Тестирование среды программирования роботов

Д.А. Мордвинов
Санкт-Петербургский государственный университет
Кафедра системного программирования
Email: mordvinov.dmitry@gmail.com

Ю.В. Литвинов
Санкт-Петербургский государственный университет
Кафедра системного программирования
Email: y.litvinov@spbu.ru

***Аннотация*—В статье представлен опыт тестирования достаточно сложной визуальной среды программирования роботов, написанной на C++ с использованием библиотеки Qt. Рассматриваются три аспекта тестирования среды — модульное тестирование, тестирование пользовательского интерфейса и функциональное тестирование посредством интерпретации программ, написанных в среде, на имитационной модели с заданными ограничениями на состояние модели. Приводится описание языка задания ограничений. Описываются другие задачи, помимо тестирования, где представленные в статье решения оказались полезны — код для тестирования пользовательского интерфейса переиспользован для реализации режима обучения, код для функционального тестирования переиспользован для проверки заданий учащихся онлайн-курса. Статья может быть полезна специалистом, занимающимся разработкой приложений на C++ с Qt и людям, занимающимся тестированием с использованием имитационных моделей физических объектов, которыми управляет программа.**

I. Введение

С 2012 года ведётся разработка робототехнического конструктора ТРИК[1] и графической среды программирования для него TRIK Studio [1]. Среда программирования появилась на базе наших исследований в области визуального моделирования ([2], [3], [4]) и их приложения к робототехнике ([5], [6], [7], [8]), так что на данный момент она представляет собой достаточно большую и богатую функциональными возможностями систему, имеющую несколько тысяч пользователей по всей стране и в мире. Среда предназначается прежде всего для обучения школьников и студентов программированию и робототехнике. Система имеет открытый исходный код, кроссплатформенная[2] (на момент написания статьи работает под управлением операционных систем Windows, Linux и Mac OS X), реализована на языке C++ с использованием библиотеки Qt 5[3], содержит как развитый пользовательский интерфейс, так и сложную внутреннюю логику работы, включающую в себя управление реальными роботами и исполнение программ на имитационной модели. Это создаёт некоторые сложности в организации автоматического тестирования системы. В этой статье будет рассказано, как эти сложности преодолевались и как вообще было организовано автоматическое тестирование, что может быть полезно читателю как с чисто технической точки зрения (тестирование приложений с пользовательским интерфейсом, реализованных с использованием библиотеки Qt), так и с научной (тестирование среды программирования роботов посредством проверки корректности исполнения программ на имитационной модели).

Среда TRIK Studio представляет собой средство визуального программирования роботов, то есть среду, где программы представляются в виде набора графических символов (или блоков), реализующих элементарные команды роботу (включить моторы, дождаться срабатывания датчика и т.д.). Блоки соединены стрелками, показывающими передачу управления. Такой подход довольно типичен для данной предметной области (ближайшие аналоги TRIK Studio — среды Robolab [9], NXT-G[4], MRDS[5]). Одна и та же нарисованная программа может быть исполнена на компьютере с посылкой команд реальному роботу, сгенерирована в программу на JavaScript, F# или C и передана роботу для исполнения в автономном режиме, или исполнена на имитационной модели, где симулируется робот и его окружение.

Автоматическое тестирование системы TRIK Studio состоит из трёх частей — модульного тестирования, тестирования пользовательского интерфейса и функционального тестирования посредством имитационной модели. Первые две части типичны для практически любых приложений с пользовательским интерфейсом, третья же специфична для предметной области робототехники — среда симулирует поведение реального робота и исполняет тестовые программы на нём, контролируя состояние среды симуляции посредством

[1]Домашняя страница конструктора ТРИК, URL: http://www.trikset.com/ (дата обращения: 17.08.2015)

[2]Репозиторий проекта на GitHub, URL: http://github.com/qreal/qreal/ (дата обращения: 17.08.2015)

[3]Домашняя страница библиотеки Qt, URL: http://www.qt.io/ (дата обращения: 17.08.2015)

[4]Среда программирования NXT-G, URL: http://www.lego.com/ru-ru/mindstorms/learn-to-program (дата обращения: 07.02.2015)

[5]Microsoft Robotics Developer Studio, домашняя страница, URL: https://msdn.microsoft.com/en-us/library/bb648760.aspx (дата обращения: 07.02.2015)

задания ограничений на её состояние, зависящих от времени и происходящих событий.

В статье будет рассказано про каждый из упомянутых выше этапов автоматического тестирования. Каждый этап представляет собой применение знаний из весьма обширной области, поэтому в данной работе не ставится цель подробно рассказать про модульное тестирование или тестирование пользовательского интерфейса вообще (это было бы слишком амбициозной задачей, к тому же уже решённой в многочисленных статьях и книгах, например, [10], [11]). Цель данной работы — представить опыт применения некоторых методик тестирования в реальном промышленном активно развивающемся проекте, представить и пояснить принятые нами решения.

II. Модульное тестирование

Для C++ существует множество библиотек модульного тестирования и единого стандарта не существует. Для выбора конкретной библиотеки мы руководствовались такими критериями, как скорость и простота написания теста, умение библиотеки работать с исключениями, наличие развитого механизма проверок, наличие поддержки создания mock-объектов.

Библиотека Qt, на основе которой разрабатывается TRIK Studio, имеет в своём составе библиотеку модульного тестирования Qt Test[6], но, несмотря на хорошую интеграцию с Qt, нам она не подошла по причине отсутствия встроенной поддержки работы с исключениями и mock-объектами. Также были рассмотрены библиотеки CPPUnit, Boost.Test, CppUnitLite, Unit++, CxxTest, Google C++ Testing Framework, подробное сравнение этих библиотек можно найти в работе [12]. В результате была выбрана библиотека Google C++ Testing Framework[7] и её расширение для написания mock-объектов Google C++ Mocking Framework[8].Эти инструменты удовлетворяют нашим критериям, кроме того, оказались просты в использовании, в том числе и в кроссплатформенном окружении.

