Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем
Системное программирование

Фадеев Виктор Юрьевич

Интеграция языка РуСи в робоконструкторы

Выпускная квалификационная работа бакалавра

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

Рецензент:
директор ООО «Новые Мобильные Технологии» В. В. Оносовский

Санкт-Петербург
2020

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems
Software Engineering

Fadeev Victor

Integration of the RuC language into robot designers

Bachelor's Thesis

Scientific supervisor:
prof. chair SE, D. Sc., professor Andrey Terekhov

Reviewer:
CEO LLC «New Mobile Technologies» Valentin Onossovski

Saint-Petersburg
2020

Оглавление

Введение

Сегодня активно развивается использование роботов для обучения школьников программированию. Это позволяет более наглядным образом показать, как работают программы. Однако программирование роботов все еще остается достаточно сложным, поэтому выбор используемых технологий и языков является крайне важным для упрощения процесса обучения.

Одной из таких технологий является визуальное программирование. Обучающийся составляет диаграмму поведения робота, после чего из нее генерируется код конкретного языка программирования и загружается, для исполнения, в робота. Диаграммы сильно упрощают понимание создаваемого алгоритма, благодаря использованию атомарных блоков с конкретными действиями. В данной работе задействована среда графического программирования TRIK Studio, разработанная на кафедре системного программирования СПбГУ. Выбор данной среды связан с ее активным использованием в робототехнических кружках и, как следствие, известностью среди преподавателей.

В качестве языка программирования, как следует из названия работы, используется язык РуСи, разработанный профессором кафедры системного программирования Андреем Николаевичем Тереховым. Ключевыми особенностями языка РуСи, являются:

- обширный список отлавливаемых им ошибок,
- высокая скорость работы,
- переносимость,
- малый вес,
- поддержка русского языка,
- наличие C-подобного синтаксиса.

Эти факторы делают РуСи перспективным языком для обучения программированию, так, например, C-подобная структура языка облегчает

будущий переход к более сложным языкам программирования, а наличие русского языка, с возможностью плавного перехода к английскому, помогает обучающимся более младшего возраста.

С технической стороны вопроса выступает контроллер ЙоТик[1], основанный на микропроцессоре ESP32[2]. Основными преимуществами данного контроллера, являются низкая цена, в сравнении с другими контроллерами, в сочетании со всей необходимой, для обучения, функциональностью. Также его отличительной особенностью является схожесть с Arduino-подобными контроллерами, облегчающая обучающимся будущее использование более сложных и общеизвестных форматов контроллеров. Для взаимодействия с TRIK Studio и языком РуСи, ЙоТик использует специальную прошивку, предоставленную компанией MGBot, позволяющую исполнять коды виртуальной машины РуСи.

[1]ЙоТик — контроллер производства компании MGBot

[2]ESP32 — микроконтроллер разработанный компанией Espressif Systems

1. Постановка задачи

Целью работы является внедрение системы программирования РуСи в TRIK Studio для использования в робоконструкторе ЙоТик. Для достижения этой цели были сформулированы следующие задачи.

- Добавить в TRIK Studio набор для контроллера ЙоТик.
- Реализовать генерацию кода на языке РуСи для контроллера ЙоТик.
- Разработать требования для новой прошивки ЙоТик 32 v2.0.
- Произвести рефакторинг проекта РуСи для выполнения требований новой прошивки.

2. Обзор

2.1. Плагинная система TRIK Studio

Среда графической разработки TRIK Studio позволяется создавать программы для роботов: TRIK[3], LEGO Mindstorms NXT[4], LEGO Mindstorms EV3[5]. Полученные программы можно исполнять последовательно, передавая команды с управляющего компьютера, в режиме интерпретации, преобразовывать в код на текстовом языке программирования и загружать в робота, либо же исполнять на виртуальной 2D-модели робота внутри среды. Сама IDE написана на C++ с использованием фреймворка Qt[6], что упрощает взаимодействие с роботом посредством Wi-Fi, Bluetooth и USB интерфейсов.

Архитектура комплектов контроллеров имеет плагинную систему, что позволяет конечному пользователю, при установке программы, выбрать только требуемые ему наборы моделей роботов и используемых языков. Данная система имеет следующую структуру:

- метамодель робота, представляющая собой описание доступных, для работы с контроллером, блоков диаграммы в формате `*.xml` файла;
- ядро комплекта, реализующее представленные в метамодели блоки, базовую модель робота и протоколы коммуникации с ним;
- плагин интерпретатора, необходимый для отображения робота в списке доступных, а также реализующий интерфейс интерпретатора и 2D-модели;
- базовый генератор конкретной модели робота, содержащий в себе шаблоны генерации кода для каждого из представленных в метамодели блоков; именно от него наследуются все генераторы поддерживаемых данным контроллером языков.

[3]TRIK — кибернетический конструктор компании КиберТех
[4]LEGO Mindstorms NXT — базовый набор для программирования на основе деталей Lego Technic
[5]LEGO Mindstorms EV3 — образовательная робототехническая платформа компании LEGO
[6]Qt — кроссплатформенный фреймворк для разработки ПО

Рассмотрим подробнее процесс генерации кода (рис. 1), так как в данной работе не будет реализовываться режим интерпретации или 2D-модель робота. В начале пользователь строит диаграмму управления роботом. Затем по ней генерируется граф потока управления программы и проверяется на корректность. При нахождении ошибки процесс останавливается, до внесения исправлений пользователем. После этого граф преобразуется в семантическое дерево, пригодное для применения шаблонов языка программирования, в соответствии с правилами определенными в базовом генераторе. В результате этих манипуляций мы получаем готовый код на требуемом языке программирования. Также, в случае необходимости дополнительной модификации кода после генерации, возможно переопределить специальные методы для изменения полученного кода перед выводом пользователю. Например, при помощи регулярных выражений.

