Техническое введение в TRIK Studio — визуальную среду программирования роботов

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

Здесь будет абстракт

Введение

Современное состояние школьного образования в области информатики можно назвать миром сбывшихся пророчеств Сеймура Пейперта. Еще в 1967 им был предложен исполнитель «черепашка Logo», до сих пор широко используемый для обучения школьников программированию. Чуть менее известно, что наряду с виртуальным исполнителем, Пейперт в своих экспериментах использовал механического робота-черепашку, управляемого с компьютера [1], что делало процесс обучения программированию более увлекательным. Сегодня идеи Пейперта получили широкое распространение: практически повсеместно происходит массовое внедрение реальных исполнителей в школы (к примеру, в России с 2015 года робототехника входит в программу обязательного школьного образования как часть предмета «технология» [2]). При этом чаще всего используются робототехнические конструкторы, такие как Lego Mindstorms NXT и Lego Mindstorms EV3[1], ТРИК[2] и т.д.

Задача программирования робота, собранного из конструктора, сложнее, чем виртуальной «черепашку»: программы должны быть написаны в терминах мощностей моторов и значений с датчиков вместо конкретных перемещений и поворотов. Поэтому при внедрении таких конструкторов в учебный

[1] Домашняя страница LEGO Mindstorms, URL: http://www.lego.com/en-us/mindstorms (дата обращения: 27.04.2016г)

[2] Домашняя страница конструктора ТРИК, URL: http://www.trikset.com/ (дата обращения: 27.04.2016г)

процесс большое внимание уделяется средствам их программирования. При этом довольно популярными являются визуальные языки, поскольку они нагляднее текстовых и проще в изучении. Программирование в таких средах осуществляется в основном перетаскиванием графических примитивов с помощью компьютерной мыши, иногда это дает возможность программировать роботов даже детям, которые еще не умеют читать.

Популярность визуальных языков в образовательной робототехнике подчеркивается количеством сред программирования учебных роботов на таких языках. К самым известным отнесем Robolab [3], NXT-G [4] и EV3-G [5], Scratch [6] и Scratch-подобные среды (S4A [7], mBlock [8], Enchanting [9], ScrathDuino [10], Blockly [11] и App Inventor [12], 12Blocks [13], Open Roberta [14]), Ardublock [15]), среды программирования менее популярных конструкторов и роботов, такие, как Robo PRO [16] для конструктора fischertechnik или Scribbler Program Maker для роботов Scribbler), а также среды программирования для дошкольников и учеников начальных школ Lego WeDo Software [17], Create [18], Wonder [19]. В наши дни научная область образовательной робототехники представляет большой интерес в мире, о чем говорит тот факт, что в 2010-х годах практически каждый ведущий университет мира занялся разработкой собственных решений. К примеру, в апреле 2016 года Гарвардский Университет представил платформу Root [20], Университет Карнеги–Меллон занимается разработкой и продвижением проекта Arts&Bots [18], Массачусетский технологический институт внес свой вклад созданием системы Scratch [6], и данный список на этом не иссякает. Подробный русскоязычный обзор всех вышеупомянутых сред можно найти в работе [21], в которой делается вывод о том, что несмотря на разнообразие всех инструментов в данной области, ни один из них не может удовлетворить всем требованиям, которые ставятся теми или иными образовательными учреждениями. Подавляющее большинство таких сред реализуют только наиболее общую и необходимую функциональность (редактор визуальных диаграмм и возможность исполнить их на роботе с отладкой на компьютере или в автономном режиме), однако в них отсутствуют более специализированные средства обучения программированию, например, возможность генерации читаемого кода по визуальной диаграмме для облегчения перехода с визуальных языков на текстовые, возможность отладки программы на виртуальном симуляторе робота перед исполнением на реальном устройстве или встроенные средства проверки корректности выполнения задания, что автоматизировало бы часть обязанностей преподавателя. В случае, если такие возможности присутствуют в системе, то не все сразу, и все такие среды проприетарны и не бесплатны: их пользователи могут попасть в ситуацию «vendor lock-in», не говоря уже о том, что многие учебные заведения не могут позволить себе покупку дорогостоящих лицензий.

Практически все вышеупомянутые языки реализуют модель вычисления с передачей управления (*control flow*). Такая модель легче воспринимается людьми, поэтому ее разумнее использовать в образовательных целях. Тем не менее, модель потока управления не самая удачная для программирования роботов. Дело в том, что роботы по своей природе реактивны: программа управления роботом представляет собой преобразование сигналов с датчиков в импульсы на приводы. Реактивные модели хорошо описываются так называемыми *языками программирования потоков данных* (*dataflow languages*) [22], в которых программа представляется множеством «черных ящиков», соединенных каналами данных. Каждый такой «ящик» (далее, *блок*) имеет фиксированный набор входов и выходов, его работа состоит в преобразовании данных на входе в данные на выходе (далее данные, посылаемые по каналу, будем называть *токенами*). К примеру, каждый датчик робота в такой схеме описывается всего одним блоком, который отправляет в выходные каналы токены значений с датчика. При этом многие исследователи отмечают удобство визуальных потоковых языков по сравнению с текстовыми (которые также известны как *реактивные*) [22], в частности, из-за наглядной визуализации самих потоков данных.

Идея использования потоковых языков программирования в робототехнике не нова, она реализована практически в каждом инструменте для визуального программирования промышленных и лабораторных систем автоматизации. К таким системам отнесем LabVIEW от компании National Instruments [23], систему Simulink [24], а также среду визуального программирования роботов Microsoft Robotics Developer Studio [25]). Все упомянутые системы хорошо решают задачу автоматизации, имеют мощные инструментарии (например, Microsoft Robotics Developer Studio используется на серверах в широко известной социальной сети MySpace [26]), однако весьма сложны в изучении и даже громоздки, поэтому если и применяются в образовании, то гораздо чаще в университетском [27, 28]. В случае с LabVIEW, к примеру, школьные образовательные эксперименты были популярны в конце 1990-х годов [29, 30], однако это привело к появлению адаптаций среды для целей образования (таких, как Robolab), в которых, в частности, модели потоков данных была предпочтена модель потока управления. Таким образом, образуется другая проблема — «разрыв» между упрощенными учебными языками и «серьезными» потоковыми. Исследователи активно пытаются адаптировать потоковую модель для образовательной робототехники, к примеру, новозеландской исследовательской группой был предложен язык RuRu [31]. Тем не менее, готовых сред программирования на визуальном потоковом языке, применяемых в образовании, все еще не существует.

Данная работа описывает новый инструмент программирования роботов TRIK Studio, являющийся попыткой решения вышеописанных проблем.

Успешность их решения косвенно подтверждается практическими применениями: на данный момент TRIK Studio широко применяется в российском образовании (десятки школ и тематических кружков), известны применения в образовательном процессе европейских стран (Англии и Франции), отдельные пользователи есть на каждом обитаемом континенте мира. Описание будет дано с технической стороны, образовательные аспекты в данной статье обсуждаться практически не будут.

1. Общее описание

TRIK Studio — среда визуального и текстового программирования популярных образовательных конструкторов роботов. В официальной версии имеется поддержка конструкторов Lego Mindstorms NXT, Lego Mindstorms EV3 и ТРИК. Каждый из этих конструкторов может быть запрограммирован на одном из двух визуальных языков — более простом, построенном на модели потока управления, или более сложном, потоковом — или на одном из нескольких текстовых: для Lego NXT доступны языки NXT OSEK C и русскоязычная версия C (для облегчения изучения текстовых языков), для ТРИК — JavaScript, F# [32] или PascalABC.NET [33], для Lego EV3 поддержан единственный официальный язык программирования стандартной прошивки, байткод виртуальной машины EV3.

Визуальная диаграмма может быть исполнена в трех режимах:

- отладка на симуляторе,
- отладки на компьютере с посылкой пакетов на робота по одному из физических каналов (USB, Bluetooth, Wi-Fi, см. главу 5),
- режим генерации из визуальной диаграммы читаемого кода на одном из вышеупомянутых текстовом языке с последующим автономным исполнением его на роботе.

В режиме отладки на симуляторе диаграмма интерпретируется на двумерной имитационной модели робота (см. раздел 8). Пользователь имеет возможность нарисовать двумерную модель мира из стенок, цветных элементов и разметки регионов. Опыт использования показал, что в редакторе модели мира можно создать большинство полей и полос препятствий, используемых на соревнованиях по спортивной робототехнике. Такая возможность, по отзывам пользователей, является очень удобной для первоначальной отладки программы перед каким-либо взаимодействием с роботом. Также наличие симулятора дает возможность обучения программированию и кибернетике в образовательных учреждениях, которые не имеют реальных роботов. Существует так-

же экспериментальная поддержка отладки на трехмерном симуляторе роботов V-Rep [34].

