Санкт-Петербургский Государственный Университет
Математико-механический факультет

Кафедра системного программирования

Переиспользование кода в визуальных языках программирования.

Курсовая работа студента 345 группы
Нефёдова Ефима Андреевича

Научный руководитель ст. пр. Литвинов Ю.В.

Санкт-Петербург
2012 год

Оглавление

Введение

CASE-системы

До сегодняшнего дня наблюдается рост интереса к CASE-системам[1].
CASE - *Computer-Aided Software Engineering* - набор инструментов и методов программной инженерии для проектирования программного обеспечения. В функции CASE входят средства анализа, проектирования и программирования. С их помощью можно осуществлять проектирование интерфейсов, документирование и производство кода. CASE-системами являются и среды визуального программирования.

Именно для создания специализированных сред визуального программирования в рамках научно-исследовательской группы исследования визуальных модельно-ориентированных технологий разработки ПО на кафедре системного программирования Санкт-Петербургского Государственного Университета разрабатывается meta-CASE средство QReal.

Показательными примерами применения средства QReal являются следующие CASE-системы:

- QReal:Ubiq[2] - технология для генерации клиент-серверных приложений со сложной логикой, клиент которых работает на мобильных устройствах. Программирование ведётся в среде QReal на визуальном языке, созданном средствами QReal, в качестве целевой платформы для генерации используется Ubiq Mobile[1] - программная платформа, предназначенная для разработки распределенных мобильных сервисов.
- QReal:Robots - среда программирования роботов Lego MindStorms[2] NXT с помощью предметно-ориентированного[3] визуального языка программирования. Имеется как возможность интерпретирования программы, так и генерация по диаграмме C++ кода для операционной системы NXT OSEK[4].

1 URL:http://ubiqmobile.com/ (дата обращения: 24.05.2012)
2 URL:http://mindstorms.lego.com/ (дата обращения: 24.05.2012)
3 Domain-specific language
4 URL:http://lejos-osek.sourceforge.net/ (дата обращения: 24.05.2012)

Переиспользование кода

Современный проект, не переиспользующий существующие разработки - исключение из общей картины.

Переиспользование кода – собственного или написанного сторонней компанией - оказывает положительное влияние на все важные аспекты процесса разработки.

1. Уменьшает время разработки - возможность реализовать отдельные модули системы на основе уже существующих разработок.
2. Уменьшает стоимость разработки - даже при использовании в проекте платных платформ или библиотек, общая стоимость проекта уменьшается за счёт:
    - Сокращения затраченного времени
    - Высокого качества кода, гарантированного большим количеством специалистов, заинтересованных в его переиспользовании
    - Отсутствие необходимости поддержки стороннего кода
3. Упрощает поддержку, поскольку позволяет одновременно исправлять ошибки в нескольких проектах, использующих общий код.

Прекрасным примером по переиспользованию может служить накопленная математиками исследовательская практика: сама структура понятий и доказательств, способы введения абстракций и применения исключительно абстрактных понятий в частных случаях.

Также важно понимать, что как и в математике качество использования готовых компонентов системы зависит от знания о них[3].

Постановка задачи

QReal:Robots - средство программирования роботов, одним из основных применений которого считается обучение основам программирования, кибернетики.

Но его можно использовать и для написания более сложных программ, в том числе QReal:Robots неоднократно использовался для программирования роботов LEGO MindStorms NXT для участия в соревнованиях, в рамках всевозможных кибернетических выставок.

При создании программ такого масштаба существующие ограничения, накладываемые на программу - одна диаграмма, элементы на которой соединены в цепь (где связь между объектами означает последовательную передачу управления (рис. 1)) вызывают неудобства.

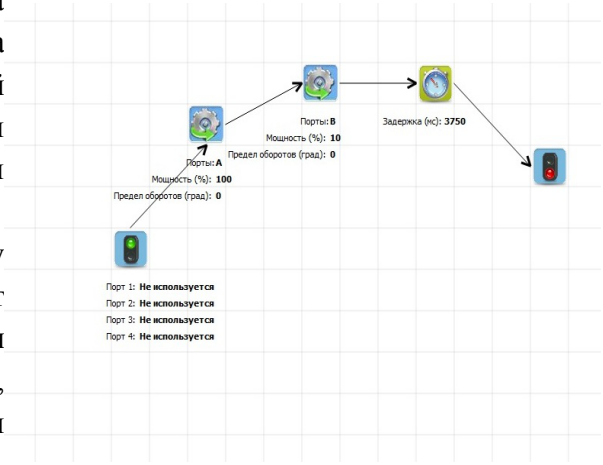

Рисунок 1

Программист, привыкший к структурному стилю программирования, интуитивно ожидает возможности использовать подпрограммы для декомпозиции сложных действий, переиспользования кода, улучшения читаемости программы.