При написании тестов мы пытались поддерживать соответствие «один исполняемый файл с тестами — один бинарный файл системы (разделяемая библиотека или исполнимый файл)», структура тестов примерно повторяет структуру основного проекта. При этом возникла проблема, что операционная система Windows не позволяет передавать в командной строке слишком длинные пути (более 260 символов), поэтому пришлось несколько упростить в тестах иерархию каталогов, иначе возникали трудности со сборкой. Модульные тесты не раз доказали свою эффективность, помогая искать ошибки на ранних этапах разработки и упрощая рефакторинг.

[6]Документация по Qt Test, URL: http://doc.qt.io/qt-5/qttest-index.html (дата обращения 17.08.2015)

[7]Домашняя страница Google C++ Testing Framework, URL: https://code.google.com/p/googletest/ (дата обращения: 07.02.2015)

[8]Домашняя страница Google C++ Mocking Framework, URL: https://code.google.com/p/googlemock/ (дата обращения: 07.02.2015)

III. Тестирование пользовательского интерфейса

Одно лишь модульное тестирование, по крайней мере в нашем случае, не может покрыть все аспекты системы. Пользовательский интерфейс также подлежит проверке корректности. Данный раздел содержит описание результатов поиска подходящего решения, подхода к тестированию пользовательского интерфейса в TRIK Studio, а также обсуждение некоторых прикладных сторон технологии, полученной в результате данного исследования.

A. Существующие подходы

Тестирование графических интерфейсов — область, возраст которой составляет четверть века. За это время было опубликовано множество статей, проведена их подробная классификация (см. [10]), разработано большое количество подходов и инструментов, реализующих эти подходы. Данная работа не затрагивает фундаментальных теоретических оснований данной области, а скорее является практическим результатом на пересечении технологий, о которых будет рассказано в этом и последующих разделах.

TRIK Studio обладает сложным и постоянно изменяющимся интерфейсом, созданным на основе инструментария Qt Widgets[9]. Кроссплатформенность системы предъявляет важное требование к технологии тестирования интерфейса — одни и те же тесты должны иметь работать на различных операционных системах, оконных менеджерах, установленных разрешениях экрана и т.д. Открытость кода и свободное распространение среды делает непригодными коммерческие системы тестирования (например, Squish[10], который пригоден по всем другим параметрам). Приведем обзор технологий, с которыми мы столкнулись в процессе поиска конечного решения задачи тестирования интерфейсов.

1) Библиотека Qt Test: Инструментарий Qt содержит набор средств автоматизации взаимодействия с пользовательским интерфейсом приложения, предоставляя библиотеку Qt Test в базовом некоммерческом модуле. Данный инструмент позволяет автоматизировать взаимодействие с графическим интерфейсом посредством задержек и эмулирования низкоуровневых событий мыши и клавиатуры. Высокоуровневые операции поиска и взаимодействия с определенными элементами управления, инструменты их поиска и определения отсутствуют в библиотеке Qt Test, тем не менее

[9]Документация по Qt Widgets, URL: http://doc.qt.io/qt-5/qtwidgets-index.html (дата обращения 17.08.2015)

[10]Домашняя страница Squish, URL: http://www.froglogic.com/squish/gui-testing/ (дата обращения 17.08.2015)

содержатся в том или ином виде в различных местах инструментария для создания самих интерфейсов.

В результате тест с использованием данной библиотеки представляет собой модуль на C++, требующий самостоятельной реализации частых операций (например, создание операции drag-and-drop из одного места в другое с последующим использованием результатов этой операции), а также перекомпиляции тестов при изменении пользовательского интерфейса, что затрудняет их поддержку. Сами по себе эти факторы не кажутся слишком проблематичными, однако на практике показали себя довольно затрудняющими процесс создания и поддержки тестов, код получался громоздким.

Тем не менее, формально данная технология удовлетворяет всем нашим требованиям, тесты не зависят от платформы, изменение интерфейса требует сопоставимых изменений тестов.

2) Windows Automation API: Для тестирования пользовательского интерфейса Windows-приложений часто используются возможности операционной системы Windows, а именно Windows Automation API[11]. Это набор функций, доступных как из .NET, так и для программ на C++, с помощью которых можно получить доступ к элементам пользовательского интерфейса, таким как кнопки, поля для ввода, списки и т.д., и симулировать события ввода. В составе Windows SDK[12] поставляются программы Inspect и AccScope, использующие Automation API для того, чтобы показывать структуру и свойства элементов управления приложений. Существуют и полноценные библиотеки GUI-тестирования, такие как White[13] или UI Automation Verify[14], скрывающие от программиста сложность использования Automation API и позволяющие удобно писать тестовые сценарии на .NET, работая в терминах элементов пользовательского интерфейса.

Основной недостаток подобного подхода в нашем случае — отсутствие кроссплатформенности и некоторая ориентированность на .NET (сам Automation API может использоваться и из кода на C++, но большинство библиотек к нему написаны на C#), что хоть и не делает невозможным, но усложняет использование такого подхода.

3) Sikuli: Sikuli [13] — это свободный инструмент, разрабатываемый в MIT для скриптования пользовательских действий на основе изображений отдельных областей экрана. Тест представляет собой скрипт на языке Python, где в качестве примитива может быть использовано изображение какого-либо фрагмента экрана. Инструмент предоставляет удобную среду программирования таких скриптов Sikuli IDE, которая позволяет быстро создавать и менять изображения прямо в коде теста.