Отладка на компьютере с посылкой команд роботу (режим *интерпретации* в терминах среды) удобна для отслеживания поведения программы на целевом устройстве в реальном времени. В режиме интерпретации можно отслеживать значения переменных в соответствующем окне среды (аналогичному, например, окну поддержки отладчика gdb в различных текстовых IDE), а также строить в реальном времени графики данных с датчиков.

Режим генерации кода позволяет перейти от визуального представления программы к текстовому. Тестовый код отображается во встроенном редакторе qscintilla[3], который обладает возможностью полноценного редактора кода (подсветка синтаксиса, автодополнение, подсветка скобок, отмена/повтор и т.д.). В дистрибутив среды входят все необходимые инструменты для построения и передачи программ на роботов (набор кросскомпиляторов, WinSCP и Putty и т.д.), поэтому процесс компиляции и взаимодействия с контроллерами роботов остается полностью «прозрачным» для пользователей. Многие из пользователей-новичков до определенного момента даже не догадываются, что при нажатии на кнопку автономного исполнения диаграммы превращаются в текстовое представления и компилируются в машинный код перед непосредственным запуском.

Пользовательский интерфейс среды представлен на рисунке 1. На нем запечатлен момент отладки программы выезда робота из лабиринта на двумерном симуляторе.

В режиме двумерной симуляции доступна возможность автоматической проверки задач (см. раздел 9). Программа проверки скриптуется на внутреннем легковесном событийном текстовом языке, файл с сохранением модели мира и программы проверки затем может распространяться как задача. На основе этой системы был запущен курс дистанционного обучения на платформе Stepic[4] с видео-лекциями по основам кибернетики, робототехники и использованию среды TRIK Studio, множеством небольших тестов и 20 задачами образовательной робототехники. Каждая задача может быть скачана, решена и проверена в TRIK Studio и сдана на сервер, где система проверки запустит решение на своих тестах (подобно формату решения задач на соревнованиях ACM ICPC).

Среда написана на языке C++ с инструментарием Qt[5], поэтому является кроссплатформенной (доступны установочные пакеты под Windows, Linux и Mac OS X). При этом, даже несмотря на то, что официальные драйвера для Lego NXT не доступны для Linux и Mac OS X x64, в TRIK Studio существует

[3] https://riverbankcomputing.com/software/qscintilla/intro (дата обращения: 14.05.2016)

[4] https://stepic.org/s/7qe3xj4Z (дата обращения: 14.05.2016)

[5] https://www.qt.io/ru/ (дата обращения: 14.05.2016)

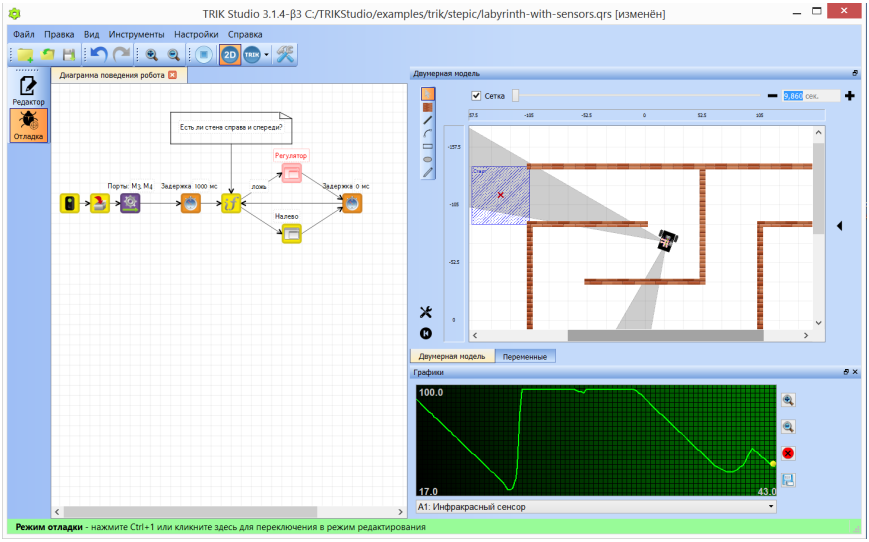

Рис. 1: Интерфейс TRIK Studio

собственная их реализация, подробнее см. главу 5. Среда полностью бесплатна, имеет открытый исходный код и распространяется под лицензией Apache License 2.0[6].

Описанные достоинства выгодно выделяют TRIK Studio на фоне всех аналогов. Фактически, среди упомянутых во введении систем, применяемых в школьном образовании, лишь среда 12Blocks обладает приближенной функциональностью, однако генерирует не читаемый код (для Lego NXT), не имеет средств проверки задач, является платной и не русифицирована. Основные недостатки TRIK Studio на данный момент — это слабая методическая поддержка среды, ограничивающаяся лишь справкой на русском язык, набором примеров и упомянутым курсом дистанционного обучения. Существуют и другие, более мелкие особенности, о которых будет упомянуто в соответствующих разделах ниже.

Остаток статьи будет построен следующим образом. В главах 2 и 3 будут кратко описаны визуальные языки TRIK Studio и их интерпретаторы. Глава 4 дает общее представление об архитектуре среды. Дальнейшие главы будут посвящены конкретным подсистемам TRIK Studio. Общие сведения о реализации механизмов коммуникации с роботами даны в главе 5. Описания интерпретаторов визуальных языков в системе даны в главе 6. Наиболее интересные детали реализации генераторов в текстовые языки из диаграмм потока

[6] http://www.apache.org/licenses/LICENSE-2.0 (дата обращения: 14.05.2016)

управления представлены в главе 7. В главе 8 рассказывается об особенностях реализации подсистемы двумерного имитационного моделирования, далее, в главе 9 описан язык описания программ автоматической проверки заданий, исполняемых на этой подсистеме. Наконец, глава 9 подводит итоги работы.

2. Язык для начинающих

Из всего множества языков, предлагаемых TRIK Studio, наиболее часто в образовательных целях используется упрощенный визуальный язык, основанный на модели потока управления (рис. 2). Язык является графовым, т.е. программирование на нем ведется в терминах сущностей и связей. Для создания программы пользователь перетаскивает необходимые ему блоки на сцену редактора, правит их свойства и соединяет стрелками. Каждый блок выполняет какое-либо примитивное действие и передает управления по стрелкам.

Блоки в описываемом языке можно поделить на четыре группы.

- В первую группу включаются блоки основных алгоритмических конструкций, такие как блоки начала и конца исполнения программы или подпрограммы, ветвления, организации арифметического цикла, выбора (switch), распараллеливания и работы с параллельными задачами (их слияния и принудительного завершения), вызова подпрограммы, генерации случайных чисел и блок текстового программирования на встроенном языке.

- Вторая группа — блоки работы с периферийными устройствами робота. Сюда включаются элементарные действия, не требующие ожидания. К примеру это блоки подачи импульсов на моторы, проигрывания звука, блоки работы с видео-зрением робота, синтезирования речи по заданной строке, управления счетчиками оборотов и лампочками на панелях контроллеров роботов, посылки сообщения на другого робота (часть поддержки мультиагентного взаимодействия), работы с файловой системой робота и т.д.

- Следующая группа включает в себя блоки, «замораживающие» исполнение текущего потока. Сюда входит блок ожидания заданное количества миллисекунд (аналог функции msleep), блоки ожидания желаемого значения с какого-либо датчика или пульта операторского управления роботом и блок ожидания сообщения с другого робота.

- Наконец, в четвертую группу входят блоки рисования графических примитивов на дисплее робота. К таким графическим примитивам относятся прямые линии, прямоугольники, эллипсы, дуги, текст, управление

цветом и толщиной кисти рисования, цветом заливки фона, рисования картинок на экране (например, грустные и веселые смайлики), а также специальные блоки управления маркером для рисования в двумерном симуляторе траектории перемещения робота, что позволяет решать на среде класс задач, предлагаемых различными популярными двумерными исполнителями, такими, как «Черепашка Logo» или «Чертежник».

