Санкт-Петербургский государственный университет
Математическое обеспечение и администрирование информационных систем

Системное программирование

Зимин Григорий Александрович

Создание языка программирования роботов в терминах потоков данных с применением DSM-подхода

Бакалаврская работа

Научный руководитель:
д. ф.-м. н., профессор Терехов А. Н.

Рецензент:
асс. каф. КСиПТ СПбПУ Беляев М. А.

Санкт-Петербург
2016

SAINT-PETERSBURG STATE UNIVERSITY
Information Systems Administration and Mathematical Support

Software Engineering

Grigorii Zimin

Development of dataflow robotics programming language using DSM-approach

Bachelor's Thesis

Scientific supervisor:
professor Andrey Terekhov

Reviewer:
assistant Mihail Belyaev

Saint-Petersburg
2016

Оглавление

Введение

В настоящее время интерес к конструированию роботов и управлению ими растет. В этой области проводится множество исследований: на крупнейших робототехнических конференциях, таких как IROS[1], ICRA[2], исследовательские группы со всего мира обсуждают новые подходы к решению различных проблем в робототехнике. На протяжении последних трех десятилетий проводится исследование возможностей применения *визуальных языков программирования* (*visual programming languages, VPLs*), результаты исследований публикуются на крупнейших конференциях, таких как симпозиум VL/HCC[3]. Пересечение этих областей образует отдельную активно развивающуюся область с множеством публикуемых работ, таких как [1, 2, 3, 4, 5]. Визуальные языки программирования в робототехнике позволяют сокращать время создания систем управления роботами, а также нагляднее их отображать. Это, в частности, используется для обучения школьников или новичков программированию роботов: есть множество сред, таких как ROBOLAB[4], NXT-G[5], TRIK Studio[6], которые позволяют программировать поведение робота с помощью *модельно-ориентированного подхода* (*model driven architecture, MDA*): для описания программы используется набор моделей, в данном случае визуальных, имитирующих высокоуровневые паттерны поведения робота.

Система управления роботом представляет собой взаимодействие трех составляющих: датчики и сенсоры, логика системы управления, приводы. Датчики и сенсоры генерируют данные, логика системы управления собирает значения, обрабатывает их и генерирует импульсы для приводов. Получается, что по своей природе программы управления роботами реактивны: они обрабатывают сигналы, непрерывно приходящие с датчиков и сенсоров, и генерируют управляющую информацию для приводов, то есть они решают задачу трансформации данных. Для программирования таких систем хорошо

[1] http://www.iros2016.org/ [Дата обращения: 22 мая 2016 г.]

[2] http://www.icra2016.org/ [Дата обращения: 22 мая 2016 г.]

[3] https://sites.google.com/site/vlhcc2016/ [Дата обращения: 22 мая 2016 г.]

[4] http://www.legoengineering.com/program/robolab/ [Дата обращения: 22 мая 2016 г.]

[5] http://www.legoengineering.com/program/nxt-g/ [Дата обращения: 22 мая 2016 г.]

[6] http://www.trikset.com/ [Дата обращения: 22 мая 2016 г.]

подходят *потоковые* или *реактивные языки программирования*, они же — *языки программирования потоков данных* (*data flow languages, DFLs*). Данные языки, в свою очередь, тоже активно эволюционировали от текстовых языков к визуальным языкам потоков данных, которые сейчас широко распространены [6, 7]. Визуальные потоковые языки превосходят текстовые хотя бы тем, что при программировании потоков данных они явно отображают потоки данных на диаграмме. В индустрии программирования роботов существует несколько широко распространенных, крупных и довольно-таки сложных сред программирования, которые позволяют программировать на потоковых языках, к примеру, Simulink[7], LabVIEW[8], Microsoft Robotics Developer Studio[9]. Эти среды предоставляют пользователю большой и даже порой громоздкий набор средств и библиотек для программирования различных роботов.

Для обучения кибернетике и робототехнике существует большое количество различных кибернетических конструкторов, к примеру, конструктор TRIK[10], конструкторы LEGO MINDSTORMS[11]. Подавляющее большинство распространенных и общеизвестных языков программирования, которые используются для обучения программированию на таких конструкторах, основаны на модели исполнения программы в модели потока управления, в то время как индустриальные среды используют языки, которые основаны на модели потока данных. В то же время, при освоении учебных языков зачастую возникает ощущение неудобства их использования для решения различных типовых задач создания систем управления роботом.

Несмотря на это, проводятся попытки адаптации парадигмы языков потоков данных к образованию, например, в работах [3, 5, 8]. Однако, учебных сред программирования роботов, позволяющих создавать системы управления на потоковых языках, и в то же время доступных для бесплатного академического использования либо нет, либо они находятся на стадии разработок.

[7] http://www.mathworks.com/products/simulink/ [Дата обращения: 22 мая 2016 г.]

[8] http://www.ni.com/labview/ [Дата обращения: 22 мая 2016 г.]

[9] https://www.microsoft.com/en-us/download/details.aspx?id=29081/ [Дата обращения: 22 мая 2016 г.]

[10] http://blog.trikset.com/p/blog-page_6355.html/ [Дата обращения: 22 мая 2016 г.]

[11] http://www.lego.com/en-us/mindstorms/products/ [Дата обращения: 22 мая 2016 г.]

Постановка задачи

Задачей данной выпускной квалификационной работы является создание языка программирования роботов в терминах потоков данных. Основная направленность языка — это программирование учебных робототехнических конструкторов, таких как TRIK, LEGO NXT, LEGO EV3. Язык должен быть достаточно простым в освоении, но в то же время обладать достаточными функциональными возможностями, чтобы создавать сложные системы управления, контролирующие различные аспекты поведения робота. По сути, он должен сочетать в себе простоту учебных языков и хотя бы частично обладать возможностями языков, используемых в индустрии. Для использования языка необходимо создать инструментарий для его поддержки, а именно редактор визуального языка и интерпретатор программ, созданных в терминах потоков данных на новом языке. Интерпретация должна осуществляться на симуляционной модели, а также непосредственно на реальном роботе. Еще одной задачей данной квалификационной работы является апробация нового языка и инструментария, созданного для него, для программирования типичных учебных задач в робототехнике, а также для более сложных систем управления роботом.

1. Среды и языки программирования роботов

В данной главе рассматриваются различные среды программирования роботов: промышленные (предоставляют обширный инструментарий для создания различных моделей и систем управления роботами), образовательные (обычно позволяют программировать небольших роботов), а также некоторые академические среды, созданные в рамках исследовательских работ для демонстрации каких-либо интересных идей.

1.1. Промышленные инструменты

Индустриальные среды программирования роботов позволяют решать широкий круг задач: создание различных динамических моделей, анализ, проектирование инструментов и т.д. Они предоставляют большой набор средств для этого, но довольно сложны для освоения. Стоит отметить, что в них чаще используется модель потока данных.

LabVIEW

LabVIEW [9] — графическая среда разработки программного обеспечения на языке G, созданная компанией National Instruments в 1986 году. LabVIEW позволяет быстро создавать приложения для задач управления, тестирования, измерения и множества других (см. рис. 1). Данная среда позволяет программировать в терминах потоков данных и позволяет использовать различные шаблоны проектирования для создания приложений, к примеру, есть возможность применить архитектуру конечного автомата.

LabVIEW поддерживает большое количество аппаратных платформ, предоставляет огромный набор библиотек, которые содержат средства для работы со сложными математическими конструкциями, средства для создания виртуальных инструментов, алгоритмы компьютерного зрения и т.д. Для взаимодействия блоков библиотеки предоставляют набор различных связей, которые отличаются типом передаваемых через них данных. Созданная в среде LabVIEW программа — это виртуальный прибор (virtual instrument), она делится на две части: блочная диаграмма, описывающая логику виртуального

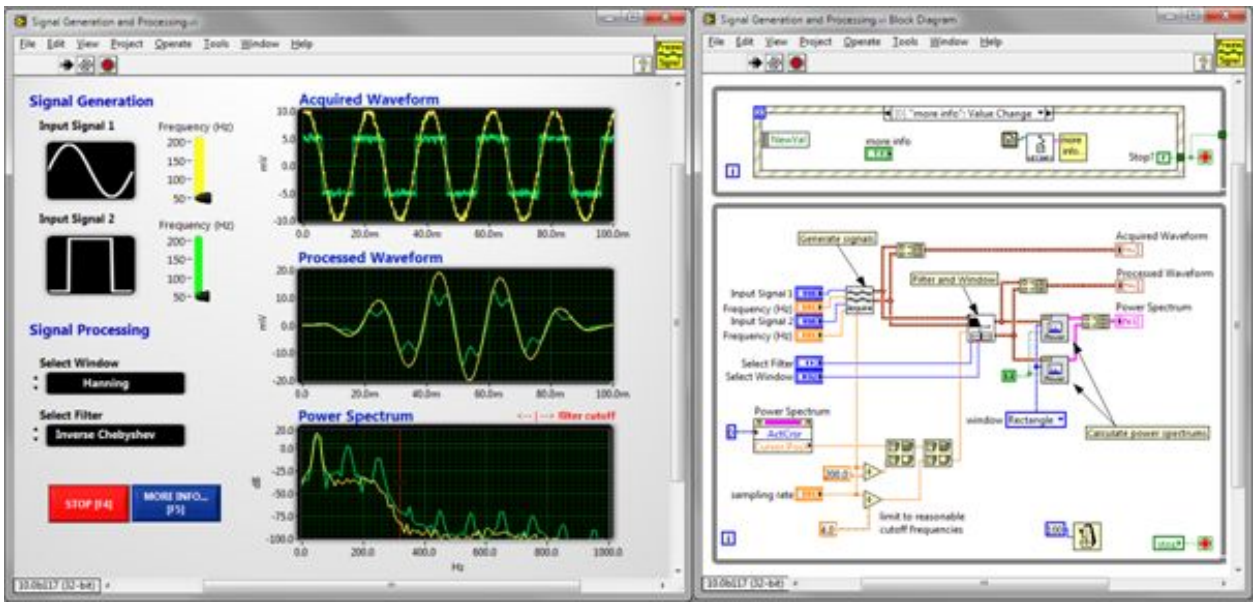


Рис. 1: Пример программы в среде LabVIEW (изображение взято с ресурса http://www.instructables.com/id/LabVIEW-Tips-Tricks-and-Resources/)

прибора, и лицевая панель, которая описывает интерфейс прибора. Важно отметить, что компилятор языка автоматически распараллеливает участки кода, имеющие параллельно расположенные блоки, создавая для их выполнения отдельные потоки.

Возможности применения данной среды обширны, есть компоненты[12], позволяющие применить данную среду в образовательных целях. Они предоставляют инструменты для работы с учебными робототехническим конструкторам LEGO MINDSTORMS NXT/EV3. LabVIEW позволяет интерпретировать программы и генерировать по ним код для автономного запуска программ на устройстве. Существуют примеры попыток применения данной среды в образовательных целях [10], но отмечается, что слишком много времени тратится на освоение самой среды (модули для работы с роботами содержат около четырехсот блоков).

Simulink

Simulink — это графическая среда программирования и моделирования (см. рис. 2), использующая блок-диаграммы. Среда создана компанией MathWorks. Принципы ее работы схожи с LabVIEW. Simulink позволяет моделировать

[12]http://sine.ni.com/nips/cds/view/p/lang/ru/nid/212785/ [Дата обращения: 22 мая 2016 г.]