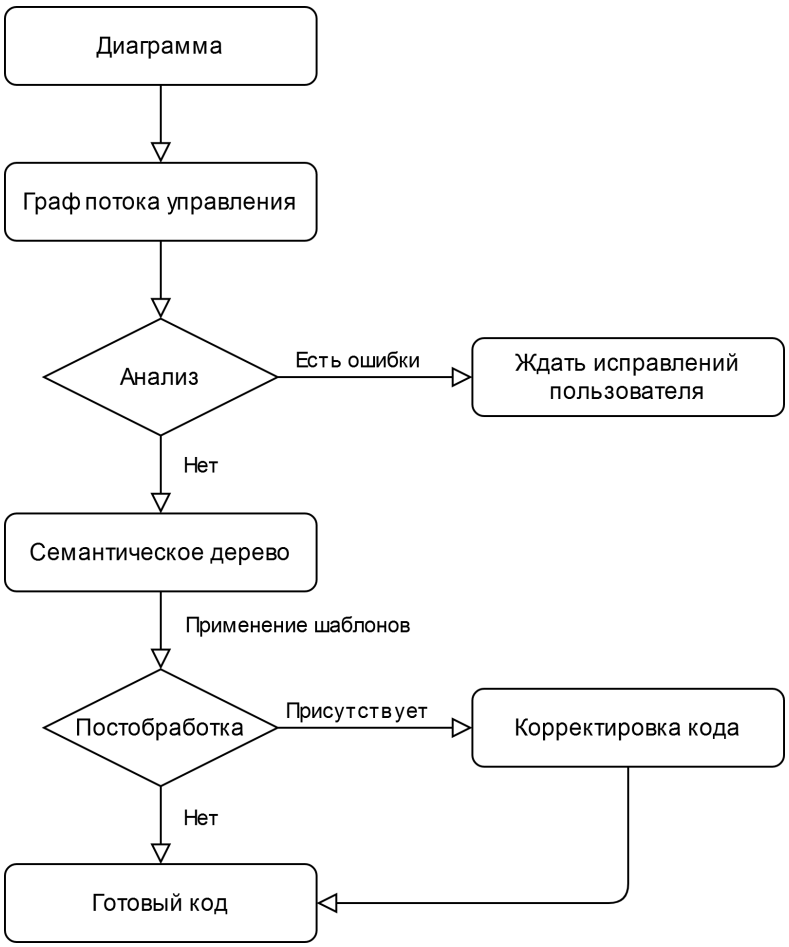

Рис. 1: Процесс генерации кода

2.2. Технология программирования РуСи

Для первоначального ознакомления с языком программирования РуСи были изучены следующие статьи [7] и [6]. Язык РуСи содержит в себе следующий функционал: базовые математические операции, функции работы со строками, интерфейс для считывания показаний с датчиков, а также технологию для работы с параллельными нитями. Помимо этого, как и большинство языков программирования, имеет возможность определения пользовательских функции, структур, массивов и других абстракций. Каждая стандартная функция РуСи имеет 4 способа записи — строчные и заглавные буквы русского и английского языков. Как и в C, программа начинает свое исполнение с функции `main()`.

С точки зрения реализации, язык РуСи состоит из двух частей — транслятора и виртуальной машины. Транслятор встраивается в среду разработки для преобразования РуСи-кода в коды виртуальной машины. Виртуальная машина устанавливается на целевую платформу и исполняет передаваемые ей коды. Данный подход облегчает портирование языка РуСи на различные системы и сокращает количество используемых, для интерпретации программы, ресурсов.

2.3. Техническое обеспечение

В данной работе используется контроллер ЙоТик 32 v2.0 (рис. 2) с прошивкой, содержащей виртуальную машину РуСи, поддерживаемые компанией MGBot. И одной из задач работы является разработка требований к следующей ее версии, в связи со сложностями в поддержке. Для углубления познаний в этой сфере, была изучена официальная документация ESP-IDF [2], на которой основана прошивка.

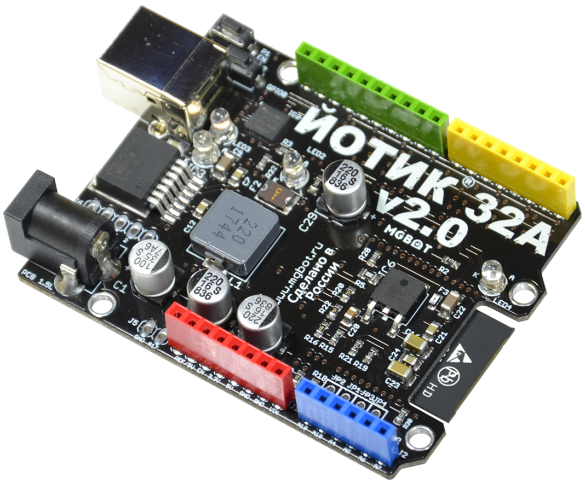

Рис. 2: Внешний вид контроллера ЙоТик 32 v2.0

Далее представлена схема работы с контроллером (рис. 3): на контроллер один раз записывается прошивка для интерпретации РуСи, в TRIK Studio разрабатывается визуальная программа и генерируется исходный код, после чего он транслируется в коды виртуальной машины и отправляется на контроллер для исполнения.