Именно из необходимости приблизить работу с QReal:Robots по удобству к существующим текстовым языкам программирования сформировалась моя задача:

1. Реализовать возможность вызова подпрограмм в QReal
2. Рассмотреть другие методы переиспользования кода
3. Адаптировать их применительно к визуальным языкам программирования
4. Добавить в QReal инструменты для применения выбранных техник переиспользования
5. Апробация в Qreal:Robots

Исследование переиспользования

Поскольку неясно, что именно должно представлять из себя переиспользование работы программиста на визуальном языке - для начала стоит рассмотреть существующие меры, разработанные для текстовых языков. И затем обобщить их, ведь текстовые языки можно рассматривать как узкое подмножество визуальных, где для изображения программы используются ключевые слова, идентификаторы и прочие синтаксические элементы, линейно упорядоченные в файле с исходным кодом.

Методы переиспользования

Поскольку требование разрабатывать программы, в которых выделены автономные, легко заменяемые фрагменты, решающие замкнутые задачи, является общелогическим, то неудивительно, что за всё время развития культуры программирования было выделено много способов переиспользования кода[4].

Переиспользование в малом

Вероятно самый старый из существующих методов переиспользования - копирование необходимого участка кода, вставка в нужное место с последующей правкой параметров, имён переменных, названий функций.

Минусы этого метода:

- Возможно разнесение ошибок, при этом для исправление ошибки её необходимо локализовать во всех размноженных местах, что заметно усложняет отладку
- Если во всех местах, где используется код, предполагается идентичное поведение, то аналогичным образом многократно увеличивается сложность модификации кода, поскольку нужно сразу же исправлять во всех местах - высока вероятность пропустить важное изменение и потратить большое количество времени на отладку

Плюсы:

- Его скорость, простота
- Очевидная альтернатива написанию схожих участков алгоритмов в императивных стилях программирования

Переиспользование по интерфейсу

Исключительно популярный метод переиспользования, его сущность заключается в том, что код, которой планируется переиспользовать, оформляется в виде функции, а необходимые параметры передаются либо по значению, либо по ссылке.

Это является частным случаем модульности системы, а также простейшим методом переиспользования, удовлетворяющим DRY[1]-принципу

Плюсы:

- Метод лишён всех минусов метода переиспользования в малом.
- При правильном оформлении имени функции в большинстве случаев отпадает необходимость изучать её тело.
- Функция может быть параметризованная - generic - для ещё большей гибкости в переиспользовании

Использование библиотеки

Статической библиотекой считаются методы, функции и классы, выделенные в отдельный модуль. Библиотека может являться самостоятельным продуктом и, хотя в использовании библиотечные сущности принципиально не отличаются от пользовательских, имеются несколько идеологических особенностей:

- Разрабатывается, отлаживается и поддерживается отдельно - наибольшая экономия собственных ресурсов
- Проект зависит только от предоставляемого интерфейса - смена функциональности может быть произведена быстро и безопасно. Это покрывает как возможность исправить поведение используемой библиотеки разработчиком, так и замену библиотеки на конкурирующую, при условии идентичности интерфейсов
- Иногда возможно собственноручное изменение библиотеки, но это исключает совместимость с последующими версиями, её поддержку.

Наследование

Этот тип переиспользования является одной из основ ООП. Суть этого метода заключается в том, что класс может быть определён “наследником” и использовать “родительские” методы как собственные, или переопределять их.

То, что “наследник” всегда может быть использован там же, где и “родитель”, также является одним из основных положений, обеспечивающих переиспользование.

Плюсы:

- Тонкое управление поведением кода в наследниках
- Широкие возможности использования паттернов

1 Don't repeat yourself// wikipedia.org – свободная энциклопения. URL:http://en.wikipedia.org/wiki/Don't_repeat_yourself (дата обращения: 24.05.2012)

- Содержит все плюсы офомления кода функцией

Минусы:

- Относительная сложность планирования и построения иерархий

Аспектно-ориентированное программирование

Появилось из необходимости переиспользовать сквозную функциональность. Типичными примерами сквозной функциональности являются поддержка транзакций и проверка прав доступа: их использование в рамках ООП подразумевает вызов соответствующих функций из всевозможных частей программы, что очень сильно ухудшает читаемость кода, усложняет отладку.

Предложено ввести формальное определение подобной функциональности: аспект. Часто в качестве аспектов используют логирование, профилирование, транзакции, проверка прав доступа. Существуют также правила применения аспектов к вызовам методов или обращениям к полям.