Удобство написания простых тестов намного выше, чем при использовании библиотеки Qt Test, а также исчезает ряд проблем взаимодействия с операционной системой, однако данный инструмент не подошел нам из-за требований кроссплатформенности тестов. Очевидно, что на различных платформах и даже в разных версиях одной и той же платформы интерфейс приложения может выглядеть совершенно по-разному. При незначительной модификации интерфейса зачастую приходилось полностью изменять содержимое тестов. В итоге, несмотря на удобство и легкость написания простых тестов, данное обстоятельство делает Sikuli непригодным в нашем случае.

B. Скриптовый программный интерфейс

В результате так и не было найдено готового решения, способного удовлетворить всем поставленным ограничениям. Наиболее близкой к желаемому оказалась библиотека Qt Test, которая представляет собой скорее основу для технологии тестирования интерфейсов, чем саму технологию. Опыт использования описанных решений позволил нам сформировать представление о платформе, обладающей всеми достоинствами Qt Test в виде кроссплатформенности и возможности создания самой изощренной логики тестирования и простотой и лаконичностью скриптового подхода наподобие Sikuli.

В нашем представлении подходу с использованием Qt Test не хватает «скриптовости» и высокоуровневости в смысле работы в терминах элементов управления, а не событий от манипуляторов. Эти идеи были учтены при реализации системы тестирования интерфейсов TRIK Studio.

В качестве основного языка написания тестов был выбран Qt Script. Это реализация стандарта ECMAScript[15] (наиболее популярной реализацией которого является язык JavaScript), расширенная механизмами интеграции с Qt-приложениями. Скриптовое окружение расширено инструментами работы с пользовательским интерфейсом системы в терминах самого интерфейса. Работа этого окружения осуществляется посредством библиотек Qt Test и Qt Widgets, таким образом, сохраняются независимость тестов от платформы и легкость доступа ко всем элементам управления, при этом решаются проблемы громоздкости кода тестов и низкоуровневости терминов, в которых они работают.

[11] Документация Windows Automation API на MSDN, URL: https://msdn.microsoft.com/en-us/library/windows/desktop/ff486375(v=vs.85).aspx (дата обращения 17.08.2015)

[12] Домашняя страница Windows 10 SDK, URL: https://dev.windows.com/en-us/downloads/windows-10-sdk (дата обращения 17.08.2015)

[13] Страница проекта White на GitHub, URL: https://github.com/TestStack/White (дата обращения 17.08.2015)

[14] Домашняя страница UI Automation Verify, URL: https://uiautomationverify.codeplex.com/ (дата обращения 17.08.2015)

[15] Домашняя страница ECMAScript, URL: http://www.ecmascript.org/ (дата обращения 17.08.2015)

Окружение предоставляет доступ ко всем частям интерфейса и методам работы с ними и содержит вспомогательную функциональность, специфичную только для TRIK Studio. К примеру, существует возможность скриптования создания визуальных диаграмм, рисования модели мира, в котором будет производиться имитационное моделирование робота, работы с жестами мышью, изменения свойств примитивов, из которых строится программа, работы с настройками приложения, управления состоянием вспомогательных панелей и т.д. Реализованы также методы произвольного доступа к элементам управления интерфейса по структурному описанию (идентификаторам и/или положению в дереве элементов управления) и координатам на экране. Управление эмуляторами устройств ввода (например, навигация и действия виртуальной мышью или указание клавиатурного фокуса и последовательностей нажатий на кнопки клавиатуры) тесно интегрировано с различными видами элементов управления. Например, присутствуют отдельные функции прокрутки и работы с выпадающими списками. При этом остаются доступными методы синтезирования низкоуровневых событий манипуляторов.

Кроме всего описанного, скриптовый интерфейс системы содержит методы показа информационных сообщений, подсвечивания отдельных элементов управления и рисования стрелок-указателей на них, анимации виртуального курсора мыши и тому подобных «показательных» функций, надобность которых будет обсуждаться в разделе III-D. Тем не менее, остаются открытыми некоторые проблемы, каждая из которых носит чисто технический характер. Одна из них — управление системными диалоговыми окнами. Дело в том, что на каждой платформе системные диалоговые окна (такие как диалоги выбора файлов, печати, выбора цвета и т.д.) свои и не поддаются контролю системы синтезирования событий, используемой в библиотеке Qt Test. Сейчас эта проблема обходится работой с этими диалогами как с «черными ящиками» — тесты задают для себя результаты выбора сущностей, редактируемых этими диалогами, не прибегая к работе с ними, «обходя», таким образом, их на уровень выше. Другая проблема — отсутствие в Qt Test возможности эмулирования ввода с клавиатуры на разных языках, что делает невозможным полноценное тестирование локализации среды. Все эти проблемы являются, скорее, проблемами библиотеки Qt Test, в то время как с нашей стороны единственной на данный момент проблемой является стоимость поддержки кодовой базы, что, безусловно, являясь значимым, не будет обсуждаться в рамках этой статьи.

Полученная технология оказалась практически полностью удовлетворяющей нашим потребностям в удобстве написания тестов и их переносимости. Важно, что разработкой автоматизированного тестирования интерфейса смогли заниматься люди, практически не

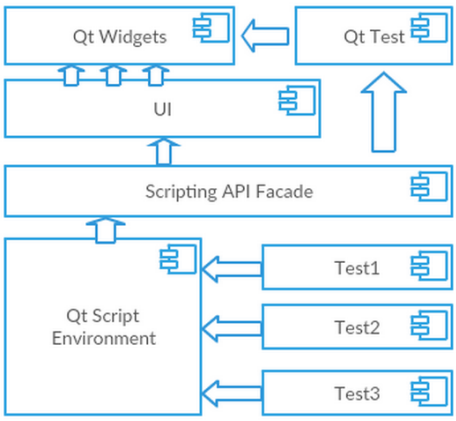

Рис. 1: Архитектура системы тестирования интерфейсов TRIK Studio.

имеющие отношения к разработке и поддержке самой системы и не имеющие опыта программирования на C++.