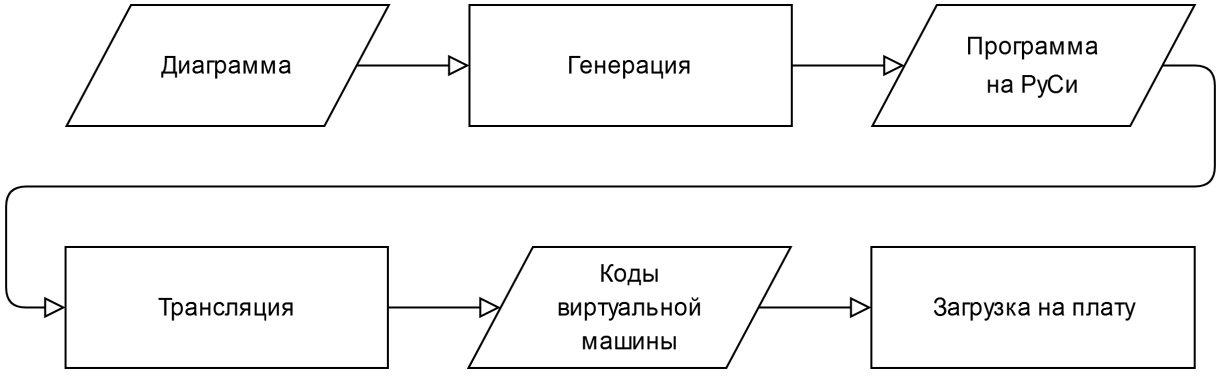

Рис. 3: Схема взаимодействия с контроллером

Также помимо контроллера были предоставлены датчики (освещенности, расстояния, ультрафиолетовый, емкостный, акселерометр, компас, гироскоп, атмосферного давления, влажности и температуры) и модули (RGB светодиод, LED матрица, реле, I2C двигатели) компании. Данные компоненты могут использовать различные способы взаимодействия, а именно: аналоговый вывод значений, цифровой ввод / вывод, универсальный режим работы (доступны аналоговый и цифровой режимы), интерфейс I2C[7]. Особый интерес в данном случае представляет именно I2C шина, так как использует две линии связи, передающая данные (SDA) и генерирующая такты (SCL). Этот интерфейс позволяет подключить до 127 устройств, имеющих разные адреса, к одним и тем же контактам, при условии достаточного питания.

Еще одним предоставленным инструментом, требующим интеграции, является сервис Blynk[8]. Данный сервис позволяет удаленно управлять микроконтроллером при помощи смартфона. Для его внедрения необходимо реализовать методы взаимодействия с сетью из клиентского кода, используя для ознакомления общедоступную документацию [1].

[7]I2C — последовательная асимметричная шина для связи между интегральными схемами

[8]Blynk — популярная облачная IoT платформа

2.4. Работы прошлых лет

В 2017 году похожая работа, под названием названием «Поддержка программирования микроконтроллера STM32 в TRIK Studio» [5], проводилась Приходько Станиславом Витальевичем. Однако данная работа имеет ряд ключевых отличий, в сравнении с этой.

- Использование контроллера ЙоТик вместо STM32[9], с кардинально отличающимся подходом к загрузке исполняемой программы. А именно, использованием постоянной прошивки, находящийся на контроллере, и передачей только кодов интерпретируемой программы, посредству как USB, так и Wi-Fi интерфейса.

- Выбор языка РуСи в качестве промежуточного, с необходимостью поддержки двух шаблонов генерации и интеграции транслятора РуСи в TRIK Studio.

- Наличие более сложной метамодели, в связи с более обширными требованиями и большим количеством датчиков.

- Возможность свободного конфигурирования контактов датчиков, допускающая не взаимно однозначное соответствие.

[9]STM32 — семейство 32-битных микроконтроллеров производства STMicroelectronics

3. Расширение функциональности TRIK Studio

3.1. Добавление контроллера ЙоТик

3.1.1. Метамодель

Для добавления контроллера ЙоТик в TRIK Studio, сначала необходимо определить его метамодель. Проект метамодели создается в папке `./plugins/robots/editor/iotik` и подключается к проекту `editor`. В начале файла метамодели описываются используемые в ней типы данных. Они позволяют выводить пользователю выпадающее меню с вариантами ответа, вместо пустой строки ввода, что сокращает количество ошибок. Метамодель ЙоТик-а использует следующие типы: `IotikDigitalSensorPort`, `IotikAnalogSensorPort`, `IotikUniversalSensorPort`, `IotikWritePort`, `IotikBlynkPort` для описания портов, а также вспомогательные типы: `IotikLedColor`, `IotikMatrixColor`, `IotikIcons` для упрощения интерфейса блоков.

В центральной части файла находятся описания отображаемых блоков и наследуемых ими абстрактных. Для описания блока доступны следующие параметры:

- `displayedName` — отображаемое пользователю название,
- `name` — внутренний уникальный идентификатор блока,
- `path` — жест сенсорного экрана,
- `description` — содержание всплывающей подсказки.

Внутри блок разделен на две части: графическое представление (`<graphics>`) и логика (`<logic>`). В графике описываются отображаемые на диаграмме элементы: пиктограмма и основные параметры блока, свойство `hard` запрещает сокрытие элемента с неактивного блока. Логика содержит в себе все используемые блоком параметры, со значениями по умолчанию, а также ссылку на блок родителя.

Метамодель ЙоТик-а обладает следующей структурой (рис. 4). Организация команд двигателей произведена по аналогии с комплектом TRIK. Остальные блоки имеют общего предка `IotikSensorBlock` от которого наследуются все поддерживаемые интерфейсы:

- `IotikAnalogSensorBlock` — аналоговые блоки,
- `IotikDigitalSensorBlock` — цифровые блоки,
- `IotikUniversalSensorBlock` — блоки поддерживающие как аналоговый так и цифровой ввод,
- `IotikI2CSensorBlock` — блок протокола I2C,
- `IotikWriteBlock` — блоки вывода сигнала,
- `IotikBlynkBlock` — блоки сервиса Blynk.