Плюсы:

- Позволяет выделить сквозную функциональность на абстрактном уровне

Обобщение на визуальные языки

Проблема, заметная практически сразу - графические языки могут очень сильно отличаться от текстовых, и дело не только в том, что программа может быть представлена большим количеством различных способов: множества визуальных блоков, графы с различными связями, а также любой более экзотический вид может оказаться удобным для решения каких-либо задач.

QReal - meta-CASE система, предоставляющая инструментарий для создания визуальных языков. И только пользователь определяет структуру, правила языка, результатом выполнения программы на котором может являться практически всё, что угодно. Также определения понятий “функция” и “вызов” в языке вовсе не обязательны.

Поэтому стоит получше рассмотреть самое основное различие выделенных методов переиспользования: вызов функции против копирования.

Переиспользование копированием

Не нуждается в модифицировании: его использование в текстовых языках заключается в полном копировании данных с последующей вставкой, именно такого же поведения стоит ожидать и при употреблении такого инструмента в визуальных средах.

Отсутствие зависимости от логики языка делает данный метод ещё более привлекательным для использования в рамках QReal.

Поскольку в графических языках программа представляется одной или несколькими диаграммами, то помимо классической пары команд “копировать\вставить”, такой функционал можно использовать для импорта подпроектов.

Импорт позволит составлять наборы шаблонов, полезных, например, с языком похожим на UML, а также добавит возможность разрабатывать проект параллельно.

В рамках данной курсовой работы было принято решение реализовать импорт проекта, разработка буфера обмена для привычных операций копировать, вырезать, вставить - ведётся параллельно.

Переиспользование по интерфейсу

При работе с QReal пользователи могут быть разделены на две группы:

- Разработчики предметно-ориентированного языка - системные программисты
- Разработчики прикладных программ

Поэтому предугадать как будет реализован вызов функции в языке или определена суть переиспользования, применительно к каждому конкретному языку практически невозможно. А поскольку для успеха meta-CASE система должна обеспечивать наибольшую гибкость создания CASE-систем, задача внедрения возможности использования библиотек сводится к созданию наиболее универсального инструмента, позволяющего строить обобщённые связи между объектами.

Получены предварительные требования для необходимой функциональности, перед внедрением следует провести поиск готовых решений и согласование с ними.

Поиск готовых решений

Для проверки полученных предварительных спецификаций следует проверить методы переиспользования, предлагаемые популярными существующими CASE-средствами.

На рассмотрение были выбраны несколько CASE-систем, позволяющих создание и редактирование UML-диаграмм, программ на собственных визуальных языках, в том числе с функцией преобразования в код на C++, C#, Java.

NClass[1]

Бесплатный редактор UML-диаграмм, поддерживается диаграмма классов для языков C# и Java, умеет генерировать код по диаграмме.

Имеется функционал, применимый для переиспользования: “import an assembly” - создаёт диаграмму, соответствующую выбранному *.dll файлу - динамической библиотеки Windows

Visual Paradigm[2]

Visual Paradigm - визуальное средсво моделирования, поддерживает все типы диаграмм UML, разработку баз данных, SysML.

Есть возможность выделить часть проекта - любой набор элементов диаграммы - и сохранить в файле с отдельным расширением для последующего импорта в проекты. Полностью соответсвует методу переиспользования копированием.

Завлена возможность помечать свойства “стереотипом” для быстрого добавления этого свойства другим элементам - проверено не было, поскольку доступна только в платной версии - аналог АОП.

Для удобной командной работы диаграммы, созданные в Visual Paradigm, подлежат версионированию, для корпоративных клиентов доступны сервера контроля версий с системой разрешения конфликтов.

Altova UModel[3]

Позволяет создавать модели приложений в UML с последующей генерацией кода Java, C#, Visual Basic .NET или документации, возможно обратное построение диаграмм по коду. Поддерживаются 14 типов диаграмм UML, диаграммы SQL БД, SysML модели, бизнесс-процессы.

1 URL: http://nclass.sourceforge.net/ (дата обращения: 24.05.2012)
2 URL: http://www.visual-paradigm.com/ (дата обращения: 24.05.2012)
3 URL: http://www.altova.com/umodel.html (дата обращения 24.05.2012)

Для переиспользования можно использовать функцию слияния проектов, или включения под-проекта, они заключаются в добавлении выбранных диаграмм в проект, добавлении выбранной диаграммы в текущую.

StarUML

Программный инструмент моделирования, который поддерживает UML. Поддерживает стандарт UML 1.4, нотацию UML 2.0 и концепцию профилей UML, позволяя создавать платформенно-независимые модели.

