Санкт-Петербургский государственный университет

Математико-механический факультет

Кафедра системного программирования

Система автоматизированного тестирования графического интерфейса в проекте QReal

Курсовая работа студента 444-ой группы

Новожилова Евгения Алексеевича

Научный руководитель Брыксин Т.А.

Санкт-Петербург

2014

Оглавление

Введение

В современном мире одним из основных этапов разработки программного обеспечения является тестирование. Данный процесс позволяет проверить различные компоненты систем на корректность их работы, выявить и исправить ошибки. Во многих компаниях существуют огромные отделы, ответственные за тестирование продуктов. Существует много различных теорий тестирования, направленных на различные уровни интеграции компонент. Одним из довольно важных видов тестирования является системное тестирование. Оно рассчитано на проверку слаженности работы всех компонент некоторой системы. Автоматизация же позволяет производить этот процесс как можно чаще без использования человеческих ресурсов. Таким образом, описав стандартные условия использования системы, можно стараться гарантировать работоспособность системы в базовых случаях.

Одним из примеров системного тестирования является тестирование графического интерфейса. В современном мире, существует не так много наборов инструментов, которые позволяют автоматизировать такой процесс. Можно отметить, что список всемирно известных инструментов для тестирования существует как страница на Википедии и содержит в себе порядка 30 элементов. На самом деле, тестирование графического интерфейса является довольно сложной задачей по причине того, что существует множество фреймворков для реализации приложений с графическим интерфейсом. Однако в случае использования определенной платформы для разработки программного продукта появляется возможность найти инструмент, который может помочь решить проблему автоматизации тестирования графического интерфейса.

В данной курсовой работе рассмотрены популярные инструменты для тестирования графического интерфейса приложений, реализованных с помощью фрейморка Qt, а также произведена реализация системы автоматизированного тестирования графического интерфейса для DSM-платформы QReal.

Глава 1. Обзор

Froglogic Squish

Froglogic Squish является платным кроссплатформенным инструментом для тестирования графического пользовательского интерфейса, созданного в помощью различных технологий. Данный инструмент разрабатывается компанией Froglogic. Для распознавания элементов графического интерфейса используется проприетарная технология, работающая с элементами, как объектами. Автоматизация тестов может быть проведена с помощью скриптовых языков, таких как: JavaScript, Perl, Python, Tcl. Инструмент состоит из двух модулей, один из которых является клиентов, а второй сервером, как показано на рисунке 2. Клиент отвечает за интерпретацию и выполнение теста. Сервер используется для контроля над приложением, находящемся в процессе тестирования. Оба компонента являются кроссплатформенными. В комплекте ПО также идет среда разработки для тестов, основанная на Eclipse. Данный инструмент обладает очень сильным недостатком в виде политики его распространения. Одна лицензия этого продукта стоит порядка $9000, что не является возможным для использования в студенческом проекте.

