}
```

Листинг 2. Код на языке QML описывающий графическое представление элемента (рис. 3)

Данный вариант имеет как свои преимущества, так и недостатки. Важным преимуществом является связывание свойств. Для иллюстрации приведем простой пример, в котором прямоугольник будет менять цвет с зеленого на красный при растяжении его свыше, чем на 100 пикселей в ширину.

```
Rectangle {
    id: rect
    width: 100; height: 100; // задаем первоначальную ширину элемента
    color: rect.width > 100 ? "red" : "green"; // задаем зависимость цвета от ширины
}
```

Листинг 3. Пример задания зависимости значения одного свойства от другого

При динамических изменениях внешний вид элементов должен зависеть от значений их свойств. В сочетании со связыванием свойств эти

элементы будут изменяться автоматически после первоначальной настройки. Это, в свою очередь, позволяет работать со сложными интерфейсами, сохранив при этом простоту кода и снизив риск появления ошибок. QML можно интегрировать с кодом на C++, то есть существует возможность использования из QML C++-методов и из C++ задавать значения свойств элементов QML. В сочетании эти два аспекта дают простое и гибкое решение проблемы с автоматическим изменением элемента по изменению свойства, и наоборот, изменения значения свойства графической модели после изменения элемента.

QML также позволяет задавать набор состояний, для каждого из которых задается набор свойств и их значений в этом состоянии, в котором может находиться элемент и переключение между ними по сигналу или по нажатию мыши. К сожалению, список компонент, описанных в данной библиотеке, недостаточен для реализации поставленной задачи, но существующие возможности по его расширению за счет регистрации пользовательских типов позволяют его дополнить необходимыми компонентами. Исходя из всего этого, можно сказать, что библиотека QtQuick 1.0 предоставляет необходимые возможности для решения поставленной задачи с некоторыми неудобствами в виде отсутствия встроенной библиотеки элементов управления.

QtQuick 2.0

QtQuick 2.x [9] является переработкой QtQuick 1.x. Как идеологическое развитие QtQuick 1.0, версия 2.0 обладает теми же плюсами. Помимо прочего, для QtQuick 2.0 описана библиотека элементов управления, содержащая требуемые элементы. Главным же отличием от QtQuick 1.0

является полная переработка низкоуровневой системы рисования, в основе которой теперь лежит QpenGL. На рис. 4 приведен пример из демонстрации[6].

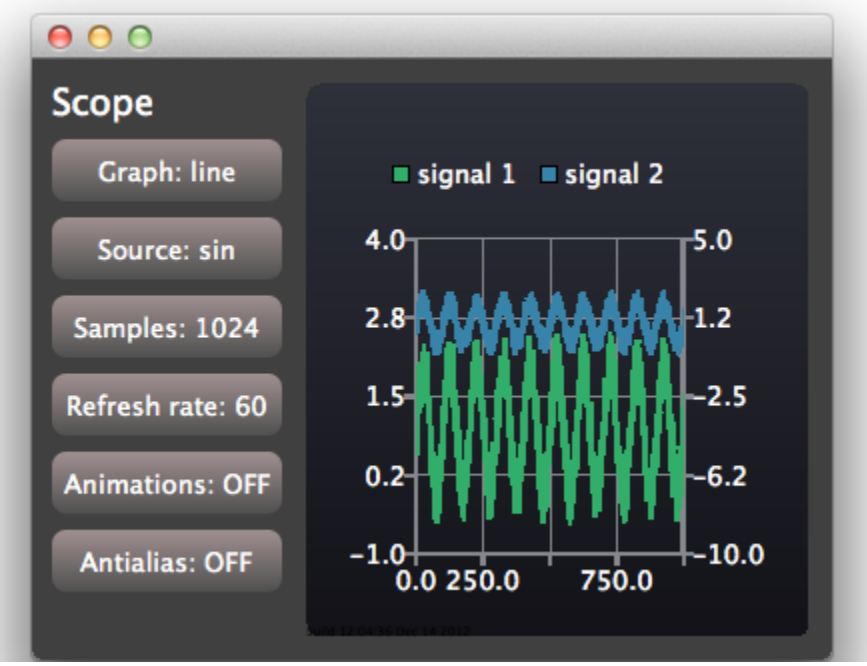

Рис. 4. Пример компоненты, реализованной с использованием QtQuick 2.0.

