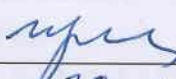


МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК
Кафедра программного обеспечения

ДОПУЩЕНО К ЗАЩИТЕ В ГЭК
И ПРОВЕРЕНО НА ОБЪЕМ
ЗАИМСТВОВАНИЯ
Заведующий кафедрой
д.п.н., профессор
 И.Г. Захарова
20 июня 2016 г.

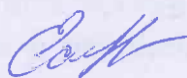
МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

РАЗРАБОТКА И ВНЕДРЕНИЕ ИНТЕГРИРОВАННОГО КУРСА
«ПРОГРАММИРОВАНИЕ В РОБОТОТЕХНИКЕ»

02.04.03. Математическое обеспечение и администрирование информационных систем

Магистерская программа «Высокопроизводительные вычислительные системы»

Выполнил работу
Студент 2 курса
очной формы обучения


(Подпись)

Есин
Тимофей
Евгеньевич

Научный руководитель
к.т.н., доцент


(Подпись)

Воробьева
Марина
Сергеевна

Рецензент
к.т.н., доцент


(Подпись)

Григорьев
Михаил
Викторович

Тюмень 2016

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	3
ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ.....	6
1.1. Роботы – инструмент обучения программированию	6
1.2. Серия конструкторов Lego Mindstorms	7
1.3. Сравнительная характеристика поколений Mindstorms.....	9
1.4. Программная оболочка LeJos	11
1.5. Управление роботом с помощью поведенческих правил	12
ГЛАВА 2. РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ УПРАВЛЕНИЯ МОБИЛЬНЫМ РОБОТОМ.....	16
2.1. Проектирование приложения.....	16
2.2. Описание используемых технологий.....	20
2.3. Техническое описание структуры приложения	25
2.4. Логическая структура приложения	33
ГЛАВА 3. ВНЕДРЕНИЕ И ИСПОЛЬЗОВАНИЕ ПРИЛОЖЕНИЯ SENSEBOT	37
3.1. Ввод приложения в эксплуатацию	37
3.2. Описание пользовательского интерфейса	41
3.3. Инструкция по программированию в SenseBot	44
ЗАКЛЮЧЕНИЕ	48
СПИСОК ЛИТЕРАТУРЫ	49
СПИСОК ИЛЛЮСТРАЦИЙ.....	51
ПРИЛОЖЕНИЯ.....	51

ВВЕДЕНИЕ

В компьютерном мире произошло огромное количество изменений за последние тридцать лет. Технологии становятся быстрее и меньше. Большие успехи достигнуты и в аппаратных средствах, и в программном обеспечении. Тем не менее, многие вопросы середины 80-х годов прошлого века остаются нерешенными по сей день. Одна из таких проблем – обучение программированию. Учить программированию в средней школе не является приоритетной задачей, хотя она и обязательная.

Национальная образовательная программа нацелена на развитие навыков написания алгоритмов и алгоритмического мышления, но способность написать компьютерную программу можно заметить лишь у школьников, обучающихся в профильных классах. По этой причине, многие абитуриенты, поступающие на специальности, связанные с информационными технологиями, начинают изучать элементы алгоритмизации и программирования только в университете. Для увеличения заинтересованности и повышения мотивации к программированию все чаще используются игровые формы обучения и роботизированные устройства.

Мобильные роботы распространены весьма широко, используются в лабораториях научных институтов и университетов, разрабатываются и модернизируются на различных предприятиях для решения специальных задач, таких как переноса груза, работы в сложных условиях, информационной разведки. Курсы по робототехнике включаются в образовательные программы, проводятся специализированные соревнования различных уровней. Роботы входят в повседневную жизнь человека в виде бытовых и игровых устройств. Но, несмотря на многие преимущества обучения программированию при помощи мобильных роботов, работа с ними не лишена некоторых недостатков.

На занятиях, в условиях ограниченного времени, возникают сложности с процессом запуска, изменением программного кода и обсуждением выполняемых роботом действий. Как правило, работа той или иной программы объясняется наблюдением за поведением робота и основывается на понимании того или иного действия, установлении соответствия с программным кодом. Изменение уже запущенной программы включает в себя ряд продолжительных шагов, за время выполнения которых разрывается связь с тем, что робот уже делал ранее. Обычно, эти шаги состоят из отключения запущенной программы, выявления причины неправильной работы, загрузки модифицированной программы и перезапуска программы непосредственно на программируемом блоке.

Чтобы этого избежать, можно применить более интерактивный подход: во-первых, выводить на экран устройства, с которого программируется блок, значения датчиков расстояния, цвета, освещенности и других. Так, если видеть, что происходит «внутри» интеллектуального блока, пока он выполняет программу, можно определить с чем связаны те или иные выполняемые им действия, или их отсутствие. Во-вторых, изменять работу программы без перезапуска, усложнять ее. Это позволит экономить время, сохранять динамику проведения занятий и, как следствие, лучше понимать программирование с использованием мобильных роботов.

Основная сложность разработки интегрированного курса состоит в выборе материального-технического оснащения и написании программного обеспечения для работы на занятиях.

Объект исследования: процесс обучения программированию.

Предмет исследования: методика использования мобильных роботов для обучения программированию в рамках курса информатики основной школы и внеурочной деятельности школьников.

Цель работы: разработка приложения и его архитектуры для повышения эффективности взаимодействия с мобильным роботом.

Главная идея создания такой платформы состоит в предоставлении пользователям возможности отслеживать значения датчиков и изменять программу для робота во время выполнения алгоритмов. С помощью платформы, у пользователей появится возможность добавлять или изменять программные модули без отрыва от того, что происходило с роботом на предыдущей итерации.

Для достижения цели необходимо решить следующие задачи:

1. Провести анализ научно-методической литературы и исследований эффективности использования роботов в обучении программированию.
2. Изучить современные роботизированные платформы, их возможности.
3. Разработать приложение, позволяющее следить за работой датчиков и имеющее возможность изменять алгоритм запущенной программы во время работы.

ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Роботы – инструмент обучения программированию

Роботизированные платформы в течение последнего десятилетия стали популярным педагогическим инструментом в школах и университетах, используются как в образовательных, так и в демонстрационных целях. Роботы играют активную роль в образовании с момента появления в 1991 году исполнителя «Черепаша», программируемого на языке ЛОГО. Сеймур Пейперт, создатель языка ЛОГО, в 1980 году написал книгу «Mindstorms: Children, Computers, and Powerful Ideas». В книге развивается две центральные темы: дети в совершенстве могут научиться пользоваться компьютером и то, что способ обучения может быть совершенно иным, отличающимся от всех тех, которыми они до этого узнавали все остальное. Пейперт высказывает точку зрения, что в классах, насыщенных технологиями есть больше возможностей для социализации, и что технологии часто способствуют более тесному взаимодействию между учениками, а также учениками и их учителями. В концепции Пейперта описывается использование компьютеров не только для помощи в обучении или проведении каких-либо расчетов, но и с другой стороны — обучение компьютера для своих целей.

Одноименная с книгой серия конструкторов Mindstorms, разработанная Lego в содружестве с MIT¹ впервые представлена в 1998 году. На сегодняшний день существует три поколения наборов. Наборы комплектуются набором стандартных деталей (балки, оси, колеса, шестерни) и набором, состоящим из сенсоров, двигателей и программируемого блока. Если только что написанная программа может дать «видимый» результат (а не только набор символов на экране), это заряжает энергией и может мотивировать к самостоятельному изучению программирования. Таким образом, программируемые роботы,

¹ MIT (*Massachusetts Institute of Technology*) — Массачусетский технологический институт

разработанные Lego для образовательных целей, кажутся одними из наиболее подходящих устройств. С помощью конструкторов этой серии удастся достичь высокого уровня заинтересованности учеников, а научиться писать программы для управления совместимыми электронными компонентами чрезвычайно просто.

1.2. Серия конструкторов Lego Mindstorms

Серия Lego Mindstorms содержит программное и аппаратное обеспечение для создания модифицируемых механических программируемых роботов. Наборы серии включают в себя интеллектуальный компьютер, так называемый "кирпич" (далее – блок), который управляет системой, ряд модульных датчиков и двигателей, и детали из линейки конструкторов Lego Technics для создания механических систем.

Первый Lego Mindstorms RCX, был выпущен в 1998, основой для которого послужил микроконтроллер H8. Следующим поколением стал Lego Mindstorms Education NXT, выпущенный в 2006, он заметно расширил функционал первого поколения набора, также в нем использовался новый микроконтроллер ARM7 32bit CPU и язык Robolab, разработанный в качестве основы для LabVIEW компанией National Instruments.

Lego Mindstorms Education EV3 — это третье поколение робототехнических конструкторов серии LEGO Mindstorms (рис.1). Данный набор разрабатывался в Массачусетском технологическом институте совместно с компанией Lego. Вы можете конструировать роботов с множеством датчиков и моторов, или измерять расстояние, освещенность, температуру, проводя научные эксперименты.

При помощи меню на интеллектуальном микроконтроллере EV3 могут быть созданы лишь самые простые программы. Более сложные программы и

звуковые файлы загружаются через порт USB или по беспроводной сети с помощью Bluetooth.



Рисунок 1. Интеллектуальные блоки различных поколений

Компания National Instruments с 1986 г. выпускает программный продукт LabVIEW. LabVIEW — это среда разработки и платформа для выполнения программ, созданных на графическом языке программирования G. Создание программ на этом языке имитирует составление блок-схем, когда программа формируется путем соединения соответствующих блоков. Таким образом, это помогает пользователям трансформировать бумажную блок-схему программы в блоки на экране. Именно этот язык был положен в основу программных продуктов Lego Mindstorms.

EV3-G это среда визуальной разработки, которая поставляется в комплекте с EV3. Среда EV3-G была разработана специально для LEGO специалистами компании National Instruments.

Программы, написанные при помощи данной среды, могут быть использованы в реальных роботах, при условии тщательного проектирования блоков и связей для инкапсуляции сложности. Программное обеспечение хорошо подходит для запуска небольшой группы параллельных циклов

датчиков/ответчиков (например, подождать 60 секунд, проиграть звук "bonk" на небольшой громкости при низком заряде батареи, повторить), или для совместной по Bluetooth и другого "внешнего управления". Язык предоставляет виртуальные инструменты для всех фирменных датчиков и компонентов LEGO, а также для большинства датчиков сторонних производителей.

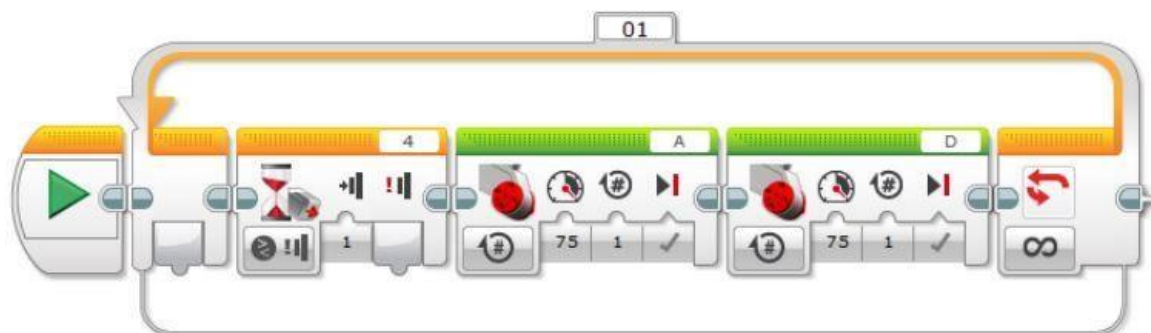


Рисунок 2. Короткая программа, написанная на стандартном языке EV3-G.

На рис.2 изображена короткая программа, написанная на стандартном языке EV3-G. Начинается с блока бесконечного цикла, внутри которого располагаются остальные блоки. Первый из них – ожидание нажатия датчика касания, подключенного в 4-й порт блока. Когда кнопка на датчике была нажата, сервомотор, подключенный в порт 'A', совершает полный оборот. Затем мотор в порту 'D' так же поворачивается на 360° со скоростью 75%.