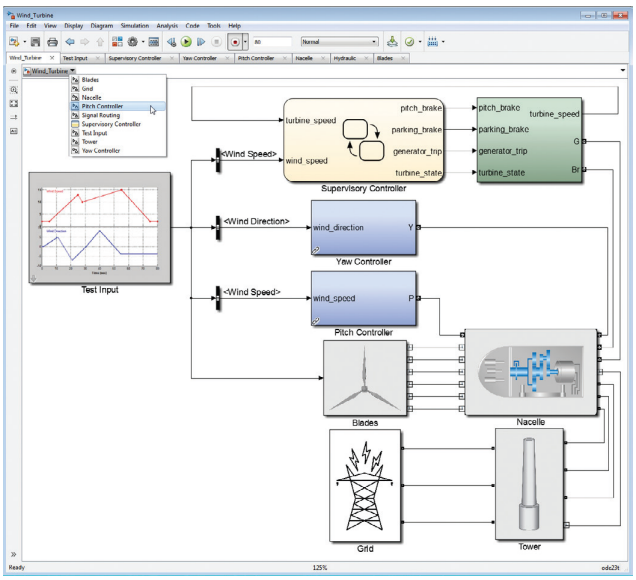


Рис. 2: Пример программы в среде Simulink (изображение взято с ресурса http://matlab.ru/products/simulink/)

различные динамические модели, проводить симуляцию и автоматическую кодогенерацию, тестирование и верификацию. Предоставляет множество библиотек с различными блоками, позволяет взаимодействовать с пакетом MATLAB, используя алгоритмы в моделях и экспортируя результаты моделирования для дальнейшего анализа. С помощью инструмента Robotics System Toolbox[13] среда Simulink имеет возможность разрабатывать программы управления для автономных роботов.

Среда предоставляет обширный набор библиотек, содержащих различные блоки (около двухсот) для верификации, взаимодействия с датчиками и другими устройствами робота, для работы с математическими операциями и другие. Также как и LabVIEW, Simulink основан на модели потоков данных, что лучше подходит для программирования роботов в силу реактивности их природы. Есть библиотеки для поддержки роботов LEGO EV3 и NXT[14], которые позволяют осуществить взаимодействие MatLab и Simulink и учебных робототехнических конструкторов.

[13]http://www.mathworks.com/products/robotics/?requestedDomain=www.mathworks.com [Дата обращения: 22 мая 2016 г.]

[14]http://www.mathworks.com/hardware-support/lego-mindstorms-ev3-simulink.html [Дата обращения: 22 мая 2016 г.]

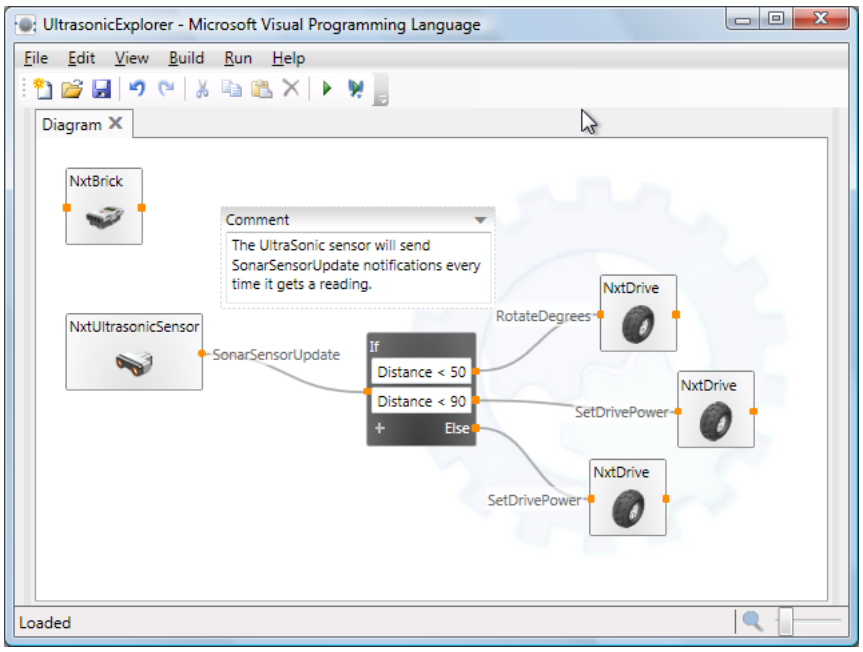


Рис. 3: Пример программы в среде Microsoft Robotics Developer Studio (изображение взято с ресурса thttps://msdn.microsoft.com/ru-ru/library/bb483027.aspx/)

Microsoft Robotics Developer Studio

Еще один пример промышленной системы — среда Microsoft Robotics Developer Studio (MRDS) (см. рис. 3) [11]. Она предназначена для визуального программирования распределенных робототехнических систем путем создания программ в терминах потоков данных. При выполнении программы система преобразует диаграмму в набор веб-сервисов, которые могут частично выполняться на роботе, а частично на пользовательской машине. Связи между блоками соответствуют взаимодействию между веб-сервисами. Таким образом, программа представляет собой набор независимо исполняемых параллельных компонент.

MRDS позволяет взаимодействовать с конструктором LEGO NXT[15]. Связь с конструктором осуществляется только удаленно по каналу Bluetooth, однако отсутствует возможность запустить выполнение программы на роботе автономно. Стоит отметить, что MRDS предоставляет возможность для взаимодействия с пользовательскими робототехническими платформами, но компания Microsoft прекратила поддержку среды с 2014 года.

[15]https://msdn.microsoft.com/ru-ru/library/bb483027.aspx

1.2. Учебные инструменты

Учебные среды программирования позволяют решать типовые задачи управления роботом: езда по линии, прохождение лабиринта и т.д. С их помощью можно достаточно легко создавать «примитивные» системы управления. Они предназначены для обучения основам управления роботами и их программирования.

LEGO MINDSTORMS Education NXT-G/EV3 software

LEGO MINDSTORMS Education NXT-G [12] — среда программирования для конструктора LEGO Mindstorms NXT. Среда основана на индустриальной среде LabVIEW и использует язык потоков данных G. NXT-G автоматически размещает блоки на диаграмме: исполнение подчиняется порядку блоков, необходимые следующим блокам данные нужно явно связать потоком. Среда предоставляет довольно-таки большой набор блоков (сто девяносто три). В NXT-G практически отсутствует поддержка математических выражений: для задания сложных выражений приходится строить блоками дерево разбора. Плюсом среды является то, что она распространяется бесплатно.

LEGO MINDSTORMS Education EV3 software[16] для конструктора LEGO Mindstorms EV3 решает часть проблем среды NXT-G, таких как задание математических формул. Среда поддерживает программирование конструктора LEGO NXT (хотя есть известные проблемы совместимости). EV3 software предоставляет пользователю небольшой набор блоков для программирования, исполнение подчиняется явно заданному потоку управления, где частично используется модель передачи данных (см. рис. 4). В языке, предоставляемом средой, используется пятьдесят три различных блока, которые отвечают за управление различными сенсорами, датчиками, приводами, кнопками контроллера, за реализацию математических функций, а также алгоритмических конструкций: развилка, цикл, выбор (switch) и т.д. Среда поддерживает не все ОС (отсутствует, например, поддержка Linux).

[16] http://www.lego.com/ru-ru/mindstorms/downloads/download-software/ [Дата обращения: 22 мая 2016 г.]

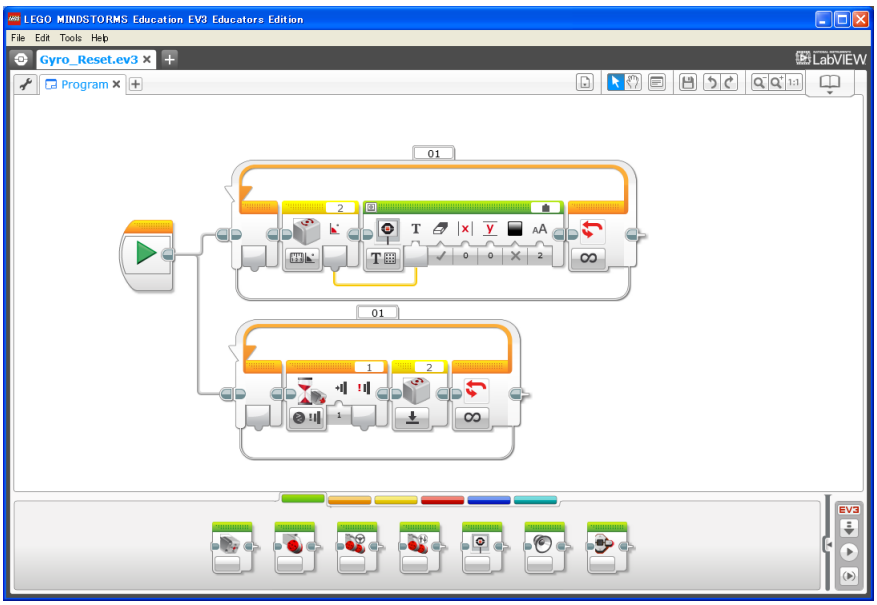


Рис. 4: Пример программы в среде EV3 Software (изображение взято с ресурса http://www.afrel.co.jp/en/archives/841/)

Robolab

Robolab [13] — это еще одна учебная среда для программирования роботов (см. рис. 5). Среда позволяет программировать несколько видов микроконтроллеров — LEGO NXT, LEGO Control Lab, LEGO RCX. Она является упрощенной версией промышленной среды программирования LabView. Среда использует визуальный язык, который в общей сложности насчитывает порядка четырехсот блоков. Чтобы не пугать начинающего пользователя громоздкой палитрой, в среде есть возможность выбора уровня использования программы. Уровни ограничивают размер используемой палитры, первый уровень, к примеру, содержит порядка двадцати элементов и позволяет только подставить блок в отведенное для него пустое место. На последнем уровне пользователю доступна вся палитра (размещение блоков никак не ограничивается). Палитра включает в себя блоки управления, блоки различных арифметических действий (математические выражения можно задать текстом явно на языке C), блоки переменных, подпрограмм, работы с потоками исполнения (распараллеливание исполнения), циклы (реализованы с помощью меток и переходов). Среда обладает устаревшим пользовательским интерфейсом. Передача управления осуществляется, также как и в среде LEGO MINDSTORMS Education EV3 Software. Блоки в Robolab окутаны сетью различных «прово-

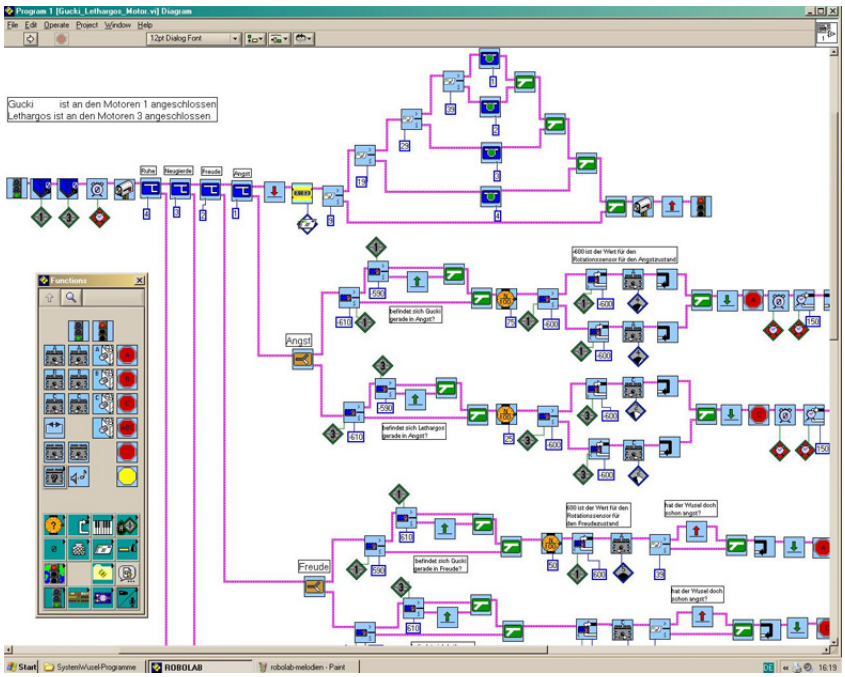


Рис. 5: Пример программы в среде Robolab (изображение взято с ресурса http://netzspannung.org/learning/artdecom/systemwusel-technology/)

дов», по ним приходят различные модификаторы, разным типам данных соответствуют разные цвета проводов. Это затрудняет понимание при работе с большой программой. Еще одним недостатком является то, что блоки для взаимодействия с разными конструкторами никак не разделены. Они перемешаны, но не все команды, к примеру, для LEGO RCX могут быть использованы с роботом LEGO NXT.

