САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра Системного Программирования

Веселков Иван Дмитриевич

Система автоматической раскладки элементов на сцене в проекте QReal

Курсовая работа

Научный руководитель:
Брыксин Т. А.

Санкт-Петербург
2014

Оглавление

Введение

Большинство применяемых в современном программировании технологий основаны на текстовых языках, например C++ и Java. Однако, в последнее время всё большую популярность приобретают визуальные средства разработки. При таком подходе предметная область описывается в виде множества моделей. Часто для конкретного круга задач создаются даже специальные предметно-ориентированные языки (Domain Specific Language, DSL). Это делает модели более наглядными и понятными, а также существенно увеличивает скорость решения задач. При этом, сами языки зачастую создаются на специальном языке метамоделирования и также формулируется в виде модели. Следовательно, возникает потребность в удобном инструменте, который позволит быстро и удобно создавать такие языки и работать с моделями на них.

Проект QReal, разрабатываемый на кафедре системного программирования математико-механического факультета Санкт-Петербургского Государственного университета, является инструментом, созданным именно для решения подобного рода задач [8] [9].

Программа, написанная в системе QReal, представляет собой граф. Графовое представление сложных программных проектов может быть очень тяжело для восприятия человеком, что затрудняет чтение и отладку программы. Таким образом, программист должен заботиться о понятном расположении узлов графа, то есть своей программы. Поэтому естественным желанием будет снять заботу о раскладке объектов с пользователей.

Как и в любой программе, с ростом её сложности, растёт количество элементов и сама модель. Соответственно растут усилия, которые необходимо приложить для поддержания читабельности и простоты модели. Поэтому внесение небольшого изменения может повлечь изменение всей структуры графа, после которой снова придется затрачивать силы на восстановление стройности и ясности модели. Естественным желанием стало автоматизировать этот процесс.

Целью данной работы явилось написание плагина к системе QReal, осуществляющего автоматическую раскладку элементов на сцене.

Постановка задачи

Функциональное представление плагина должно выглядеть как кнопка на панели инструментов, при нажатии на которую граф на сцене изменяется по следующим параметрам:

- Снижение числа пересечений дуг.
- Ортогональное их представление: либо прямые линии, либо с несколькими примерно прямыми углами.
- Уменьшение среднего расстояния между узлами.

Всё это должно привести к большей наглядности и читабельности графа

Три инструмента

Инструменты отбирались на основе следующих критериев:

- Написаны на C++.
- Открытый исходный код.
- Лицензия совместима с лицензией QReal (GPLv3).
- Мультиплатформенность.

GraphViz

Это программное обеспечение с открытом исходным кодом для визуализации графов и других структурированных данных[1]. Пакет разработан специалистами лаборатории ATT и распространяется с открытыми исходными файлами.

В его состав входит несколько программ верстки, которые используют простое текстовое описание графов для создания диаграмм в различных форматах: текст, изображение, PDF и т.д.

Одной из основных библиотек является dot, предназначенная для многоуровневой отрисовки ориентированных графов. Она располагает рёбра в одном направлении, уменьшает их длину и пытается избежать пересечений.

Для того, чтобы граф выглядел наиболее просто и понятно человеческому глазу библиотека использует различные алгоритмы. Поэтому задачей было научиться применять их к нашему графу. Но для этого нужного было бы описать его на языке DOT, который использует GraphViz [7].

В первую очередь, выбор пал именно на этот инструмент, потому что он достаточно распространен, есть множество приложений, использующих его в своей работе, на которые можно было ориентироваться в качестве примера.

Однако, возникли сложности по подключению GraphViz'а как простой библиотеки, и получению из него конкретных алгоритмов раскладки. Также, работу ограничивало малое количество документации.

Поэтому были предприняты поиски альтернативы.