Рис. 2: Программа выхода из лабиринта по правилу правой руки. Первые четыре блока инициализируют программу: устанавливают константу порога близости стены, включают моторы робота, после чего робот продолжает движение одну секунду, чтобы заехать в лабиринт. Вторые четыре описывают главный цикл управления. Через каждые десять миллисекунд робот проверяет с помощью инфракрасных датчиков, может ли он продолжать движение вперед или направо (условие sensorA2 <= threshold && sensorA1 <= threshold). Если движение возможно, то оно будет осуществляться посредством пропорционального регулирования значений правого датчика (подпрограмма «Регулятор»), если стены есть и слева, и справа, то робот поворачивается на 90 градусов влево (подпрограмма «Налево»)

.

Свойства каждого блока могут быть отредактированы как на самой сцене редактора, так и на отдельной панели редактора свойств. Во всех свойствах блоков, где это уместно, имеется возможность задать вычислимое значение на встроенном в TRIK Studio текстовом языке — статически типизируемом диалекте языка Lua[7]. Модуль синтаксического разбора этого языка написан

[7] http://www.lua.ru/ (дата обращения: 14.05.2016)

с применением библиотеки парсер-комбинаторов для языка C++ стандарта 2011 года, созданной также в рамках проекта TRIK Studio. Вывод типов узлов результирующего абстрактного синтаксического дерева осуществляется алгоритмом Хиндли-Милнера [35], в реализации которого сделаны упрощения, которые здесь из соображений краткости обсуждаться не будут.

Для реализации визуального языка был применен *предметно-ориентированный подход* (*DSM-approach*) [36]. Редактор языка был создан при помощи DSM-платформы QReal [37,38]. Метамодель языка была описана на визуальном метаредакторе DSM-платформы, далее по ней самой платформой был сгенерирован подключаемый модуль с редактором визуального языка. Результатом подключения такого модуля к ядру DSM-платформы является полноценная среда визуального программирования на языке, описанном метамоделью. Эта среда «наследует» все преимущества DSM-платформы, включая современный пользовательский интерфейс, поддержку рисования элементов жестами мышью [39, 40], операции отмены-повтора в редакторе, возможности копирования-вставки групп элементов, изменения масштабов сцены редактора, различные способы рисования и отображения связей между элементами, панели-обозреватели моделей, поддержка особого режима работы на сенсорных-экранах и т.д. По отзывам пользователей, по эргономичности и современности визуальный редактор TRIK Studio превосходит многие аналоги, при этом время на создание первой версии визуального редактора заняло порядка трех человеко-дней. Все это говорит о разумности выбора DSM-платформы QReal в качестве базовой технологии, однако отметим, что во время выполнения описываемой работы в ядро DSM-платформы было внесено множество улучшений и исправлений.

3. Язык для «продвинутых» пользователей

Для пользователей, освоивших программирования на более простом языке, в среде существует возможность программирования на более сложном, но более удобном визуальном языке. Этот язык построен на модели потока данных. В отличие от предыдущего языка, где управление в диаграмме явно передается по стрелкам, блоки в данном языке исполняются параллельно, обмениваясь друг с другом токенами по каналам данных. К примеру, у программы на таком языке может быть сразу несколько точек входа (чаще всего, это датчики робота, которые посылают данные далее по цепочке их обработки), в то время как у каждой программы на языке с передачей управления есть лишь одна входная точка. Также как в предыдущем, в данном языке присутствуют блоки поддержки основных алгоритмических конструкций, блоки взаимодействия со всеми устройствами роботов, блоки рисования и т.д. Как правило, диаграммы на потоковом языке получаются лаконичнее, чем аналогичные диаграммы

на языке с передачей управления, к примеру, на рисунке 3 пропорциональный регулятор для следования вдоль стены на языке с передачей управления, сопоставленный пропорционально-дифференциальному регулятору на потоковом языке.

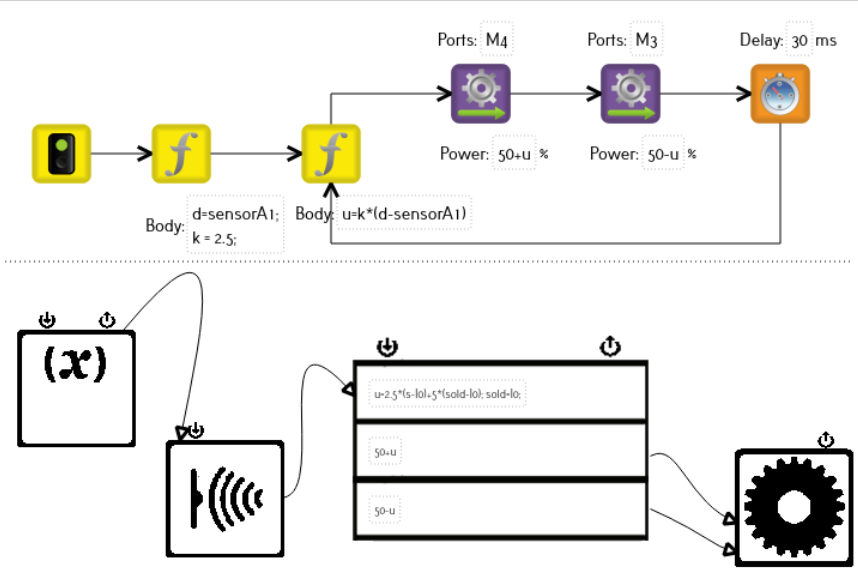

Рис. 3: Регуляторы на двух языках

Для упрощения изучения «продвинутого» языка в нем сохраняется концептуальная совместимость с моделью потока управления. Блоки здесь также могут быть выстроены в цепочку с явной передачей управления, для этого практически на всех блоках есть специальный порт активации, который игнорирует данные, приходящие на него и попросту передает управление блоку подобно тому, как это происходит в языке для начинающих. Поначалу программирование на нем может вестись сходно с тем, как оно велось на предыдущем языке, и лишь с приходом должного опыта пользователь будет выражать свои идеи меньшим количеством блоков.

Выразительная сила языка позволяет описывать на нем широко известные в робототехническом сообществе подходы к построению сложных систем управления роботов, такие как категориальная архитектура, предложенная Родни Бруксом [41], архитектура «колония» Джонатана Коннелла [42], схема поведенческой навигации Рональда Аркина [43], или распределенная схема навигации DAMN [44]. Доказательство этих фактов является скорее материалом для отдельной статьи и здесь приведено не будет (при желании читатель

может убедиться в этом на собственном опыте). Некоторые общие соображения по этому поводу могут быть найдены в статье английских исследователей [45] и диссертации [46].

Следует отметить, что на момент написания статьи поддержка потокового языка в среде реализована в экспериментальном режиме и не входит в официальную версию, распространяемую между конечными пользователями. Исходный код его редактора и инструментальной поддержки находится в открытом доступе[8] и может быть собран вместе с исходным кодом TRIK Studio.

4. Общая архитектура

Цель данной главы — структурирование информации предыдущих разделов в виде архитектурного описания: большинство описываемых здесь деталей в каком-либо виде уже были упомянуты выше. Все диаграммы, представленные в данной главе абстрагированы от множества деталей, которые, хоть и представляют интерес в контексте данной статьи, здесь представлены не будут из соображений краткости.

На рисунке 4 представлена часть общей архитектура среды TRIK Studio. Как из него видно, система разбита на несколько «слоев» абстракции. Каждый из слоев содержит код, реализующий сложные элементы функциональности среды и при этом имеет свой строго определенный и задокументированный API. На самом низком уровне такими «слоями» являются библиотеки взаимодействия с реальными и виртуальными роботами. На основе такого взаимодействия строится иерархия устройств робота (датчиков, приводов, дисплеев, динамиков, пультов управления, кнопок на контроллере и т.д.), из которых, в свою очередь, составляется описание модели того или иного конструктора роботов. Описания модели роботов группируются в подключаемые к ядру TRIK Studio модули, они используются другими подсистемами TRIK Studio (интерпретаторами и генераторами, которые также представляют собой подключаемые к ядру TRIK Studio модули).