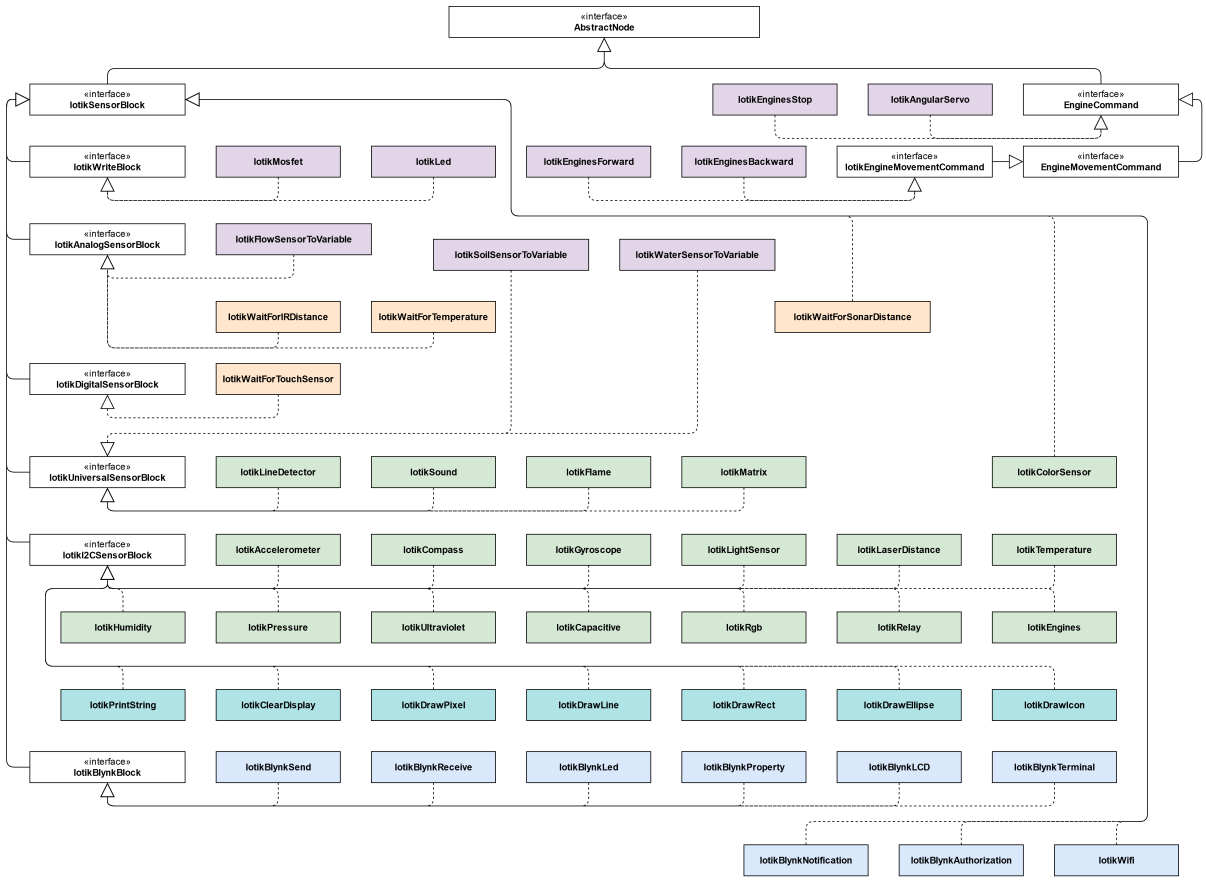

Рис. 4: Метамодель контроллера ЙоТик

В самом конце файла настраивается порядок отображения и производится распределение созданных блоков по группам. Метамодель ЙоТик-а содержит следующие группы: блоки немедленного исполнения, блоки ожидания события, блоки работы с дисплеем, модули компании MGBot, блоки работы с сетью.

3.1.2. Ядро комплекта

После создания метамодели необходимо реализовать ядро комплекта, содержащее фабрику блоков. По аналогии с предыдущей частью, в папке `./plugins/robots/common/iotikKit` создается подпроект `iotikKit`. В нем нужно реализовать классы `IotikBlocksFactory` и `IotikRobotModelBase` наследующие, соответственно, `blocksBase::CommonBlocksFactory` и `robotModel::CommonRobotModel`. Также для каждого зарегистрированного блока необходимо создать соответствующий класс, реализующий интерфейс `interpretation::Block`. После этого определить функции produceBlock и providedBlocks, возвращающие эти классы. Дополнительно можно реализовать функции `blocksToDisable` и `blocksToHide`, для отключения или сокрытия устаревших блоков (для полного отключения блока, его нужно указать в обеих функциях).

В этом же подпроекте реализуются протоколы коммуникации. Они располагаются в каталоге `communication` и наследуют `robotCommunication::RobotCommunicationThreadInterface`. У контроллера ЙоТик были определены USB и Wi-Fi протоколы связи.

3.1.3. Плагин интерпретатора

Чтобы подключить созданный робототехнический комплект в TRIK Studio, необходимо создать плагин интерпретатора в `./plugins/robots/interpreters/iotikKitInterpreter`. За основу можно взять `nullKitInterpreter`. Однако, в случае отсутствия режима интерпретации, его необходимо модифицировать, оставив только класс интерпретатора и возвращая пустой список в методе `robotModels`. Это

позволяет скрыть режим интерпретации от конечного пользователя, что и было сделано для комплекта ЙоТик (в случае еще не определенного генератора, данное действие скроет весь комплект).

3.2. Генерация РуСи-кода

3.2.1. Базовый генератор

Архитектура генераторов имеет следующую структуру (рис. 5), цветом выделены реализуемые, в данной работе, компоненты. Перед созданием генератора языка РуСи, необходимо создать базовый генератор комплекта в директории `./plugins/robots/generators/iotik/iotikGeneratorBase`. Для этого необходимо определить классы: `IotikMasterGeneratorBase`, `IotikGeneratorPluginBase`, `IotikGeneratorFactory`, `IotikGeneratorCustomizer`, а также задать модель робота. Наибольший интерес представляют: фабрика блоков и главный генератор.