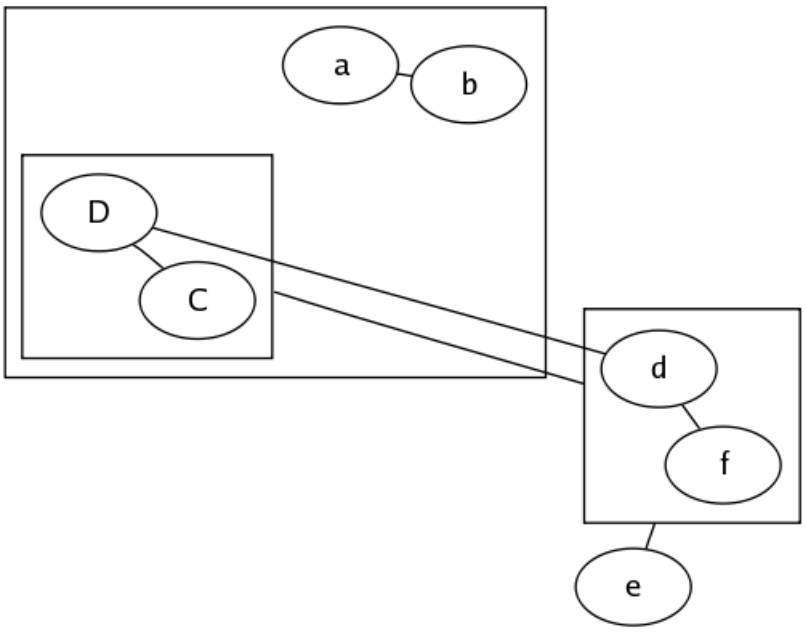

Рис. 1: Пример результата работы GraphViz

[1]http://www.graphviz.org/Documentation.php

Tulip

Фреймворк для анализа и визуализации реляционных данных.[2]

Написанный на C++, фреймворк позволяет разрабатывать алгоритмы, визуальные кодировки, методы взаимодействия, модели данных и предметно-ориентированные визуализации. Легкое повторное использование компонентов является одной из целей Tulip'а и позволяет сосредоточиться на реализации задуманных проектов.

Фрейморк представляет из себя большое количество связанных библиотек, утилит и приложений, которые предоставляют огромный выбор по видам визуализации данных. Встроенные алгоритмы позволяют изменять расположение элементов, размер, цвет, вычислять метрики многое другое [1].

Однако это обилие функциональности сказалось на удобстве работы. Чтобы применить один конкретный алгоритм нужно также подключить немалое количество зависимостей, которые сильно увеличивают конечный размер плагина, примерно на несколько десятков мегабайт.

[2]http://tulip.labri.fr/TulipDrupal/

OGDF

Open Graph Drawing Framework[3] – самодостаточная библиотека C++ с открытым исходным кодом для автоматической разметки диаграмм, при помощи встроенных алгоритмов и структур данных. Разработкой и поддержкой библиотеки занимается сообщество, в основном состоящее из студентов и преподавателей университетов Германии [3].

OGDF был создан на основе C++-библиотеке AGD [6], популярной в прошлом и покрывающей широкий спектр структур и алгоритмов по отрисовке, в которой, однако, были проблемы с кроссплатформенностью и расширяемостью [2]. Создатели OGDF сохранили основные преимущества предшественника и избавилась от его недостатков.

Таким образом библиотека:

- Обладает разнообразным набором алгоритмов, которые можно неоднократно использовать и изменять отдельные фазы, засчёт их модульной структуры.
- Оснащена различными открытыми интерфейсами структур данных, часто используемых при отрисовке графов.
- Имеет самодостаточный код, не требующий внешних библиотек.
- Написана на C++, который поддерживает наиболее важные компиляторы для операционных систем Windows, Linux и MacOS.
- Предоставляет доступ к исходному коду под лицензией GNUv3.

Сравнительный анализ библиотек

GraphViz	Неудобно использовать в качестве только библиотеки и мало документации.
Tulip	Фреймворк для визуализации данных с большим количеством ненужных нам функций.
OGDF	Кросплатформенная библиотека с различными алгоритмами отрисовки и открытым исходным кодом.

На основе выше приведенного анализа, было принято решение задействовать именно OGDF.

[3]http://www.ogdf.net/doku.php

Строение OGDF

Библиотека состоит из базовых структур данных, различных классов графов и алгоритмов расположения. Последние можно настраивать и изменять с помощью встроенных модулей.

В базовые помимо массивов, строк, очередей и других структур входят элементы для параллельного программирования: мьютекс, барьер, критическая секция и поток.