Ядро TRIK Studio, в свою очередь, является подключаемым к DSM-платформе QReal модулем[9]. Оно подстраивает интерфейс QReal под себя, добавляя к нему множество панелей и окон, таких, как окно двумерного симулятора, панели конфигурирования датчиков робота, просмотра их значений и построения графиков с них и т.д., подгружает все описания моделей роботов и отвечает за их рассылку всем необходимым подсистемам, предоставляет пользователю информацию о работе процесса генерации и интерпретации, и

[8] https://github.com/ZiminGrigory/DFVPL (дата обращения: 14.05.2016)

[9] Таким образом, в проекте реализована двухуровневая схема подключаемых модулей, подобно тому, как это сделано в системе ReSharper от компании JetBrains: ReSharper может расширяться подключаемыми модулями, при этом сам является подключаемым к Visual Studio модулем

т.д. (полный список функций ядра слишком длинный и только усложнит восприятие). Наряду с ядром инструментальной поддержки TRIK Studio, к ядру QReal подключаются модули, описывающие редакторы визуальных языков, которые сгенерированы самой платформой QReal. Подробности про подсистемы более низкого уровня на рис. 4 даны в главах ниже.

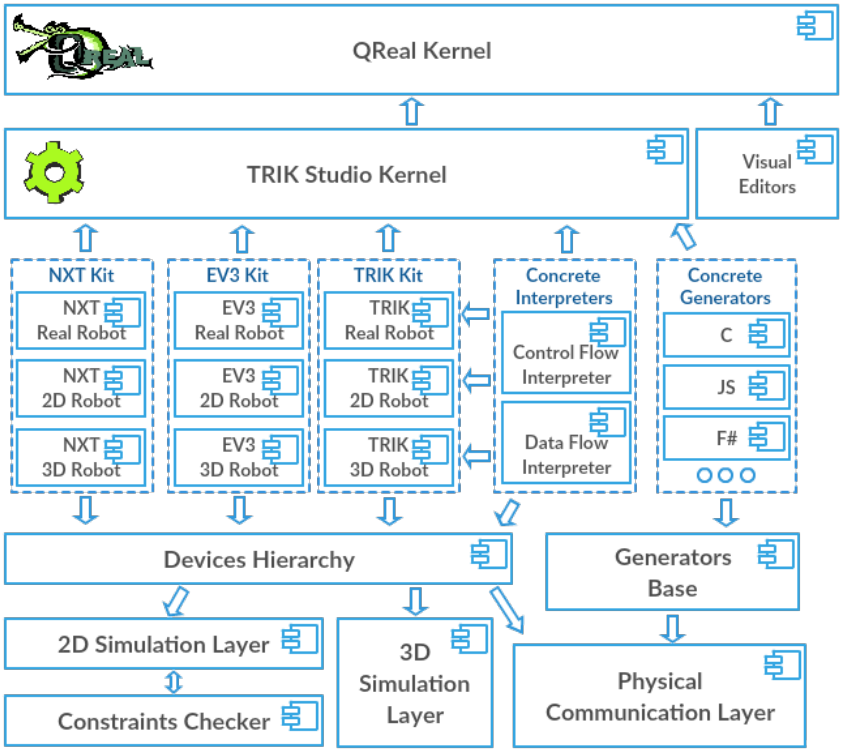

Рис. 4: Общая архитектура системы

Как можно заметить, вся среда построена на принципе подключаемых модулей. Отметим, что такая идея позволила сделать более гибкой систему конфигурирования установки: пользователь может выбрать, какие компоненты ему интересны, а остальные отключить (например, если у него есть лишь конструктор Lego EV3, то ему не интересны модули поддержки конструкторов Lego NXT и TRIK, в таком случае к нему на компьютер даже не попадут файлы, эту поддержку предоставляющие). Более того, такой подход хорошо совместим с пакетной организацией систем установки программ на различные дистрибутивы Linux. Это хорошо и с архитектурной точки зрения, так как ис-

чезает большинство зависимостей между компонентами — ядро TRIK Studio, к примеру, «минималистично», оно «не знает» обо всех возможностях среды и лишь предоставляет и реализует общие объекты и интерфейсы для каждого робототехнического конструктора. API каждого «слоя» абстракции зафиксировано и задокументировано, это может позволить создавать модули поддержки новых языков и конструкторов в среде даже независимым разработчикам из сообщества проекта.

5. Коммуникации с контроллерами роботов

Одна из важных подсистем TRIK Studio — это модуль взаимодействия с контроллерами роботов по тому или иному физическому протоколу. На диаграмме компонентов 4 это один из самых низких «слоев» абстракции среды. API модуля коммуникаций предоставляет следующие операции:

- переключение текущего физического способа взаимодействия с целевым устройством (контроллеру робота) между Bluetooth, USB и Wi-Fi,
- указание адреса устройства в терминах физического протокола (это может быть номер COM-порта в случае с взаимодействием по Bluetooth, или IP-адрес или имя хоста в случае с взаимодействием по Wi-Fi),
- подключение-отключение к целевому устройству,
- посылка набора байтов на целевое устройство,
- события получения набора байтов с целевого устройства,
- события изменения статуса подключения (такие события вырабатываются, если связь с роботом установилась или разорвалась),
- события, оповещающие обо всех произошедших ошибках с локализованным их описанием.

Взаимодействие по USB происходит при помощи библиотеки libusb[10], для реализации обмена по Bluetooth используется библиотека QextSerialPort[11], взаимодействие по Wi-Fi происходит поверх протоколов TCP и UPD посредством библиотеки QtNetwork. Процесс общения с целевым устройствам происходит в отдельном потоке, чтобы не «замораживать» пользовательский интерфейс. Конкретные реализации механизмов коммуникации в подсистеме

[10] http://libusb.org/ (дата обращения: 14.05.2016)
[11] https://github.com/qextserialport/ (дата обращения: 14.05.2016)

могут быть расширены и подменены другими подсистемами «извне», это используется, например, для совместимости TRIK Studio со стандартным драйвером Lego NXT. Отметим, что взаимодействие через него — это не единственный способ общения с контроллером NXT по USB в TRIK Studio, в среде реализован собственный, аналогичный драйвер Lego NXT, однако, в отличие от официального, работающий на всех поддерживаемых операционных системах через libusb.

6. Интерпретаторы

Интерпретатор преобразует визуальную диаграмму в последовательность команд на целевом устройстве (рис. 5). При этом иерархия устройств в системе построена таким образом, что не делается различий, реальное ли это устройство или симулируемое (рис. 6). Каждое устройство представляет собой класс C++, содержащий код взаимодействия с целевым механизмом. Реализации устройств какого-либо конструктора находятся в соответствующих библиотеках подключаемых модулей. Для общения по физическим каналам устройства используют подсистему физических коммуникаций (гл. 5), для передачи команд двумерному и трехмерному симуляторам — API этих симуляторов.

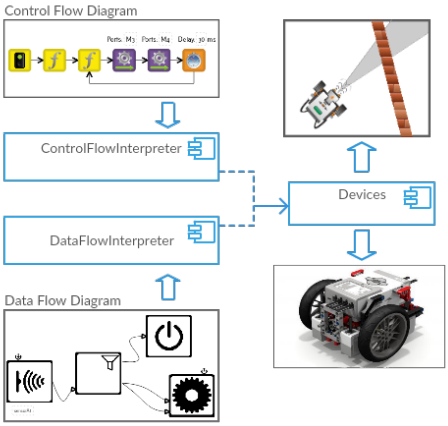

Рис. 5: Общий принцип работы систем интерпретации

Интерпретаторы, также как и вся система в целом, написан на языке C++ с использованием инструментария Qt. На вход подсистеме интерпретации поступает описание созданной пользователем диаграммы в некотором внутрен-

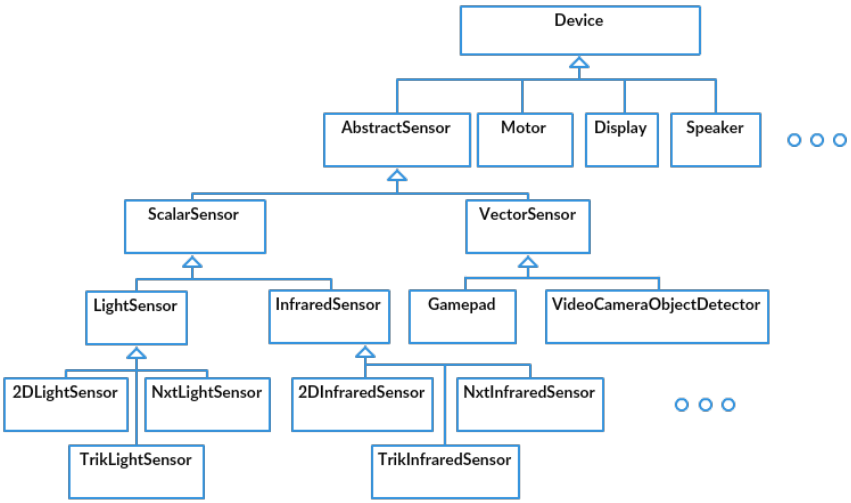

Рис. 6: Часть иерархии устройств в системе

нем представлении. В среде реализовано два интерпретатора: интерпретатор языка для начинающих с передачей управления и интерпретатор потокового языка.