C. Инфраструктура

В данном разделе будет кратко описана структура технологии тестирования, описанной в разделе III-B. Общая архитектура всего комплекса представлена на рисунке 1. Основная идея состоит в создании новой «прослойки» системы, стоящей над всей технологией, интерфейса программирования поведения пользовательского интерфейса. Этот уровень не модифицирует кодовую базу системы, тем не менее агрегируя в себе все ее отдельные компоненты. Здесь реализуются все операции, специфичные только для данной платформы в удобном для использования в скриптах виде, а также происходит «оборачивание» операций библиотеки Qt Test. Интерпретатор Qt Script, занимающийся исполнением кода самих тестов, импортирует символы скриптового API. Очевидно, что необходимости в Qt Script в такой архитектуре нет, можно создавать тесты и на компилируемых языках, компонуясь с интерфейсом программирования, в этом случае такой новый уровень можно рассматривать просто как инструмент повторного использования кода в тестах.

D. Использование технологии для обучения пользователей

Полученная технология обладает одним интересным приложением. На базе одной и той же инфраструктуры могут быть получены как тесты пользовательского интерфейса, так и автоматизированные демонстрации процесса работы со средой, встроенные прямо в нее. Это способ обучения пользователя возможностям среды, используемый, в основном, в компьютерных играх — про возможности продукта рассказывает сам продукт (в противоположность, например, таким подходам как текстовая справка или видео уроки).

Возможно, такой подход будет полезен разработчикам приложений с развитым, нестандартным графическим интерфейсом. Написанием скриптов тестирования и скриптов демонстрации могут заниматься одни и те же люди, даже не участвующие в написании кода самого приложения.

E. Обсуждение

Дальнейшее развитие этой технологии предполагает генерацию скриптов симуляции поведения пользователей. В самом простом случае тесты графического интерфейса и встроенные демонстрации возможностей среды можно создавать, записывая действия пользователя, работающего со средой. Если же требуется более сложная логика работы сценария, её можно описать с помощью специального визуального языка, по которому будет автоматически сгенерирован код скрипта тестирования. На данный момент эти технологии реализованы в системе TRIK Studio и проходят апробацию, но в силу того, что окончательный результат ещё не получен, они не будут подробно обсуждаться в рамках данной работы.

Разумеется, создание специального визуального языка для нужд тестирования пользовательского интерфейса «с нуля» было бы неоправданно трудозатратно. Однако такой подход полностью соответствует парадигме предметно-ориентированного моделирования (Domain-Specific Modeling [14]), а визуальный язык и инструментальные средства к нему можно быстро реализовать с помощью так называемых DSM-платформ ([15], [16]). Проект, на котором основывается TRIK Studio, QReal — это пример такой DSM-платформы (на самом деле, сам язык программирования роботов и редактор к нему были созданы с помощью QReal в течение нескольких часов). Создать ещё один визуальный язык для тестирования пользовательского интерфейса TRIK Studio не представляло особой сложности.

Идея записи пользовательских действий с последующей генерацией теста уже находила свою реализацию, например, в проекте TPTP[16] тестирования приложений, созданных в среде разработке Eclipse[17]. Более того, существуют работы, в которых эта методика расширяется добавлением элементов искусственного интеллекта, например [17], [18]. Интеграция подобных подходов в наше решение является интересной темой дальнейших исследований.

IV. Функциональное тестирование

A. Краткое описание возможностей среды

Далее в этом разделе пойдёт речь о тестировании функциональности, специфичной для TRIK Studio (и аналогичных систем, предназначенных для управления некоторым объектом), поэтому следует описать подробнее функциональность, которую требуется тестировать.

TRIK Studio представляет собой среду программирования роботов на визуальном языке. Система поддерживает программирование робототехнических конструкторов Lego Mindstorms NXT 2.0, Lego Mindstorms EV3 и ТРИК, в разработке находится поддержка платформы Arduino. Для каждой из поддерживаемых платформ имеется возможность отладки визуальной программы на компьютере с посылкой команд роботу, генерации кода на каком-либо текстовом языке программирования (например, C, JavaScript, F#, ШАЯ) с последующей загрузкой кода на робота по различным интерфейсам (Bluetooth, USB, Wi-Fi) для автономного исполнения. Существует также возможность отладки программы на двумерном виртуальном исполнителе. Общий вид интерфейса среды представлен на рисунке 2

Программа представляется в виде набора блоков, каждый из которых представляет собой элементарную команду роботу или элементарный шаг вычисления. Команды роботу могут быть как простыми (например, включить мотор на таком-то порту с такой-то мощностью), так и довольно сложными (например, получить данные датчика линии, использующего видеокамеру и алгоритмы распознавания изображений), сложные алгоритмы, как правило, реализуются на самом роботе и среда лишь вызывает их и обрабатывает результаты. Во всех блоках можно использовать математические выражения на текстовом языке, представляющем собой подмножество языка Lua 5.3[18], синтаксический анализатор и интерпретатор которого были специально реализованы в TRIK Studio (готовые решения нас не устраивали тем, что невозможно было использовать их абстрактное синтаксическое дерево для генерации кода на других языках программирования, кроме того, над Lua пришлось реализовать свою систему типов, потому что некоторые целевые языки генерации, такие как C, статически типизированы). В выражениях можно использовать текущие значения датчиков робота, доступные как специальные переменные, они же отображаются в виде таблицы или графиков при запущенной в режиме интерпретации программе. В режиме генерации и автономного исполнения обращения к таким переменным транслируются в команды чтения значений датчиков и их показания на компьютере не отображаются (чтобы дать возможность программе исполняться автономно).