Среди классов графов нас интересуют только два: Graph и GraphAttributes. Первый представляет из себя класс "обычного" графа, который составляет основу для других классов графов, и содержащий всю структурную информацию (рёбра, узлы), направление рёбер и т.д. Он осуществляет это с помощью богатой реализации, использующей представление в виде списка смежности и допускающей динамические графы. По умолчанию всё ребра в OGDF являются направленными дугами, но их можно интерпретировать и как неориентированные рёбра.

Graph содержит чисто структурную информацию. Для хранения планировочной информации (координанты узлов, размер, цвет и т.д.), используются объекты класса GraphAttributes, которые связаны с графом. Несколько таких атрибутных объектов могут быть привязаны к одному графу. При создании такого объекта, мы определяем, с какими параметрами мы хотим работать в этом объекте (цвет узла, например). Стоит отметить, что даже если исходный граф изменяется (например, добавляется новый узел), то атрибуты изменятся таким образом, чтобы новый узел обладал всеми теми же полями, что и другие узлы данного графа.

В библиотеке представлено множество типов алгоритмов планировки: многоуровневые и основанные на силе (force-directed), восходящие, для деревьев, планарные и ортогональные с ломаными рёбрами. Теперь о каждом подробнее.

- Многоуровневые и основанные на силе – алгоритмы, основанные на иерархическом разбиении графа на компоненты разной детализации и представления рёбер графа в виде силовых векторов [4].
- Восходящие – алгоритмы представления графа в виде, при котором большая часть рёбер направлена в одну сторону.
- Для отрисовки Деревьев.
- Планарные – предназаначены для отрисовки графов без пересечений рёбер, когда это возможно.
- Ортогональные с ломаными рёбрами – оптимальное представление структурированных схем.

Нас интересует последний вариант, так как мы работаем с программами, а значит нам важна последовательная структура представления.

К этому типу алгоритмов относятся:

- PlanarizationLayout – применяет подход сглаживания для минимизации пересечений и подход topology-shape-metrics [5] для ортогонального расположения графа.
- PlanarizationGridLayout – то же, что и предыдущий, но отрисовывает граф по сетке.
- ClusterPlanarizationLayout – объединяет подходы topology-shape-metrics и сглаживания для отрисовки кластерных графов.

Алгоритм topology-shape-metrics, на котором основаны вышеописаный тип алгоритов, состоит из трёх шагов:

- Планаризация – определяется топология отрисовки и уменьшается количество пересечений, а также добавляются незначащие узлы на места пересечений рёбер.
- Ортогонализация – уменьшается количество изгибов рёбер, без изменения топологии графа.
- Компактизация – вычисляются финальные координаты вершин и рёбер, а также убираются незначащие узлы с пересечений.

Выбран был алгоритм PlanarizationLayout. Алгоритм отрисовывания по сетке нам не подходит, так как результат его работы выглядит хуже чем выбранный из-за особенностей сетки OGDF. А кластерный алгоритм оказался несовместим из-за отсутствия кластеров (по определению OGDF) в системе QReal.

К основным достоинствам библиотеки стоит также отнести то, что она предоставляет только нужные нам функции и, соответственно, её небольшой размер и совместимость

Описание реализации

Архитектура QReal

QReal обладает модульной архитектурой. Это позволяет добавлять функциональность в систему QReal в виде плагинов, которые могут обращаться к конкретным частям системы, не усложняя общую работу программы.

Основными модулями QReal являются:

- **qrkernel** содержит основные классы системы (Id, Exception и т.д.).
- **qrutils** содержит код, который может быть полезен в других компонентах. системы. Также, сюда же выносятся общие части подсистем, используемых в разных плагинах.
- **qrgui** – пользовательский интерфейс и система управления плагинами.
- **qrrepo** – репозиторий, где хранятся нарисованные диаграммы. Но плагины получают к нему доступ через qrgui, чтобы пользовательский интерфейс знал о вносимых изменениях.
- **qrxc**, **qrmc** – компиляторы метамоделей.