TRIK Studio

Еще один пример учебной среды программирования — среда программирования роботов TRIK Studio (см. рис. 6). Она позволяет программировать несколько видов микроконтроллеров — LEGO NXT, LEGO EV3, TRIK — с помощью последовательности пиктограмм. Всего в языке около ста различных блоков, отвечающих за взаимодействие с роботом и алгоритмическую и математическую поддержку. Среда имеет современный пользовательский интерфейс. Для удобства программирования блоки в палитре разделены на группы по функциональному значению. Язык программирования в TRIK Studio полностью основан на модели потока управления, потоки данных не используются. Стоит отменить, что в отличие от предыдущих рассмотренных

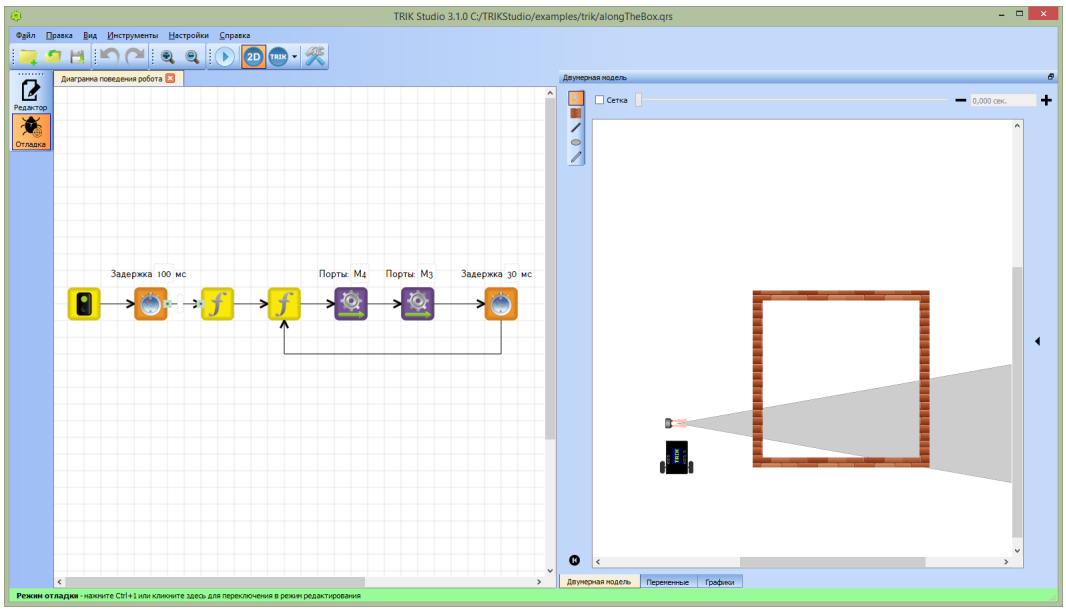


Рис. 6: Пример программы в среде TRIK Studio (изображение взято с ресурса http://blog.trikset.com/)

сред, исходный код среды находится в открытом доступе.

1.3. Академические инструменты

Создание потоковых языков программирования для робототехники — активно развивающаяся область.

Например, в диссертации [1] был предложен визуальный язык программирования в терминах потоков данных, основанный на математическом формализме — расширенной машине Мура. Прототип среды для демонстрации идей работы крайне неудобен в использовании.

Авторы данных работ [3, 4] описывают визуальный потоковый язык программирования для обучения универсантов параллельному программированию, инструмент привязан к языку программирования $occam$-π и инструментарию $Transterpreter$, вторая работа описывает применение архитектуры категорий (см. 2) и их языка к управлению «роем» роботов. Также авторы в своих работах говорят о важности программирования роботов в терминах потоков данных и о пользе применения категориальной архитектуры Р.Брукса. Созданная в рамках работы среда программирования, скорее является макетом, основная работа проводится на языке $occam$-π.

Еще одна работа [5], предложенная на симпозиуме VL/HCC, описывает

визуальный язык RuRu в терминах потоков данных и среду с интуитивно-понятным интерфейсом для обучения программированию роботов. Описанная в работе среда предоставляет простой в освоении интерфейс, но сама технология не обладает достаточными функциональными возможностями и требует доработки. Более того, среда недоступна для конечных пользователей.

1.4. Выводы

После рассмотрения всех перечисленных выше инструментов становится ясно, что учебные среды визуального программирования роботов обычно представляют собой небольшой набор блоков, с помощью которого можно создавать программы для решения типовых учебных задач, используя при этом простую для понимания модель исполнения — модель потока управления (возможно с частичным использованием модели потока данных). Индустриальные же среды, предоставляют куда больший набор блоков и обширный набор средств для программирования и моделирования различных устройств. Они, в основном, основаны на модели исполнения в терминах потоков данных, где полезная работа блока осуществляется только тогда, когда на него приходят данные. Хотя имеется ряд примеров применения этих сред для программирования робототехнических учебных конструкторов, они все еще кажутся непригодными для целей обучения: они сложны, большинство времени тратится на освоение самой системы. Академические же инструменты, созданные в рамках научных исследований, ненадежны, подчас неудобны для обычного пользователя и, более того, часто недоступны для использования.

Также при обзоре было отмечено, что создание нового языка программирования роботов и удобного в использовании инструментария для него «с нуля» — это задача, которая потребовала бы большого числа человеко-лет разработки. Поэтому во время рассмотрения сред искалась среда, которую можно использовать как основу для нового визуального языка. Было отмечено, что среда программирования роботов TRIK Studio, созданная на кафедре Системного программирования СПбГУ, является средой программирования роботов с открытым исходным кодом, которая предоставляет воз-

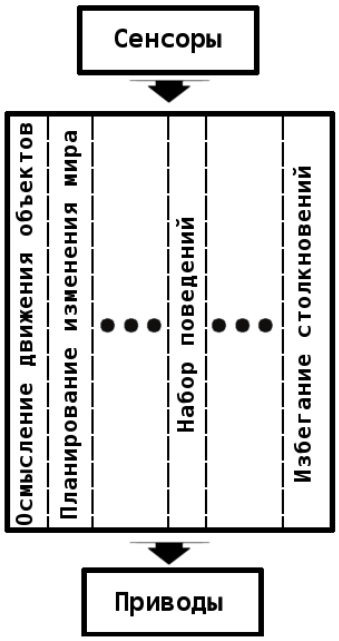
можность расширения себя новым потоковым визуальным языком программирования роботов, и позволяет использовать кодовую базу операций, отвечающих за взаимодействие с различными роботами.

2. Архитектуры построения систем управления роботами

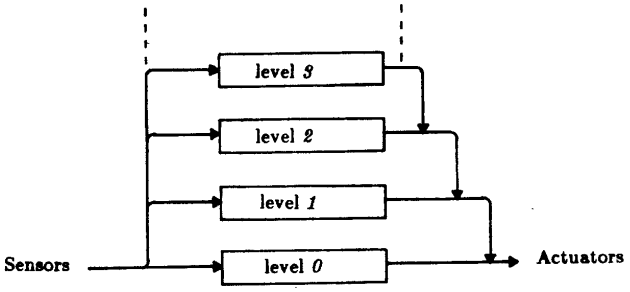
Задача создания сложных и масштабируемых систем управления роботами весьма нетривиальна. Начиная с середины 80-х годов множество исследователей пытались решить эту задачу и предоставляли свои решения [14]. Часть этих подходов быстро стала популярной в робототехнике, некоторые из них актуальны до сих пор. Например, работа представленная Родни Бруксом, рассказывающая об архитектуре категорий (subsumption architecture) [15], является одной из наиболее цитируемых работ в области робототехники. Поэтому, одним из требований, которым должен удовлетворять разрабатываемый язык, является возможность выразить, по крайней мере некоторые идеи популярных архитектур для построения систем управления роботами.

Брукс предложил делить систему управления по уровням ответственности (см. рис. 7a). Каждый уровень непосредственно взаимодействует с датчиками и сенсорами, а также с приводами робота. На основании такого разделения им была предложена архитектура категорий (см. рис. 7b). Уровни в такой архитектуре образуют иерархию уровней ответственности, каждый уровень отвечает за новое поведение робота. Верхние уровни могут влиять на работу нижних, но не наоборот, таким образом отказы в работе верхних уровней не влияют на нижние. Это особенно важно в робототехнике, к примеру, отказ работы уровня, отвечающего за хват робота, все еще позволит роботу вернуться на станцию. Такая архитектура легко масштабируема. Взаимодействие верхних и нижних уровней осуществляется за счет механизмов *замещения* (*suppression*) и *подавления* (*inhibition*) данных. С помощью них верхние уровни корректируют поведение, генерируемое нижними.

Другая популярная архитектура — «Колония» Джонотана Коннеля [16]. Эта модель похожа на архитектуру категорий Р. Брукса, но решает некоторые проблемы ее масштабирования (см. рис. 8). В ней система также разделяется на набор взаимодействующих, параллельно работающих уровней, которые, однако, не требуют явной упорядоченности. Еще одним отличием является то, что здесь не используется подавление (inhibition) потока данных. Подавление должно быть реализовано путем различных предикатов на более



(a) Модель декомпозиции системы управления Р. Брукса



(b) Архитектура категорий Р. Брукса

Рис. 7: Идеи Р. Брукса

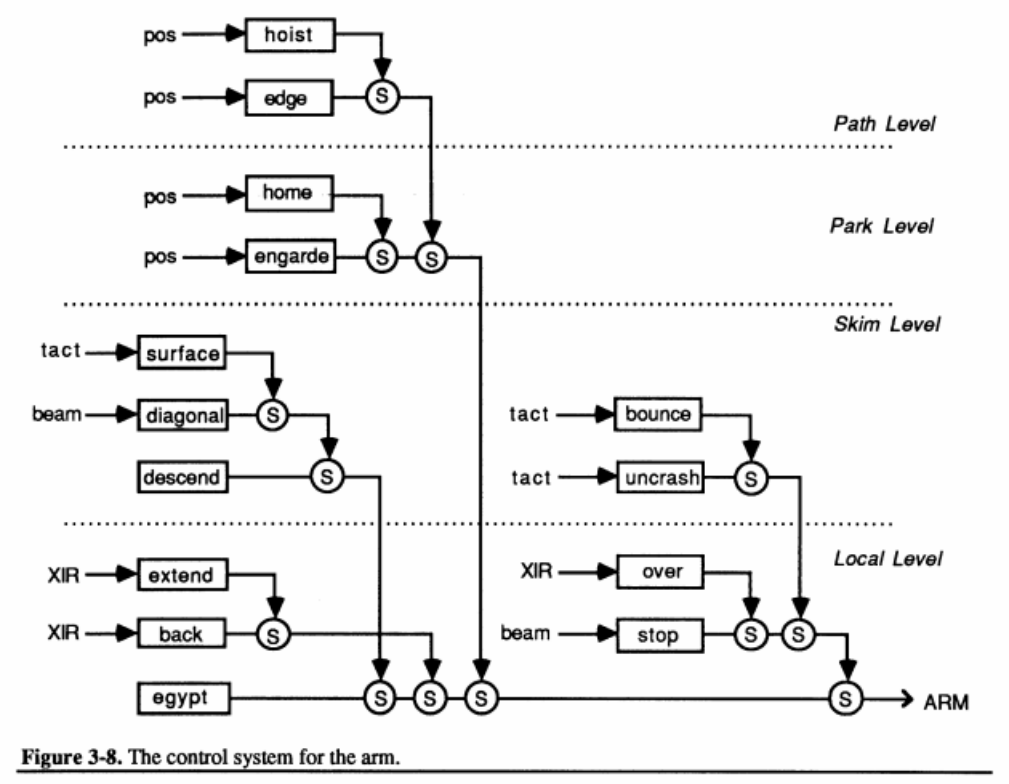


Figure 3-8. The control system for the arm.