Двумерная модель позволяет исполнять большую часть программ, которые могут быть исполнены на реальном роботе. Пользователь может нарисовать в области редактирования двумерной модели стены, цветные

[16]Страница TPTP GUI Recorder, URL: http://www.eclipse.org/tptp/test/documents/userguides/Intro-Auto-GUI.html (дата обращения 17.08.2015)

[17]Домашняя страница проекта Eclipse, URL: https://eclipse.org/ (дата обращения 17.08.2015)

[18]Домашняя страница языка Lua, URL: http://www.lua.org/ (дата обращения 17.08.2015)

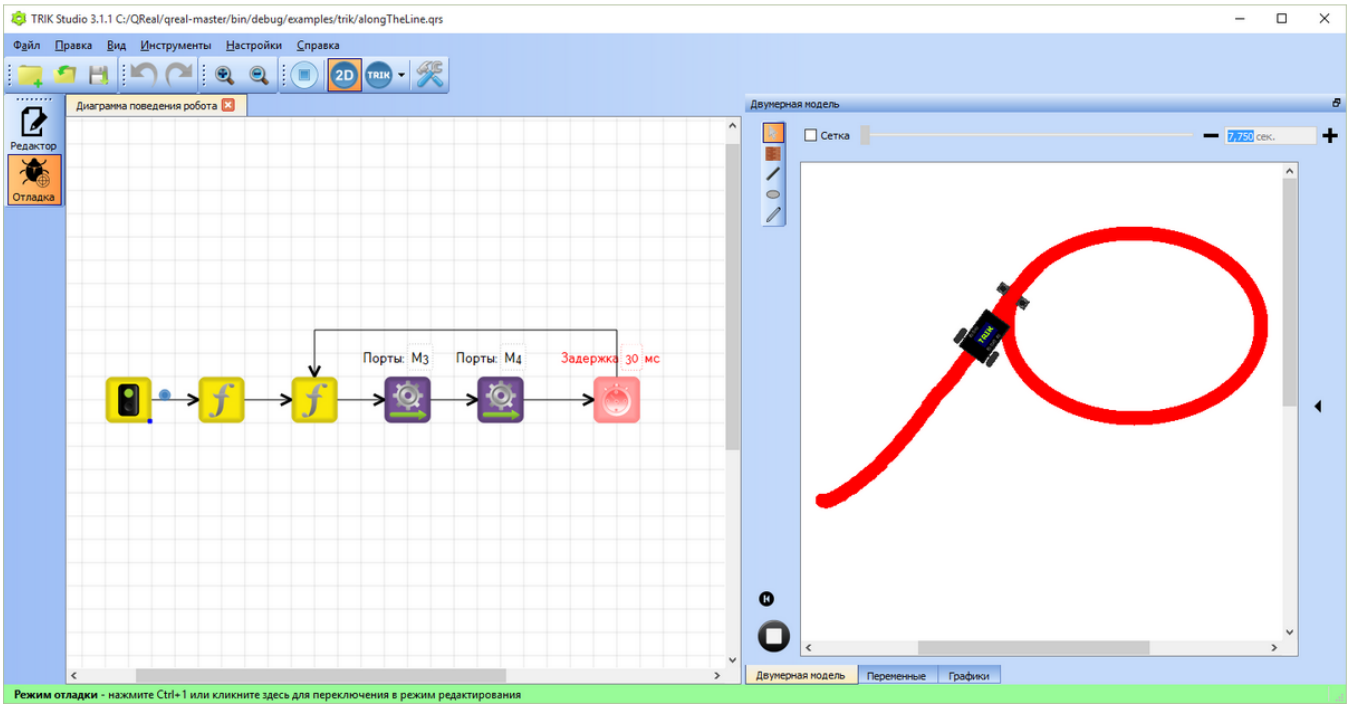

Рис. 2: Интерфейс TRIK Studio.

линии или области на полу, разместить самого робота и его датчики, задать им направление. Датчики будут взаимодействовать со стенами и цветными линиями (для ТРИК поддержаны инфракрасные и ультразвуковые датчики расстояния, датчик освещённости и детектор линии по видеокамере, для Lego NXT или Lego EV3 — все датчики, входящие в стандартный набор: касания, света, цвета и ультразвуковой датчик расстояния). Часть функциональности, доступная только на реальном роботе, имитационной моделью не поддерживается, например, на данный момент на имитационной модели может быть только один робот, поэтому в поддержке функциональности общения между роботами нет смысла. Тем не менее, имитационная модель позволяет решать большинство задач из курсов по робототехнике для Lego NXT (например, [19]) и ТРИК[19], и существенно упрощает отладку, давая возможность отладить алгоритм без реального робота, а потом лишь подобрать коэффициенты для исполнения той же программы в реальном мире.

Отдельно стоит отметить наличие достаточно развитых средств обеспечения удобства пользовательского интерфейса, что связано с образовательной направленностью среды. Например, при рисовании диаграмм можно пользоваться механизмом распознавания жестов мышью ([20]) — с зажатой правой кнопкой рисуется схематичная фигура, которая распознаётся средой и на месте нарисованной фигуры создаётся соответствующий ей блок, что позволяет не искать нужный блок в палитре и сделать использование среды интереснее для детей. Для TRIK Studio проводились исследования по юзабилити и вносились правки в интерфейс по их результатам ([21], [22]).

Генерация кода для автономного исполнения на данный момент реализована для Lego NXT в язык C (с использованием библиотеки ECRobot[20] и ОСРВ nxtOSEK[21]) и для ТРИК в языки JavaScript ([23]) и F# ([24]). Для ТРИК также реализованы библиотеки поддержки времени выполнения на C++ и F#, которые работают непосредственно на роботе и, собственно, реализуют функции, вызываемые программами, сгенерированными в TRIK Studio.