К сожалению, по умолчанию блок не предназначен для использования других языков программирования. Для того, чтобы использовать популярные языки программирования, такие как Java, Python или C#, внутреннее программное обеспечение блока должно быть заменено.

1.3. Сравнительная характеристика поколений Mindstorms

В рамках данной работы использовался интеллектуальный блок третьего поколения EV3, как наиболее подходящая платформа. Блок из образовательной версии набора не отличается от домашней комплектации, поэтому разработанное приложение будет работать на обеих версиях. Базовый набор включает в себя один программируемый блок, два больших сервомотора, один средний мотор,

ультразвуковой датчик, два датчика касания, один датчик наклона и датчик цвета. Все перечисленные электронные компоненты изображены на Рис.3.



Рисунок 3. Электронные компоненты набора EV3

Наибольшее отличие поколений Lego Mindstorms кроется внутри интеллектуального блока. Во-первых, он стал намного производительнее, а также появилось несколько новых функций (характеристики EV3 в сравнении с предыдущим поколением можно видеть на Табл.1). Во-вторых, в модели нового поколения появилась возможность загружать дополнительную операционную систему, установленную на карте памяти microSD. Нововведение позволяет очень просто произвести замену стандартной оболочки, так как для этого необходима лишь загрузочная флеш-карта. Кроме того, такой способ позволяет загружать также и стандартную операционную систему (рис.4), т.к. удалять ее не обязательно. Предыдущие поколения требовали «прошивать» новое внутреннее программное обеспечение в блок напрямую, заменяя старое.

Несмотря на то, что EV3 вышел сравнительно недавно, уже появилось множество открытых разработок, позволяющих работать с иными языками

программирования. К сожалению, многие из них находятся в стадии ранних альфа- и бета-версий, а также полны недоработок.



Рисунок 4. Встроенное программное обеспечение Lego Mindstorms EV3

1.4. Программная оболочка LeJos

Одна из наиболее стабильных оболочек, доступных для EV3 – leJOS². LeJOS – это java-совместимая программная среда, ее релизы представлены для каждого поколения Lego Mindstorms. Данная оболочка не только использует возможности EV3, но и расширяет их, предоставляя полный набор функций JVM (виртуальной машины Java) через операционную систему Linux. В рамках релиза для 3-го поколения появляется возможность управления и настройки блока через SSH, что было недоступно в предыдущих версиях.

Таблица 1. Сравнительная характеристика EV3 и NXT

	EV3	NXT
Дисплей	Монохромный LCD, 178x128	Монохромный LCD, 100x64
Процессор	300 МГц	48 МГц
Память	64 Мб RAM, 16 Мб Flash	64 Кб RAM, 256 Кб Flash

² leJOS – это среда программирования для Lego Mindstorms, позволяет программировать роботов на языке Java. Так же в рамках данной среды предоставляются поведенческие (behavior) классы, которые поддерживают категориальную (subsumption) архитектуру для более удобного программирования поведения роботов.

USB-хост	1 порт (480 мб/с)	Нет
Wi-Fi	Опционально, через USB	Нет
SD-карта	Micro-SD	Нет
Bluetooth	Есть	Есть
Операционная система	Linux	Проприетарная

Первая бета-версия системы вышла 18 апреля 2014 года. На сегодняшний день самой актуальной версией является 0.9.1-beta. У leJOS имеется официальный форум, где разработчики размещают новости и отвечают на вопросы пользователей.

1.5. Управление роботом с помощью поведенческих правил

В традиционной робототехнике робот – механизм, управляемый центральным контроллером, который постоянно обновляет свое представление об окружающем мире и вырабатывает план поведения, исходя из этого представления. Новая информация о мире поступает от сенсоров, например, осязания, света, ультразвука и т.д. Контроллер анализирует всю информацию от сенсоров и обновляет представление об окружающей среде, а затем принимает решение о том или ином действии.

Большинство тех, кто впервые решил программировать робота, обычно думает о разработке программы, как о серии условий «если-то», которые напоминают структурное программирование (Рис.5).