Рис. 8: Пример решения задачи в архитектуре Д. Коннеля (рисунок взят из работы автора)

низких уровнях. Средства поддержки этих двух архитектур в языке коротко упомянуты в разделе 3.1.

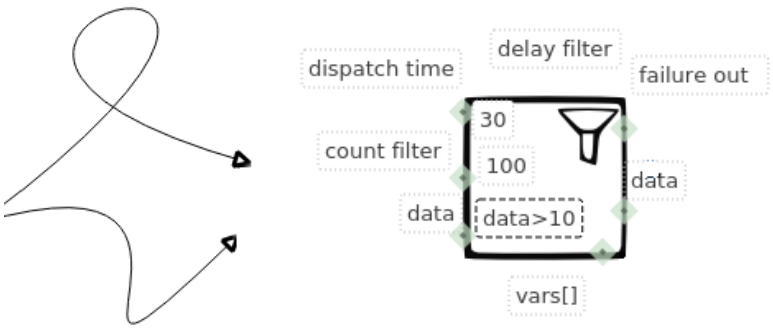
Существуют и другие решения (архитектуры) для построения систем управления роботами, которые могут быть выражены средствами созданного языка, к примеру, «Схема мотора» («Motor Schema») Рональда Аркина [17], «Распределенная архитектура» («distributed architecture») Джулио Розенблата [18]. Подробное рассмотрение возможности выражения средствами созданного языка главных идей для каждой из существующих архитектур, основанной на взаимодействии поведений, скорее является темой отдельной работы.

3. Реализация

Глава содержит детальное описание инструмента, полученного в рамках выполнения данной выпускной квалификационной работы. Первая часть описывает визуальный язык, а вторая — реализацию средств его инструментарной поддержки.

3.1. Спецификация языка

Предложенный визуальный язык является графовым: его элементы делятся на два класса — вершины (*сущности*, они же *блоки*) и *связи*. Блоки представляют собой «черные ящики», которые принимают, обрабатывают и генерируют данные (см. рис. 9b). Связи, соединяющие блоки, представляют собой однонаправленные каналы для передачи данных между ними (см. рис. 9a).



(a) Связь (b) Блок

Рис. 9: Элементы языка

На графических представлениях блоков слева и снизу могут быть входные порты для данных, справа — выходные порты. Для совместимости с учебными языками, полностью или частично основанными на модели потока управления, часть блоков поддерживает явную передачу управления. Для этого на графическом представлении блоков сверху могут быть отдельные порты для передачи контроля управления, например см. рис. 10a. Также в графических представлениях блоков может быть одно или несколько текстовых полей, в которых пользователь может писать выражения на статически типизированном диалекте Lua [19], чья поддержка «унаследована» от среды

TRIK Studio.

Ниже представлены все блоки, используемые в языке, они разделены на несколько групп. Для каждого блока дано описание его работы.

3.1.1. Блоки управления

Блоки «управления» (см. рис. 10) реализуют некоторые базовые алгоритмические конструкции и полезные алгоритмические функции.

К примеру, блок «Константа» (см. рис. 10a) при приходе на него любых данных генерирует значение, установленное пользователем.

Блок «Завершение исполнения» (см. рис. 10b) останавливает выполнение программы, в каком бы состоянии ни находился робот.

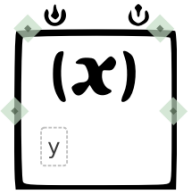
Блок «Глобальная переменная» (см. рис. 10c) позволяет пользователю устанавливать значения глобальных переменных. Его поведение разнится в зависимости от того, на какой порт пришли данные и от того, что ввел пользователь в текстовое поле: если в текстовом поле указана переменная, и она была до этого определена, то при приходе любых данных на верхний левый порт блок вырабатывает значение этой переменной на выходной правый порт, при приходе данных на входной порт слева, переменная инициализируется этими данными и блок генерирует значение переменной на выходной правый порт. В текстовом поле переменная может быть сразу проинициализирована: `x = 10`; тогда при приходе данных на верхний входной порт генерируется это значение. Исходящая связь из блока, начинающаяся в верхнем выходном порте, передает пустые данные одновременно с тем, как блок генерирует данные на выходной правый порт.

Блок «Распараллеливание» (см. рис. 10d) нужен для явного запуска связанных с ним (его исходящими связями) блоков в новых потоках исполнения (причина добавления этого блока будет объяснена далее).

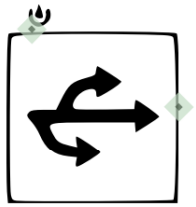
Блок «Условие» (см. рис. 10e) — это реализация условной развилки; если условие на данные, введенное пользователем в текстовом поле с возможным использованием значений переменных, пришедших на нижний входной порт, истинно, то данные будут отправлены на выходной порт `TRUE`, иначе — на порт `FALSE`. Не обязательно, чтобы к каждому порту были присоединены