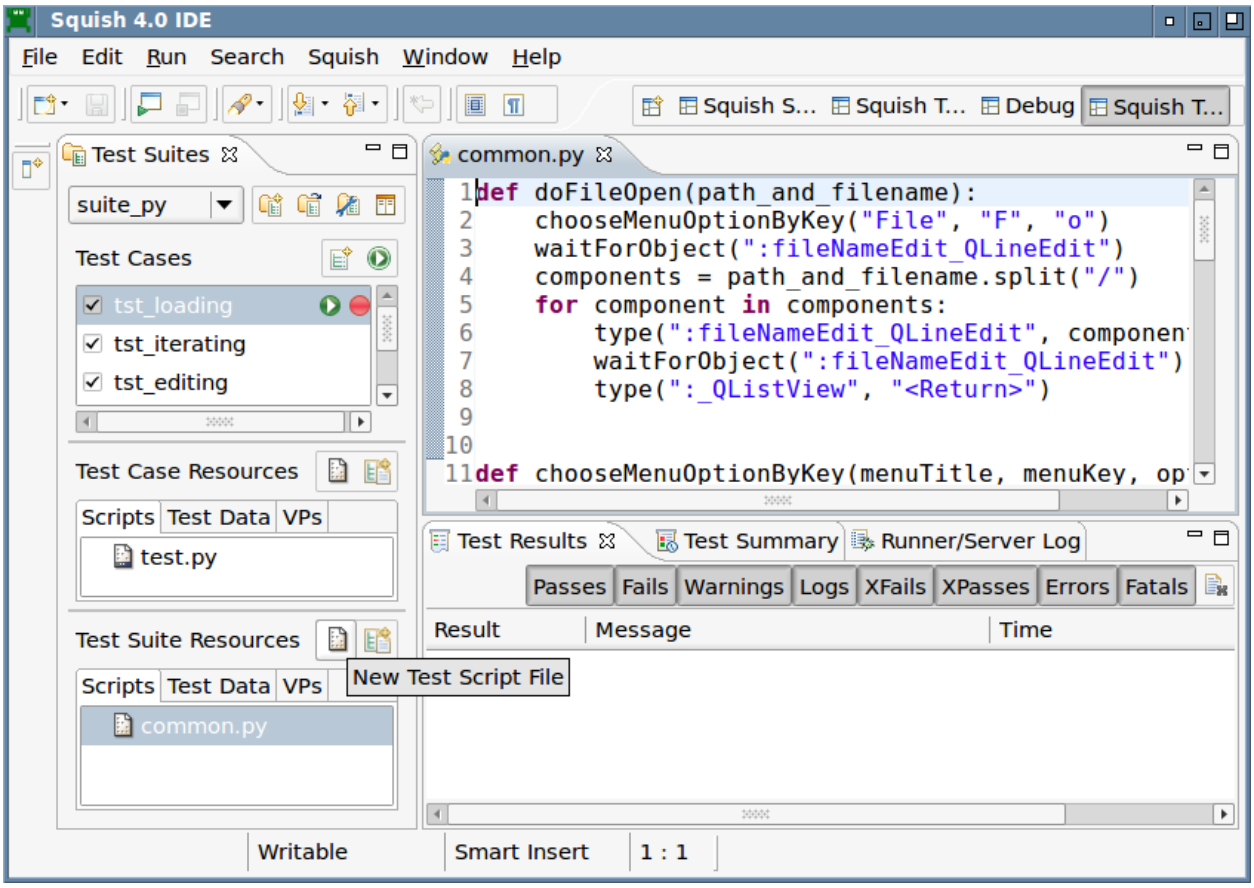

Рис. 1. Среда разработки Squish

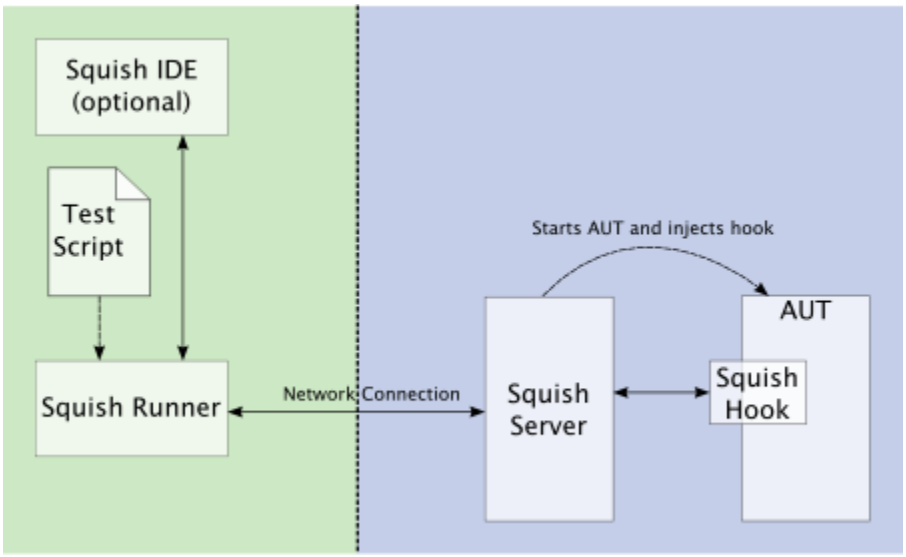

Рис. 2. Алгоритм работы Squish

QTestLib

QTestLib является набором инструментов для unit-тестирования приложений, реализованных с помощью Qt. QTestLib включает в себя все функциональность, которую можно найти во всех инструментах для unit-тестирования, а также библиотеку для тестирования графического интерфейса. В связи с тем, что данная библиотека встроена во фреймворк Qt, она позволяет оперировать одновременно как с графическим интерфейсом, так и с объектами реализации продукта. Стоит отметить недостаток данного инструмента, заключающийся в том, что поддержка событий графического интерфейса реализована достаточно низкоуровнево.