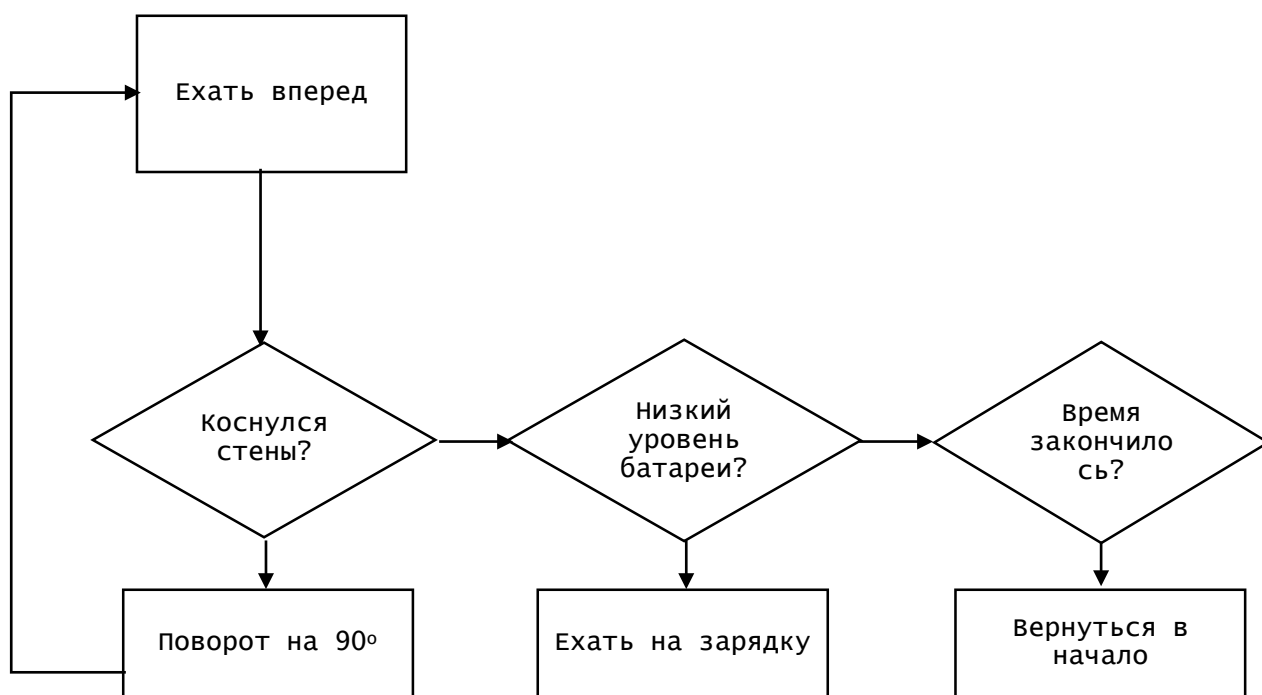


Рисунок 5. Блок-схема, отражающая вид структурного программирования

Такой тип программирования очень прост для начинающих, но сопряжен с определенными трудностями. Во-первых, такой подход требует большего объема вычислений и продолжительного проектирования, но, как правило, в итоге получается т.н. спагетти-код – запутанная и трудная для понимания программа. Во-вторых, поддержка актуальной картины окружающего мира – задача очень сложная, т.к. мир меняется постоянно. При этом известно, что многие организмы, например, насекомые, благополучно существуют и без поддержки полной картины мира, более того, даже не имея памяти как таковой. Подобные размышления стали отправной точкой нового течения в робототехнике, доминирующего в настоящее время. Оно получило название "поведенческая робототехника" (behavior-based robotics - BBR). [1]

Одним из способов организации BBR-роботов является архитектура поглощения, предложенная в 1986 г. Родни Бруксом (Rodney A. Brooks) – в настоящее время главой лаборатории искусственного интеллекта в Массачусетском Технологическом Институте (MIT) – в его фундаментальной статье под названием "Слоны не играют в шахматы". Согласно Бруксу,

поведенческие роботы можно рассматривать как набор простых и независимых поведенческих модулей (behaviors). Модули часто имеют реакционный характер: в контексте отсутствия внутреннего представления мира, они могут только реагировать на информацию, поступающую с различных датчиков. Поведения могут наслаиваться друг на друга, а также конфликтовать между собой. В этом случае, в действие вступает специальный механизм арбитража, который решает, какое поведение в данный момент является приоритетным. Ключевым моментом является то, что поведение робота, как единого целого, не закладывается заранее, а вырисовывается из взаимодействия его поведенческих модулей.

Брукс выдвинул гипотезу о том, что интеллект и рациональное поведение не возникает в нематериальных системах, подобных системам доказательства теорем, или даже в традиционных экспертных системах. Эту гипотезу автор подтвердил на примерах созданных им роботов. Он считает, что интеллект является продуктом взаимодействия определенной многослойной системы со своим окружением. Более того, он придерживается точки зрения, что интеллектуальное поведение возникает при взаимодействии архитектур, организованных из более простых сущностей. В этом состоит основная идея *категориальной архитектуры* (subsumption architecture). [2]

Категориальная архитектура представляет собой совокупность модулей поведения, предназначенных для решения отдельных задач. Каждый такой модуль является конечным автоматом, который непрерывно преобразует воспринимаемую входную информацию в выходное управляющее воздействие и зачастую реализуется с помощью простого множества продукционных правил.

В одном из первых роботов Родни Брукса было представлена трехслойная архитектура. Робот был оснащен кольцом из двенадцати ультразвуковых датчиков. Если не углубляться в подробности, то можно отметить, что на самом

низком уровне архитектуры была реализован модуль Avoid (избежать), которая предотвращает столкновение робота с объектами. Следующий слой Wander (перемещение) определяет направление каждые 10 секунд. Верхний уровень Explore (исследовать) позволяет изучить окружение робота. (Рис. 6).

Брукс утверждает, что поведение верхнего уровня возникает в результате разработки и тестирования отдельных слоев более низкого уровня архитектуры. Он также полагает, что «явные представления и модели мира можно строить лишь на низких уровнях интеллекта. Оказывается, в качестве модели мира лучше использовать сам мир» [3].

Программная оболочка leJOS предоставляет API³ для работы с архитектурой subsumption и программирования модулей поведения. Концепция программирования таких модулей в leJOS очень проста:

- только один модуль может быть задействован и управлять роботом одновременно;
- каждый модуль имеет заранее заданный приоритет;
- для каждого модуля самостоятельно определяются условия, при которых ему нужно получить контроль над роботом.

³ API (от англ. application programming interface, интерфейс программирования приложений) — набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) или операционной системой для использования во внешних программных продуктах.

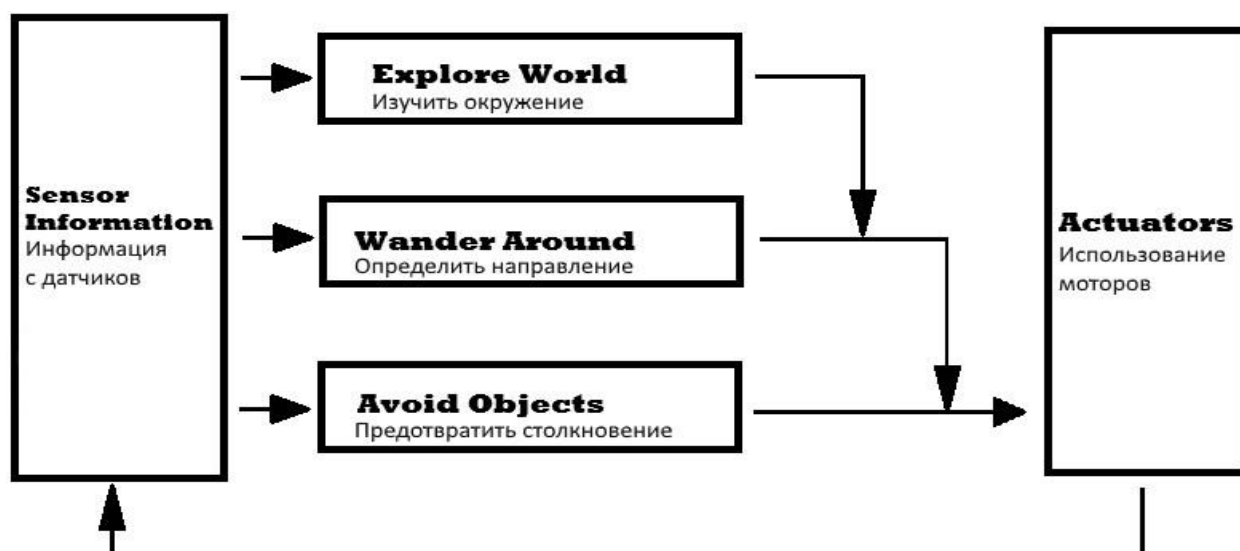


Рисунок 6. Схема программирования модулей поведения

В работе использован именно такой способ программирования, т.к. работа с модулями поведения обеспечивает непрерывность процесса загрузки/обновления программ.