(a) Константа

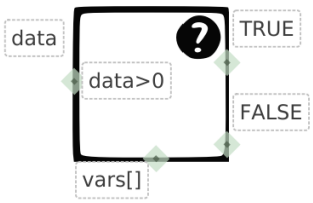
(b) Завершение исполнения



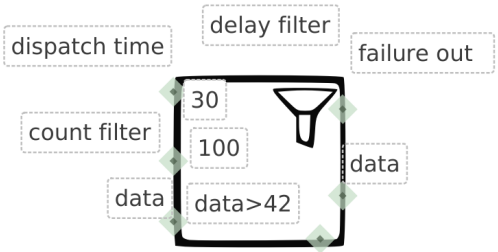
(c) Глобальная переменная



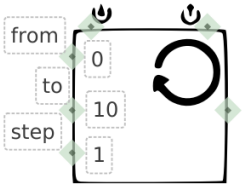
(d) Распараллеливание



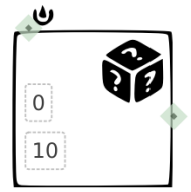
(e) Условие



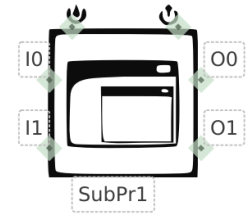
(f) Фильтр



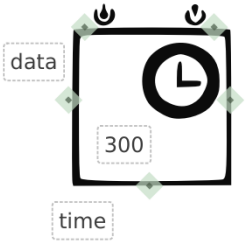
(g) Цикл



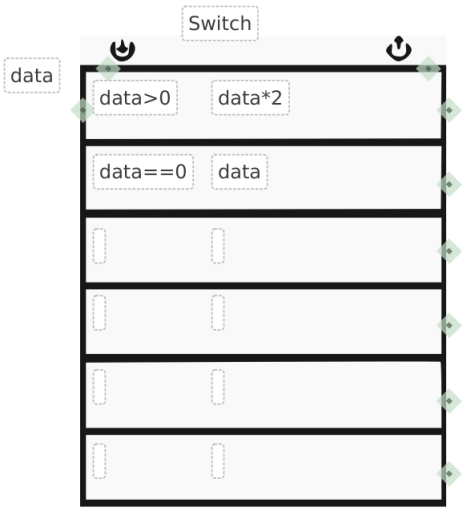
(h) Случайное число



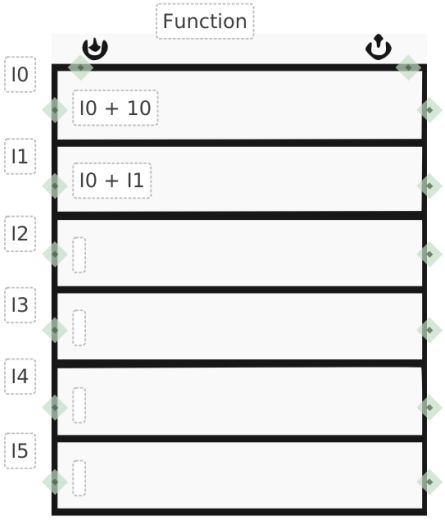
(i) Пользовательский блок



(j) Задержка

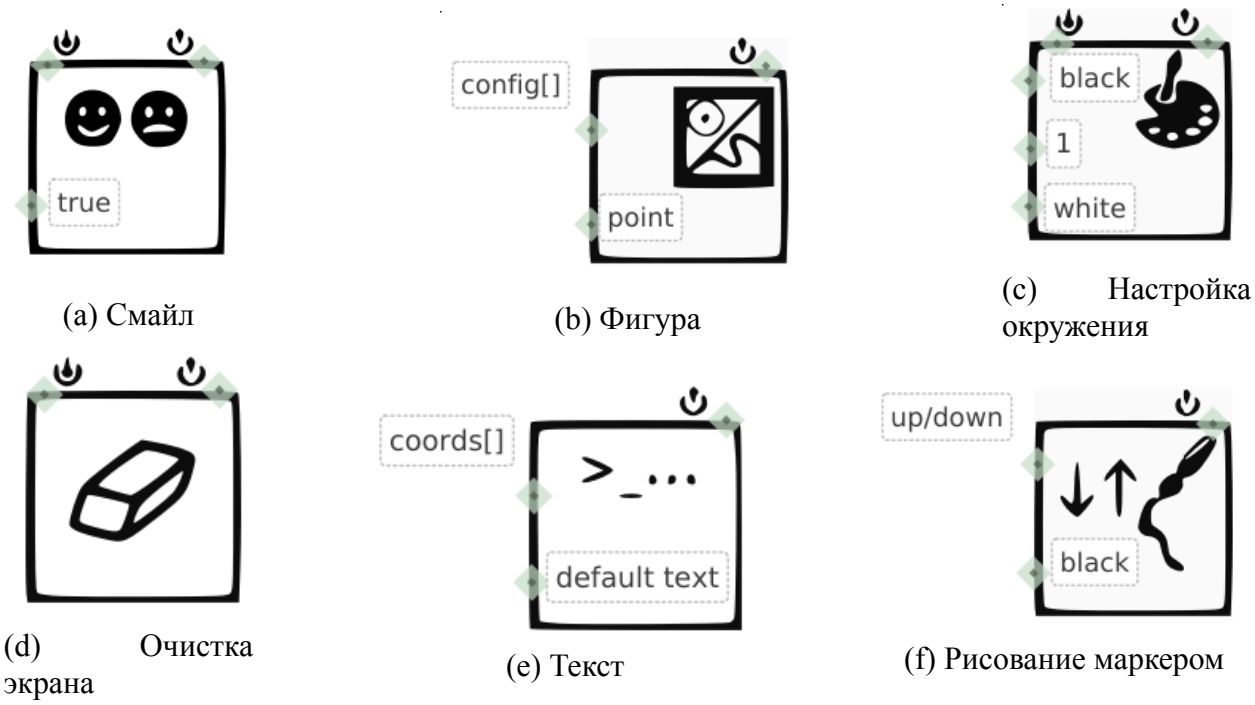


(k) Множественный выбор



(l) Текстовое программирование

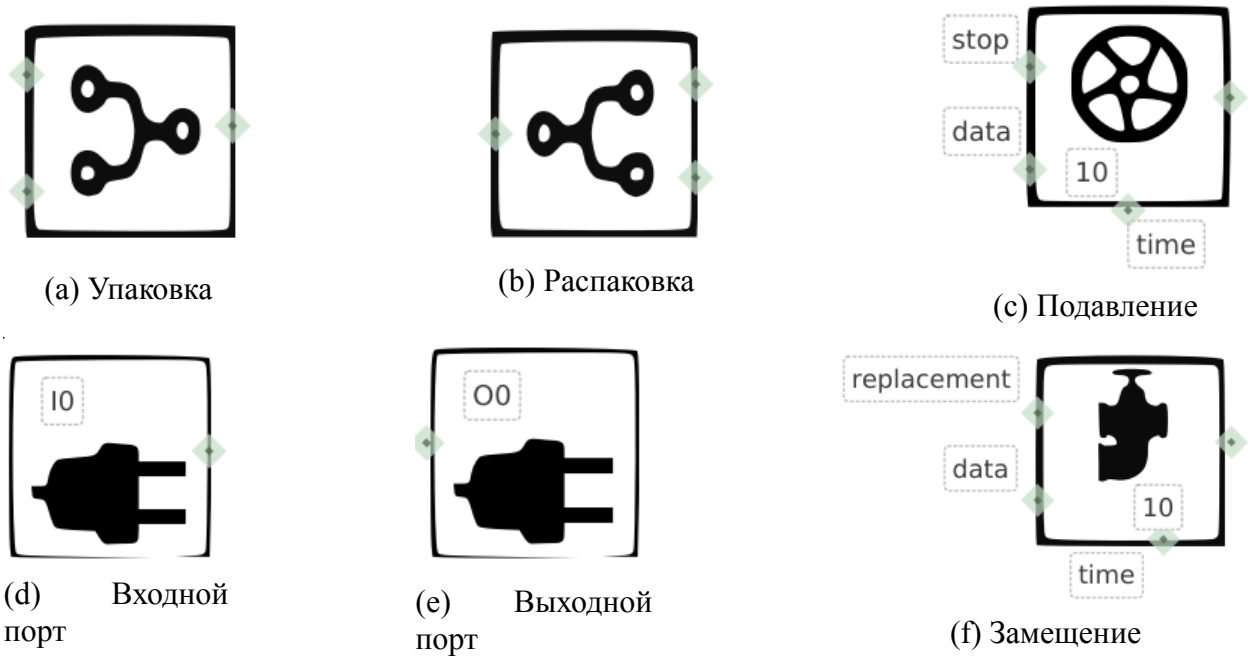
Рис. 10: Группа блоков управления



(a) Смайл (b) Фигура (c) Настройка окружения

(d) Очистка экрана (e) Текст (f) Рисование маркером

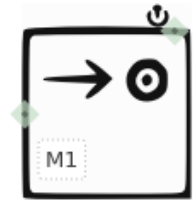
Рис. 11: Группа блоков рисования на экране



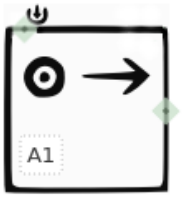
(a) Упаковка (b) Распаковка (c) Подавление

(d) Входной порт (e) Выходной порт (f) Замещение

Рис. 12: Группа блоков управления потоками



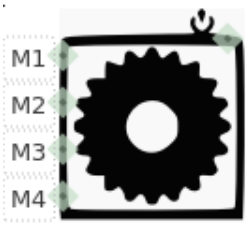
(a) Запись в порт



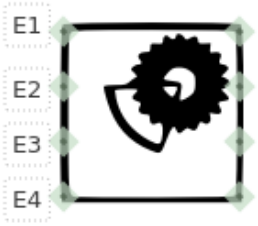
(b) Чтение с порта

(c) Сенсор

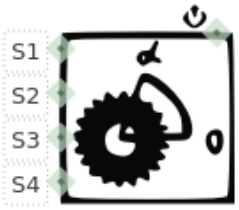
(d) Проиграть музыку



(e) Силовые моторы

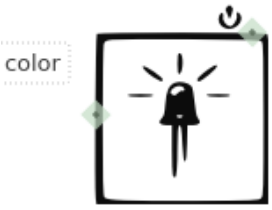


(f) Энкодеры

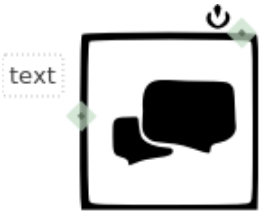


(g) Сервомоторы

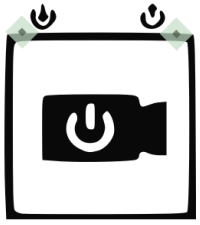
(h) Инициализация камеры



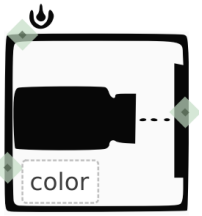
(i) Диод



(j) Синтез речи



(k) Трансляция

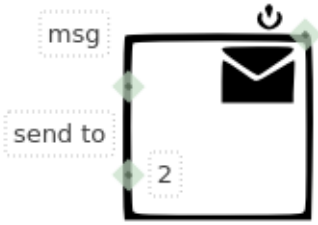


(l) Детектор объектов

(m) Запись в файл

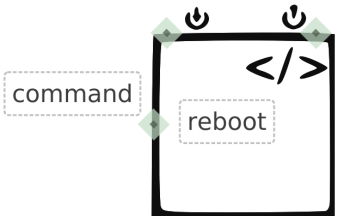
(n) Чтение из файла

(o) Удаление файла

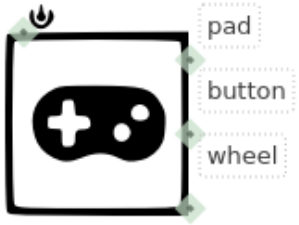


(p) Сообщение роботу

(q) Сообщение от робота



(r) Системный вызов



(s) Операторский контроль

Рис. 13: Группа блоков взаимодействия

связи.

Блок «Задержка» (см. рис. 10j) позволяет пропускать данные с заданной периодичностью, и, в зависимости от настроек пользователя, либо выстраивать приходящие данные в очередь по времени, либо терять приходящие данные между отправками. Пользователь может задать время задержки в текстовом поле. Время также может быть обновлено данными, приходящими на соответствующий порт, и задержка будет высчитываться по новому времени.

Блок «Фильтр» (см. рис. 10f) — объединение предыдущих двух с добавлением счетчика «успешных срабатываний». Если значение счетчика указано как `-1`, то он игнорируется. В противном случае, когда счетчик обнулится, то, даже при истинности проверяемого условия, данные будут отправлены в порт `failure out`.

Блок «Цикл» (см. рис. 10g) последовательно генерирует числа в заданном диапазоне. Значение границ и шага может быть установлено в текстовых полях, а также получено из приходящих на соответствующие порты данных; шаг может быть нулевым, в таком случае цикл будет бесконечным.

Блок «Случайное число» (см. рис. 10h) работает аналогично блоку «Константа» с тем различием, что блок генерирует случайные целочисленные данные в заданных границах.

«Пользовательский блок» позволяет выделить диаграмму в подпрограмму, которая затем может быть переиспользована. В выделяемой диаграмме для связи с входными и выходными портами «Пользовательского блока» используются специальные блоки (см. 3.1.3).

Для реализации множественного выбора (switch), в языке присутствует блок «Множественный выбор» (см. рис. 10k). Его работа состоит в построчной проверке условий. В зависимости от своей настройки блок может сгенерировать значение, соответствующее первому истинному, условию либо всем истинным условиям. Если для проверки условий не нужны приходящие данные, к примеру, если условия заданы на глобальные переменные, установленные пользователем ранее, то блок может проверять значение при приходе данных на верхний входной порт.

В блоке «Текстовое программирование» (см. рис. 10l) пользователь может написать множество выражений, которые будут выполняться сверху вниз

построчно. При этом результаты верхних строк могут быть использованы в строках ниже. Блок допускает два вида поведения (зависит от настроек блока): он может ожидать прихода данных по всем входным портам, имеющим входящие связи, работая как барьер до получения всех данных. Либо каждая строка может работать независимо, но тогда данные приходящие на нее недоступны другим строкам, равно как и значения, полученные при вычислении выражения на строке. Каждая строка может иметь вид `[переменная=выражение>;]<генерируемое выражение>`. Блок вырабатывает `<генерируемое выражение>`.

3.1.2. Блоки рисования на экране

Следующая группа блоков отвечает за растровое и векторное рисование графических примитивов на экране робота (см. рис. 11).

К примеру, блок «Смайл» (см. рис. 11a) рисует на экране робота грустный/веселый смайл в зависимости от установленного пользователем флага (либо полученного значения в виде данных). При приходе любых данных на верхний входной порт на экране робота отображается соответствующий рисунок и генерируются данные на выходной верхний порт.

Блок «Фигура» (см. рис. 11b) позволяет рисовать точки, эллипсы, прямоугольники и кривые в соответствие с заданным типом фигуры и набором необходимых параметров для рисования: координаты, линейные размеры и другие.

Блок «Настройка окружения» (см. рис. 11c) позволяет устанавливать параметры для рисования: цвет и толщину кисти, цвет фона. Эти настройки применяются по одной, когда данные приходят на соответствующие порты, или все разом, если данные пришли на верхний входной порт. При обновлении настроек генерируются данные для выходного порта.

Чтобы удалить все изменения на экране используется блок «Очистка экрана» (см. рис. 11d).

Блок «Текст» (см. рис. 11e) аналогичен блоку «Фигура» с тем отличием, что вместо типа фигуры ожидается текст.

Так как в большинстве обучающих языков программирования роботов

присутствует опция рисования маркером на поверхности (на симуляционных моделях и даже порой на реальных роботах), в язык был добавлен блок «Рисование маркером» (см. рис. 11f). Приняв любые данные, соответствующие истине, робот опустит маркер и поднимет, если данные интерпретируются как ложь.

3.1.3. Блоки управления потоками

Чтобы иметь возможность более «тонко» настраивать работу потоков данных, «циркулирующих» между блоками, в языке присутствует ряд блоков, чья работа заключается в манипуляции данными без их трансформации — блоки «управления потоками» (см. рис. 12).

Блок «Упаковка» (см. рис. 12a) работает как барьер: ждет данные на входные порты, а затем конкатенирует их в кортеж и отправляет дальше.

Блок «Распаковка» (см. рис. 12b) является противоположностью предыдущему блоку, он декомпозирует кортеж данных на части.

Для того, чтобы использовать диаграмму как новый блок в группе «управления», есть «Пользовательский блок». Чтобы связать входные и выходные порты «Пользовательского блока» с данными в пользовательской программе в языке присутствует два блока, соответствующие входным и выходным портам: «Входной порт» (см. рис. 12d) и «Выходной порт» (см. рис. 12e) соответственно.

Для остановки данных в потоке существует блок «Подавление» (см. рис. 12c). В обычном режиме данные, приходящие на нижний левый порт, пропускаются через правый выходной порт дальше. Если приходят любые данные на левый верхний порт, то включается таймер, и в течение установленного времени данные, приходящие на нижний левый порт, теряются. По сути, данный блок «перекрывает» поток данных.

Чтобы не просто «перекрыть» поток данных, а еще и заменить его другим потоком, в языке присутствует блок «Замещение» (см. рис. 12f); его работа аналогична предыдущему, с тем различием, что поток не просто приостанавливается, а еще и заменяется потоком, соединенным с верхним левым входным портом.

«Пользовательский блок», блок «Условие», и блоки «Замещение» и «Подавление» позволяют выражать архитектуру категорий Брукса и архитектуру «Колония», упомянутые в разделе 2.

3.1.4. Блоки взаимодействия

Блоки «взаимодействия» предоставляют доступ к чтению данных с различных устройств робота, а также отвечают за возможность изменять их состояние (см. рис. 13).

Чтобы непрерывно получать изменяющиеся данные с любых сенсоров робота, в языке существует блок «Сенсор» (см. рис. 13c). Как только на входной порт блока придут данные, он подпишется на заданный пользователем сенсор и начнет непрерывно генерировать данные, поступающие с соответствующего датчика робота.

Если в контроллере робота присутствует динамик и соответствующее программное обеспечение, то с помощью блока «Проиграть музыку» (см. рис. 13d) можно проиграть аудио-файл, или с помощью блока «Синтез речи» (см. рис. 13j) сгенерировать и воспроизвести текст, пришедший в виде текстовых данных на входной порт.