Есть возможность разделить проект на несколько секций, разрабатывать их раздельно и, при необходимости, объединить обратно в один проект. В качестве секций могуть быть сохранены пакеты, модели и подсистемы.

Simulink[1]

Simulink - среда для модельно-ориентированого дизайна динамических или встроенных систем. Она предоставляет графический интерфейс и настраиваемые наборы блоков, которые позволяют составлять, симулировать, реализовывать и тестировать различные динамические системы: управления, связи, обработки данных.

Simulink имеет широкие возможности переиспользования по интерфейсу: его язык имеет абстракции “вход” и “выход” - они позволяют рассматривать любой связанный набор блоков, начинающийся “входом” и заканчивающийся “выходом” в качестве единого блока. Поддерживается автоматическое сворачивание таких схем и возможность использования этого пользовательского блока наравне с заранее заготовленными.

Это является прямой аналогией оформления функций и библиотечных модулей.

Итог

В существующих продуктах встречаются различные способы переиспользования, самым популярным был метод основанный на копировании, как самый универсальный.
Поскольку большинство рассмотренных CASE-систем ориентируются либо на создание небольших программ, где переиспользование не сыграет большой роли, либо на создание больших, но уникальных проектов, например баз данных, где с точки зрения дизайна нет смысла переиспользовать предыдущие разработки - переиспользование по интерфейсу не распространено. Оно встретилось лишь в среде Simulink, которая ориентирована на разработку сложных программ.

Обзор существующих CASE-систем не выявил других неочевидных способов переиспользования разработок в графических средах и подтвердил жизнеспособность полученных ранее обобщений методов переиспользования.

1 URL:http://www.mathworks.com/products/simulink/ (дата обращения 24.05.2012)

Реализация переиспользования в QReal

Определение спецификаций

Поскольку предметно-зависимые языки, на создание которых ориентирован QReal, могут создаваться для решения принципиально различных задач и, вследствие этого, разительно отличаться друг от друга, желательно сохранять как можно большую гибкость в предоставляемых инструментариях.

Поэтому для внедрения был выбран следующий функционал:

- Возможность импорта проекта - переиспользование копированием
- Возможность создания библиотек из сущностей языка с последующим однообразным их использованием в диаграммах
- Работа с буфером обмена

Буфер обмена

Разработка буфера обмена началась до начала работы над данной курсовой и происходит параллельно. Поскольку интуитивность пользовательского интерфейса чётко специфицирует поведение связанных с работой буфера обмена методов, их разработка является чисто технологической и не сопряжена с исследованиями, проведёнными в рамках этой работы.

Импорт проекта

Функция, импортирующая выбранный проект в текущий, присутствует в следующих рассмотренных средтвах: Altova UModel, Visual Paradigm, StarUML. Она позволяет вести параллельную разработку непересекающихся модулей с последующим слиянием или использовать заранее заготовленные наборы шаблонов для решения типичных задач.

При широких возможностях переиспользования простота реализации обеспечивается тем, что реализация импорта не зависит от логики построения языков, а это значит, что при добавлении этой функциональности необходимы минимальные изменения существующего кода.

Работа по реализации была исключительно технического характера, позволила ознакомиться с архитектурой подсистемы QReal, работающей с файловым представлением диаграмм.

Переиспользование по интерфейсу

Добавление возможности использования подпрограмм и пользовательских библиотек в QReal:Robots послужило мотивацией к этой работе.