B. Общая схема функционального тестирования

Поскольку имитационная модель покрывает достаточно большой процент функциональности среды и ввиду сложности использования реального робота в автоматических тестах, именно имитационная модель используется нами для функционального тестирования. Обычно в аналогичных ситуациях пишется эмулятор реального устройства, у нас же эмулятор фактически является неотъемлемой частью системы и активно используется конечными пользователями. Предлагаемый подход к функциональному тестированию TRIK Studio таков:

[19] например, набор уроков И.Ю. Широколобова, URL: http://bit.ly/TRIK_presentations (дата обращения 17.08.2015)

[20] Домашняя страница ECRobot, URL: http://lejos-osek.sourceforge.net/html/index.html (дата обращения 17.08.2015)

[21] Домашняя страница nxtOSEK, URL: http://lejos-osek.sourceforge.net/ (дата обращения 17.08.2015)

1) На визуальном языке рисуется программа для исполнения на имитационной модели, решающая ту или иную практическую задачу, а также рисуется модель мира для имитационной модели, с которым эта программа будет работать.
2) Программа снабжается описанием ограничений на состояние системы на специальном XML-языке, описание которого приводится ниже. Наборов ограничений для одной программы может быть несколько, при этом они могут использовать разные модели мира.
3) Файл с сохранённой программой запускается на интерпретацию в двумерной модели, для этого используется специально созданный для нужд тестирования исполняемый файл, представляющий собой по сути TRIK Studio без пользовательского интерфейса. Тестирующая программа перед запуском тестируемой программы подменяет модель мира и набор ограничений в сохранении, что даёт возможность использовать несколько тестов для одной программы. Программа исполняется так, как она исполнялась бы, будучи запущенной пользователем в TRIK Studio, и в случае, если одно из ограничений не выполнено, программа работает некорректно — делается вывод, что поведение интерпретатора или среды имитационного моделирования отличается от эталонного, тест не пройден.
4) Тестирующая программа вызывается системой непрерывной интеграции Travis[22] каждый раз после сборки проекта, для всех файлов сохранений и моделей мира для них, имеющихся в репозитории.
5) Результаты тестов рассылаются по электронной почте разработчикам.

Архитектурно тестирующая система представляет собой отдельную программу, использующую в качестве разделяемых библиотек ядро и имитационную модель системы TRIK Studio. При этом она активно использует рефлексию, реализованную для C++ в библиотеке Qt для доступа к свойствам объектов имитационной модели. Таким образом, при расширении возможностей системы или добавлении нового вида симулированного оборудования новые объекты и свойства будут немедленно доступны из описаний ограничений, без дополнительной модификации кода. Тестовая система подписывается на события имитационной модели и вызывает проверку ограничений на каждом «тике» модельного времени. При этом проверяются только активные ограничения, которых в каждый конкретный момент времени немного, поэтому скорость работы оказывается вполне удовлетворительной.

[22]Система непрерывной интеграции Travis, URL: https://travis-ci.org/ (дата обращения 17.08.2015)

C. Описание языка задания ограничений

Описание ограничений на имитационную модель робота производится на событийном языке, основанном на XML, разработанном как дополнение к механизму имитационного моделирования в среде. Выбор в пользу XML-фронтэнда был произведен из чисто практических соображений удобства интеграции с описанием остальной имитационной модели.

Программа на таком языке представляет собой множество $\{e_1, e_2, ..., e_n\}$ событий, где каждое событие e_i — это тройка (id_i, c_i, T_i).

- id_i — идентификатор события, внутренняя метка, по которой к событию e_i смогут обращаться другие события.
- c_i — условие срабатывания события, формула логики первого порядка, о предикатных и функциональных символах которой будет рассказано чуть ниже.
- T_i — упорядоченный список элементарных триггеров $[t_{i1}, t_{i2}, ..., t_{in}]$. Элементарный триггер — это некоторое действие, выполняемое в момент истинности условия срабатывания события c_i.

Одному событию соответствует один элемент в XML-спецификации программы. Каждое событие в текстовом описании программы может быть представлено либо в виде ограничения, либо в каноническом виде.

Событие в канонической форме — уже описанная тройка (id_i, c_i, T_i).

Событие в форме ограничения — это событие вида $(id_i, !c_i, [fail(message_j)])$, где «!» означает логическое отрицание, а $fail(message_j)$ — триггер прекращения исполнения имитационной модели с выдачей сообщения об ошибке $message_j$. Другими словами, ограничение — это событие, которое срабатывает тогда, когда нарушается какое-то условие, и которое сообщает об этом нарушении. Особый случай такого ограничения — лимит времени на выполнение программы. Такое ограничение должно присутствовать в любой программе проверки ограничений имитационной модели TRIK Studio (что проверяется на уровне самого языка задания ограничений), так как проверка, очевидно, не может осуществляться бесконечное время.

Хоть в выделении ограничений в отдельную группу событий особой надобности нет, на практике удобно описывать утверждения вида «робот x не должен выезжать за пределы зоны z» или «к роботу x должен быть подключен набор датчиков s» именно в терминах ограничений, а не событий.

Коротко опишем множество доступных в языке предикатных и функциональных символов и элементарных триггеров. Предикатные символы можно поделить на несколько групп:

- Предикаты сравнения значений функциональных символов $>$, $<$, $<=$, $>=$, $=$, $!=$.

- Пространственные предикаты. Имеют вид «тело x находится внутри области y».
- Предикаты состояния событий $settedUp(id_i)$ и $dropped(id_i)$, описывающие состояние события (взведено/опущено). Во взведенном состоянии событие может быть выполнено, в опущенном оно остается неактивным, не выполняя триггеры даже при выполнении условия срабатывания события.
- Предикат времени $timer(t)$, истинный в моменты времени t и позже.
- Прочие предикаты, являющиеся надстройками над уже описанными и поддержанные в целях удобства.