Аналогично, если на роботе есть светодиоды, то ими можно управлять с помощью блока «Диод» (см. рис. 13i). Он зажжет светодиод цветом, который будет получен на входном порте.

Для взаимодействия с моторами в языке есть блоки «Моторы» (см. рис. 13e) и «Сервомоторы» (см. рис. 13g). Они работают идентично, отличие лишь в том, что взаимодействуют они с разными моторами робота. У этих блоков есть два варианта поведения: синхронный и асинхронный. В синхронном режиме блок ожидает прихода данных на все подключенные входные порты, в асинхронном — активирует мотор в момент прихода данных на соответствующий порт.

Для работы со счетчиком числа оборотов моторов в языке есть блок «Энкодеры» (см. рис. 13f). Он совмещает в себе блок «Запись в порт» и блок «Сенсор» для каждого порта. Пользователь может записать стартовое значение для каждого энкодера, к примеру, обнулить, и получать обновляющиеся

данные с энкодеров. Чтобы не перегружать каналы данных, пользователь может настроить число оборотов, при увеличении на которое у конкретного энкодера, блок сгенерирует новые данные на соответствующий порт. Этот блок, так же как и предыдущие два, может принимать данные, ожидая как барьер либо независимо для каждого входного порта.

Для работы с файловой системой робота есть три блока: «Запись в файл» (см. рис. 13m), «Чтение из файла» (см. рис. 13n) и «Удаление файла» (см. рис. 13o). Первый записывает приходящий или предустановленный текст в файл с установленным или приходящим на порт именем. Второй читает текст и генерирует строки или слова из файла (в зависимости от настроек) как данные. Третий удаляет файл с указанным именем.

Для поддержки мультиагентного взаимодействия в языке присутствуют два блока: «Сообщение роботу» (см. рис. 13p) и «Сообщение от робота» (см. рис. 13q). Первый блок позволяет роботу отправить текстовое сообщение другому роботу с указанным номером. Второй блок начинает генерировать сообщения, полученные от других роботов.

Если робот позволяет управлять собой с помощью устройств операторского контроля, то для этого в языке есть блок «Операторский контроль» (см. рис. 13s). Блок реагирует на манипуляции оператора джойстиком и генерирует соответствующие данные на порты.

Блок «Системный вызов» (см. рис. 13r) выполняет команду с помощью командного интерпретатора робота (например, bash на роботе TRIK). К примеру, команда `reboot` перезапустит робота.

Блоки «Инициализация камеры» (см. рис. 13h), «Трансляция» (см. рис. 13k) и «Детектор объектов» (см. рис. 13l) позволяют использовать алгоритмы видеозрения для детекции различных объектов. Первый инициализирует камеру в режиме отслеживания `цвета`/`объекта`/`линии`, второй транслирует видео с видеокамеры, третий работает как блок «Сенсор»: после инициализации камеры, он генерирует данные, соответствующие выбранному алгоритму.

Для поддержки низкоуровневой работы с периферийными устройствами робота, в языке есть блоки чтения и записи в отдельные порты робота, к примеру, для робота TRIK порт `M1` отвечает за силовой мотор с первым номером,

а `A1` за аналоговый датчик с первым номером. В данном случае блок «Запись в порт» (см. рис. 13a) позволяет подать импульс на первый силовой мотор, а блок «Чтение с порта» (см. рис. 13b) считать текущее значение с аналогового датчика `A1`.

3.2. Реализация инструментария

Инструментарий для работы с языком представлен двумя средствами. Во-первых, это редактор визуального языка, а во-вторых, интерпретатор программ, созданных на нем. Оба инструмента реализованы как подключаемые модули (плагины) для среды программирования TRIK Studio.

3.2.1. Визуальный редактор

Эволюция *предметно-ориентированного моделирования* (*domain-specific modeling*, *DSM*) позволяет в короткие сроки создавать довольно сложные визуальные языки программирования [20]. TRIK Studio — это пример среды программирования, которая создана с помощью применения данного подхода на базе платформы для предметно-ориентированного моделирования QReal [21, 22] (также как и среда TRIK Studio, она создана на кафедре Системного программирования СПбГУ). С помощью платформы QReal был создан подключаемый модуль, который описывает визуальный язык — содержит метамодель потокового визуального языка, и предоставляет визуальный редактор для системы TRIK Studio. Будучи подключенным к платформе, он предоставляет пользователю все особенности визуального редактора, которые есть у среды TRIK Studio, такие как современный пользовательский интерфейс (см. рис. 14), возможность создавать элементы жестами мышью [23], различные стили связей и т.д. Выбор применения DSM-подхода для разработки редактора визуального языка дает ряд очевидных преимуществ: разработка аналогичного инструмента «с нуля» заняла бы на порядок больше времени.

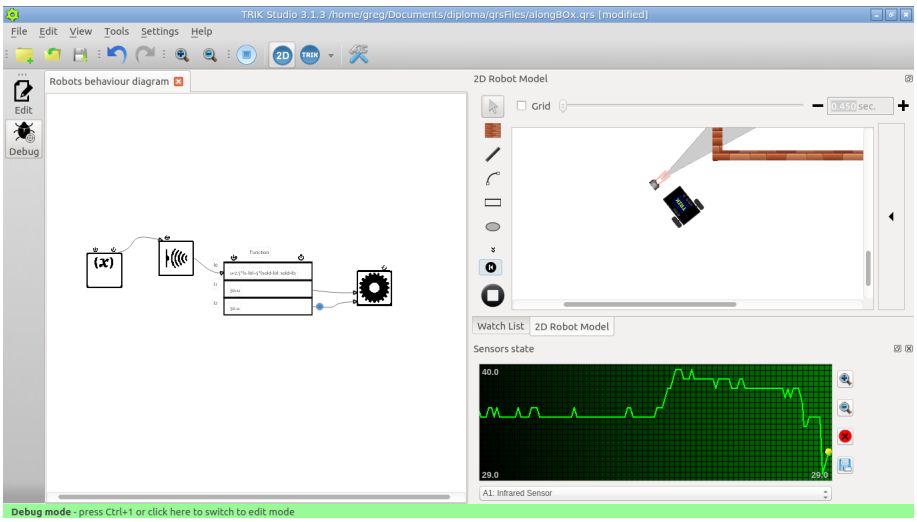


Рис. 14: Пользовательский интерфейс системы

3.2.2. Интерпретатор

Второй подключаемый модуль содержит реализацию интерпретатора диаграмм в стиле потоков данных. Получив диаграмму, созданную первым подключаемым модулем, интерпретатор преобразует ее в последовательность команд, которые посылаются выбранному роботу (см. рис. 15).

Робот может быть одним из поддерживаемых средой TRIK Studio: робот LEGO NXT или EV3, робот TRIK, TRIK Studio 2D симулятор или V-REP 3D симулятор [24]. Команды посылаются с помощью высокоуровнего API[17] устройств, реализованных в среде, частично API представлен на рисунке 16.

Основная архитектура подключаемого модуля, реализующего интерпретацию потоковых диаграмм, представлена на рисунке 17. Процесс интерпретации делится на две части: подготовка и непосредственно интерпретация. На первом шаге интерпретатор, получив диаграмму, обходит ее, валидирует и подготавливает для процесса интерпретации. Сначала производится обход графа диаграммы, для каждого блока, создается объект, реализующий его функциональность[18]. Созданием объектов занимается соответствующая фабрика блоков. Объекты подписываются друг к другу в соответствии с тем, как они связаны потоками на диаграмме. Здесь используется поведенческий шаблон проектирования *издатель-подписчик*. Блоки, содержащие текстовые

[17] Application Programming Interface (интерфейс программирования приложений).
[18] Объекты реализованы на языке C++.

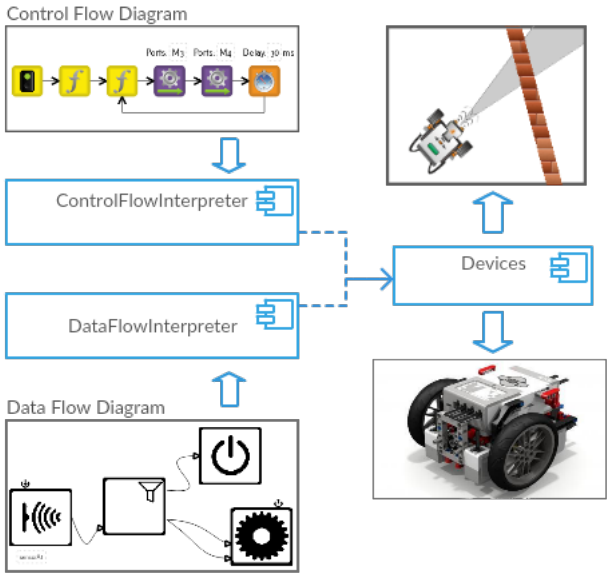


Рис. 15: Общая архитектура работы модулей

поля, проверяются на корректность выражений. Стартовыми блоками будут все вершины в графе, которые не имеют входящих связей. После завершения всех этих действий процесс интерпретации готов к исполнению. На втором шаге стартовым блокам посылаются необходимые для начала работы данные (обычно это «пустые» данные для верхних входных портов), и начинается процесс интерпретации.

Процесс интерпретации отличается от общепринятого асинхронного исполнения, используемого в большинстве сред потокового программирования. Обычно компоненты потоковых диаграмм исполняются параллельно в разных потоках, процессах или даже машинах (к примеру, Microsoft Robotics Developer Studio разворачивает диаграмму в набор веб-сервисов). Применение такого подхода хорошо подходит для платформ с мощным аппаратным обеспечением, но не в случае, когда речь идет о встраиваемых системах (embedded devices). Данная работа направленна именно на встраиваемые системы (роботы LEGO NXT, EV3, TRIK), поэтому был использован другой способ исполнения потоковых диаграмм. Главная идея — завести глобальную очередь сообщений и цикл обработки событий для обработки сообщений. Когда данные публикуются каким-нибудь блоком, они помещаются в очередь сообщений и ожидают доставки подписчикам (см. рис. 18). Таким образом параллельная модель исполнения заменяется на псевдо-параллельную, где интер-

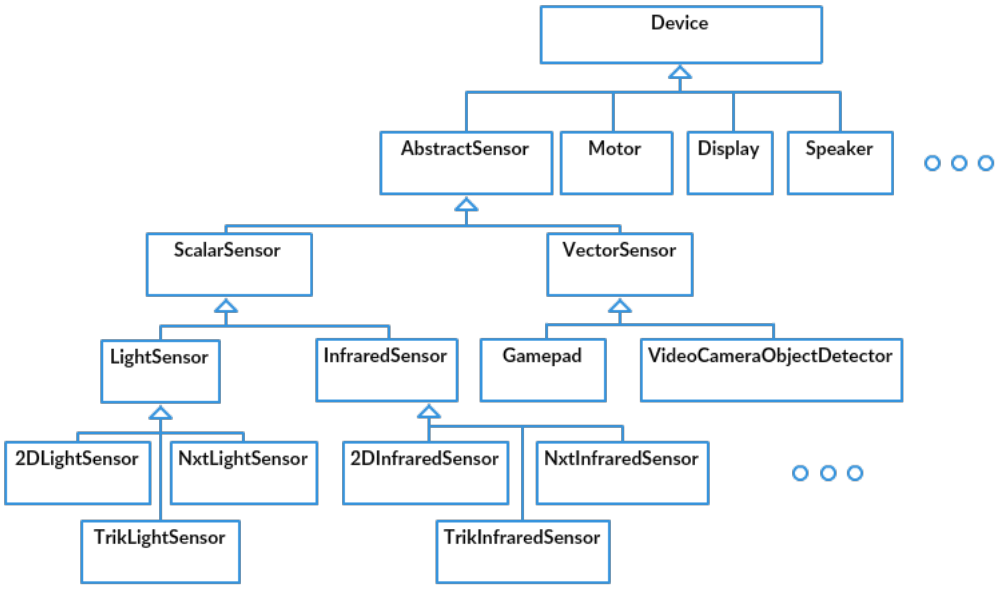


Рис. 16: Частичная архитектура устройств используемых в интерпретаторе потоковых диаграмм

претатор сам формирует расписание работы блоков. Данный механизм не был написан «с нуля». Был использован похожий механизм, который используется платформой Qt: обработка событий полностью возложена на класс $QEventLoop$, а доставка сообщений выполняется посредством механизма сигналов и слотов системы Qt, в режиме $QueuedConnection$.