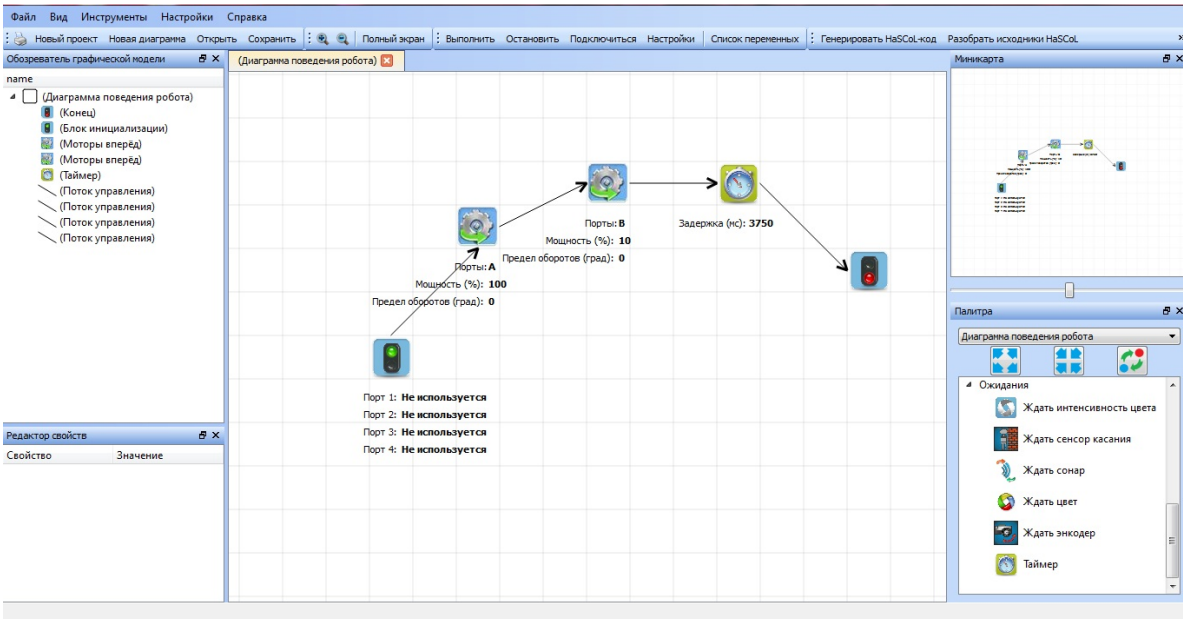

Рисунок 2

На рис. 2 изображён интерфейс QReal:Robots.

Программа является связанным между собой набором блоков, который предоставляет основную функциональность, достаточную для написания программ: управление моторами, получение данных с датчиков, а также такие логические элементы как таймер, условный переход.

Прямым конкурентом QReal:Robots является RoboLab - визуальная среда программирования роботов Lego MindStorms NXT. В нём также используется блочная система - составление программы представляет собой построение цепочки или дерева из блоков, каждый из которых инкапсулирует определённую функциональность.

Но в отличие от RoboLab, где, например, существуют блоки “Ждать 10мс”, “Ждать 50мс”, и т.д. В QReal все эти действия выполняются блоком “Ждать таймер” с изменяемым параметром длительности. Это обеспечивает большую гибкость, по сравнению с RoboLab, но для достижения этой гибкости блоки должны выполнять самую базовую функциональность: так, например программа поворота по дуге заключается в последовательном соединении следующих блоков:

1. “Моторы вперёд” активирующий мотор A на мощности 100
2. “Моторы вперёд” активирующий мотор B на мощности 10
3. “Ждать таймер” с соответствующей требуемому углу задержкой

Поскольку в сложной программе может требоваться большое количество различных поворотов, более чем логично реализовать возможность объединения некоторой части диаграммы и использования её как единого блока.

Следующим логичным шагом является выделение таких функций в отдельный файл, чтобы иметь возможность не реализовывать их каждый раз заново.

Принципиальное отличие QReal от других рассмотренных систем заключается в том, что QReal является meta-CASE системой - это среда разработки предметно-ориентированных визуальных языков программирования, поэтому частные решение не устраивают: необходимо оценить, как могут выражаться требования к библиотечному переиспользованию в различных графических языках, и предоставить как можно более общий инструмент, позволяющий авторам этих языков самостоятельно реализовать необходимую функциональность.

Вызовы в QReal:Robots

Основные сущности QReal:

- Элемент - некоторая сущность языка.
- Язык определяется набором определённых в нём типов элементов. Тип элемента определяет свойства, которыми может владеть конкретный элемент - представитель.
- Модель - представляет собой набор конкретных элементов, со всеми их свойствами

Модели разделены на логическую и графическую[5] - основная идея разделения состоит в том, что элемент модели и его визуальное представление - разные вещи. Некторые элементы могут не иметь визуального представления (например, специальные служебные сущности вроде настроек генератора) или могут иметь несколько визуальных отображений (один и тот же класс может присутствовать на нескольких разных диаграммах, причём выглядеть по-разному), другие, наоборот, могут вообще не иметь визуального представления и на логику никак не влияют (например, просто линии и прямоугольники на диаграмме).

Логическая и графическая модели тесно связаны - при работе с диаграммой происходит их постоянная синхронизация.

В QReal:Robots исполнение программы реализовано следующим образом: поток исполнения проходит последовательно по элементам графической модели, запуская интерпретацию каждого по очереди.

чтобы некоторый графический элемент мог служить вызовом подпрограммы, ему необходимо дополнительно уметь возвращать первый блок цепочки этой подпрограммы. Тогда изменения в классе, представляющем поток исполнения, будут минимальны - при

анализе блока, полученного на исполнение, может быть принято решение запустить подпрограмму по полученной точке входа, останется лишь поместить текущее место исполнения программы в стек вызовов.

Диаграмма на рис.3 представляет вариант перехода потока исполнения в подпрограмму при унифицированном подходе к получению следующего к исполнению блока.