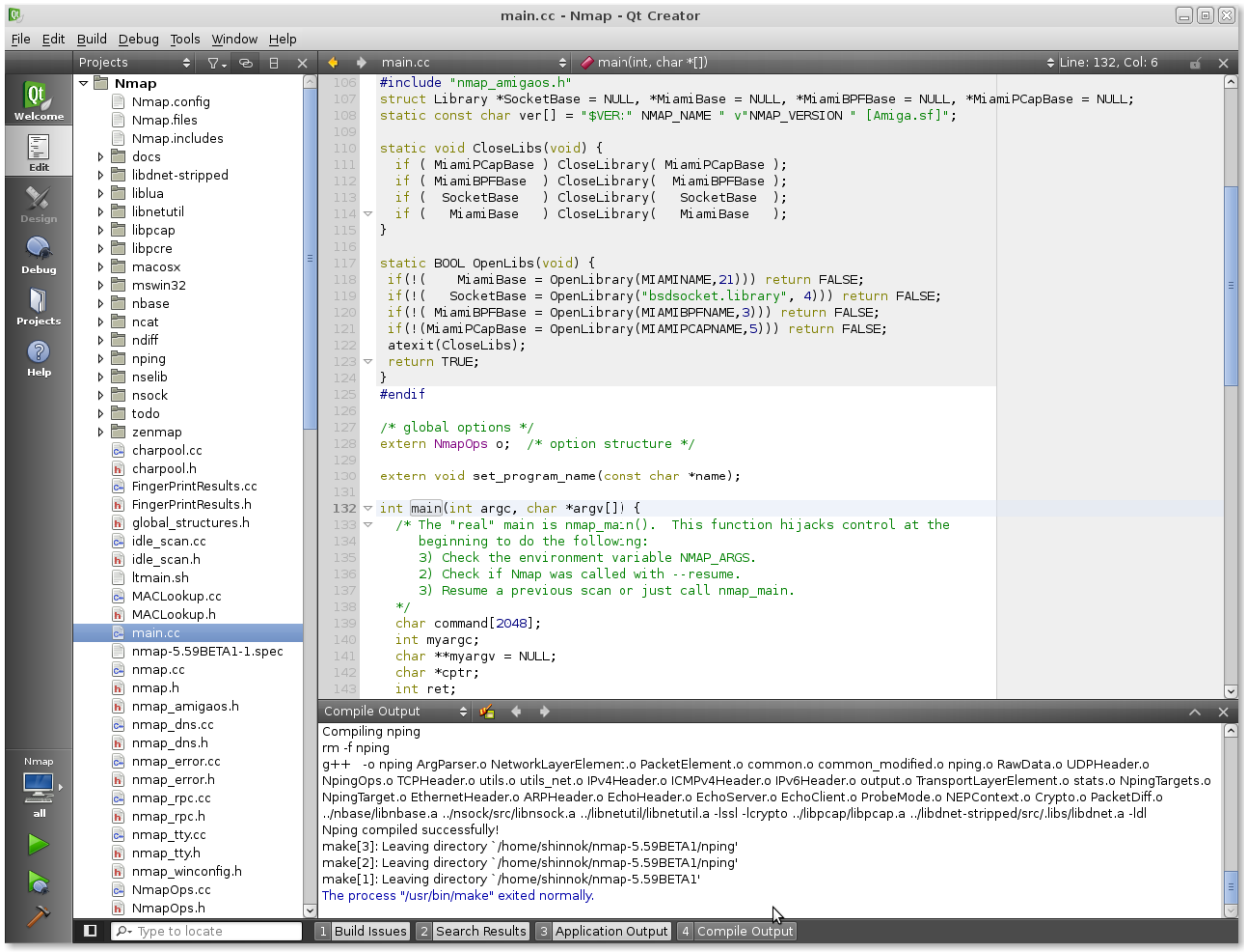

Рис. 3. Среда разработки Qt Creator, в которой можно реализовывать тесты с помощью QTestLib.

Sikuli

Sikuli является технологией, разработанной в университете MIT группой, занимающейся дизайном пользовательских интерфейсов, которая позволяет автоматизировать процесс тестирования с помощью компьютерного зрения. Компьютерное зрение используется для распознавания паттернов со снимков экранов пользовательского интерфейса. Для автоматизации необходимо использовать диалект языка Python – Jython. Язык Jython использует графические элементы, как конструкции языка, в связи с чем разработку процесса тестирования лучше всего вести в специальной среде разработки, поставляющейся в наборе. Очевидным недостатком использования данной технологии в проекте QReal является использование скриншотов в качестве инструмента для сравнения, в связи с тем, что проект QReal довольно часто меняет свой внешний вид, и тесты необходимо постоянно обновлять.