ГЛАВА 2. РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ УПРАВЛЕНИЯ МОБИЛЬНЫМ РОБОТОМ

2.1. Проектирование приложения

2.1.1. Назначение и основные возможности проектируемого приложения

Приложение Sensebot разрабатывается для использования при программировании мобильных роботов, в частности, на занятиях по робототехнике. Название приложения – это комбинация слов *sense* (пер.с англ. – чувство или сознание) и *robot*, которая в полной мере отражает назначение разработки: все, что робот «чувствует» непрерывно отображается на экране в виде значений датчиков; второе значение слова говорит о возможности создавать интеллектуальное поведение, которое лежит в основе поведенческой робототехники.

Основными составляющими проектируемого приложения являются:

- модуль для удаленного просмотра значений датчиков в реальном времени;
- редактор кода для написания и загрузки программ, отвечающих требованиям категориальной архитектуры.

Приложение должно позволять изменять алгоритм работы мобильного робота без перезагрузки программы, путем добавления и изменения т.н. модулей поведения.

2.1.2. Архитектура приложения

В основе разрабатываемой системы лежит архитектура «клиент-сервер», в которой задачи и сетевая нагрузка распределены между поставщиками услуг, называемых серверами, и заказчиками услуг, называемых клиентами. В качестве среды взаимодействия клиента с сервером используется локальная сеть или Интернет.

Приложение состоит из трех взаимодействующих подсистем: клиент, внешний сервер, внутренний сервер. Клиент взаимодействует только с внешним сервером, внешний сервер взаимодействует как с клиентом, так и с внутренним сервером (внешний сервер является клиентом внутреннего сервера). Внешний сервер взаимодействует с внутренним для непосредственного управления мобильным роботом (рис.7).



Рисунок 7. Общая схема архитектуры системы

Такая трехуровневая архитектура позволяет нескольким пользователям получать доступ и совместно работать с одним и тем же мобильным роботом.

2.1.3. Проектирование серверной части

Внутренний сервер – программа, работающая на блоке EV3, которая принимает запросы на управление и выполняет их согласно требованиям. Для обеспечения одновременной работы нескольких клиентов, требуется разработать внешний сервер, который будет являться клиентом для внутреннего, и основным сервером для пользователей. Внешний сервер должен состоять из веб-сервера и интерфейса взаимодействия пользователя с программноаппаратной частью.

Веб-сервер — аппаратно-программный комплекс, предназначенный для обслуживания HTTP-запросов. HTTP-запрос — сформированный согласно протоколу HTTP/1.1 запрос на сервер на заранее определенный порт (по умолчанию, порты 80 и 8080) с целью выполнения какого-либо удаленного действия. Как правило, такие запросы посылает интернет-браузер клиента.

В задачи веб-сервера входит:

- Обработка данных и предоставление интерфейса;
- Получение и ответ на HTTP-запросы;
- Формирование запросов JSON;
- Динамическое обновление данных у подключенных пользователей.

2.1.4. Проектирование клиентской части

Клиентская часть приложения должна поддерживать следующие технологии:

- Возможность работы по протоколу HTTP/1.1;
- Доступ к сети, в которой находится сервер;
- Поддержка «горячей замены кода»;
- Масштабируемость под различные клиентские устройства;

- Обработка ошибок в работе программы

Общая схема работы клиентской части представлена на рисунке 8.



Рисунок 8. Схема работы клиентской части

2.1.5. Проектирование пользовательского интерфейса

Для создания макета веб-приложения использовалась сервис «Balsamic Москит 3». Сервис позволяет создавать мокапы⁴ пользовательских интерфейсов веб-сайтов, мобильных приложений и компьютерных программ.

Интерфейс состоит из трех блоков (рис.9):

1. Сервисный блок для обновления списка адресов EV3 и соединения с ними.
2. Блок отображения значений датчиков. Максимум – 4 датчика, по количеству доступных портов.
3. Блок программирования модулей поведения робота.

⁴ Москит или mock-up (макет) — неработающая модель, выполненная в натуральную величину и выглядящая так, как будет выглядеть работающий экземпляр.