Интерпретатор первого языка находит на диаграмме начальный блок, и далее трассирует поток управления диаграммы в том порядке, как он задан стрелками. Текущий блок исполнения подсвечивается на визуальной диаграмме, чтобы пользователю был виден прогресс исполнения. Для каждого посещенного блока специальные фабрики интерпретатора создают объекты, которые реализуют его поведение. В общем случае, такой объект-реализация ищет среди готовых к работе устройств робота необходимые ему, вызывает соответствующие команды и передает управление следующим блокам по какой-либо исходящей ветке (которая определяется, опять-таки, самой реализацией блока). При этом не делается различий, является ли найденное устройство частью реального робота или симулируемым, таким образом, одна и та же подсистема интерпретации используется для исполнения диаграммы в двух из трех режимов (гл. 1).

Если на каком-то шаге интерпретации повстречался блок распараллеливания, интерпретатор пополняет список запущенных потоков и запускает новые потоки исполнения с соответствующих точек, создавая для каждого свой стек вызовов. Если встретился вызов подпрограммы, вычисляются ее параметры, укладываются на текущий стек вызовов, и далее процесс интерпретации повторяется рекурсивно. При достижении любого завершающего блока со сте-

ка снимается верхний фрейм, и исполнение продолжается с соответствующей точки на предыдущей диаграмме. Если при снятии очередного фрейма стек вызовов стал пустым, текущий поток исполнения считается завершенным. Таким образом, программа исполняется по мере того, как она «открывается» интерпретатору.

Отличительной чертой такой реализации является тот факт, что все изменения, вносимые пользователем на диаграмму во время процесса интерпретации, «подхватываются на лету». Это является удобным, например, в случае подбора коэффициентов пропорциональности какого-либо регулятора, реализованного в программе. При этом процесс исполнения не изменившихся кусков диаграммы оптимизирован: объекты-реализации блоков кэшируются в отдельную таблицу, то же происходит с абстрактными синтаксическими деревьями и информацией о выводе типов выражений на текстовом языке. Таким образом, если содержимое программы не меняется во время процесса интерпретации, повторная передача управления в ветки программы повлечет использование уже созданных сущностей. Валидация диаграммы также осуществляется «на лету», в случае, если в диаграмме найдено некорректное место, пользователю отображается локализованное сообщение об ошибке. Последняя черта может рассматриваться как отрицательная, так как сообщение об ошибке появляется лишь в момент ее достижения в программе.

Интерпретатор потокового языка работает в два этапа. На первом, подготовительном этапе диаграмма проходит валидацию, создаются объекты-реализации элементарных блоков, преобразующие диаграмму в команды на устройствах роботов подобно тому, как это происходит в интерпретаторе языка, описанного в предыдущей главе. Объекты-реализации соединяются друг с другом посредством механизма сигналов и слотов инструментария Qt в соответствии с тем, как они соединены потоками данных на диаграмме; здесь оказался полезным паттерн проектирования «издатель-подписчик». На втором этапе интерпретатор запускает на автономное исполнение каждый из блоков, в который не входит ни один поток данных, каждый из которых, в свою очередь, будет активировать блоки, соединенные с ним выходными потоками данных при выработке токенов. Исполнение блоков происходит псевдопараллельно с централизованной очередью сообщений (рис. 7). Это решение отличает данный язык от всех его промышленных аналогов (к примеру, в Microsoft Robotics Developer Studio диаграмма разворачивается в набор независимых веб-сервисов [25]), его причины состоят в узкой направленности языка на «слабые» контроллеры учебных роботов, в которых параллельное исполнение большого количества блоков затруднительно или вовсе невозможно. Тем не менее, в языке все же присутствует блок распараллеливания, который полезен, например, для выражения вышеупомянутых подходов к проектированию сложных систем управления. Такой блок может рассматриваться как меха-

низм низкоуровневого управления вычислительными ресурсами в языке.

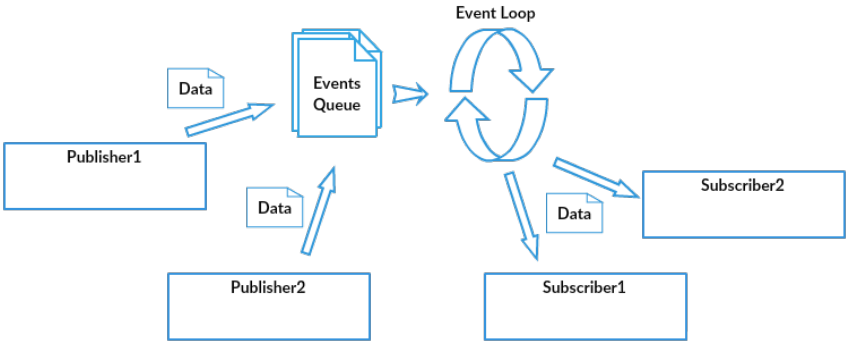

Рис. 7: Механизм исполнения потоковой программы в TRIK Studio

7. Генераторы

Одна из наиболее важных и востребованных функций среды TRIK Studio — генерация читаемого кода по визуальной диаграмме. По одной и той же диаграмме может быть сгенерирован код на любом поддерживаемом средой текстовом языке (C, JavaScript, F#, Pascal, РуСи [47], байткод BM EV3). Под визуальными диаграммами в данном разделе будем подразумевать программы на языке для начинающих (с передачей управления), под генераторами — модули генерации кода по визуальным диаграммам именно этого языка. Общие принципы построения системы генераторов для потокового языка похожи на изложенные и здесь обсуждаться не будут.

Визуальная диаграмма состоит из блоков и стрелок, таким образом, описывает *граф потока управления* программы [48]. Все основные алгоритмические конструкции (развилки, циклы, конструкции выбора и т.д.) складываются из стрелок. Например, чтобы задать бесконечный цикл, достаточно лишь провести стрелку от блока к одному из предыдущих, для описания циклов вида while-do, do-while или с инструкцией break внутри тела достаточно лишь провести одну из веток развилки из тела цикла к соответствующему блоку вне его. Очевидно, что одна стрелка на диаграмме соответствует инструкции goto в коде. Однако код, содержащий инструкции goto весьма труден для чтения человеком, тем более не подходит для обучения азам текстового программирования, поэтому следует избегать их появления в сгенерированном коде. Тем не менее, на TRIK Studio возможно написать программу, не выразимую общепринятыми алгоритмическими конструкциями, в таком случае появления

инструкций goto в коде не избежать. С другой стороны, далеко не все современные текстовые языки поддерживают инструкцию goto (например, в языке JavaScript такой поддержки нет). Будем называть *структурированными* диаграммы, которые можно выразить стандартными алгоритмическими блоками (if-then, if-then-else, while-do loop, do-while loop, while-break loop, switch), код на текстовом языке, соответствующий структурированной диаграмме также будем называть *структурированным*. В противном случае будем говорить, что диаграмма *неприводима*, а код будем называть *неструктурированным*. В случае, если диаграмму не удается сгенерировать в структурированный код, пользователю должно быть показано предупреждение. Возможность генерации в неструктурированный код должна быть, так как требуется запускать на автономное исполнение даже запутанные диаграммы.

Таким образом, реализация системы генераторов в TRIK Studio потребовала решения двух нетривиальных задач, которые сформулированы ниже в виде требований.

- Система генераторов должна быть организована таким образом, чтобы на добавление в систему генератора в новый текстовый язык занимало минимальное количество времени и сил. По возможности, это должно быть по силам даже людям, не знакомым с кодом системы.

- Для каждого языка (где это возможно) система должна поддерживать два режима генерации: структурированные диаграммы должны быть сгенерированы в структурированный код, запутанные — в неструктурированный. При этом успешность структурирования диаграммы не должна зависеть от ее размера и сложности.

Первая задача является чисто архитектурной. Ее решение представлено на рисунке 8. Основная идея — разделение процесса генерации на два этапа. На первом диаграмма преобразуется в представление, независимое от целевого языка генерации — *семантическое дерево*. Семантическое дерево упорядочивает структуру графа потока управления до «древесной», в независимости от того, является ли диаграмма структурированной или нет. В случае, если диаграмма структурирована, родительский узел семантического дерева всегда соответствует блоку вычислений в целевом коде, дети — инструкциям этого блока, среди которых нет goto. Опишем последовательность действий, отображенных на рисунке 8.