Но модульность задает определенные требования к плагинам:

- Для реализации нового плагина не должно возникать необходимости изменять исходный код системы, и весь относящийся к QReal код должен иметь возможность храниться в отдельной ветке репозитория.
- Плагин может сам обращаться к функциональности общей части, например, для подсветки текущего исполняемого элемента.
- Плагин должен обладать возможностью настраивать внешний вид GUI.
  Если подключено одновременно несколько плагинов с такими возможностями, пользователь должен иметь возможность выбрать, с чем он сейчас работает.

Всю систему можно описать следующим образом (здесь стрелакми обозначены зависимости при сборке):

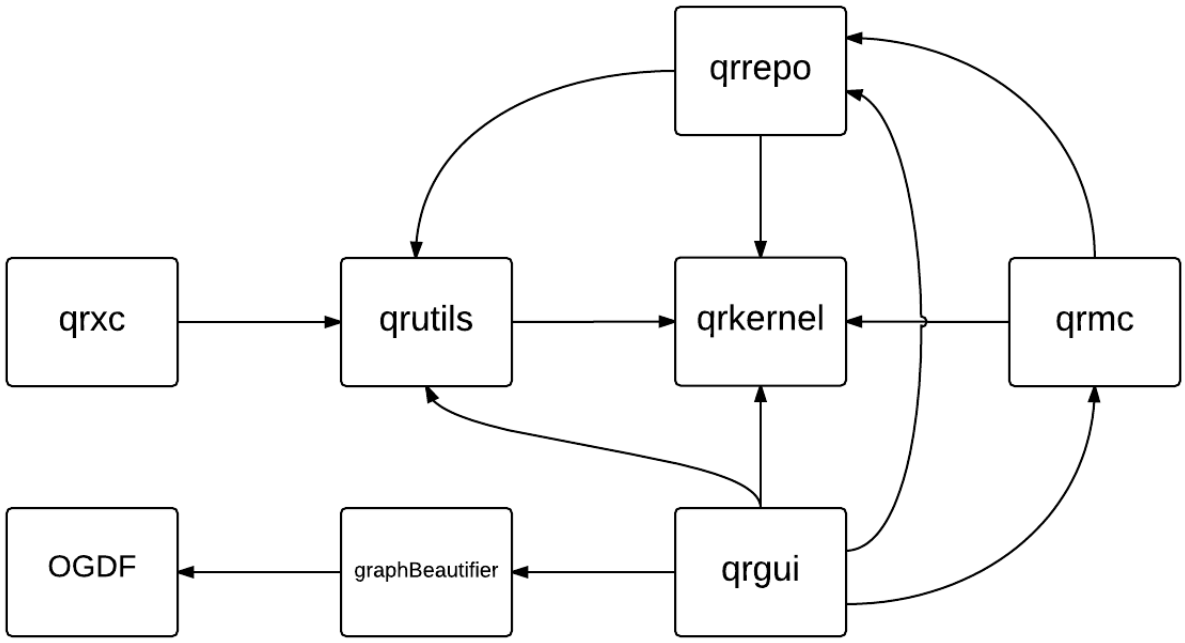

Рис. 2: Архитекрутра модулей с подключенным плагином

Строение плагина

Основная идея плагинов заключается в том, что они добавляют свои кнопки в панели инструментов и меню и предоставляют обработчики для событий нажатия на эти кнопки. Плагины реализуют специальный интерфейс toolPluginInterface, который позволяет GUI и системе управления плагинами QReal получить информацию о функциональности плагина, а также информацию о том, в какие меню и на какие панели инструментов эта функциональность должна быть добавлена. Объекты класса ActionInfo содержат методы, которые возвращают виджет конфигурации плагина (он будет добавлен в окно настроек), метод инициализации плагина (он вызывается системой управления плагинами в начале работы), а также метод, который возвращает интерфейс для настройки внешнего вида пользовательского интерфейса под конкретный плагин [10].

Через toolPluginInterface интерфейс осуществляется доступ к графическому интерфейсу GraphicalModelAssistInterface, который предоставляет методы для получения информации о графе проекта: количестве узлов и соответствующих им рёбер.

Порядок действия для этого таков:

- Из mainWindowInterpretersInterface извлекается информация об активной в текущий момент диаграмме (метод activeDiagram()).
- Через графический интерфейс узнаем список её “детей” (children()). В ответ получаем список элементов на сцене.
- Проходя циклом по этому списку, преобразовываем его в граф.