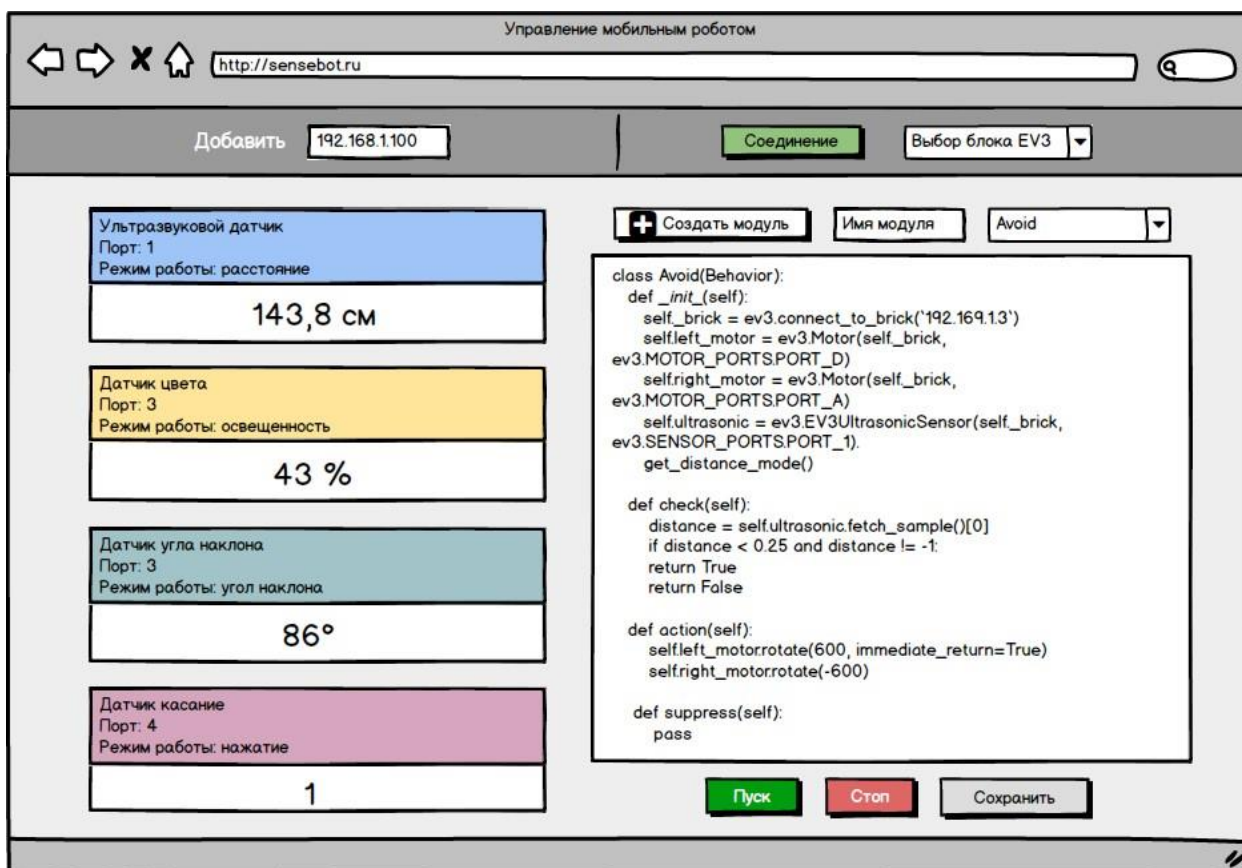


Рисунок 9. Макет пользовательского интерфейса

Интерфейс должен масштабироваться в зависимости от разрешения экрана пользователя, не теряя при этом в удобстве использования.

2.2. Описание используемых технологий

2.2.1. Языки программирования и способы взаимодействия

Для создания внутреннего сервера использован язык Java ⁵. Java представляет собой язык программирования и платформу вычислений, которая была впервые выпущена Sun Microsystems в 1995 г. Приложения Java транслируются в специальный байт-код, поэтому они могут работать на любой виртуальной Java-машине вне зависимости от компьютерной архитектуры.

Программы на Java транслируются в байт-код, выполняемый виртуальной машиной Java (JVM) — программой,

⁵ <http://www.java.com/>

обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор.

Достоинством подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина.

Основной модуль конструктора Lego Mindstorms EV3 может работать с прошивкой leJOS, позволяющей запускать Java-приложения. Специально для этого Oracle, которой на данный момент принадлежат права на Java, выпустил и поддерживает отдельную версию полноценной Java Standard Edition Embedded. Таким образом, блок EV3 может запускать java-приложения.

Разработка приложения внутреннего сервера ConnectBot велась в среде Eclipse ⁶. Eclipse – свободная интегрированная среда разработки, leJOS предоставляет плагин для работы с ней, позволяющий загружать и запускать разработанные программы.

Для разработки внешнего сервера использован язык Python⁷. Python — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода.

Разработка клиентской части приложения производилась на следующих языках:

HTML (от англ. HyperText Markup Language — «язык разметки гипертекста») — стандартный язык разметки документов во Всемирной паутине.

⁶ <https://eclipse.org/>

⁷ <http://www.python.org>

Javascript⁸ — это язык программирования, с помощью которого веб-страницам придается интерактивность. С его помощью создаются приложения, которые включаются в HTML-код.

С помощью Javascript можно изменять страницу, изменять стили элементов, удалять или добавлять теги. С его помощью можно узнать о любых манипуляциях пользователя на странице (прокрутка страницы, нажатие любой клавиши, клики мышкой, увеличение или уменьшение рабочей области экрана). С его помощью, можно к любому элементу HTML-кода получить доступ и делать с этим элементом множество манипуляций. Javascript позволяет загружать данные не перезагружая страницу, выводить сообщения, считывать или устанавливать cookie и выполнять множество других действий. При разработке веб-интерфейса SenseBot, язык JavaScript был использован для динамического обновления элементов интерфейса.

Связь между подсистемами осуществляется в формате **JSON**.

JSON⁹ (JavaScript Object Notation) — простой формат обмена данными, удобный для чтения и написания как человеком, так и компьютером. JSON — текстовый формат, полностью независимый от языка реализации, но он использует соглашения, знакомые программистам C-подобных языков, таких как C, C++, C#, Java, JavaScript, Perl, Python и многих других. Эти свойства делают JSON идеальным языком обмена данными.

2.1.2. Библиотеки и компоненты

□ Flask

Flask¹⁰ является микрофреймворком для создания веб-сайтов на языке Python. Основная причина почему Flask называется «микрофреймворком» — это идея сохранить ядро простым, но расширяемым. В нем нет абстрактного

⁸ <http://www.w3schools.com/js/>

⁹ <http://www.json.org/>

¹⁰ <http://flask.pocoo.org/>

уровня базы данных, нет валидации форм или всего такого, что уже есть в других библиотеках к которым вы можете обращаться. Однако Flask поддерживает расширения, которые могут добавить необходимую функциональность и имплементирует их так, как будто они уже были встроены изначально.

□ SimpleWebSocketServer

Протокол WebSocket (стандарт RFC 6455) протокол полнодуплексной связи поверх TCP-соединения, предназначен для снятия ограничений обмена данными между браузером и веб-сервером в режиме реального времени, позволяет пересылать данные безопасно, без лишнего сетевого трафика. SimpleWebSocketServer¹¹ — веб-сокет-сервер для Python.