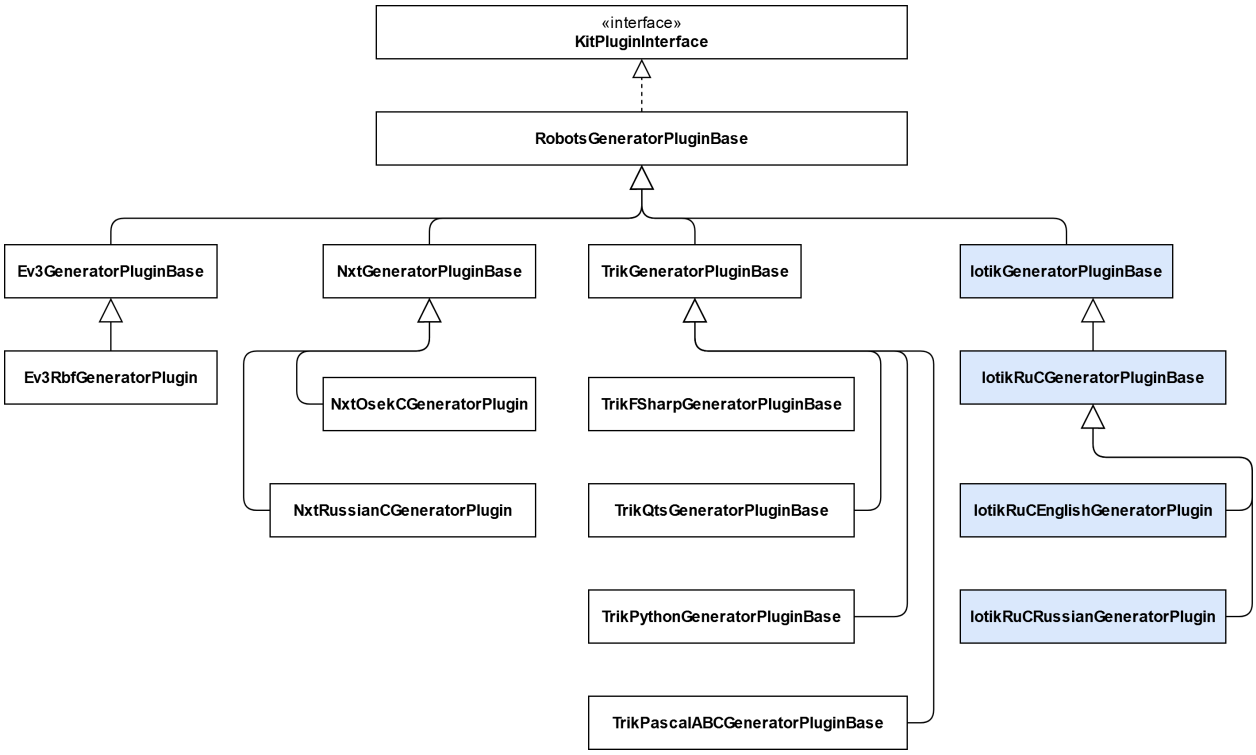

Рис. 5: Архитектура генераторов

Механика работы фабрики блоков аналогична фабрике из ядра комплекта. Для каждого из блоков метамодели необходимо определить класс простого генератора в подпапке `simpleGenerators`. Таким образом функция `simpleGenerator`, класса фабрики, возвращает необ-

ходимые для генерации блока элементы. Дополнительно хотелось бы отметить следующие особенности реализации простых генераторов:

- основной принцип работы заключается в считывании шаблона генерации, с последующей заменой ключевых слов на пользовательские значения; получение пользовательских данных производится функцией `repo.property(id, ''Variable'')`, где `''Variable''` — название запрашиваемого параметра (полученные таким образом значения можно преобразовать в строки, а также использовать в условном операторе);

- при реализации блоков считывания данных в переменную, следует проводить инициализацию этой переменной с помощью `Binding::createStaticConverting`, используя значение 0 для целочисленных и 0.0 для вещественных типов;

- также доступна возможность переопределения генераторов блоков по умолчанию, так, например, в случае с комплектом ЙоТик возникла необходимость в создании собственного генератора циклов `for`.

Второй интересующей нас частью, является главный генератор комплекта, наследуемый всеми поддерживаемыми языками. Перегрузка метода `afterGeneration` позволяет, как следует из названия, модифицировать код после процесса генерации. Данная возможность была задействована дважды для языка РуСи, а именно:

- для исправления генерации параллельных нитей, из-за использования языком целочисленных идентификаторов, вместо названий функции, потоками РуСи;

- для нормализации объявлений массивов, так как здание многомерных массивов в РуСи требует статического объявления младших размерностей.

3.2.2. Генератор языка

В общем случае, для реализации генератора языка достаточно создать классы, наследующие `MasterGeneratorBase`, `GeneratorPluginBase` базового генератора, в которых определить интерфейс панели инструментов и указать путь до шаблонов генерации. Однако, так как РуСи имеет два варианта написания: русский и английский, было решено создать промежуточный класс, в котором реализовать весь необходимый функционал, а зачем унаследовать от него два класса переопределяющих пути к шаблонам.

Транслятор языка РуСи был добавлен в качестве git[10] подмодуля и подключен к `IotikRuCGeneratorPluginBase`. Таким образом, при активации кнопки загрузки, сгенерированный код передается в транслятор, после чего полученные коды виртуальной машины отправляются на робота, либо, в случае ошибки компиляции, выводится сообщение пользователю.

[10]git — распределенная система контроля версий

3.3. Апробация

3.3.1. Проект «Теплица»

Для подтверждения работоспособности полученного решения, компанией MGBot был предоставлен проект «Теплица». Этот проект представляет собой агротехнический макет теплицы для выращивания растений (рис. 6). Так как изначально об был разработан для использования с Arduino IDE[11], в ходе апробации необходимо было достичь той же функциональности средствами TRIK Studio.