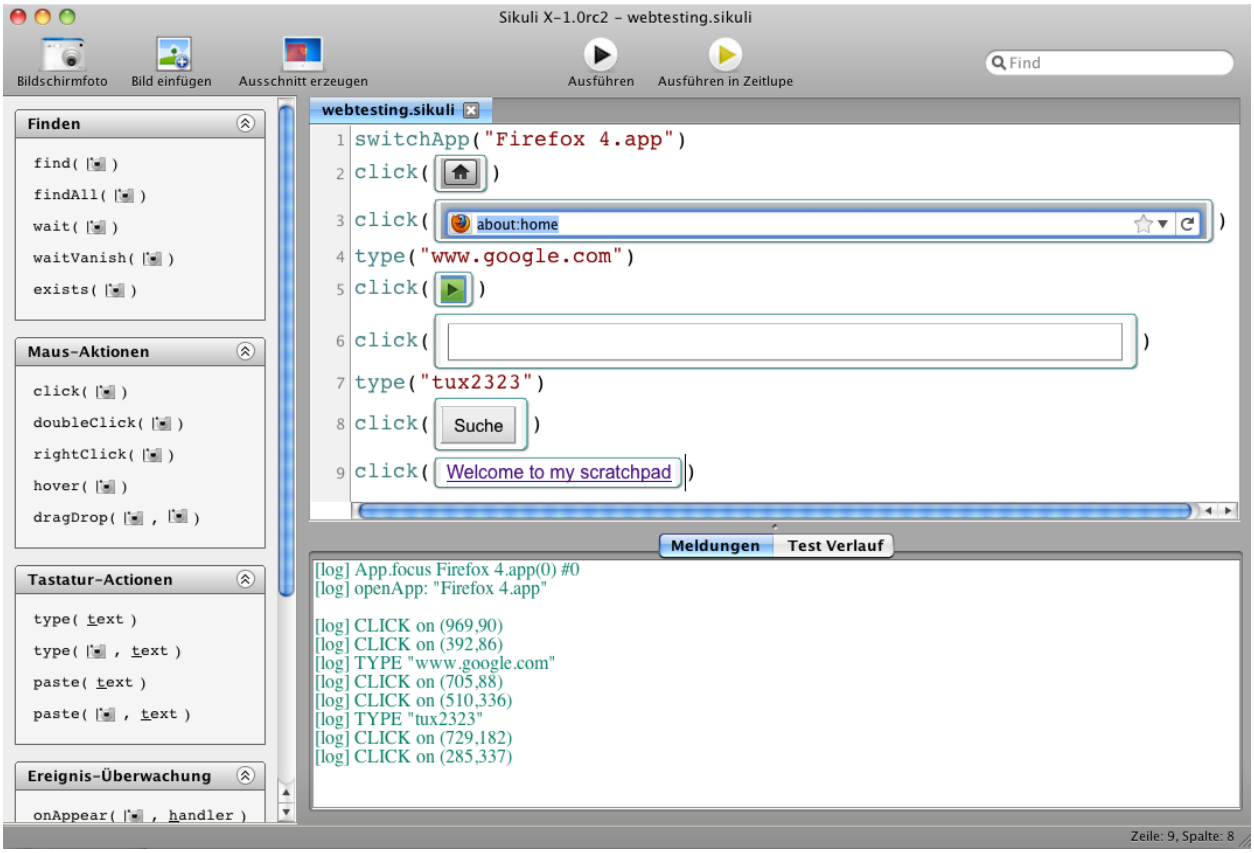

Рис. 4. Среда разработки для языка Jython в инструменте Sikuli

Xnee

Xnee – это набор программ, которые могут записывать и воспроизводить последовательность действий пользователя в X11 оконной системе. В оконной системе X11 каждое нажатие кнопки или клик мыши являются системным событием, которые, в случае записи тестового сценария, сериализованно помещаются в файл. В последующем, модуль, который реализует воспроизведение событий, использует данные из файла, который был создан на предыдущем этапе. Xnee является свободным программным обеспечением, разрабатываемым GNU Project. Для проекта QReal данный инструмент обладает недостатком в виде привязки событий к координате на экране, что вынуждает обновлять сценарии после изменений графического интерфейса.

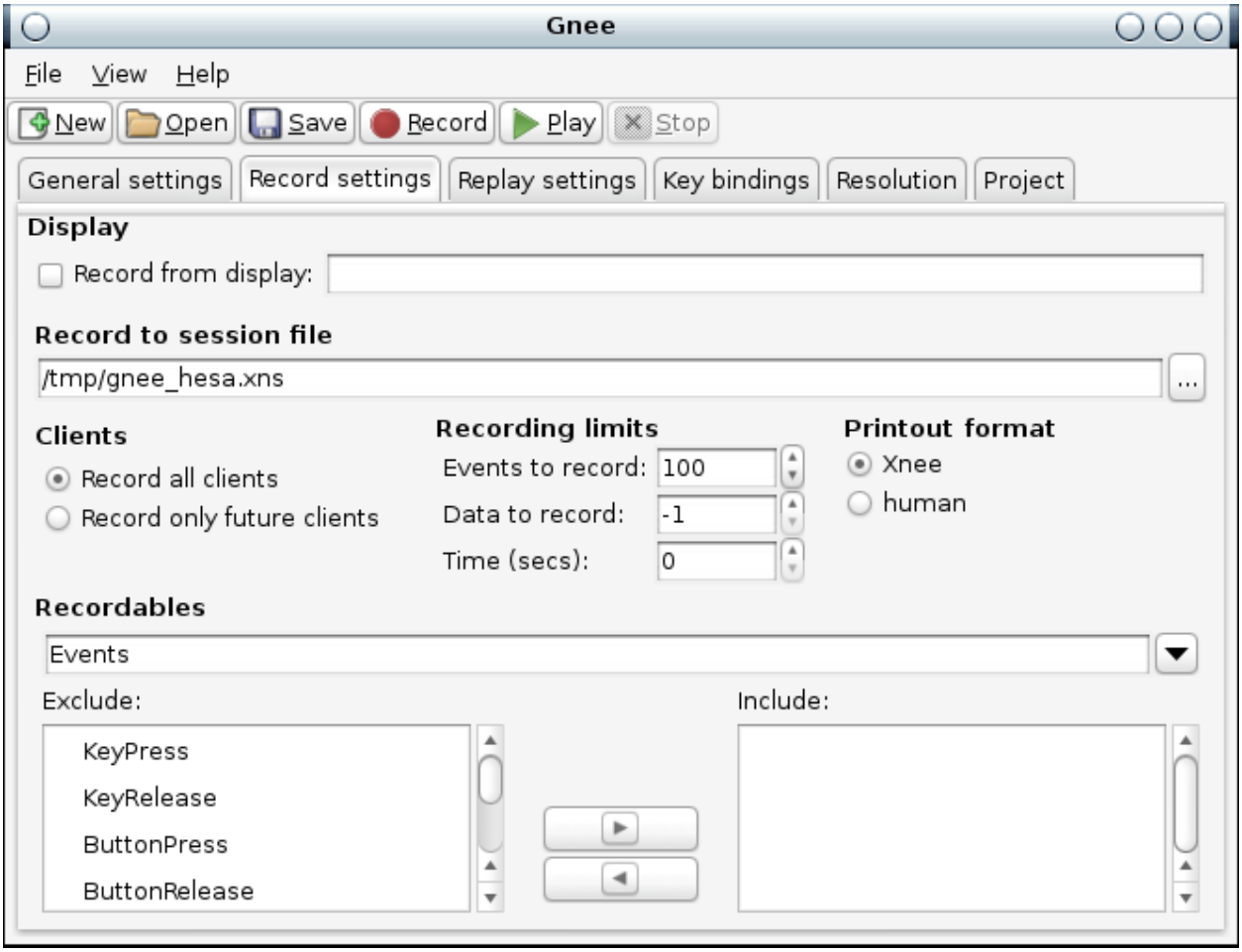

Рис. 5. Утилита gnee, предназначенная для записи и воспроизведения событий пользовательского интерфейса