Функциональные символы бывают следующих видов:

- Константы различных типов (целочисленные, с плавающей точкой, строковые, логические, цветовые, геометрические и пр.).
- Символ $variableValue(id)$ для получения значения переменной с идентификатором id. Переменные могут быть полезны при сложной логике проверки, например, при подсчете числа проделанных роботом итераций.
- Арифметические и геометрические операции над значениями других символов, например, модуль числа, подсчет расстояния между двумя точками или выпуклая оболочка множества точек.
- Символы сравнения формы двух объектов с использованием расстояния Левенштейна, полезные при проверке схожести фигур, нарисованных роботом.
- Символ $objectState(path)$ — основное средство получения информации о состоянии устройств робота и свойствах предмета из внешнего мира. Аргумент $path$ — путь к желаемому свойству в иерархии объектов в модели мира. Подробный рассказ обо всех свойствах, предоставляемых окружением, довольно громоздкий и выходит за рамки данной статьи.

Наконец, элементарные триггеры делятся на следующие категории:

- $success$, $fail(message)$ — управление состоянием проверки. Первый помечает результат проверки как успешный, второй — как неудавшийся, отображая заданное сообщение об ошибке.
- Триггеры задания значения переменных и изменения значения свойств объектов из имитационной модели мира.
- Триггеры управления состоянием событий. Каждое событие может взвести или опустить другое или себя, с помощью этого механизма можно задавать сложную логику проверки.

Пример простейшей программы на языке задания ограничений имитационной модели мира в TRIK Studio приведён в листинге 1.

D. Использование для проверки заданий в онлайн-курсе

Та же система, что используется для тестирования среды, используется и для проверки задач, решаемых пользователями в рамках онлайн-курса по робототехнике, подготавливаемого в данный момент для платформы Stepic[23]. На сервере размещён набор задач в виде файлов сохранения TRIK Studio, в которых готова (и закрыта для модификации) модель мира для имитационной модели, некий набор ограничений, но не до конца нарисована программа (или отсутствует вовсе). Учащимся предлагается скачать файл с сохранением, открыть его в TRIK Studio, дорисовать программу, проверить, что программа работает у них в имитационной модели и удовлетворяет ограничениям, записанным в файле сохранения, и загрузить программу на сервер.

На сервере используется та же программа тестирования, которая подменяет модель мира и набор ограничений на тестовые и запускает на исполнение пользовательскую программу. Возможность подменить модель мира позволяет проверить программу на работоспособность в различных (возможно, не известных пользователю заранее) условиях, а различные ограничения позволяют управлять случайными факторами, влияющими на работу программы (например, датчиком случайных чисел, или симулировать ввод пользователя). По результатам пользователю отправляется сообщение с результатом проверки.

Результат работы программы может быть отображён в браузере пользователя с помощью JavaScript-программы, рисующей двумерную модель и поведение робота в ходе выполнения задачи. Для этого сервер также готовит и отправляет по запросу файл с трассой робота — набором пройденных им точек и состоянием подключенных к нему устройств. Некоторые задачи могут быть решены прямо в браузере, без открытия сохранения в TRIK Studio, для этого используется среда программирования, частично реализующая возможности TRIK Studio на JavaScript. Среда позволяет нарисовать диаграмму, после чего формирует файл сохранения TRIK Studio, который открывается системой проверки.

E. Результаты

На данный момент с помощью предлагаемой технологии реализовано 20 тестовых задач и тестовая утилита, запускающая все задачи и контролирующая корректность работы системы. Эти же тестовые задачи используются как упражнения для онлайн-курса[24] и покрывают функциональность от самой простой (проехать вперёд) до довольно сложной (движение по лаби-

[23]Платформа онлайн-обучения Stepic, URL: https://stepic.org/ (дата обращения 17.08.2015)

[24]пример такой задачи доступен по ссылке http://bit.ly/online_task_demo (дата обращения: 01.11.2015)

```
<!-- Корневой элемент, означающий начало программы проверки ограничений -->
<constraints>

    <!-- Обязательное в любой программе ограничение на время работы -->
    <timelimit value="2000"/>

    <!-- Ограничение на местоположение робота -->
    <constraint failMessage="Робот покинул допустимую зону!">
        <inside objectId="robot1" regionId="warzone"/>
    </constraint>

    <!-- Условие успешности программы: робот должен сказать "Привет" с помощью
        встроенного механизма синтезирования речи и нарисовать улыбку на дисплее -->
    <event settedUpInitially="true">
        <conditions glue="and">
            <equals>
                <objectState object="robot1.shell.lastPhrase"/>
                <string value="Привет"/>
            </equals>
            <equals>
                <objectState object="robot1.display.smiles"/>
                <bool value="true"/>
            </equals>
        </conditions>
        <trigger>
            <success/>
        </trigger>
    </event>

</constraints>
```

Листинг 1: Пример программы проверки ограничений имитационной модели мира в TRIK Studio.

ринту с использованием ультразвукового датчика расстояния). Все блоки, доступные пользователю, встречаются хотя бы один раз в наборе тестов, кроме того, тесты фактически строятся по материалам годового курса обучения школьников, что даёт уверенность в удовлетворительном покрытии типичных случаев использования имитационной модели. Прогон полного набора существующих на данный момент тестов занимает порядка минуты, что позволяет запускать тесты после каждого коммита в систему контроля версий.

На создание задач, ограничений на XML-языке и эталонных решений, которые, собственно, и используются для тестирования среды, ушло примерно две человеконедели. В результате тестирования было выявлено и исправлено порядка 10 дефектов имитационной модели, в том числе и редко проявляющиеся при «обычном» использовании проблемы с зависимостью показаний, возвращаемых некоторыми датчиками, от случайных факторов. Таким образом, технология уже оказалась нам полезна.

V. Заключение

В статье было рассмотрено применение ряда методик тестирования в проекте TRIK Studio. Было показано, что некоторые технологии полезны не только для тестирования, но и для реализации пользовательской функциональности: средство для тестирования пользовательских интерфейсов переиспользовано для реализации режима обучения пользователей, а средство для тестирования работы интерпретатора программ и имитационной модели — для проверки заданий учащихся в онлайн-курсе.