В QtQuick 2 использует граф сцены (screen graph) для эффективной отрисовки на GPU. К сожалению, это делает невозможным интеграцию с классическими виджетами, в отличие от QtQuick 1.x, где интеграция с ними возможна с помощью особого механизма в инструментарии Qt (QGraphicProxyWidget). В данный момент ведется активная разработка набора виджетов специально для QML, но они пока не могут полностью компенсировать классические виджеты. Здесь так же могут возникнуть трудности технического характера, так как оборудование для обработки графики должно поддерживать технологию OpenGL. Таким образом, данная технология не подходит для внедрения в систему QReal.

[6] http://doc.qt.io/QtCharts/qtcharts-qmloscilloscope-example.html

1.4. Выбор технологии

Технология QtQuick 1.0 подходит для решения поставленной задачи. Помимо преимуществ, описанных в обзоре, низкоуровневое управление внешним видом происходящее посредством C++ кода, на котором написана платформа QReal, и возможность описание логики поведения элемента в QML-коде позволяют эффективно решить поставленную задачу.

Глава 2. Реализация

2.1. Подход к реализации

Данную задачу можно разбить на несколько этапов. На каждом из этапов технология поддерживается на определенном уровне иерархии метамоделирования. Первый уровень - это уровень модели. На этом уровне необходимо внедрить технологию QtQuick в CASE-движок. Дальше необходимо поддержать данную технологию на уровне метамодели. На данном этапе требуется переработать визуальное представление для уже описанных визуальных языков. Для этого необходимо разработать инструмент трансляции из текущего формата в формат QML. На третьей стадии реализуется поддержка технологии в метаметамодели. Здесь необходимо разработать возможность задания визуального представления сущностей на языке QML при создании нового визуального языка. Для этого следует переработать редактор форм и реализовать генерацию QML кода по диаграмме.

2.2. Внедрение QtQuick в инфраструктуру графической сцены

На первом этапе была внедрена технология QtQuick в движок графической сцены. Как и ожидалось, технология легко встроилась в инфраструктуру, не потребовав при этом больших усилий для модификации движка в соответствии с требованиями QML (в отличие от WTF, где возникали значительные сложности с интеграцией). При внедрении технологии, были переработаны механизмы инициализации, передачи визуального представления, синхронизации размеров, отображения иконок, способ записи визуального представления в файл метамодели, механизм

кэширования изображений и было реализовано взаимодействие между графической моделью и QML-окружением.

Добавление пользовательских компонентов

Так как в QtQuick 1.0 не существует готовой к использованию библиотеки, содержащей все необходимые элементы управления, то инструментарий данной технологии был расширен недостающими элементами, с помощью описания пользовательских элементов в среде QML [11]. Для этого задается набор сущностей, содержащих описание логики работы. Для каждой сущности определяется набор свойств, которые могут хранить значения и оповещать всех об их изменении. Диаграмма классов представлена на рис. 5.

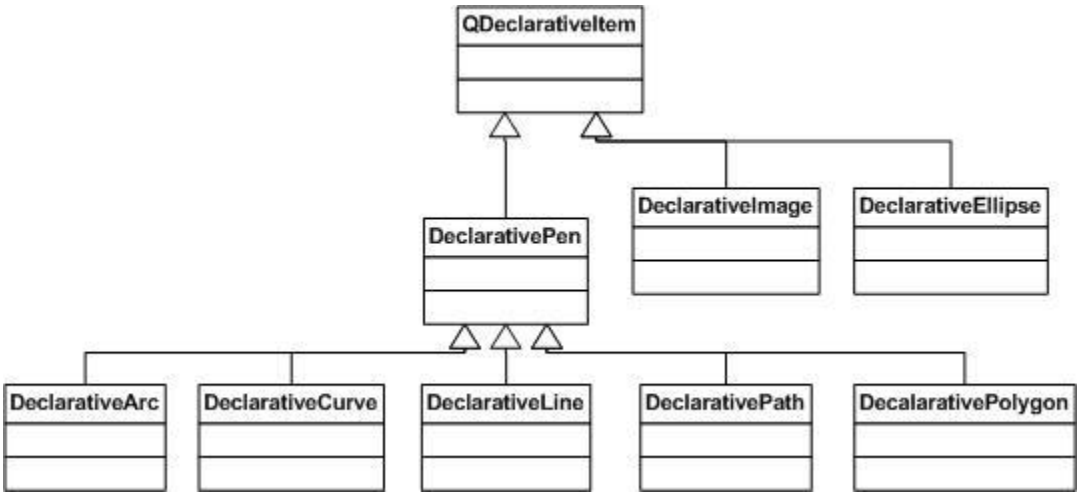

Рис. 5. Диаграмма классов новых примитивов

Низкоуровневые примитивы, такие как эллипс и многоугольник, были описаны на языке C++, остальные же элементы управления, такие как кнопка и флажок, были реализованы средствами языка QML из уже интегрированных компонент.