LDTP

LDTP – Linux Desktop Testing Project является свободно распространяемым инструментом, который использует технологию помощи ограниченным людям для управления графическим интерфейсом. Данный инструмент рассчитан на использование под ОС GNU\Linux, но существуют различные портирования на платформы Mac OS и Windows. LDTP может использоваться для тестирования приложений в GNOME и KDE, при условии включенной функции помощи ограниченным людям. Авторы заявляют поддержку фрейморка Qt версии 4.8. В качестве разработки тестов необходимо использовать язык Python. Для проекта QReal этот инструмент обладает недостатком в виде отсутствия поддержки фреймворка Qt версии 5 и выше. Однако, существует вторая версия LDTP, которая кардинально отличается по реализации, но пока что не содержит в себе поддержки никаких других окружений, кроме как GNOME.

Итоги

Как итог данного обзора хотелось бы заметить несколько вещей. Во-первых, на данный момент не существует универсального бесплатного инструмента, который позволил бы тестировать приложение, реализованное с помощью Qt, без дополнительных действий на популярных платформах. Кардинальные различия платформ по реализации графических интерфейсов делают существование такого инструмента практически невозможным. Большинству организаций, разрабатывающих ПО с графическими интерфейсами, возможно, инструмент Sikuli будет оптимальным решением, но для проекта QReal он не удовлетворяет изначальным требованиям. Абсолютно также не удовлетворяют требованиям инструмент Xnee, которые обладает жесткой привязкой к координатам элементов. Возможно, инструмент LDTP, обладает хорошим потенциалом, но на данный момент в нем отсутствует поддержка последней версии Qt, что опять же не удовлетворяет исходным критериям. В таких условиях приходится выбирать QTestLib, не смотря на низкоуровневую поддержку событий графического интерфейса и несколько различное поведение тестов под ОС Windows и ОС GNU\Linux.

Глава 2. Реализация

DSM-платформа QReal

DSM-платформа QReal предназначена для описания визуальных языков и дальнейшего использования созданных языков в различных целях. В основном пользователь взаимодействует с приложением QReal с помощью мыши и клавиатуры. Большинство задач, выполняемых пользователем, состоит в создании и редактировании элементов визуального языка. Из всех этих задач можно выбрать подмножество, которые необходимы для демонстрации корректности работы системы. Для начала стоит рассмотреть графический интерфейс и выделить зоны, обращения к которым происходят чаще всего при использовании QReal. Большую часть графического интерфейса занимает сцена, на которой отображается диаграмма из объектов, содержащихся в модели текущего проекта.