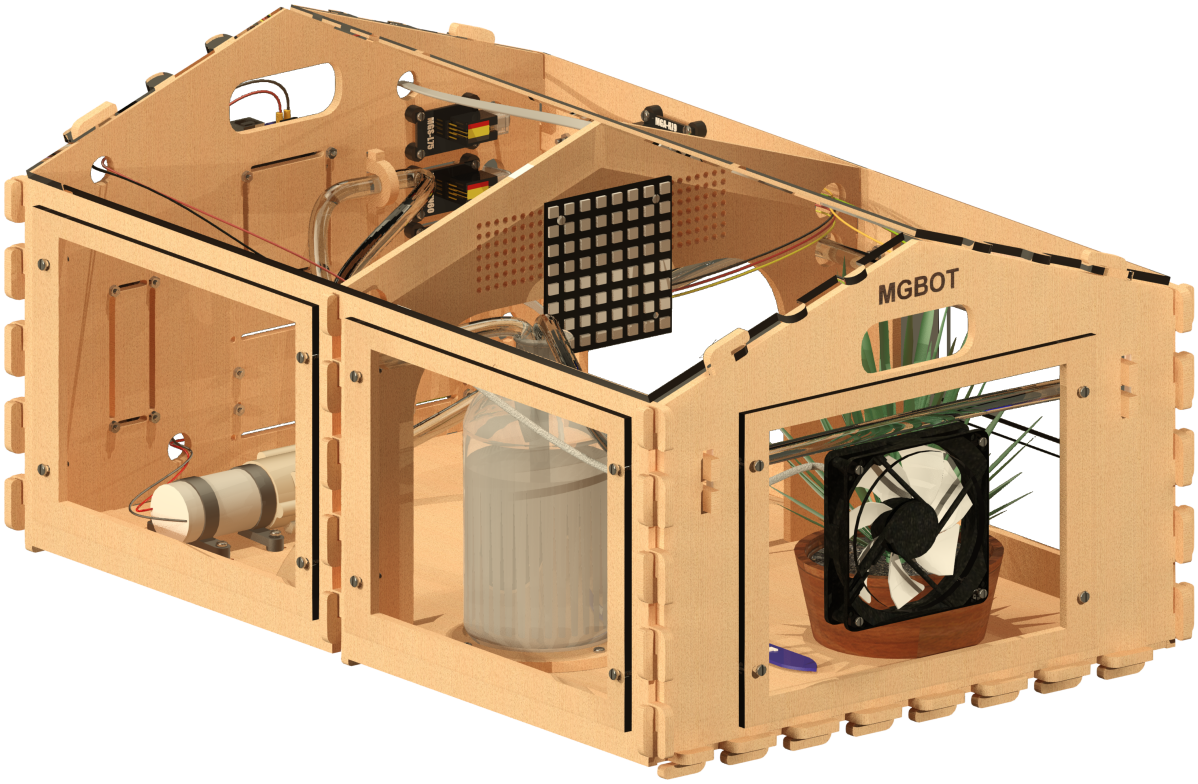

Рис. 6: Внешний вид макета теплицы

С этой целью была разработана следующая диаграмма (рис. 7). В начале программы контроллер подключается к сети Wi-Fi и авторизуется в сервисе Blynk. Затем, в бесконечном цикле, отправляет показания датчиков на сервер и принимает команды управления электроникой (рис. 8).

[11]Arduino IDE — приложение для написания и загрузки программ на различные платы и контроллеры