2.3. Переработка графической части уже готовых визуальных языков

На следующем этапе работы было переработано визуальное представление элементов для уже готовых языков. Так как на данный момент

существует большое количество визуальных языков, реализованных на основе DSM-платформы QReal, то переводить описание внешнего вида элементов с языка SDF на язык QML вручную - это неприемлемый вариант, учитывая схожесть формата SDF и подмножества языка QML. Поэтому был разработан инструмент для автоматического преобразования графического представления. Данный инструмент получает на вход файл, содержащий описание сущностей в XML-подобном формате с данными метамодели, а на выходе для каждой сущности генерирует файл, содержащий описание её визуального представления на языке QML. Каждую из сущностей, в свою очередь, можно разбить на набор графических базовых элементов, с определенными свойствами. Под базовыми элементами понимаются следующие элементы: линия, прямоугольник, эллипс, кривая, текст, изображение, ломаная линия и многоугольник.

Так как переданный входной файл описан в формате, имеющим иерархическую структуру, то обработка начинается с корневого элемента. Дальше выделяются вложенные в него элементы. Для каждого элемента в метамодели содержится описание его графической составляющей и его свойств. Графическая часть состоит из перечисления описаний элементарных блоков. По каждому элементарному блоку в формате SDF строится эквивалентный блок, описанный на языке QML. Объединяя их, получим графическое представление сущности. В этой части также могут содержаться условия отображения конкретного блока в данном элементе.

При переработке необходимо учитывать, что координаты, определенные для блоков, могут быть заданы как в абсолютных значениях (при растяжении-сжатии элемента они не меняются), так и в относительных (меняются при изменении размеров элемента). В случае относительных координат необходимо найти соотношение между координатами блока и

длиной/шириной родительского элемента, для того, чтобы при изменении размеров элемента на сцене, форма элемента сохранялась в тех же пропорциях. В случае абсолютных координат, координаты переносятся без изменений в сгенерированный файл графической составляющей элемента на языке QML.

Условия отображения

Как уже было сказано, для некоторых элементов заданы условия отображения в зависимости от значений свойств, описанных в логической части. В этом случае строится выражение, значение которого является видимостью элемента, присваиваемое соответствующему свойству (visible). Значения свойств элементов, содержащихся в метамодели можно получить из графической модели.

2.4. Переработка редактора форм

Следующим уровнем внедрения является визуальный редактор внешнего вида элементов. Стоит отметить, что редактор форм в текущей реализации, сложно модифицируемый и имеет отличную от сцены редактора диаграмм кодовую базу, что не позволяет использовать преимущества платформы QReal (к примеру, возможность создания элементов при помощи жестов мыши [4]), в то время как любой визуальный язык, созданный на платформе QReal, наследует преимущества платформы как CASE-системы. Для увеличения возможностей графической подсистемы на основе DSM-платформы QReal был реализован визуальный язык редактора форм (рис. 6), содержащий все примитивы редактора форм и новые сложные интерактивные элементы управления.