Получив общее представление о структуре графа, мы передаем эти данные в методы библиотеки OGDF. На выходе получаем граф, с чётко определенными координатами узлов и ломанных рёбер, значения которых передаем в исходный граф на нашей сцене.

Примеры

Рассмотрим пример работы плагина. Примеры состоят из двух рисунков: исходный граф и он же, но после завершения работы алгоритма.

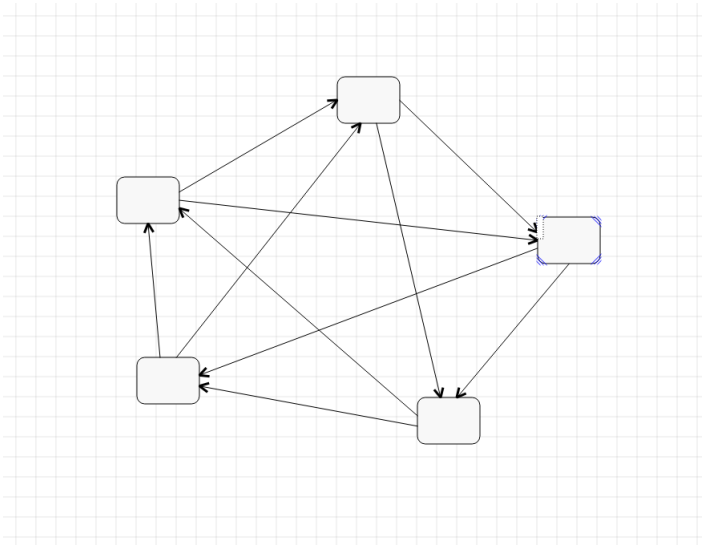

Рис. 3: Простой граф (исходный)

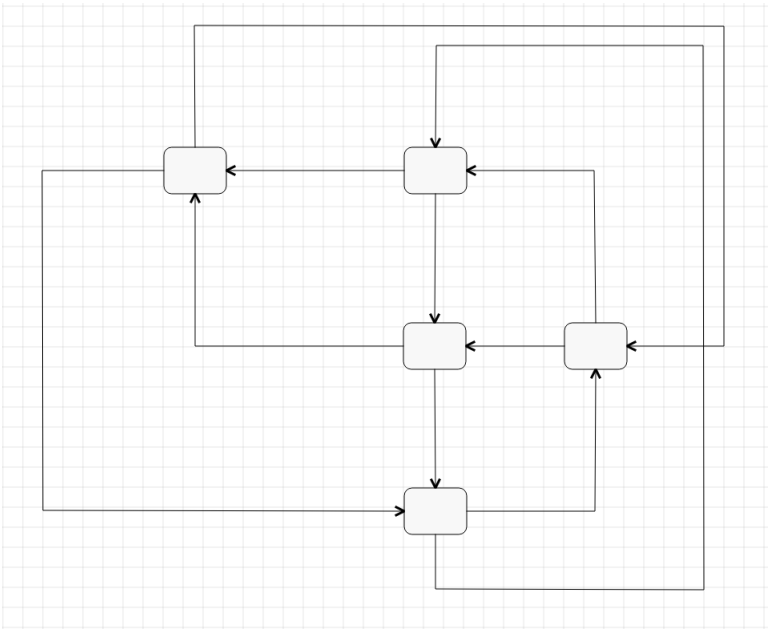

Рис. 4: Простой граф (после обработки)