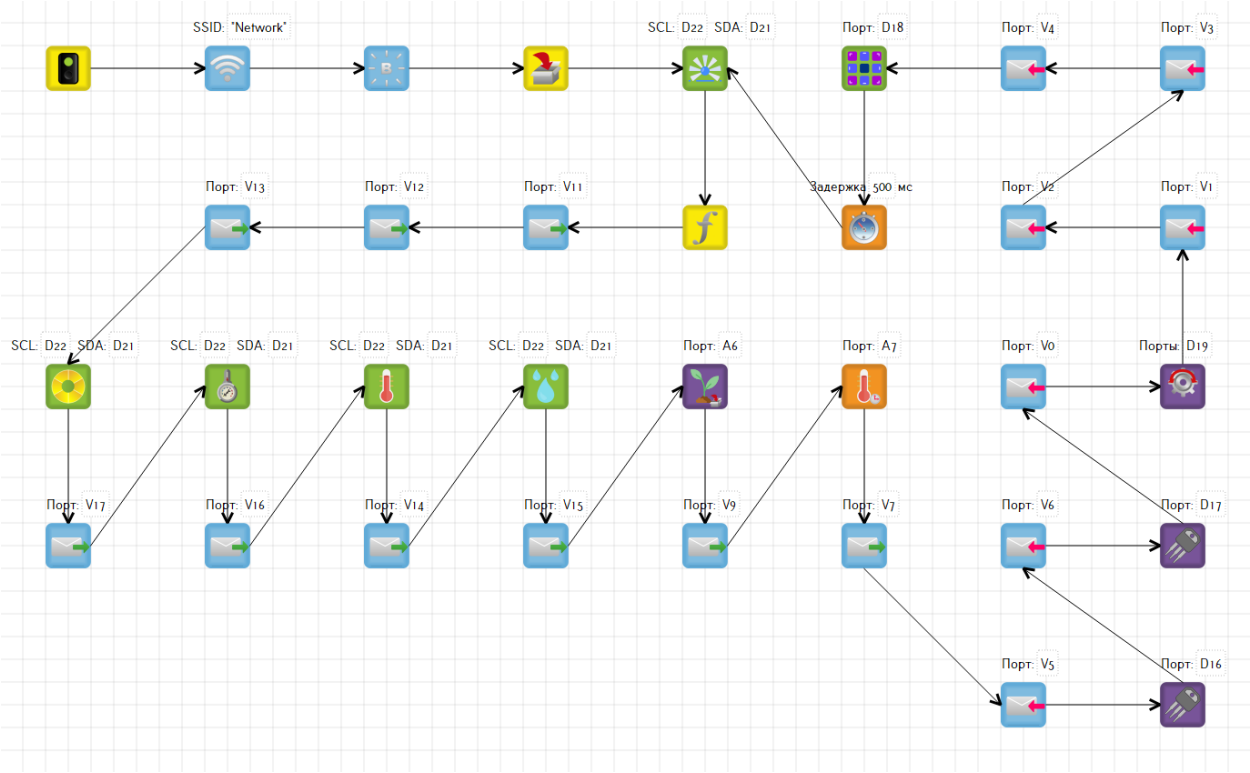

Рис. 7: Визуальная программа теплицы

Рис. 8: Интерфейс управления теплицей

В результате апробации была выявлена необходимость, в подтверждении загрузки программа на контроллер, и реализована в TRIK Studio.

3.3.2. Проект «Танк»

Следующим проектом для тестирования возможностей TRIK Studio, выступает модель танка. На основе предоставленных модулей была собрана платформа, содержащая в себе: драйвер двигателей (L298P) для управления шасси, угловой сервомотор (MG995) для поворота башни, лазерный датчик расстояния (VL56L0X) и реле для включения лазерного светодиода. Диаграмма (рис. 9) и интерфейс (рис. 10) проекта.

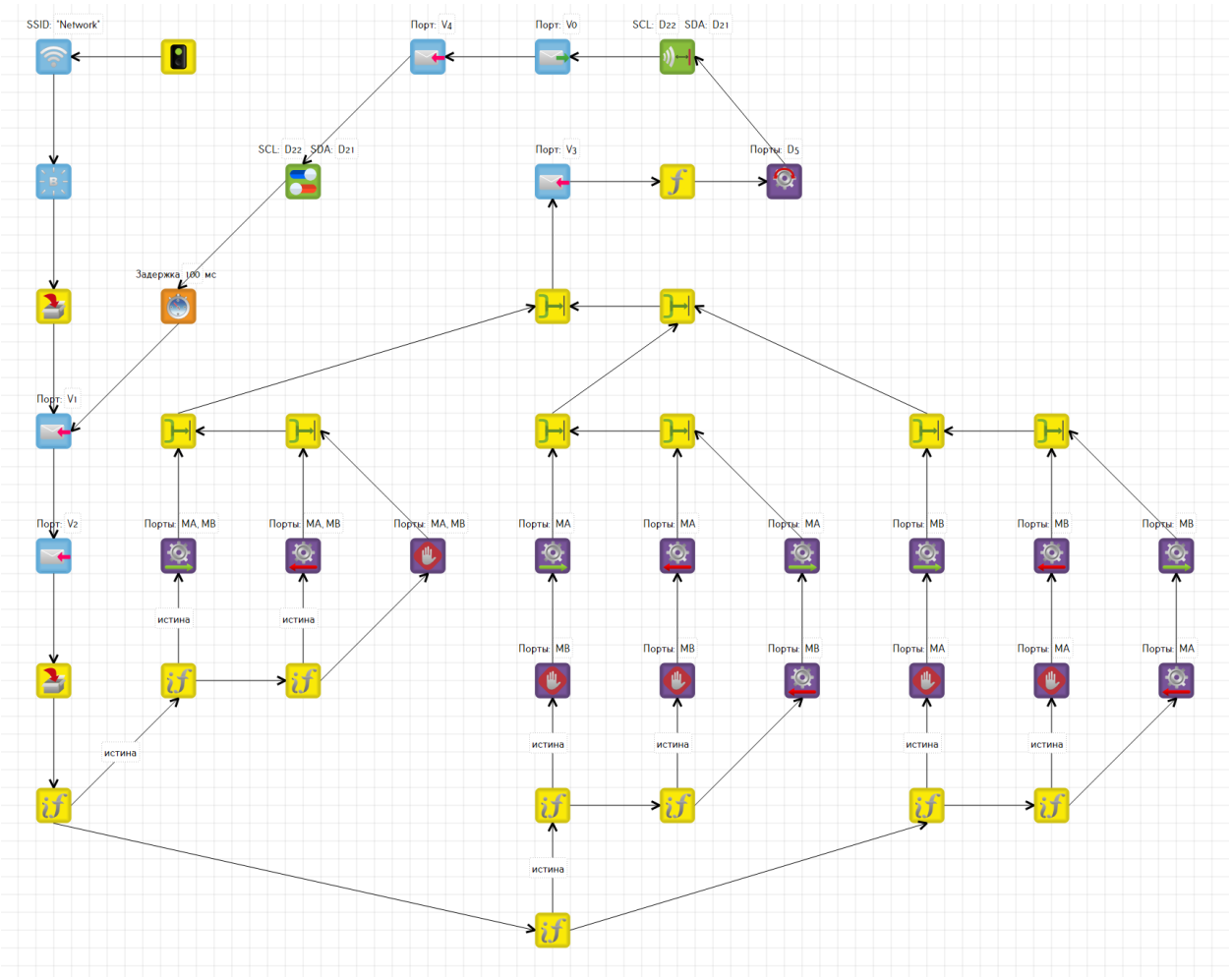

Рис. 9: Визуальная программа танка

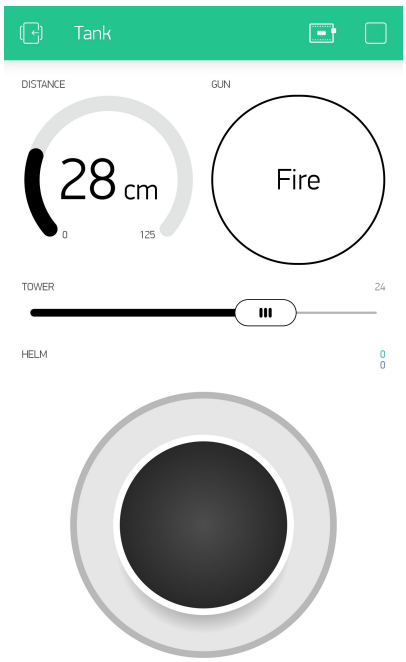

Рис. 10: Интерфейс управления танком

Тестирование показало работоспособность данной модели, а также недостатки использования данного драйвера, в следствии его перегревания и потери управления роботом.