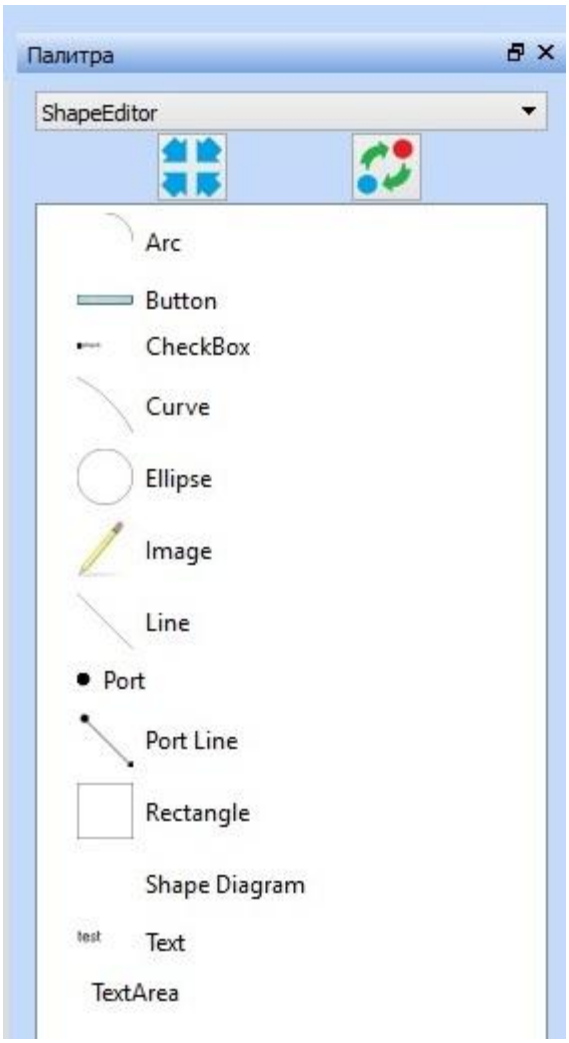

Рис 6. Палитра визуального языка редактора форм

Для каждого элемента было задано графическое представление на языке QML и набор свойств, изменяя которые можно манипулировать с внешним видом элемента на сцене. Помимо этого было описано обратное взаимодействие, то есть при изменении размеров элемента на сцене свойства, хранящиеся в логической модели, аналогично изменяются. Для этого при изменении элемента на сцене (изменяется ширина и высота элемента), проверяются какие из свойств, которые необходимо изменить, заданы (к примеру, свойства startx и starty для кривой), и происходит их изменение с сохранением пропорций.

2.5. Генератор

Следующий этап - это генерация кода. По диаграмме редактора форм необходимо уметь генерировать код на языке QML. Для начала упомянем о разделении визуальной модели элемента и его логической модели. Элемент может, как не иметь визуального представления, так и иметь несколько представлений. Следовательно, при генерации необходимо из репозитория получить список как логических, так и графических элементов. В свою очередь каждый графический элемент представляет из себя дерево графических элементов.

Задачу по генерации кода можно разбить на три подзадачи:

1. Генерация кода подключения библиотек,
2. Генерация кода “объемлющего” элемента,
3. Генерация кода графического представления каждого из элементов диаграммы.

Помимо подключения стандартной библиотеки QtQuick подключается библиотека пользовательских элементов и компоненты, содержащихся в файле ресурсов. Чтобы генерировать код элементов, необходимо уметь задавать соотношения, для этого выделяем параметры фона (длина и ширина объемлющего элемента) и координаты на сцене верхней левой точки для каждого элемента. Для вычисления размеров фона обходятся все графические элементы, во время обхода будем запоминать минимальное и максимальное значение координат x и y. Зная эти значения, можно определить размер фона и сгенерировать объемлющую компоненту. Потом повторно обходим графические элементы для генерации кода каждого из элементов. Так как графическая модель содержит в себе идентификатор логической модели, образом которой она является, то во время обхода можно вызвать методы для получения значений свойств элемента из логической

модели. При обходе сравнивается тип текущего элемента и вызывается соответствующий метод, где и генерируется код элемента с заданными свойствами.

После реализации генератора визуальный язык редактора форм был интегрирован в метаредактор для возможности задания визуального представления с использования этого визуального языка. То есть узловой элемент метаредактора теперь по двойному клику раскрывается в диаграмму, где пользователь может задать визуальное представления сущности. При генерации редактора в качестве графического представления будет записываться результат работы генератора QML-кода. Для диаграммы (рис. 7) генератор породит следующий код (рис. 8) графического представления.