Подходы, рассматриваемые в этой работе допускают интересные, на наш взгляд, обобщения и углубленное исследования. В частности, интересными представляются идеи использования формальной верификации по методу Model Checking в функциональном тестировании среды программирования. Данное направление исследований интересно еще и потому, что на момент написания статьи находится в разработке событийный визуальный язык для программирования сложных мультиагентных систем, где результаты такого исследования, как представляется, могут быть эффективно

применены.

Список литературы

[1] Литвинов Ю.В., Кириленко Я.А. TRIK Studio: среда обучения программированию с применением роботов // V Всероссийская конференция «Современное технологическое обучение: от компьютера к роботу» (сборник тезисов). ЗАО «Полиграфическое предприятие № 3», 2015. С. 5–7.

[2] QReal DSM platform-An Environment for Creation of Specific Visual IDEs. / Anastasiia Kuzenkova, Anna Deripaska, Timofey Bryksin [и др.] // ENASE. 2013. С. 205–211.

[3] Средства быстрой разработки предметно-ориентированных решений в metaCASE-средстве QReal / А.С. Кузенкова, А.О. Дерипаска, К.С. Таран [и др.] // Научно-технические ведомости СПбГПУ, Информатика, телекоммуникации, управление. 2011. № 4 (128). С. 142–145.

[4] Архитектура среды визуального моделирования QReal / А.Н. Терехов, Т.А. Брыксин, Ю.В. Литвинов [и др.] // Системное программирование. 2009. № 4. С. 171–196.

[5] Терехов А.Н., Брыксин Т.А., Литвинов Ю.В. Среда визуального программирования роботов QReal:Robots // III Всероссийская конференция "Современное технологическое обучение: от компьютера к роботу"(сборник тезисов). 2013. С. 2–5.

[6] Литвинов Ю.В. Визуальные средства программирования роботов и их использование в школах // Современные информационные технологии и ИТ-образование, сборник избранных трудов VII Международной научно - практической конференции. ИНТУИТ.РУ, 2012. С. 858–868.

[7] Тихонова М.В. Среда программирования QReal:Robots // Список-2012: Материалы всероссийской научной конференции по проблемам информатики. 25-27 апр. 2012г, Санкт-Петербург. Изд-во ВВМ, 2012. С. 70–75.

[8] Брыксин Т.А., Литвинов Ю.В. Среда визуального программирования роботов QReal:Robots // Материалы международной конференции "Информационные технологии в образовании и науке". Самарский филиал МГПУ, МГПУ, 2011. С. 332–334.

[9] Portsmore Merredith. ROBOLAB: Intuitive Robotic Programming Software to Support Life Long Learning // APPLE Learning Technology Review. 1999.

[10] Graphical user interface (GUI) testing: Systematic mapping and repository / Ishan Banerjee, Bao Nguyen, Vahid Garousi [и др.] // Information and Software Technology. 2013. Т. 55, № 10. С. 1679–1694.

[11] Котляров В.П., Коликова Т.В. Основы тестирования программного обеспечения. Бином. С. 285.

[12] Бажутин М.С. Фреймворки юнит-тестирования для C++ // курс. работа. 2012.

[13] Yeh Tom, Chang Tsung-Hsiang, Miller Robert C. Sikuli: Using GUI Screenshots for Search and Automation // Proceedings of the 22Nd Annual ACM Symposium on User Interface Software and Technology. UIST '09. New York, NY, USA: ACM, 2009. С. 183–192. URL: http://doi.acm.org/10.1145/1622176.1622213.

[14] Mernik Marjan, Heering Jan, Sloane Anthony M. When and how to develop domain-specific languages // ACM computing surveys (CSUR). 2005. Т. 37, № 4. С. 316–344.

[15] О средствах разработки проблемно-ориентированных визуальных языков / А.А. Павлинов, Д.В. Кознов, А.Ф. Перегудов [и др.] // Системное программирование. 2006. Т. 2, № 1. С. 116–141.

[16] Kelly Steven, Tolvanen Juha-Pekka. Domain-specific modeling: enabling full code generation. Wiley-IEEE Computer Society Press, 2008. С. 446.

[17] Memon Atif M, Pollack Martha E, Soffa Mary Lou. Hierarchical GUI test case generation using automated planning // Software Engineering, IEEE Transactions on. 2001. Т. 27, № 2. С. 144–155.

[18] Sugiura Atsushi, Koseki Yoshiyuki. Simplifying macro definition in programming by demonstration // Proceedings of the 9th annual ACM symposium on User interface software and technology / ACM. 1996. С. 173–182.

[19] Филиппов С.А. Робототехника для детей и их родителей. НАУКА, 2013. С. 319.

[20] Osechkina Maria, Litvinov Yuri, Bryksin Timofey. Multistroke Mouse Gestures Recognition in QReal metaCASE Technology // SYRCoSE 2012: Proceedings of the 6th Spring/Summer Young Researchers' Colloquium on Software Engineering. ISPRAS, 2012. С. 194–200.

[21] Соковикова Н.А. Usability в проекте QReal:Robots // Список-2012: Материалы всероссийской научной конференции по проблемам информатики. 25-27 апр. 2012г., Санкт-Петербург. Изд-во ВВМ, 2012. С. 66–69.

[22] Кузенкова А.С. Анализ пользовательского интерфейса QReal:Robots // курс. работа. 2013. URL: http://se.math.spbu.ru/SE/YearlyProjects/2013/445/445-Kuzenkova-report.pdf.

[23] Terekhov Andrey, Litvinov Yurii, Bryksin Timofey. QReal: Robots an environment for teaching computer science and robotics in schools // Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia / ACM. 2013. С. 10.

[24] Kirsanov Alexander, Kirilenko Iakov, Melentyev Kirill. Robotics reactive programming with F#/Mono // Proceedings of the 10th Central and Eastern European Software Engineering Conference in Russia / ACM. 2014. С. 16.