На вход генератору поступает визуальная диаграмма. На первом этапе диаграмма обходится в глубину компонентой-валидатором (построенной с применением шаблона проектирования «посетитель»). Для всех выражений на встроенном текстовом языке в свойствах блоков происходит их синтаксический разбор и вывод типов. В случае, если диаграмма или код в свойстве содержит ошибки, о них сообщается пользователю в локализованном виде и про-

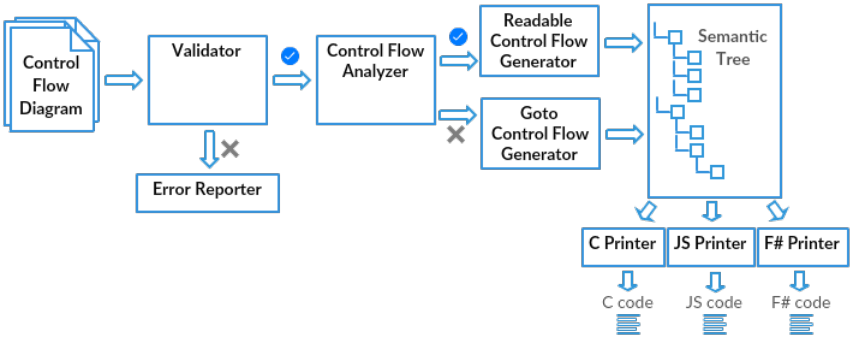

Рис. 8: Архитектура подсистемы генерации кода в TRIK Studio

цесс завершается. В случае, если диаграмма синтаксически корректна, она поступает на вход анализатору потока управления, о котором будет рассказано абзацем ниже. В зависимости от того, может ли диаграмма быть сгенерирована в структурированный код, выбирается промежуточный генератор независимого представления, который извлекает семантическое дерева из потока управления диаграммы. Если целевой язык не поддерживает инструкцию goto или, наоборот, высокоуровневые конструкции, как, например, ассемблер-подобный байткод BM EV3 (что указывается в конфигурации генератора), соответствующему генератору будет предпочтена его альтернатива. Наконец, на последнем шаге процесса генерации семантическое дерево печатается в целевой текстовый язык. Для этого используется подход с заданием шаблонов генерации для целевого текстового языка. Все, чтобы необходимо сделать для реализации нового генератора — переопределить набор этих шаблонов и указать конфигурацию генератора. К примеру, шаблон генерации условного оператора в язык C выглядит следующим образом:

```
if (@@CONDITION@@) {
    @@THEN_BODY@@
} else {
    @@ELSE_BODY@@
}
```

Вторая из решенных проблем — реализация модуль анализатора потока управления. В терминах текстовых языков задача звучит так: по данной программе, поток управления которой описан исключительно инструкциями goto требуется выдать эквивалентную структурированную программу. Данная задача решалась различными исследователями, работающими в области деком-

пиляции программного кода [48, 49]. Приемлемый способ ее решения по методу *интервального анализа* графа потока управления описан, к примеру, в монографии [48]. Алгоритм интервального анализа графа потока управления диаграммы с незначительными модификациями реализован в TRIK Studio. Коротко опишем его.

Неформально говоря, *интервал* — это участок графа потока управления с одним входом и одним выходом. Целью алгоритма является построение дерева вложенности интервалов графа потока управления программы, которое в нашем случае и является семантическим деревом. Алгоритм работает на рекурсивном подъеме поиска в глубину в графе потока управления. На каждом шагу алгоритм пробует сопоставить подграф, выходящий из текущей вершины с каждым из шаблонов элементарных интервалов (алгоритмических конструкций, в терминах которых и будет структурирована программа). Если такой шаблон удается найти, весь подграф сворачивается в одну вершину и процесс продолжается. Самый простой пример такого шаблона — два интервала, соединенные ровно одной дугой, что соответствует последовательной композиции операторов. Если шаблона интервала не нашлось, то диаграмма считается неприводимой.

Очевидными ограничениями на целевой языка в текущей реализации является его императивность: цепочка блоков на диаграмме должна преобразовываться в последовательное исполнение утверждений в текстовом языке, что проще всего достигается применением оператора последовательной композиции. Тем не менее, идея подхода с независимым представлением диаграммы в виде может помочь даже здесь. К примеру, для поддержки генерации в чисто функциональные языки достаточно расширить элементы независимого представления конструкцией продолжения (continuation passing style, CPS) и добавить промежуточный генератор диаграммы в независимое представление в форме CPS. После этого достаточно переопределить шаблоны печати независимого представления во все необходимые языки.

8. Двумерное имитационное моделирование

Особая часть среды TRIK Studio, отделимая от всей остальной системы — подсистема двумерного имитационного моделирования робота (двумерный симулятор). Окно симулятора встроено в интерфейс TRIK Studio, но может и использоваться отдельно. Основная часть симулятора — редактор модели мира. В специальном меню можно выбрать инструмент рисования (подобно тому, как это происходит в большинстве графических редакторов). Инструменты рисования делятся на стены (твердые предметы, на которые реагируют ультразвуковые и инфракрасные датчики роботов и сквозь которые робот не может проехать) и цветные маркеры на полу (элементы, на которые реагируют

датчики цвета, освещенности и эмуляторы систем видео-зрения) — прямые линии и кубические кривые безье, прямоугольники, эллипсы и стилус (произвольная растровая фигура, рисующаяся мышью как карандашом). Для каждого маркера на полу может быть задана его толщина, цвет и заливка. Модель робота всегда присутствует на сцене редактора мира, на нем произвольным образом можно размещать в виртуальные датчики. Робот представляет собой двухколесную тележку с дифференциальным приводом. Для удобства регионы сканирования датчиков расстояния подсвечиваются. На отдельной панели для каждой поддерживаемой модели робота (NXT, EV3 или TRIK) присутствует эмулятор панели контроллера: фронтальное изображение его лицевой панели с кнопками, нажатие мышью на которые эмулирует нажатие на кнопку реального контроллера, цветными светодиодами, меняющими свой цвет соответственно тому, как того требует написанная пользователем программа и эмулятор дисплея, изображение на котором можно менять блоками из группы «Рисование».

Важной особенностью симулятора является неотличимость процесса исполнения программы от процесса создания модели мира. Модель мира может редактироваться в произвольный момент, на любое добавление пользователем стенок и цветных элементов во время исполнения программы датчики начнут реагировать немедленно, положение и направление робота и его датчиков в пространстве может быть изменено в любой момент, что соответствует в реальном мире физическому воздействию на корпус робота.

Симулятор построен на основе архитектуры MVC. Модель содержит сериализуемое описание мира и робота, уведомляет о любом изменении любого свойства любого предмета. На события модели подписывается представление, таким образом, что все изменения автоматически отображаются на сцене и панелях симулятора. Пользовательские действия в представлении выполняются через контролирующую компоненту, которая кладет очередное действие на вершину специального стека для возможности их отмены и повтора (undo-redo).

Важной частью модели является ее «физика». В симуляторе реализовано две физические модели поведения робота: идеальная и реалистичная. В идеальной физике игнорируются силы трения о пол и стены, импульсы моторов, при неизменных скоростях моторов робот будет двигаться равномерно, (без ускорения и замедления). Любое, даже самое легкое столкновение со стеной в такой модели остановит робота. В реалистичной модели ведется полный учет сил тяги и трения колес робота о пол и стены, при столкновении со стеной робот поведет себя сходно с тем, как это происходит в реальности. Описание физики такого поведения является скорее материалом для отдельной статьи и здесь приведено не будет. Переключение между физическими моделями происходит при выставлении пользователем флажка «реалистичная физика»

и также может произойти в любой момент, даже во время исполнения программы. Также пользователь в любой момент может добавить гауссов шум к показаниям датчиков и значениям импульсов на моторах.

Время в двумерном симуляторе не соответствует процессорному: в нем существует централизованная временная прямая, темп хода которой может меняться на специальной панели. Система построена таким образом, что поведение робота не зависит от темпа хода модельного времени. Та же самая временная прямая используется в интерпретаторах визуальных языков TRIK Studio для соответствия блоков работы со временем модельному, а не процессорному времени.