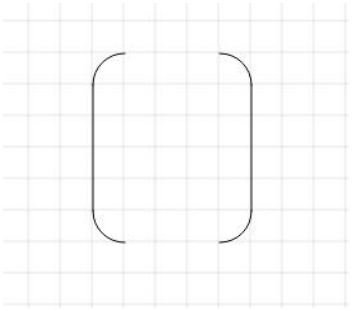

Рис. 7. Пример диаграммы, реализованной с помощью визуального редактора форм

```
Rectangle {
    width: 125; height: 150; color: "transparent";
    Line {
        x1: 0; y1: parent.height / 6;
        x2: 0; y2: 5 * parent.height / 6;
        color: "black"; width: 1; style: "solid";
    }
    Line {
        x1: parent.width; y1: parent.height / 6;
        x2: parent.width; y2: 5 * parent.height / 6;
        color: "black"; width: 1; style: "solid";
    }
    Arc {
        x1: 3 * parent.width / 5; y1: 0;
        x2: parent.width; y2: parent.height / 3;
        color: "black"; width: 1; style: "solid";
        startAngle: 0; spanAngle: 1440;
    }
    Arc {
        x1: 0; y1: 0;
        x2: 2 * parent.width / 5; y2: parent.height / 3;
        color: "black"; width: 1; style: "solid";
        startAngle: 1440; spanAngle: 1440;
    }
    Arc {
        x1: 0; y1: 2 * parent.height / 3;
        x2: 2 * parent.width / 5; y2: parent.height;
        color: "black"; width: 1; style: "solid";
       startAngle: 2880; spanAngle: 1440;
    }
    Arc {
        x1: 3 * parent.width / 5; y1: 2 * parent.height / 3;
        x2: parent.width; y2: parent.height;
        color: "black"; width: 1; style: "solid";
        startAngle: 4320; spanAngle: 1440;
    }
}
```

Рис. 8. Пример сгенерированного кода

Глава 3. Апробация

В результате реализации данного набора инструментов получается цельная, замкнутая и самодостаточная технология разработки. Необходимо апробировать полученный стек технологии. Для начала была проведена апробация на метаредакторе. Был создан визуальный язык содержащий элементы метаредактора, чьё визуальное представление было задано с использованием визуального языка редактора форм (рис. 9 и рис. 10).

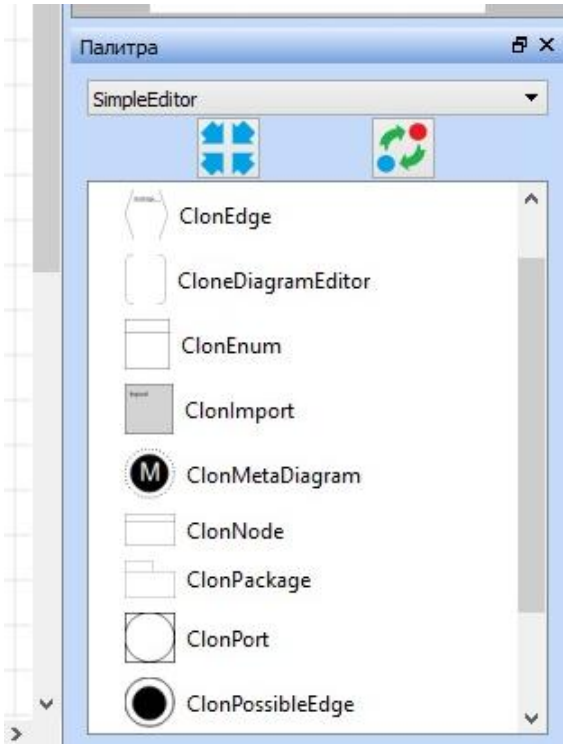

Рис. 9. Палитра элементов копии метаредактора, визуальное представление которых задано визуальным редактором форм.

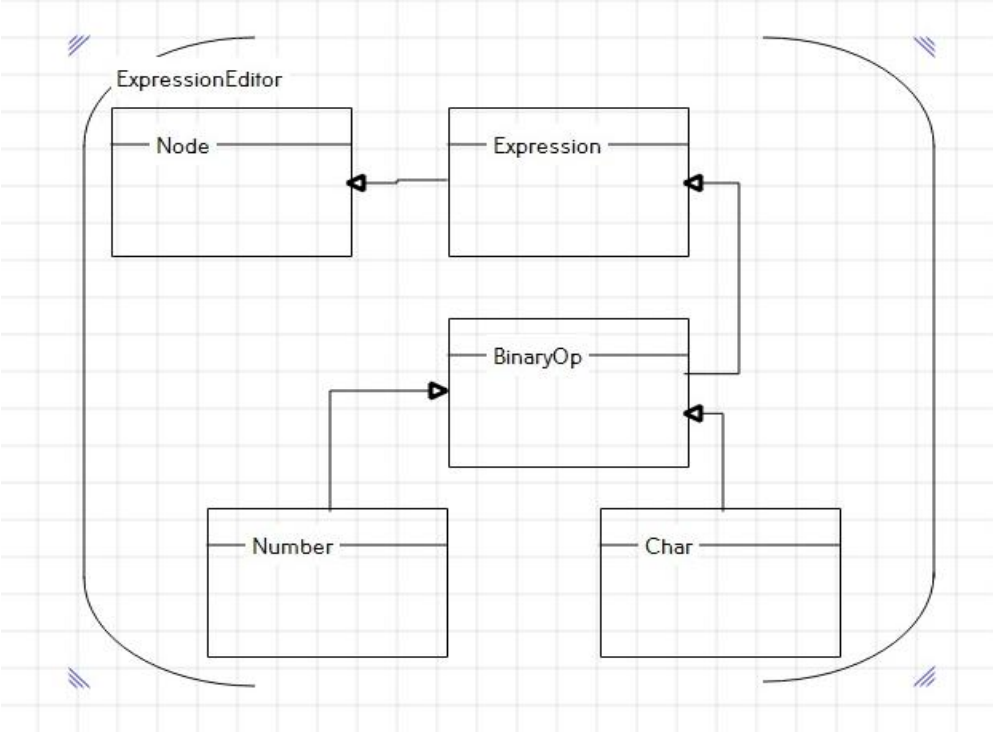

Рис. 10. Диаграмма составленная из элементов (рис. 9).