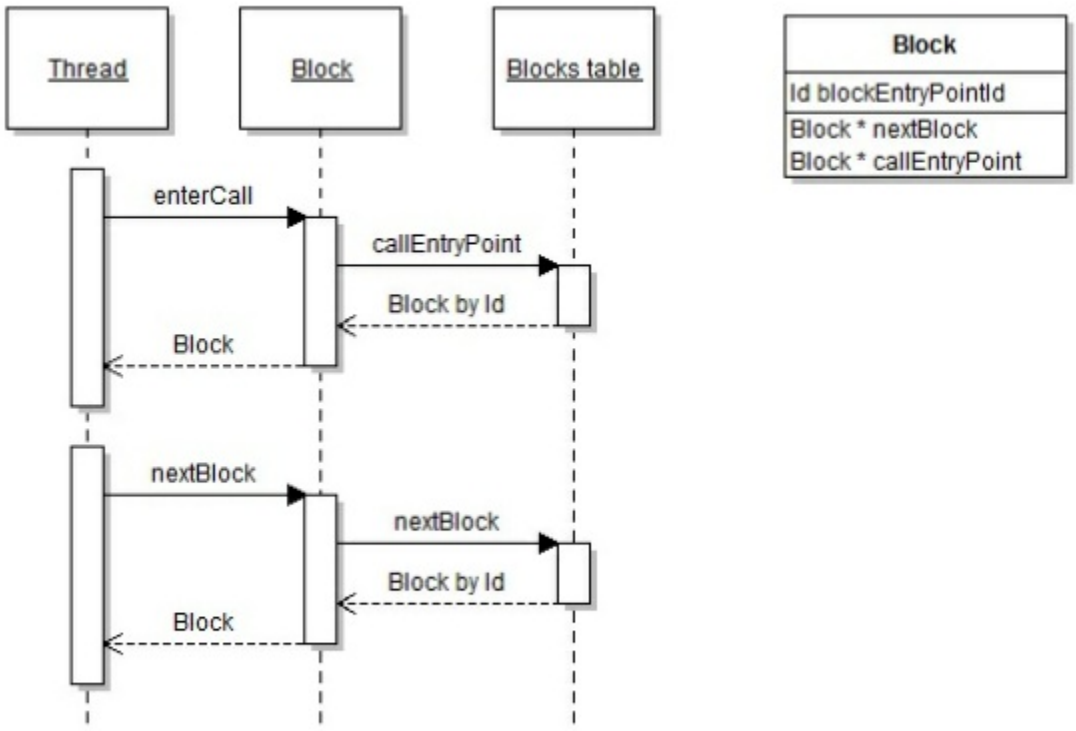

Рисунок 3

Общий случай переиспользования

В текстовых языках программирования статическая библиотека представляет из себя либо исходный код, подключаемый программистом на этапе написания, либо объектные файлы, линкуемые к исполняемой программе на этапе компиляции.

Получение наибольшей выгоды от использования библиотеки происходит из-за идеологических особенностей её использования, обсуждающихся в разделе “Исследование переиспользования”.

Поэтому были выделены аспекты, которым должно удовлетворять решение:

1. Однородность - на его основе должно быть возможно построение переиспользования так, чтобы с точки зрения пользователя не было разницы в применении стандартных или библиотечных функций
2. Закрытость - редактирование библиотеки должно происходить изолированно от работы с проектами, в которых она используется
3. Слабая связность - проект, использующий библиотеку, должен зависеть только от её интерфейса и не должен замечать подмены её реализации.

Поскольку в любом языке, созданном в QReal, программа - логически завершённое высказывание на этом языке - будет состоять из некоторого набора взаимосвязанных логических и графических элементов, возможно представить как может выглядеть связь, обеспечивающая достаточную для реализации линковки.