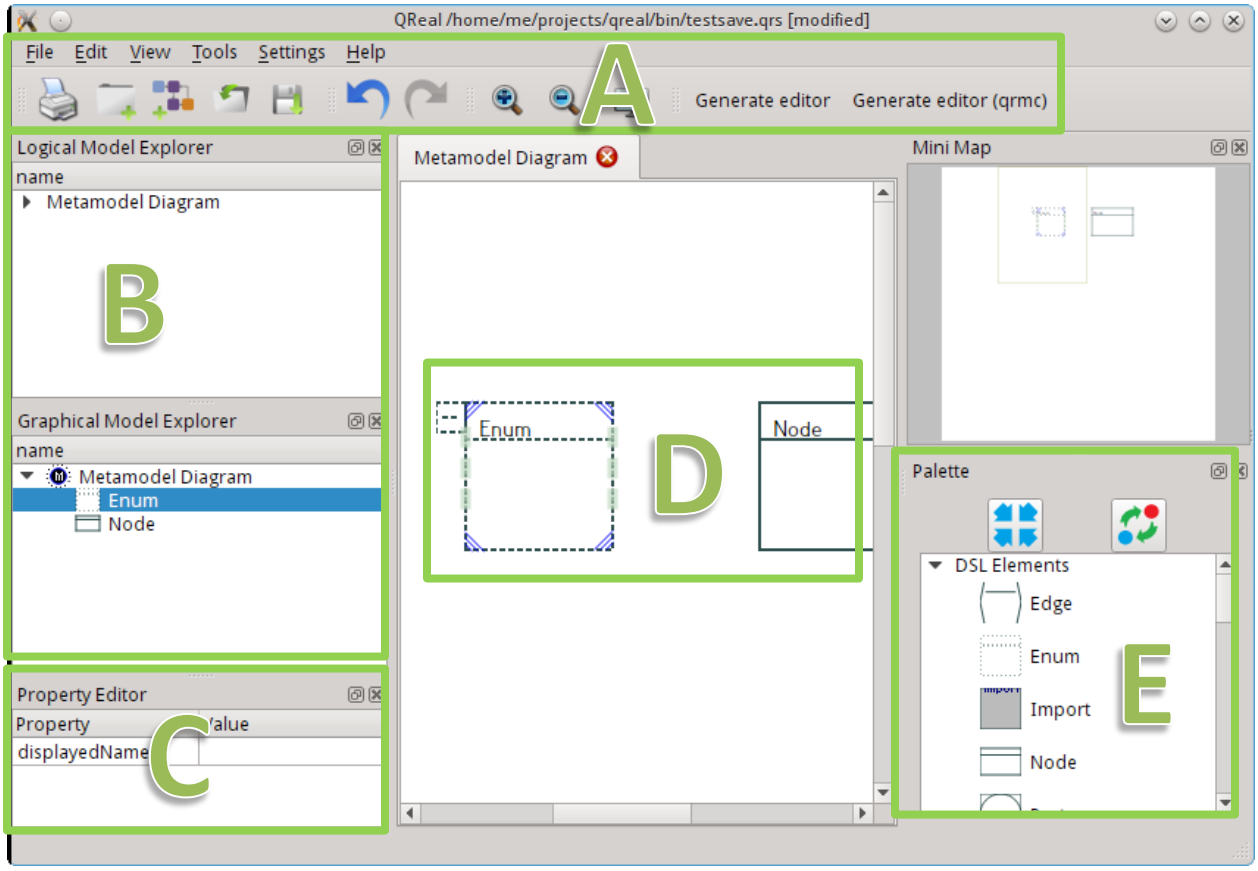

Рис. 6. Скриншот графического интерфейса приложения QReal с наложенными зонами частого обращения пользователем

На представленном выше скриншоте выделены зоны, с которыми пользователь производит большинство взаимодействий при работе с приложением QReal. Таковыми зонами являются:

A. Меню и список действий;
B. Проводники по моделям;
C. Редактор свойств;
D. Сцена и элементы на ней;
E. Палитра.

Основными сценариями использования, затрагивающие данные зоны являются:

1. Создание новой диаграммы или проекта;
2. Взаимодействие с настройками проекта или системы;
3. Поиск элемента в моделях;
4. Изменение свойств элементов;
5. Взаимодействие с элементом на сцене;
6. Создание новых элементов.

Архитектура компонент в модуле qrgui

Вся реализация графического интерфейса в проекте QReal содержится в модуле qrgui. Данный модуль состоит примерно из 10-15 компонент, которые тесно взаимосвязаны друг другом.

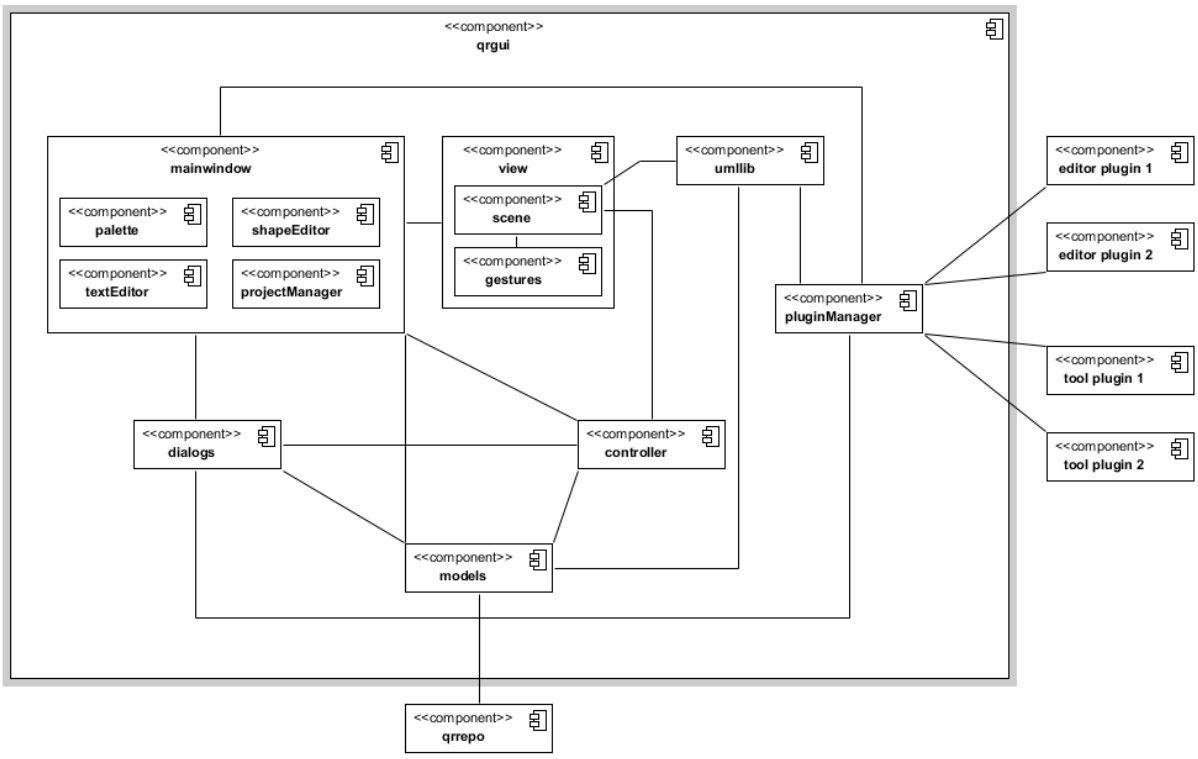

Рис. 7. Архитектура модуля qrgui

Модуль qrgui интегрирует в себе компоненты, отвечающие за взаимодействие с моделью (models и repoApi), отображение модели (view), взаимодействие с плагинами (pluginManager), использующимися при разработке визуальных конструкций, редактор свойств объектов (propertyEditor) и т.д. Несложно заметить, что в такой несколько запутанной архитектуре, содержащей в себе множество интеграционных связей, довольно велика вероятность события некорректного взаимодействия некоторых компонент. В связи с этим было принято решение организовать системное тестирование графического интерфейса проекта QReal с целью автоматизировать выявление некорректной интеграции компонент.