После апробации на метаредакторе, данный процесс разработки был апробирован на языке программирования роботов. Была реализована возможность редактирования свойств элементов прямо на сцене с помощью элементов управления (рис. 11).

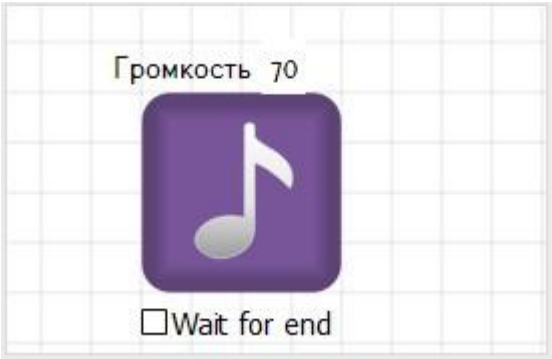

Рис. 11. Пример редактирование свойства элемента на сцене с помощью элемента управления

Глава 4. Заключение

В результате выполнения данной работы была переработана графическая подсистема DSM-платформы QReal. Внедрена технология QtQuick и была поддержана на всех уровнях иерархии метамоделирования. На основе DSM-платформы QReal был разработан редактор внешнего вида, предоставляющий возможность динамической манипуляции внешним видом элемента путем изменения его свойств и различные элементы управления. Полученный стек технологий был апробирован на двух самых часто-используемых визуальных редакторах – метаредакторе и языке программирования роботов.

Список литературы

[1] А.Н. Терехов, Т.А. Брыксин, Ю.В. Литвинов и др., Архитектура среды визуального моделирования QReal.

[2] Мордвинов Д.А., Реализация настраиваемого графического представления элемента на диаграмме в QReal, http://se.math.spbu.ru/SE/YearlyProjects/2012/YearlyProjects/2012/445/445_Mordvinov_report.pdf

[3] Мордвинов Д.А., Средства разработки пользовательских интерфейсов в DSM-платформе QReal, http://se.math.spbu.ru/SE/diploma/2013/s/MordvinovDmitry_thesis.pdf

[4] М.С. Осечкина, Т.А. Брыксин, Ю.В. Литвинов и др., Поддержка жестов мышью в мета-CASE-системах // Системное программирование. Вып. 5. СПб.: Изд-во СПбГУ. 2010. С. 52-75.

[5] Т.А. Брыксин, О генеративном подходе к созданию визуальных редакторов // Материалы Второй всероссийской научно-практической конференции, посвященной памяти засл. деятеля науки РФ, профессора В.Ф. Волкодавова. Самара: Изд-во ПГСГА, 2009. С. 207-209

[6] Т.А.Брыксин, Ю.В.Литвинов, Технология визуального предметно-ориентированного проектирования и разработки ПО QReal

[7] Такун Е.И., курсовая работа 3 курса, Разработка формата представления графики в среде визуального моделирования QReal

[8] QtQuick 1.0, http://doc.qt.io/qt-4.8/qtquick.html

[9] QtQuick 2.0, http://doc.qt.io/qt-5/qtquick-index.html

[10] QtQuick 1.0 vs QtQuick 2.0 https://wiki.qt.io/QML1-vs-QML2

[11] Using QML Bindings in C++ Applications http://doc.qt.io/qt-4.8/qtbinding.html