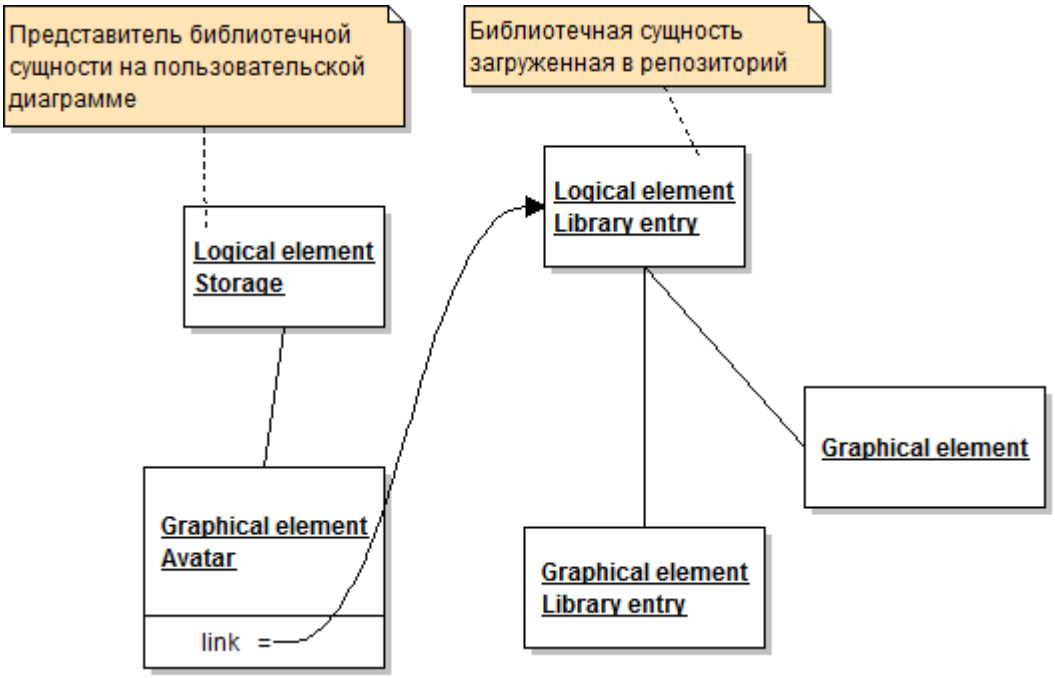

Рисунок 4

Связь между элементами, представленная на рис.4, позволит реализовать исполнение программы, рассредоточенной между несколькими диаграммами, с минимальными отличиями от исполнения программы из одного файла.

Также при такой организации связи аватар (элемент диаграммы, представляющий интерпретатору возможность выполнить библиотечный код) является полноценным элементом пользовательского проекта, работа с ним для пользователя не отличается от работы с любым другим блоком.

Загруженные библиотечные сущности хранятся в отдельной модели. Эта модель, являясь частью паттерна Model-View позволяет, с помощью стандартных средств Qt реализовать специальную палитру библиотечных элементов, работа с которой не отличается от работы с палитрой элементов QReal.

Второе ограничение, наложенное на библиотеки может быть легко удовлетворено, добавлением в язык дополнительного блока, обозначающего вызов - тогда при любой модификации библиотеки связь, используемая для линковки, останется неповреждённой, и при изменении реализации подмена в пользовательских проектах произойдёт автоматически.

Последним необходимым изменением QReal является добавление возможности работы с несколькими одновременно загруженными файлами.

Выполнение третьего ограничения наложенного на библиотеки обеспечивается здесь: хоть библиотека и загружается в общий с пользовательским проектом репозиторий, но сохранение проекта никак не затрагивает библиотечных файлов. Даже если настройки языка позволят каким-либо образом влиять на библиотечные сущности, эти сохранения не будут утверждены.

Такая реализация может использоваться в широком классе визуальных языков. Одним из примеров нестандартного языка, использовавшегося при проверке пригодности связи, обеспечивающей линковку основного проекта с библиотеками, является язык точек.

Элементами языка считаются точки и линии их соединяющие, не обладающие особыми свойствами. На рис.5 показано, как может выглядеть программа на таком языке:

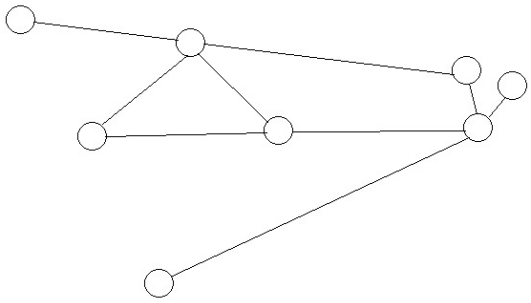

Рисунок 5

Результатом исполнения такой программы условимся считать сопоставленное каждой точке число: количество доступных из этой точки узлов, по путям длины 2.

Даже в таком языке с помощью предоставленной связи можно организовать переиспользование. Пометив любой узел связанной компоненты как библиотечную сущность, мы получаем возможность работать с этой компонентой как с единым элементом и использовать её в других проектах.

При этом требуются минимальные изменения в интерпретаторе модели этого языка.

Апробация

На рис.6 можно увидеть результат апробации полученной технологии в QReal:Robots - в контекстном меню любого графического элемента добавлено подменю “Add to library”, которое содержит индикатор текущего состояния элемента - является ли он библиотечной сущностью, и кнопку “add/remove”.

Элементы, помещённые в библиотеку при загрузке файла с сохранением, помещены в панель “Library explorer”, которая служит палитрой: перетаскивая с неё элементы на диаграмму создаётся аватар.