3.3.3. Проект «Робо-футболист»

Последним тестируемым проектом, является робо-футболист, предназначенный для оценки субъективного времени отклика системы в целом. Данный робот построен на базе нового I2C драйвера, во избежание ранее выявленных проблем, и более мощных двигателей. Построенная диаграмма не приводятся, по причине большого сходства с предыдущей. Тестирование показало, что не смотря на минимальную задержку сервиса Blynk, в 100 миллисекунд, робот сохраняет свою управляемость.

4. Проектирование прошивки для ЙоТик 32 v2.0

Накопленный опыт использования и внесения модификаций в прошивку контроллера ЙоТик, позволяет сделать вывод о необходимости создания новой прошивки. В пользу принятия данного решения действуют следующие факторы: отсутствие четкой архитектуры в имеющемся решении, сложность внесения изменений и исправлений, а также отсутствие документации. В связи с этим возникла необходимость разработать следующие требования к новой версии прошивки:

- документирование методов и подходов к реализации необходимого функционала, для упрощения поддержки кодовой базы;
- локализация сообщений об ошибках, для ускорения процесса отладки;
- простота и атомарность функций, для более тесной интеграции с виртуальной машиной РуСи;
- самостоятельная инициализация и деинициализация компонентов, связанная с отсутствием данного механизма для генерируемого кода;
- обработка одновременных вызовов одинаковых команд из параллельных задач, для предотвращения пользовательских ошибок при построении параллельных программ;
- корректное завершение программы и очищение памяти при ошибке, для продолжения стабильной работы в штатном режиме.

В состав же новой прошивки обязаны войти следующие компоненты:

- виртуальная машина РуСи с возможностью независимого обновления,
- сетевой интерфейс для передачи файлов и возможностью полного отключения,
- файловая система с возможностью управления файлами,
- система распределения портов и I2C каналов с реализацией Arduino IDE интерфейса.

5. Рефакторинг проекта РуСи

Проект РуСи развивается уже более 5 лет, в нем приняли участие несколько поколений студентов, придерживающихся различных стилей программирования. Поэтому научным руководителем была поставлена задача рефакторинга проекта в целом.

Одним из первых шагов по улучшению проекта было создание автоматической системы тестирования. На момент начала работы репозиторий проекта уже содержал в себе сборник различных пользовательских тестов. Однако они запускались только вручную и не имели никакой структуры. Поэтому на языке Bash[12] был написан специальный скрипт для проверки работоспособности проекта и запуска тестов. В случае корректной сборки проекта и прохождения хотя бы одного теста сборка считается успешной. Это связано с тем, что тесты ошибок и тесты функционала невозможно отличить друг от друга. После этого также была настроена система непрерывной интеграции Travis CI[13].

Следующим шагом по улучшению общей структуры проекта было введение автоматизированной системы проверки оформления кода. В качестве такой системы был выбран ClangFormat[14], в связи со своей распространенностью, понятностью и простотой настройки [3], а также Clang-Tidy[15] для расширения недостающего функционала [4]. Дополнительно, для повышения качества кода, исходные файлы проекта были разделены на взаимодействующие друг с другом библиотеки.

В завершение, с целью упрощения интеграции РуСи в другие проекты, была добавлена система сборки CMake[16], а репозиторий разделен на три части: транслятор, виртуальную машину и IDE. В сборку транслятора и виртуальной машины была добавлена возможность создания динамической библиотеки, для упрощения встраивания проекта.

[12]Bash — командная оболочка UNIX

[13]Travis CI — распределенный веб-сервис сборки и тестирования ПО

[14]ClangFormat — инструмент автоматического форматирования исходных кодов

[15]Clang-Tidy — среда для диагностики и исправления ошибок программирования

[16]CMake — кроссплатформенная система автоматизации сборки ПО

Заключение

В ходе данной работы были получены следующие результаты.

- В TRIK Studio для ЙоТик добавлены все требуемые компоненты.
- Выработаны требования к новой прошивке.
- Проведен рефакторинг проекта РуСи.
- Осуществлена апробация на содержательных промышленных примерах («Теплица», «Танк», «Робо-футболист»).

Список литературы

[1] Blynk. Blynk Docs // Blynk. — 2020. — Access mode: https://docs.blynk.cc.

[2] Guide ESP-IDF Programming. API Reference // Espressif. — 2016-2020. — Access mode: https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/index.html.

[3] documentation Clang 11. ClangFormat // clang.llvm.org. — 2007-2020. — Access mode: https://clang.llvm.org/docs/ClangFormat.html.

[4] documentation Extra Clang Tools 11. Clang-Tidy // clang.llvm.org. — 2007-2020. — Access mode: https://clang.llvm.org/extra/clang-tidy.

[5] С.В. Приходько. Поддержка программирования микроконтроллера STM32 в TRIK Studio. — Кафедра Системного программирования Математико-механического факультета СПбГУ, 2017.

[6] Терехов А.Н. Параллельные программы в РуСи // GitHub. — 2020. — Access mode: https://github.com/andrey-terekhov/RuC/wiki/Параллельность.

[7] Терехов А.Н. Терехов М.А. Проект РуСи для обучения и создания высоконадежных программных систем. — Известия высших учебных заведений, Северо-Кавказский регион, 2017.