9. Язык проверки ограничений

Последняя важная часть TRIK Studio, о которой будет рассказано в данной работе — механизм автоматической проверки заданий на TRIK Studio. Модель мира в двумерном симуляторе может быть превращена специальными средствами среды в упражнение, распространяемое между учениками. Для этого достаточно задать два набора описаний:

- описание того, какие части упражнения нельзя менять ученику (модель мира, начальное положение робота и его датчиков, сам набор датчиков, привязку моторов и физические настройки симуляции),
- программа проверки корректности решения задачи.

Программа проверки ограничений описывается на специальном текстовом XML-подобном языке. Данный язык интересен сам по себе и может рассматриваться как отдельный результат работы. Программа на таком языке представляет собой множество событий $\{e_1, e_2, ..., e_n\}$, где каждое событие e_i представляет собой это тройку (id_i, c_i, T_i):

- id_i — идентификатор события, внутренняя метка, по которой другие события могут получать информацию о событии e_i, может быть пустым;
- c_i — условие срабатывания события, представляющее собой формулу логики первого порядка без кванторов, об интерпретации предикатных и функциональных символах которой будет рассказано ниже;
- T_i — упорядоченный список элементарных триггеров $[t_{i1}, t_{i2}, ..., t_{in}]$. Элементарный триггер задает действие это некоторое действие, выполняемое в момент истинности условия срабатывания события c_i.

Одно событие задается одним элементом в XML-спецификации программы. Каждое событие в текстовом описании программы может быть представлено либо в *каноническом* виде, либо в виде *ограничения*. Событие в канонической форме — уже описанная тройка (id_i, c_i, T_i).

Событие в форме ограничения — это тройка $(id_j, c_j, message_j)$. Ограничение интерпретируется как событие $(id_i, !c_i, [fail(message_j)])$, 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 заданное условие, и которое сообщает пользователю 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, так как проверка, очевидно, не может осуществляться бесконечное время. Наличие ограничения на лимит времени проверяется как часть синтаксиса языка. Описание события в форме ограничения введено в язык из чисто прагматических соображений, так как на практике удобно описывать утверждения вида «робот x не должен покидать пределы зоны z» или «к роботу x должен быть подключен набор датчиков s» именно в терминах ограничений, а не событий.

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

- Предикаты сравнения значений термов $>$, $<$, $\leq$, $\geq$, $=$, $\neq$.
- Пространственные предикаты. Имеют вид «предмет x находится внутри области y».
- Предикаты состояния событий $settedUp(id_i)$ и $dropped(id_i)$, описывающие состояние события (активно/неактивно). В активном состоянии событие может быть выполнено, в неактивном оно не выполнет триггеры даже при выполнении условия срабатывания события.
- Предикат времени $timer(t)$, который становится истинным спустя t отсчетов модельного времени и остается истинным после, а до этого момента ложен.
- Прочие предикаты, выражаемые уже описанными и введенные исключительно 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 для реализации сложной логики проверки, например, при подсчете числа проделанных роботом итераций.

- Арифметические и геометрические операции над значениями других термов, например, модуль числа, подсчет расстояния между двумя точками или выпуклая оболочка фиксированного 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. Такой путь будет преобразован системой в последовательность получений значений свойств объектов C++, реализующих данные объекты в модели мира посредством механизма рефлексии Qt. Подробное описание всех свойств, предоставляемых окружением, довольно громоздкое и не будет здесь приведено.

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 элементов имитационной модели мира.

- Триггеры управления состоянием событий. Каждое событие может активировать или деактивировать другое или себя; с помощью этого можно задавать сложную логику проверки.