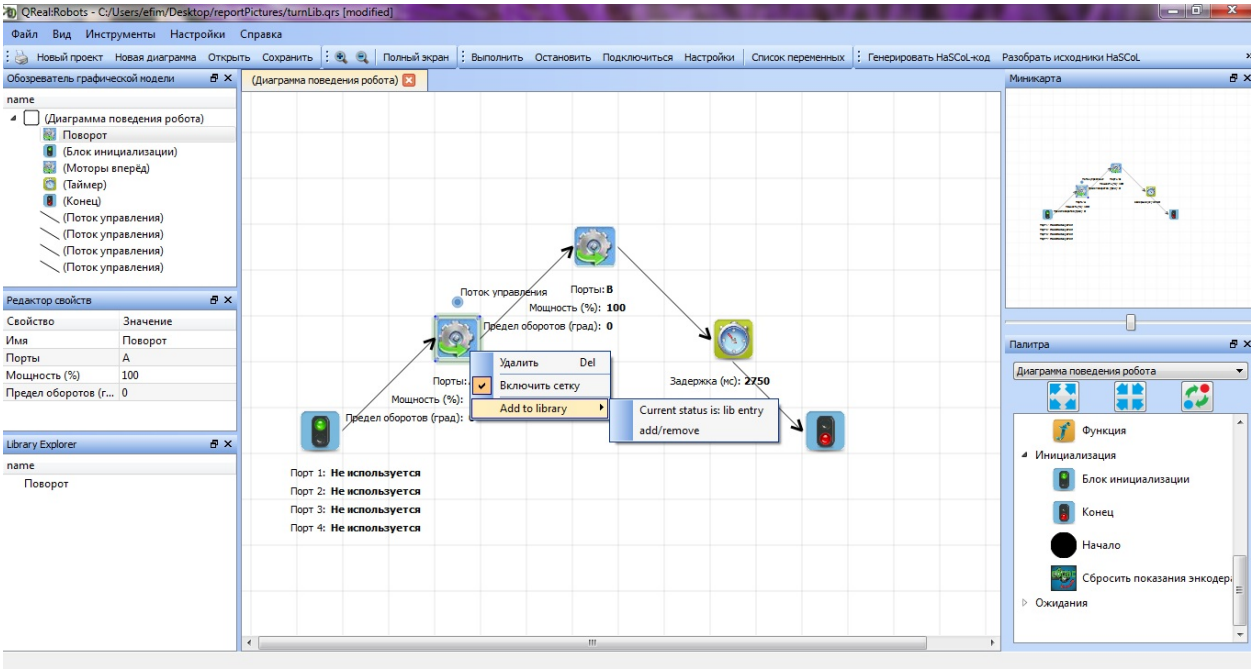

Рисунок 6

Возможно проведение юзабилити исследования и разработка полноценного интерфейса.

Желательно введение отдельного блока “вызов”, который бы упростил идентифицирование библиотечных функций на пользовательской диаграмме.

Результаты

1. Проведён анализ существующих методов переиспользования кода.
2. Построены их адаптации применительно к визуальным языкам программирования
3. В QReal добавлена следующая функциональность:
    - Импорт проекта
    - Средство построение библиотечных связей
    - Загрузка нескольких файлов одновременно
4. Проведена апробация технологии построения библиотек в QReal:Robots

Список литературы

[1] CASE //en.wikipedia.org: свободная энциплопения. URL:http://en.wikipedia.org/wiki/Computer-aided_software_engineering (дата обращения: 24.05.2012)

[2] Timofey Bryksin, Yuri Litvinov, Valentin Onossovski, Andrey N. Terekhov, Ubuq Mobile + QReal a Technology for Development of Distributed Mobile Services // 10th Conference of Open Innovations Association FRUCT and the 2nd Finnish-Russian Mobile Linux Summit: Proceedings, printed by State University of Aerospace Instrumentation (SUAI). 2011. 232 p. pp 27-35

[3] Программирование от переиспользования//excode.ru/: Программирование на высоком уровне. URL:http://www.excode.ru/art6030p2.html (дата обращения: 24.05.12)

[4] Повторное использование кода//http://ru.wikipedia.org : свободная энциклопения. URL:http://en.wikipedia.org/wiki/Code_reuse (дата обращения: 24.05.2012)

[5] Разделение графической и логической моделей//github.com/qreal/qreal/wiki:База знаний QReal. URL:https://github.com/qreal/qreal/wiki/%D0%A0%D0%B0%D0%B7%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%BB%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B9-%D0%B8-%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B9-%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B5%D0%B9 (дата обращения 24.05.2012)