Модель потока управления явно поддерживается в языке, что особенно важно в контексте направленности языка на обучение пользователей: блок «активируется» приходящими на него данными. Еще одной особенностью интерпретатора языка является то, что если в каком-то пути потока данных повторно встречается блок, отвечающий за генерацию данных с одного и того же устройства, то, чтобы избежать «лавинного эффекта» и сделать исполнение более интуитивно-понятным, как только данные приходят в повторно встречающийся блок, он «деактивирует» остальные блоки, соответствующие этому же устройству. Например, рассмотрим рисунок 19, где блок «Энкодеры» встречается дважды на пути потока данных. Когда запускается процесс интерпретации, блок «Константа» генерирует данные для блока «Моторы», а блок «Энкодеры» (a) начинает генерировать значения, соответствующие числу оборотов силовых моторов. Когда блок «Энкодеры» (b) получает данные, блок «Энкодеры» (a) прекращает генерацию данных.

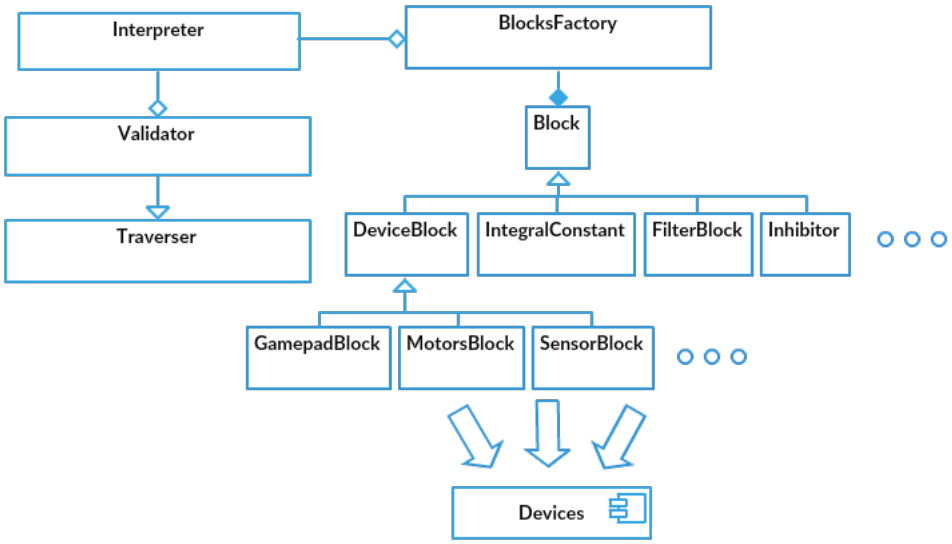


Рис. 17: Архитектура интерпретатора потоковых диаграмм

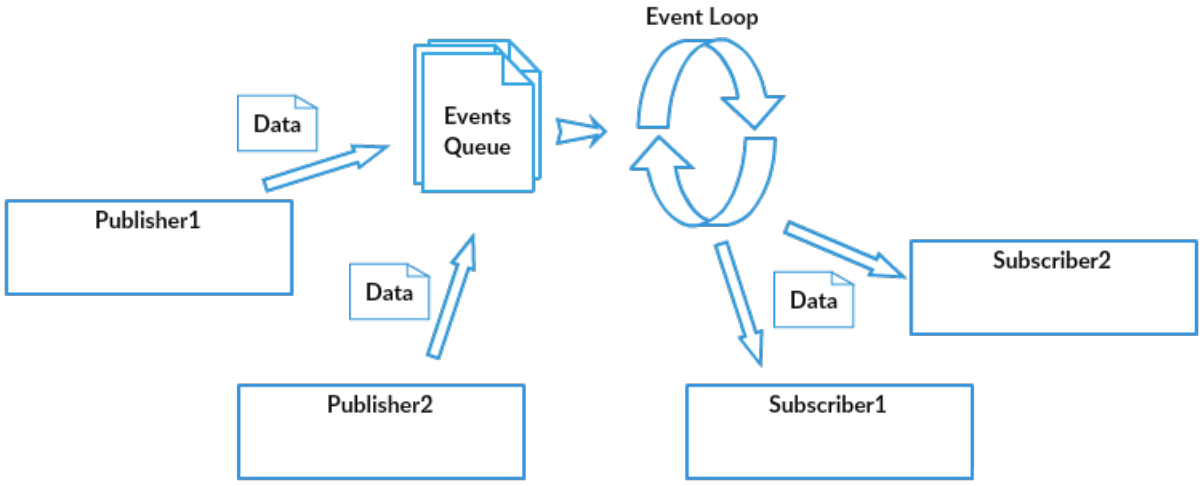


Рис. 18: Механизм псевдо-параллельной интерпретации потоковых диаграмм

Псевдо-параллельная модель хорошо работает на встраиваемых системах, но иногда, пользователю все еще может потребоваться явное выполнение некоторых блоков в отдельных потоках, это, в частности, относится к «Пользовательскому блоку», который может быть использован в качестве уровня для создания системы управления в архитектуре категорий. Для этих целей в язык включен блок «Распараллеливание», который обычно не включается в потоковые языки. В режиме интерпретации в использовании этого блока нет явной необходимости, так как исполнение диаграммы происходит на компьютере, а роботу посылаются низкоуровневые команды. Однако, он (блок) будет весьма полезен, когда появится поддержка автономного испол-

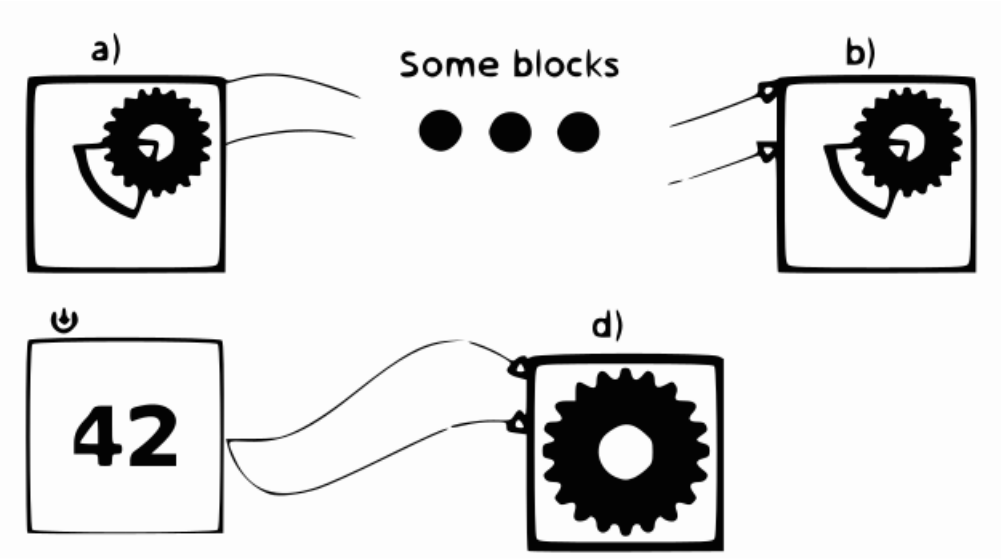


Рис. 19: Пример повторно встречающихся блоков

нения программ на реальном роботе. Его работа будет заключаться в создании платформо-зависимых элементов многопоточного исполнения, например *pthread* для UNIX или *tasks* для NXT OSEK.

4. Апробация

Для апробации были выбраны два различных по сложности примера. Первый — это типичная учебная задача в робототехнике: мобильный робот должен проехать вдоль стены (объехать коробку). Второй — сложнее: нужно реализовать алгоритм управления мобильным роботом, который случайно блуждает и избегает лобовых столкновений, при этом нужно иметь возможность управлять им с помощью устройства операторского контроля.

4.1. ПД-регулятор для движения вдоль стены

В новом языке пропорционально-дифференциальный регулятор был создан с помощью четырех блоков (см. рис. 20). В блоке «Глобальная переменная», мы задаем уставку регулятора (расстояния до стены, которого робот должен придерживаться), а также инициализируем необходимые значения используемых в регуляторе переменных. После этого блок «Сенсор» начинает непрерывно посылать данные в блок «Текстовое программирование», в нем данные с сенсора преобразуются в импульсы для блока «Моторы».

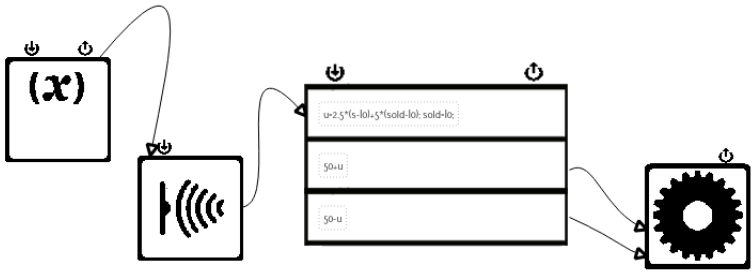
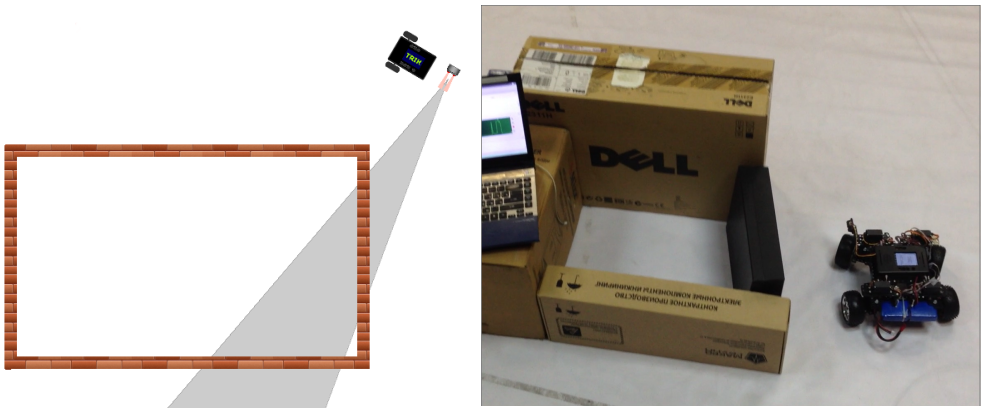


Рис. 20: Диаграмма ПД-регулятора на новом языке

После тестирования работоспособности регулятора для движения вдоль стены на двумерной модели робота (см. рис. 21a), можно запустить интерпретацию на реальном роботе (см. рис. 21b). Порядок запуска интерпретации: симуляция на 2D модели, использование реального робота — несущественен.



(a) Симуляция езды вдоль стены (b) Езда вдоль стены

Рис. 21: Интерпретация ПД-регулятора

4.2. Трехуровневая система управления

Рассмотрим более сложную задачу. Была реализована система управления случайно блуждающим роботом, но при этом избегающим лобовых столкновений. Как только оператор начнет управлять роботом с джойстика, робот должен следовать командам оператора, но все еще избегать лобовых столкновений. Если оператор перестанет управлять роботом, он снова должен начать случайно блуждать.

Такая система управления хорошо выражается в архитектуре категорий. Рассмотрим ее подробнее. Нулевой уровень будет отвечать за случайное блуждание робота (см. рис. 22). Здесь с помощью блока «Задержка» мы повторно посылаем данные на блоки «Случайное число», которые генерируют импульсы для моторов.