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

Архитектурно система проверки ограничений представляет собой отдельно стоящий модуль, использующий в качестве разделяемых библиотек ядро и имитационную модель системы TRIK Studio. Использование рефлексию, реализованную для C++ в библиотеке Qt для доступа к свойствам объектов имитационной модели позволяет намного проще масштабировать саму модель: при расширении возможностей системы или добавлении нового симулируемых устройств новые объекты и свойства будут немедленно доступны из языка описания ограничений, без дополнительной модификации кода проверяющей

```
<!-- 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 область!">
    <inside objectId="robot1" regionId="myspace"/>
  </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.

системы. Тестовая система подписывается на события имитационной модели подобно тому, как это делает представление симулятора (в смысле архитектуры MVC), и вызывает проверку ограничений на каждом отсчете модельного времени. При этом проверяются только активные ограничения, которых в каждый конкретный момент времени немного, поэтому общая скорость работы симулятора при наличии проверяющей программы практически не меня-

ется.

Описанный язык удобен для задания пространственных и временных ограничений на поведение системы. Он является значительно более удобным и выразительным, чем, к примеру, язык темпоральных логик или топологико-темпоральных логик, используемых для описания формальных ограничений на поведение робота в работах последних лет [50–53]. При этом, язык не тьюринг-полон (на нем, к примеру, нельзя выразить нормальные алгоритмы маркова из-за отсутствия возможностей работы с коллекциями произвольной длины, что делает невозможным реализацию на нем замены в строке), это говорит о возможности автоматического анализа интересных свойств программ на нем. Ко всему прочему, подмножество этого языка может использоваться для описания требований к программе для последующей ее формальной верификации. В будущих версиях TRIK Studio планируется реализация визуального конструктора ограничений на этом языке. Появление такого конструктора и интеграция мощного верификатора даст возможность формально доказывать корректность визуальных программ, не написав ни одной формулы формальной логики. Подробное исследование этого вопроса — тема для будущих работ.

Система проверки ограничений используется для функционального тестирования среды на сервере системы непрерывной интеграции. Об автоматизированном тестировании среды TRIK Studio подробнее рассказано в работе [54].

Заключение

В статье было дано техническое описание среды программирования роботов TRIK Studio. Среда разработана на кафедре системного программирования Санкт-Петербургского Государственного Университета и по данным системы Google Analytics в настоящее время насчитывает около десятка тысяч пользователей на всех обитаемых континентах мира. На данный момент интерфейс среды переведен на три языка (русский, английский, французский), в том числе силами сообщества пользователей. Система активно развивается, исходный код среды находится в открытом доступе[12]. Без учета ядра DSM-платформы, код системы содержит порядка 100 тысяч строк, включая техническую документацию всех компонентов, написанную в doxygen[13]-формате в заголовочных файлах среды.

Будущие направления работы разделяются на два больших класса: технические улучшения среды и исследовательские проекты. Задачи из первого класса появляются из общения с педагогами и учениками, пользующими-

[12] https://github.com/qreal/qreal (дата обращения: 14.05.2016)
[13] http://www.doxygen.org/ (дата обращения: 14.05.2016)

ся средой, и характеризуются большой численностью (на системе управления процессом разработки их записано порядка сотен). К примеру, это задачи по поддержке новых робототехнических платформ в среде (таких, как, Arduino), улучшение двумерного симулятора (доработка физической модели, поддержка симуляции мультиагентных систем и т.д.), поддержка новых текстовых языков, общие улучшения ядра DSM-платформы QReal и т.д. Задачи из второго класса требуют применение большого количества теоретических научных работ в области компьютерных наук. На данный момент исследовательская работа идет по двум «ветвям»: ветви предметно-ориентированного визуального моделирования и ветви формальных методов анализа программных систем. К первой ветви относится, к примеру, работа по автоматической генерации метамодели потокового языка из описания доступных пакетов системы промежуточного уровня на роботе (такой, как ROS [55]). В рамках второго направления ведутся работы по выражению строгой семантики языков с целью применения подходов к формальной верификации диаграмм.

Список литературы

[1] Papert Seymour. Mindstorms: Children, Computers, and Powerful Ideas. New York, NY, USA: Basic Books, Inc., 1980. С. 230.

[2] Черёмухин Пётр Сергеевич. Внедрение робототехники в образовательное пространство школы с целью формирования инновационного мышления учащихся // статья в сборнике материалов IX международной научно-практической конференции «Актуальные вопросы развития образовательной области «Технология. 2014. С. 175–182.

[3] Erwin Ben, Cyr Martha, Rogers Chris. Lego engineer and robolab: Teaching engineering with labview from kindergarten to graduate school // International Journal of Engineering Education. 2000. Т. 16, № 3. С. 181–192.

[4] Kelly James Floyd. Lego Mindstorms NXT-G Programming Guide. Apress, 2007. С. 336.

[5] Valk Laurens. LEGO MINDSTORMS EV3 Discovery Book: A Beginner's Guide to Building and Programming Robots. No Starch Press, 2014.

[6] Scratch: programming for all / Mitchel Resnick, John Maloney, Andrés Monroy-Hernández [и др.] // Communications of the ACM. 2009. Т. 52, № 11. С. 60–67.

[7] S4A. 2016. URL: http://s4a.cat/.

[8] mBlock. 2016. URL: http://www.mblock.cc/.

[9] Enchanting. 2016. URL: http://enchanting.robotclub.ab.ca/tiki-index.php.

[10] СкретчДуино — образовательная робототехника. 2016. URL: http://www.scratchduino.ru/.

[11] Программирование на Blockly. 2016. URL: http://blockly.ru/.

[12] Wolber David. App inventor and real-world motivation // Proceedings of the 42nd ACM technical symposium on Computer science education / ACM. 2011. С. 601–606.

[13] 12Blocks | OneRobot. 2016. URL: http://onerobot.org/products/12blocks/.

[14] Graphical Programming Environments for Educational Robots: Open Roberta-Yet Another One? / Beate Jost, Markus Ketterl, Reinhard Budde [и др.] // Multimedia (ISM), 2014 IEEE International Symposium on / IEEE. 2014. С. 381–386.

[15] Ardublock | A Graphical Programming Language for Arduino. 2016. URL: http://blog.ardublock.com/.

[16] Incorporating the Fischertechnik bricks into undergraduate mechatronics courses / GK Chang, SY Fan, RL Shue [и др.] // EE 2006, International Conference on Innovation, Good Practice and Research in Engineering Education. 2006.

[17] Mayerová Karolina. Pilot activities: LEGO WeDo at primary school // 3rd International Workshop, Teaching Robotics, Teaching with Robotics. 2012. С. 32–39.

[18] A visual robot-programming environment for multidisciplinary education / James Dorian Cross, Christopher Bartley, Emily Hamner [и др.] // 2013 IEEE International Conference on Robotics and Automation (ICRA) / IEEE. 2013. С. 445–452.

[19] Wonder Workshop | Wonder. 2016. URL: https://www.makewonder.com/apps/wonder.

[20] Root : Wyss Institute at Harvard. 2016. URL: http://http://wyss.harvard.edu/viewpage/629/.

[21] Мордвинов Дмитрий Александрович, Литвинов Юрий Викторович. Сравнение образовательных сред визуального программирования роботов // Компьютерные инструменты в образовании. СПб., 2016.

[22] Johnston Wesley M, Hanna JR, Millar Richard J. Advances in dataflow programming languages // ACM Computing Surveys (CSUR). 2004. Т. 36, № 1. С. 1–34.

[23] Kodosky Jeffrey, MacCrisken Jack, Rymar Gary. Visual programming using structured data flow // Visual Languages, 1991., Proceedings. 1991 IEEE Workshop on / IEEE. 1991. С. 34–39.

[24] Dabney James B, Harman Thomas L. Mastering simulink. Pearson/Prentice Hall, 2004.

[25] Jackson Jared. Microsoft robotics studio: A technical introduction // Robotics & Automation Magazine, IEEE. 2007. Т. 14, № 4. С. 82–87.

[26] Michael S. Scherotter. CCR at MySpace. 2009. URL: http://channel9.msdn.com/Shows/Communicating/CCR-at-MySpace.

[27] A LabVIEW-based remote laboratory experiments for control engineering education / Miladin Stefanovic, Vladimir Cvijetkovic, Milan Matijevic [и др.] // Computer Applications in Engineering Education. 2011. Т. 19, № 3. С. 538–549.

[28] Yi Zhou, Jian-Jun Jiang, Shao-Chun Fan. A LabVIEW-based, interactive virtual laboratory for electronic engineering education // International Journal of Engineering Education. 2005. Т. 21, № 1. С. 94–102.

[29] A low-cost, innovative methodology for teaching engineering through experimentation / M. Cyr, V. Miragila, T. Nocera [и др.] // Journal of Engineering Education. Washington, 1997. Т. 86. С. 167–172.

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

[31] Diprose James P, MacDonald Bruce A, Hosking John G. Ruru: A spatial and interactive visual programming language for novice robot programming // Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on / IEEE. 2011. С. 25–32.

[32] 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.

[33] Долинер ЛИ. Основы программирования в среде PascalABC. NET: учебное пособие. Издательство Уральского университета, 2014.

[34] Rohmer Eric, Singh Surya PN, Freese Marc. V-REP: A versatile and scalable robot simulation framework // Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on / IEEE. 2013. С. 1321–1326.

[35] Damas Luis, Milner Robin. Principal type-schemes for functional programs // Proceedings of the 9th ACM SIGPLAN-SIGACT symposium on Principles of programming languages / ACM. 1982. С. 207–212.

[36] Кознов Дмитрий Владимирович. Основы визуального моделирования // М.: Изд-во Интернетуниверситета информационных технологий, ИНТУИТ. ру, БИНОМ, Лаборатория знаний. 2008.

[37] Средства быстрой разработки предметно-ориентированных решений в metaCASE-средстве QReal / АС Кузенкова, АО Дерипаска, КС Таран [и др.] // Научно-технические ведомости СПбГПУ. С. 142.

[38] QReal DSM platform-An Environment for Creation of Specific Visual IDEs / Anastasiia Kuzenkova, Anna Deripaska, Timofey Bryksin [и др.] // ENASE 2013 — Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering. Setubal, Portugal: SciTePress, 2013. С. 205–211.

[39] Поддержка жестов мышью в мета-CASE-системах / М.С. Осечкина, Т.А. Брыксин, Ю.В. Литвинов [и др.] // Системное программирование. СПб., 2010. № 5. С. 52–75.

[40] 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. Perm: ISPRAS, 2012. С. 194–200.

[41] Brooks R. A robust layered control system for a mobile robot // IEEE J. Robot. Automat. 1986. Т. 2, № 1. С. 14–23.

[42] Connell Jonathan H. A colony architecture for an artificial creature: Tech. Rep.: : DTIC Document, 1989.

[43] Arkin Ronald C. Motor schema based navigation for a mobile robot: An approach to programming by behavior // Robotics and Automation. Proceedings. 1987 IEEE International Conference on / IEEE. Т. 4. 1987. С. 264–271.

[44] Rosenblatt Julio K. DAMN: A distributed architecture for mobile navigation // Journal of Experimental & Theoretical Artificial Intelligence. 1997. Т. 9, № 2-3. С. 339–360.

[45] Simpson Jonathan, Ritson Carl G. Toward Process Architectures for Behavioural Robotics. // CPA. 2009. С. 375–386.

[46] Banyasad Omid. A Visual Programming Environment for Autonomous Robots: Master's thesis: DalTech, Dalhousie University. Halifax, Nova Scotia, 2000.

[47] Терехов Андрей Николаевич. Отечественные инструментальные средства обучения программированию // Информационные технологии для Новой школы. Мат-лы VI Ме. С. 64.

[48] Muchnick Steven S. Advanced compiler design implementation. Morgan Kaufmann, 1997.

[49] Деревенец Егор Олегович, Трошина Екатерина Николаевна. Структурный анализ в задаче декомпиляции // Прикладная информатика. 2009. № 4.

[50] Мордвинов Дмитрий Александрович, Литвинов Юрий Викторович. Обзор применения формальных методов в робототехнике // Научно-технические ведомости СПбГПУ Информатика. Телекоммуникации. Управление. 2016. Т. 2016. 1236. С. 84–107.

[51] Kress-Gazit Hadas, Fainekos Georgios E, Pappas George J. Where's Waldo? Sensor-based temporal logic motion planning // 2007 IEEE International Conference on Robotics and Automation (ICRA) / IEEE. 2007. С. 3116–3121.

[52] Бугайченко Дмитрий Юрьевич. Разработка и реализация методов формально-логической спецификации самонастраивающихся мультиагентных систем с временными ограничениями // Сайт математико-механического факультета СпбГУ.–Санкт-Петербургский государственный университет. 2007. Т. 2010.

[53] Дмитриев АС. Адаптация комбинированной пространственно-темпоральной логики для обработки в системах машинного обучения // Известия Волгоградского Государственного Технологического Университета. 2013. Т. 16, № 8 (111).

[54] Мордвинов Дмитрий Александрович, Литвинов Юрий Викторович. Тестирование среды программирования роботов // Инструменты и методы анализа программ, материалы международной научно-практической конференции 12-14 ноября 2015 года. СПб.: Издательство Политехнического университета, 2015. С. 176–185.

[55] ROS: an open-source Robot Operating System / Morgan Quigley, Ken Conley, Brian Gerkey [и др.] // ICRA workshop on open source software. Т. 3. 2009. С. 5.