□ Bootstrap

Twitter Bootstrap¹² — набор инструментов для создания сайтов и вебприложений. Включает в себя HTML и CSS шаблоны оформления для типографики, веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейсов, включая JavaScript расширения.

Bootstrap содержит все необходимые инструменты для создания качественных и удобных пользовательских интерфейсов. С помощью использования шаблона Bootstrap, пользовательский интерфейс может масштабироваться под любой формат устройства.

□ CodeMirror

Редактор кода работает благодаря библиотеке CodeMirror¹³. CodeMirror

¹¹ <https://github.com/opiate/SimpleWebSocketServer>

¹² <http://getbootstrap.com/>

¹³ <http://codemirror.net/>

— это Javascript библиотека, которая превращает textarea в подобие IDE, с подсветкой синтаксиса, табуляцией, автоотступами, автодополнением и другими полезными функциями.

□ **jQuery** jQuery ¹⁴ – это JavaScript-библиотека, фокусирующаяся на взаимодействии JavaScript, HTML и CSS.

JavaScript является фундаментальным языком, и чтобы создать всплывающее окно, разворачивающийся список – потребуется много времени и программного кода. Библиотека jQuery служит, чтобы уменьшить и облегчить работу разработчика. jQuery использует краткий синтаксис с переменными в форме селекторов CSS для эффективного подключения к любым целевым элементам DOM¹⁵. Поскольку библиотека jQuery — это JavaScript, она может быть встроена в любой проект, в котором используется JavaScript.

□ **RequireJS**

Классический способ подгрузки JS файлов — использование `<script>` тегов. Несмотря на то, что этот способ является самым простым и популярным, он имеет ряд минусов:

- список загружаемых скриптов определяется в html-файле, тогда как основной код пишется в файлах JS – такое разделение приоритетов не всегда удобно;
- подгружаемые библиотеки «засоряют» глобальное пространство имен;
- зачастую при подключении библиотек важно соблюдать определенный порядок загрузки, иначе код может перестать работать.

¹⁴ <http://jquery.com/>

¹⁵ DOM (Document Object Model) - объектная модель, используемая для XML/HTML-документов.

Таким образом для эффективной работы нужен другой подход – более динамичный, подходящий для сложных, разветвленных проектов с большим числом зависимостей. Такое решение предоставляет библиотека RequireJS¹⁶.

2.3. Техническое описание структуры приложения

В этом параграфе будет представлена архитектура приложения и компонентов, которые оно использует.

2.3.1. Внешний сервер SenseBot

SenseBot – это клиент-серверное веб-приложение для программирования и наблюдения за датчиками EV3 в реальном времени, включающее в себя вебсервер и пользовательский интерфейс. SenseBot использует готовую библиотеку `ev3-python`¹⁷, чтобы управлять блоком. В этом параграфе будет представлено

проектирование и внедрение различных компонентов в веб-приложение.

Работа SenseBot строится по следующей схеме: несколько блоков EV3 и несколько пользователей могут быть присоединены к одному компьютеру-серверу (рис.10). Пользователи могут подключаться к серверу, используя веббраузер. Через веб-интерфейс пользователь получает полный контроль над вебсервером, его ресурсами и интеллектуальными блоками. Веб-приложение позволяет добавлять, подключаться, следить за работой датчиков и программировать блоки EV3.

Веб-сервер разработан на языке Python, в то время как веб-интерфейс реализован с использованием HTML, CSS и JavaScript. С точки зрения пользователя, веб-сервер невидим, а веб-интерфейс дает прямой доступ к EV3. На самом же деле, все коммуникации проходят через веб-сервер. В

¹⁶ <http://requirejs.org/>

¹⁷ <https://github.com/topikachu/python-ev3>

вебинтерфейсе ничего не запускается напрямую, все коммуникации транслируются в форме JSON-запросов на веб-сервер. Это относится и к коду, написанному в веб-интерфейсе.

Теоретически, пользователи могли бы получать прямой доступ к EV3 без необходимости обращаться к веб-серверу. Но это не было реализовано, потому что веб-страница имеет ограниченные ресурсы, а javascript, запускаемый с веб-страницы, не дает полный доступ к ресурсам компьютера. При подключении через java-приложение доступно лишь одно соединение, а через веб-сервер становится возможна совместная работа нескольких пользователей на одном EV3.

Таким образом, архитектура приложения SenseBot является модульной. Расширение списка поддерживаемых роботов может быть достигнуто добавлением набора библиотек, используемых веб-сервером.