Решение

Для дальнейшней реализации была выбрана библиотека QTestLib, которая лучше всего подходит для данной задачи. Она обладает такими преимуществами как:

- полноценное взаимодействие с внутренними объектами приложения
- поддержка кросс-платформенного тестирования

- отсутствие зависимостей от третьестороннего программного обеспечения

Однако, как было сказано, у данной библиотеки есть серьезный недостаток в виде низкоуровненности в работе с системными событиями. Если осмотреть данный момент подробнее, то библиотека представляет собой обертку над обработчиками событий мыши и клавиатуры у объектов фреймворка Qt. Это означает, что для симуляции этого события необходимо иметь ссылку на объект, у которого это событие вызывается. Этот недостаток порождает некоторые неудобства при реализации тестов, которые заключаются в необходимости поиска требуемого объекта. Также можно заметить, что иерархия реальных объектов в системе сложнее, чем в графическом интерфейсе во много раз. К примеру, если кнопка меню для конечного пользователя представляется как сущность прямо на поверхности графического интерфейса, то в системе эту иерархию можно представить как: Главное окно -> Панель меню -> Объект меню -> Кнопка этого объекта. Очевидно, что для таких простых действий необходимо создать общий код, который будет использоваться при написании тестов и облегчит их разработку. В связи с этим было принято решение разработать вспомогательную библиотеку, в которой будут реализованы необходимые на данный момент функции для взаимодействия с графическим интерфейсом.

Вспомогательная библиотека

Как было описано ранее, для последующей реализации тестов возникла необходимость в библиотеке, которая будет предоставлять более удобный пользовательский интерфейс для взаимодействия с элементами графического интерфейса. Технология Qt без особых проблем позволяет реализовать свою библиотеку в дополнение к имеющейся. Таким образом разработчики технологии Qt поставляют базовый функционал, с помощью которого можно будет реализовать свою часть фреймворка для тестирования графического интерфейса. В качестве необходимых для реализации в библиотеке функций были выбраны:

- Взаимодействие с панелью действий и активация действий на ней
- Взаимодействие с элементами на сцене
- Взаимодействие с меню и подменю
- Взаимодействие с элементами на палитре

С помощью этих действий можно проверить некоторые базовые функции проекта QReal такие как:

- Создание нового проекта или диаграммы
- Изменение настроек пользовательского интерфейса QReal
- Изменение свойств элемента на сцене
- Проверка работоспособности загрузки проекта
- Добавление элемента из палитры на сцену

В процессе разработки данной библиотеки нашлось несколько ошибок в технологии Qt. Для большинства из них нашлись способы обойти их, с помощью стандартных средств.

Одним из интересных моментов в реализации стал факт присутствия синхронных событий в графическом интерфейсе. К примеру, такими являются появление всплывающих окон или контекстного меню при клике правой кнопкой мыши. Архитектура приложения с графическим интерфейсом требует, чтобы объекты графического интерфейса создавались только в потоке основного выполнения программы. Помимо этого, накладывается еще требование того, что объекты графического интерфейса не могут быть доступны из других потоков. Вызов синхронного метода ставит поток на ожидание, пока пользователь не вмешается в поток выполнения своим действием, к примеру, кликом мышки по кнопке закрытия всплывающего окна. В силу наложенных ограничений не представляется возможным воспроизвести событие, прерывающее выполнение синхронного метода из другого потока. Но, во фреймворке Qt существуют особые объекты слоты, которые поддерживают асинхронные отложенные вызовы. Таким образом, создав вспомогательный объект со слотом в контексте основного выполнения программы, можно отложенно выполнить этот слот, в реализации которого будет действие, прерывающее синхронный метод.

Помимо этой особенности взаимодействия с объектами графического интерфейса была обнаружена ошибка, которая повлияла на реализацию функциональности, связанной с взаимодействием с элементами на палитре и последующим перемещением их на сцену. При старте разработки данной библиотеки была придумана идея воспроизвести событие перетаскивания элемента как захват элемента нажатием кнопки мыши, перемещение и бросок элемента на сцену отпусканием мыши. При перемещении элемента использовался

подход, описанный выше, так как событие перемещения также является синхронным. Однако проблема возникла на стадии вызова события отпускания мыши. Внутри технологии Qt архитектурно реализовано так, что событие отпускания мыши и событие конца перемещения объекта являются различными. А в библиотеке Qt реализована только поддержка событий отпускания мыши. Следствием из всего этого является невозможность прервать событие перемещения и бросить элемент на сцену. Можно заметить, что поддержка события перемещения была недавно реализована для Qt Qml. Вполне возможно, что данная функциональность может появиться в ближайших версиях технологии Qt. Но учитывая то, что в процессе тестирования можно взаимодействовать с объектами внутри проекта QReal, то существует возможность реализовать самостоятельно эмуляцию события броска элемента с помощью применения фильтров событий. Но при этом остается вопрос, как же принудительно завершить событие перемещения, начатое ранее.