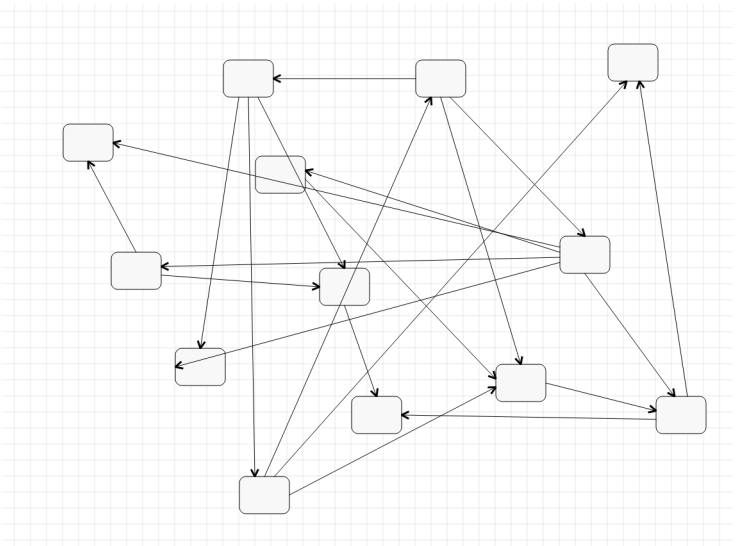

Рис. 5: Граф некой программы (исходный)

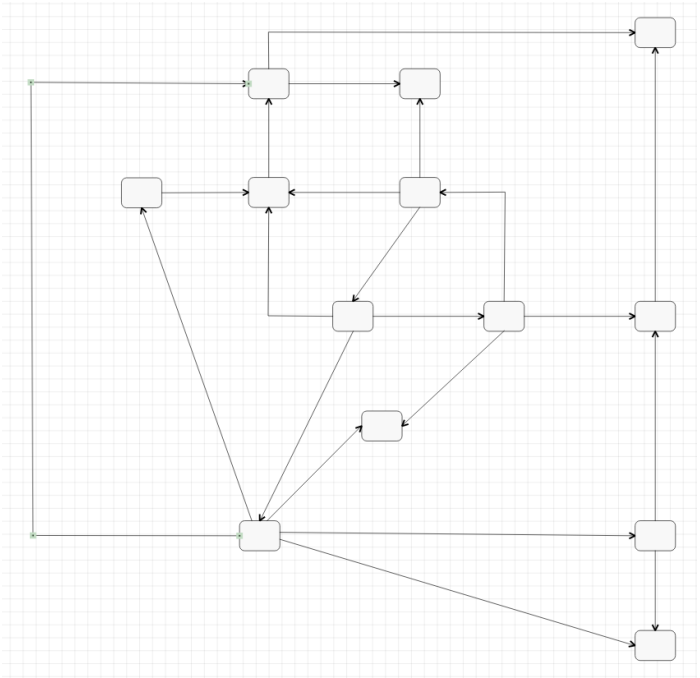

Рис. 6: Граф некой программы (после обработки)

Заключение

В результате проделанной нами работы:

- Проведен анализ различных инструментов и библиотек для отрисовки графов.
- На основе методов topology-shape-metrics библиотеки OGDF, разработан перестраивающий плагин.
- Создан плагин, встроенный в приложение QReal.
- При его запуске обрабатывается сцена и изменяются координаты исходных узлов и рёбер, приводя исходный граф к виду более читабельному и удобному для восприятия.

Список литературы

[1] David Auber, Patrick Mary. Tulip Description // Official cite. — http://goo.gl/PpbVS.

[2] Lehrstuhle Unterschiedlicher Universitäten Deutschlands. About OGDF // Official cite. — http://goo.gl/hNknRP.

[3] Lehrstuhle Unterschiedlicher Universitäten Deutschlands. OGDF Description // Official cite. — http://goo.gl/ewjcNJ.

[4] OGDF Community. Description of FMMMLayout Class // Documentation. — http://goo.gl/mOCWQN.

[5] Paul Klose. A Generic Framework for the Topology-Shape-Metrics Based Layout. — 2012.

[6] Prof. Dr. Michael Junger. AGD - Algorithms for Graph Drawing // Official cite. — http://goo.gl/1Xg1Ot.

[7] Team Graphiz. Graphviz Documentation // Official cite. — http://goo.gl/ThtUZ.

[8] А.Н. Терехов, Т.А. Брыксин, Ю.В. Литвинов и др. Архитектура среды визуального моделирования QReal Системное программирование. Вып. 4. — СПб : Изд-во СПбГУ, 2009. — С. 171–196.

[9] Т.А. Брыксин. Введение в предметную область проекта // GitHub. — http://goo.gl/1eYaF7.

[10] Ю.В. Литвинов. Плагины в QReal // GitHub. — http://goo.gl/LtnHUK.