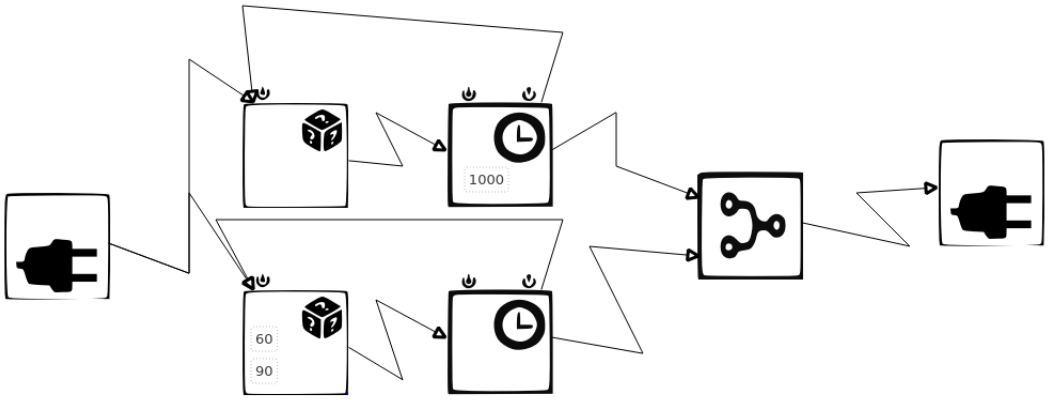


Рис. 22: Уровень бесцельного блуждания

Первый уровень будет отвечать за управление роботом с помощью устройства операторского контроля (см. рис. 23). Оператор с помощью манипуляторов джойстика задает направление для движения робота, а с помощью кнопок может остановить выполнение системы управления. Блок «Операторский контроль» генерирует соответствующие значения, а блок «Текстовое программирование» преобразует эти данные в импульсы для моторов.

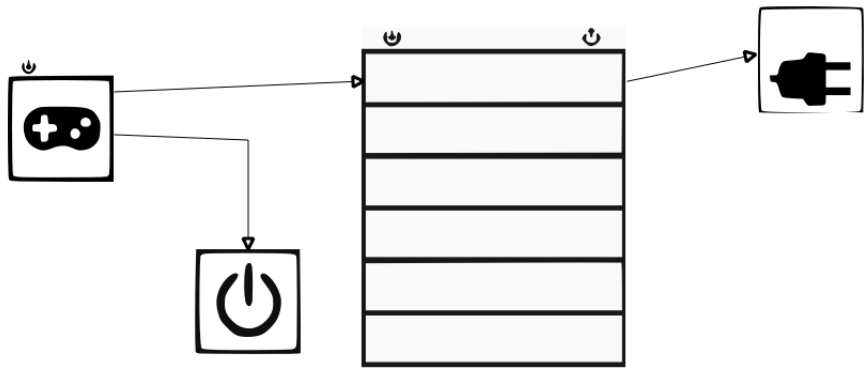


Рис. 23: Уровень управления с устройства операторского контроля

Оставшийся второй уровень будет отвечать за избегание столкновений (см. рис. 24). Два сенсора, установленных на передней панели робота, генерируют данные, их значения синхронно собираются в пары, после чего проверяется, близок ли робот к столкновению, если да, то блок «Текстовое программирование» высчитывает необходимые импульсы, которые нужно подать на моторы, чтобы избежать столкновения.

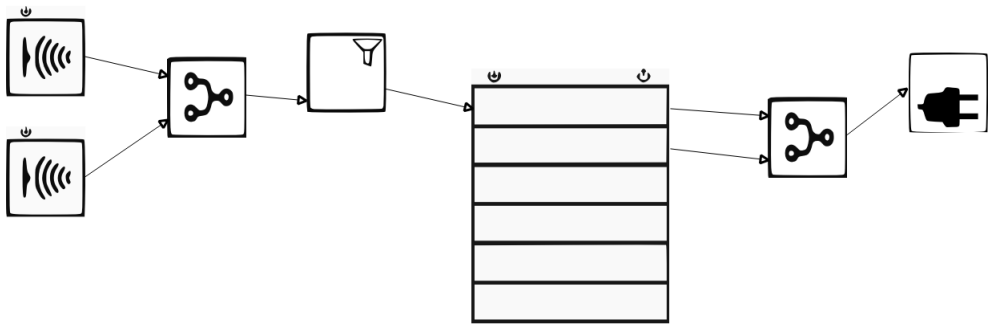


Рис. 24: Уровень избегания столкновений

Теперь, имея три уровня поведения, «соберем» систему управления роботом (см. рис. 25). Уровни представлены «Пользовательскими блоками» и сле-

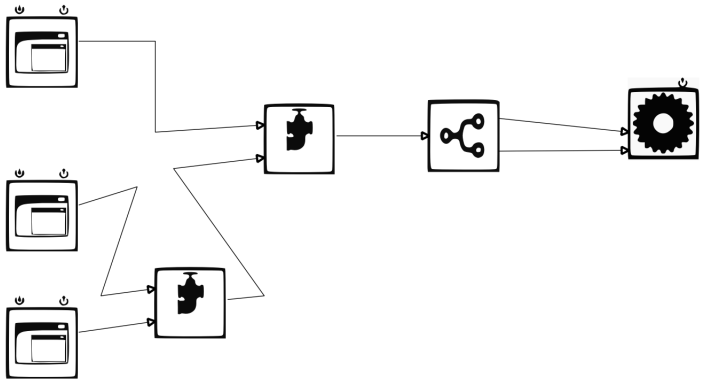
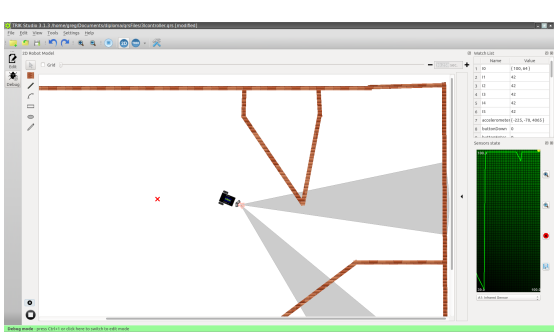
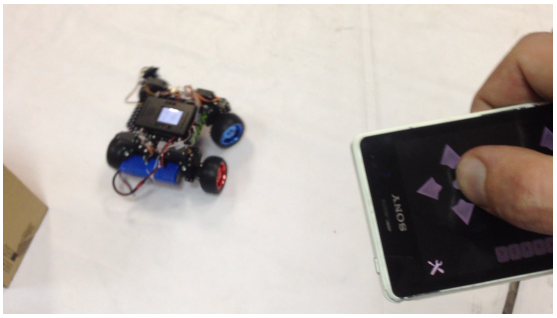


Рис. 25: Диаграмма трехуровневой системы управления



(a) Симуляция на 2D модели



(b) Запуск интерпретации на реальном роботе

Рис. 26: Интерпретация трехуровневой системы управления

дуют снизу вверх. Данные, посылаемые на моторы нулевым уровнем, могут быть заменены данными от первого уровня с помощью блока «Замещение». Высший приоритет имеет второй уровень — уровень избегания столкновений, он может замещать данные от нижних двух. Если пользователь перестает управлять роботом с джойстика, то первый уровень перестает генерировать данные для моторов, тем самым подавляя нулевой, и нулевой уровень снова получит управление, при условии, что робот не близок к столкновению. Этот пример, так же как и пример выше, был протестирован на симуляторе и на реальном роботе (см. рис. 26).

Заключение

В рамках данной выпускной квалификационной работы был создан новый язык программирования роботов в терминах потоков данных для учебных робототехнических конструкторов TRIK, LEGO NXT, LEGO EV3. Для использования языка был создан редактор визуального языка с применением модельно-ориентированного подхода на базе DSM-платформы QReal, созданной на кафедре Системного программирования СПбГУ. Им была расширена среда программирования роботов TRIK Studio. Для исполнения созданных с помощью визуального редактора программ была разработана компонента, позволяющая их интерпретировать. Данная компонента тоже является расширением для TRIK Studio. Интерпретация программ, созданных на новом языке может быть осуществлена на двумерной симуляционной модели робота, а также непосредственно на реальных роботах. Язык оказался достаточно простым в освоении, и в то же время обладает достаточными функциональными возможностями, чтобы создавать сложные системы управления, управляющие различными аспектами поведения робота. Это доказывает проведенная апробация языка и инструментария его поддержки для программирования типичных учебных задач в робототехнике, а также для более сложных систем управления роботом.

Результаты работы, а именно программный код[19] и видеоролики апробации[20] находятся в открытом доступе.

Дальнейшие перспективы

В качестве дальнейшего развитие нового инструмента планируется реализовать генераторы нового визуального языка в текстовые языки программирования, уже поддерживаемые средой TRIK Studio, к примеру, NXT OSEK C для LEGO NXT, байткод для LEGO EV3, JavaScript, F# [25] и Kotlin для TRIK, чтобы позволить программам выполняться на роботах автономно.

Созданную систему можно рассматривать в качестве платформы для последующих академических исследований. Во-первых, требуется формализа-

[19] https://github.com/ZiminGrigory/qreal/tree/DFVPL

[20] https://www.youtube.com/channel/UCrcri-bcBsOnHYvziXvctiA

ция семантики языка для того, чтобы иметь возможность применения различных формальных методов анализа программ, выраженных в новом языке. Другая ветвь развития — это применение предметно-ориентированного моделирования для автоматической генерации метамодели языка по спецификациям модулей промежуточного программного обеспечения (middleware) ROS [26] или Player [27].

Список литературы

[1] Banyasad Omid. A Visual Programming Environment for Autonomous Robots. 2000.

[2] Simpson Jonathan, Jacobsen Christian L, Jadud Matthew C. Mobile robot control // Communicating Process Architectures. 2006. C. 225.

[3] Simpson Jonathan, Jacobsen Christian L. Visual Process-Oriented Programming for Robotics. // CPA. 2008. C. 365–380.

[4] Process-Oriented Subsumption Architectures in Swarm Robotic Systems. / Jeremy C Posso, Adam T Sampson, Jonathan Simpson [и др.] // CPA. 2011. C. 303–316.

[5] 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. C. 25–32.

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

[7] Hils Daniel D. Visual languages and computing survey: Data flow visual programming languages // Journal of Visual Languages & Computing. 1992. T. 3, № 1. C. 69–101.

[8] Tools for teaching introductory programming: what works? / Kris Powers, Paul Gross, Steve Cooper [и др.] // ACM SIGCSE Bulletin / ACM. T. 38. 2006. C. 560–561.

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

[10] Using LEGO NXT Mobile Robots With LabVIEW for Undergraduate Courses on Mechatronics / Jesús M. Gomez-de Gabriel, Anthony Mandow,

Jesús Fernandez-Lozano [и др.] // IEEE Trans. Educ. 2011. Т. 54, № 1. С. 41–47.

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

[12] Kelly James Floyd. Lego Mindstorms NXT-G Programming Guide. Apress, 2010.

[13] Cyr. Martha N. Robolab. Справочное пособие к программному обеспечению. Москва: ИНТ [перевод], 2001.

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

[15] Brooks Rodney [и др.]. A robust layered control system for a mobile robot // Robotics and Automation, IEEE Journal of. 1986. Т. 2, № 1. С. 14–23.

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

[17] 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.

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

[19] Ierusalimschy Roberto. Programming in lua. Roberto Ierusalimschy, 2006.

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

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

[22] 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.

[23] Поддержка жестов мышью в мета-CASE-системах / Мария Сергеевна Осечкина, Тимофей Александрович Брыксин, Юрий Викторович Литвинов [и др.] // Системное программирование. 2010. Т. 5, № 1.

[24] 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.

[25] 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.

[26] 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.

[27] Gerkey Brian, Vaughan Richard T, Howard Andrew. The player/stage project: Tools for multi-robot and distributed sensor systems // Proceedings of the 11th international conference on advanced robotics. Т. 1. 2003. С. 317–323.