Также была обнаружена ошибка во фреймворке Qt, которая заключается в том, что не существует возможности получить объект для взамодействия в элементах графического интерфейса таких как:

- QListWidget и его внутренний элемент QListWidgetItem;
- QTreeWidget и его внутренний элемент QTreeWidgetItem.

Однако при реализации был найден некоторый подход, позволяющий получить не объект, а место в родительском элементе для взаимодействия с внутренним элементом.

Базовое тестирование

Для проверки реализации вспомогательной библиотеки, а также для создание примеров тестов, было предложено создать несколько тестов, демонстрирующих работу с QTestLib.

Был создан набор тестов, который включает в себя:

- Создание нового проекта и валидацию
- Создание новой диаграммы и валидацию
- Загрузка нового проекта
- Переименовывание элемента на сцене
- Отрисовка жеста
- Изменение системных настроек

Ниже можно рассмотреть пример, насколько сильно упрощает написание тестов реализованная библиотека. Рассмотрим реализацию теста по изменению свойства объекта в редакторе свойств без использования описываемой библиотеки.

```
void DemoGuiTestSet::testPropertyEditor() {
    MainWindow w("testsave.qrs");
    QTest::qWaitForWindowExposed(&w);
    QTest::qWait(1000);
    QTabWidget *tab = mainWindow->findChild<qReal::gui::TabWidget *>();
    qReal::EditorView *ew =
               dynamic_cast<qReal::EditorView *>(tab->currentWidget());
    Q_ASSERT(view != nullptr);

    QList<QGraphicsItem *> l = view->items();
    foreach (QGraphicsItem *i, l) {
        qReal::Label *label = dynamic_cast<qReal::Label *>(i);
        if (label != nullptr) {
            if (label->toPlainText() == name) {
                QPoint corner = label->parentItem()->scenePos().toPoint();
                QPoint relCenter =
                       label->parentItem()->boundingRect().center().toPoint();
                QTest::mouseClick(view->viewport(), Qt::LeftButton, 0, corner
+ relCenter);

                QTest::qWait(2000);
            }
        }
    }

    PropertyEditorView *dock =
            mainwindow->findChild<PropertyEditorView *>("propertyEditor");
```

```
    QtTreePropertyBrowser *tb = dock->findChild<QtTreePropertyBrowser *>();
    QTreeWidget *treeWidget = tb->findChild<QTreeWidget *>();
    qDebug() << treeWidget;
    QList<QTreeWidgetItem *> items =
            treeWidget->findItems(propertyName, Qt::MatchExactly);
    qDebug() << items.at(0);
    Q_ASSERT(items.size() == 1);

    QTest::mouseClick(treeWidget->viewport(), Qt::LeftButton
, 0, treeWidget->visualItemRect(items.at(0)).center() + QPoint(10, 0));
    QTest::keyClicks(nullptr, propertyValue, Qt::NoModifier, 200);
    QTest::keyClick((QWidget *) nullptr, Qt::Key_Enter, Qt::NoModifier, 200);
}
```

Пример 1. Реализация теста с использование только QTestLib

Далее приведен пример реализации теста с использованием специфичных для проекта QReal методов, реализованных во вспомогательной библиотеке.

```
void DemoGuiTestSet::testPropertyEditor() {
        MainWindow w("testsave.qrs");
        QTest::qWaitForWindowExposed(&w);
        QTest::qWait(1000);
        qrtestlib::clickObjectOnScene(&w, "Enum");
        qrtestlib::setPropertyInPropertyEditor(&w, "displayedName", "new
name");
      }
```

Пример 2. Реализация теста с использованием библиотеки qrtestlib.

На приведенном примере можно заметить результат работы над созданием вспомогательной библиотеки. Большинство функций, реализованных в этой библиотеке являются часто используемыми в описанных ранее сценариях, что позволяет сократить количество кода при написании тестов более, чем на 50%, при том уменьшив вероятность создания ошибки в тесте.

Результаты

Как результат данной курсовой работы был проведен обзор существующих инструментов тестирования графического интерфейса, включая платные и свободно распространяющиеся. Помимо этого была реализована система тестирования графического интерфейса для проекта QReal, которая включала в себя:

- Исследование поведения приложения с графическим интерфейсом на ОС Windows и GNU\Linux
- Вспомогательную библиотеку для взаимодействия с графическим интерфейсом, которая намного упрощает процесс реализации тестов
- Набор тестов, демонстрирующий работу данной библиотеки и библиотеки QTestLib

Дальнейшие направление

Проведенная работа открывает возможность наладить полное тестирование графического интерфейса в проекте QReal. Помимо этого, появляются возможности добавлять новые функции в библиотеку QTestLib, которые основные разработчики технологии Qt не успевают добавить из-за активной разработки над другими компонентами. Тем самым это позволяет многим другим пользователям этой технологии пользоваться новым функционалом. Также здесь становится возможным изучать работу приложений, созданных с помощью технологии Qt на различных платформах по части графического интерфейса.

Список литературы

[1] http://www.froglogic.com/squish/gui-testing/index.php Froglogic Squish

[2] http://qt-project.org/doc/qt-5/QTest.html QTestLib

[3] http://www.sikuli.org/ Sikuli Script

[4] http://www.gnu.org/software/xnee/ Xnee

[5] http://ldtp.freedesktop.org/wiki/ LDTP

[6] https://qt.gitorious.org/qt Исходный код технологии Qt