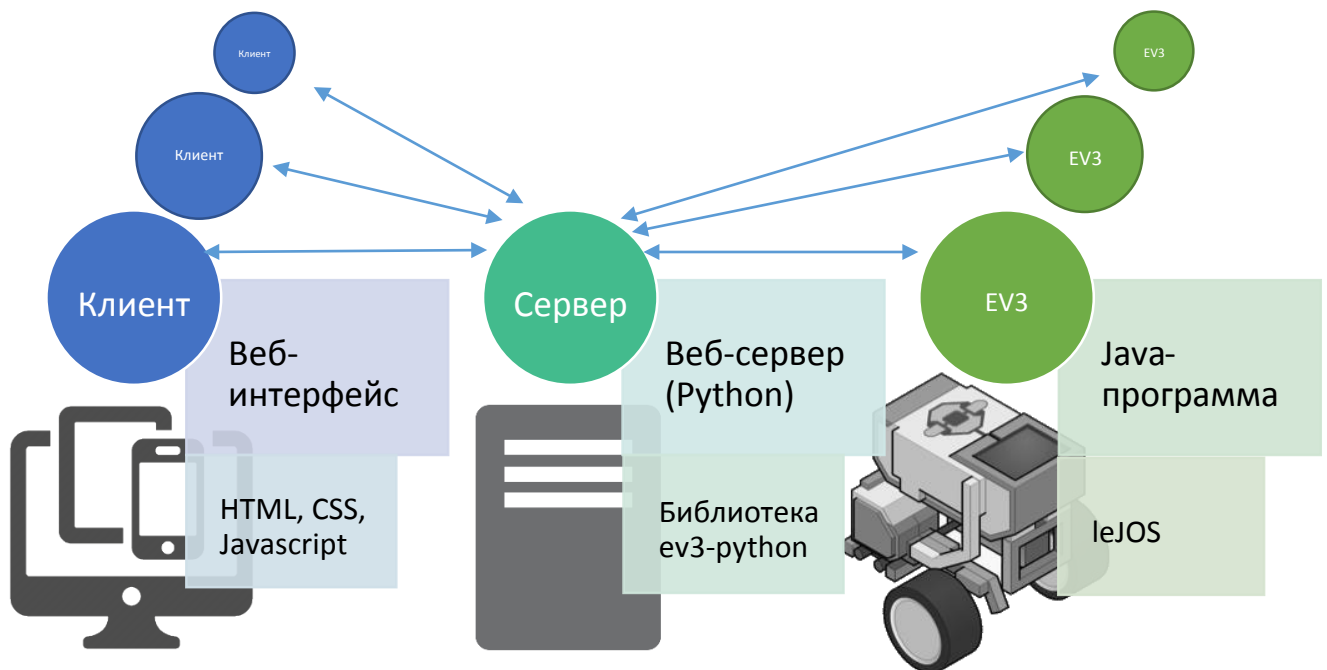


Рисунок 10. Общая схема работы приложения.

Библиотека `ev3-python` позволяет пользователю выполнять соединение и осуществлять контроль над EV3, используя язык программирования Python. С помощью библиотеки, команды языка Python интерпретируется на EV3, таким образом, пользователь может выполнять команды на присоединенном блоке.

На рис.11 схематично изображена архитектура библиотеки, связанной с блоком EV3. В архитектуре так же используется клиент-серверная модель, где блок EV3 выступает в качестве сервера, а библиотека является клиентом. Внешний доступ к ресурсам блока происходит при помощи программы `ConnectBot`. `ConnectBot` – серверное приложение, разработанное на языке Java в рамках данной работы. Оно запускается на операционной системе Linux при помощи оболочки `leJOS`.

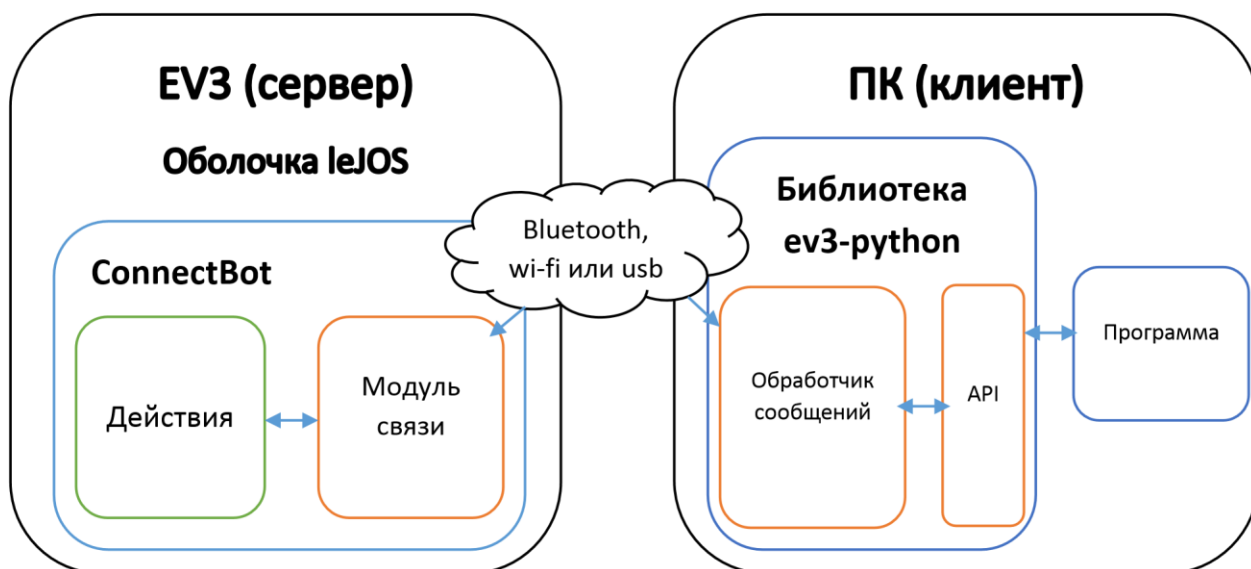


Рисунок 11. Архитектура работы библиотеки `ev3-python`

Посылая запросы JSON, пользователь на самом деле не программирует сам блок – подключенный EV3 только следует командам, отдаваемым ему через запущенную программу `ConnectBot`. Архитектура создает только иллюзию программирования EV3 на языке Python. Это сделано по причине отсутствия оболочек, позволяющих программировать блок непосредственно на Python. Тем не менее, достигнутый результат совершенно не отличается от программирования блока напрямую. Кроме того, использование промежуточной

программы ConnectBot перекладывает задачу компиляции и вычислений на сравнительно мощный персональный компьютер, вместо выполнения этих действий на слабом блоке.

2.3.2. Внутренний сервер ConnectBot

ConnectBot, программа для блока EV3, разработана с помощью языка программирования Java и leJOS. LeJOS используется для управления действиями EV3. Приложение является внутренним сервером, согласно трехуровневой архитектуре, описанной в параграфе 2.1.

Внешние приложения имеют возможность подключаться к ConnectBot, используя три различных канала: USB, Bluetooth или wi-fi, используя соответствующего сокета. Сокеты могут использовать либо Transmission Control Protocol (TCP)¹⁸ для связи по Wi-Fi или USB, либо протокол RFCOMM¹⁹ для связи через Bluetooth. Оба протокола обеспечивают надежный обмен связи между клиентом и ConnectBot.

После подключения внешнего приложения к блоку, оно может контролировать EV3 с помощью сообщений, посылаемых в формате JSON. Обмен данными осуществляется в парах запрос-ответ, где каждый запрос от клиента будет получать ответ от ConnectBot. Для каждого действия, которое клиентское приложение хочет выполнить в ConnectBot, существует пара запросответ.

¹⁸ <http://www.lissyara.su/doc/rfc/rfc793/> – RFC793 Протокол управления пересылкой

¹⁹ <https://developer.bluetooth.org/TechnologyOverview/Pages/RFCOMM.aspx>

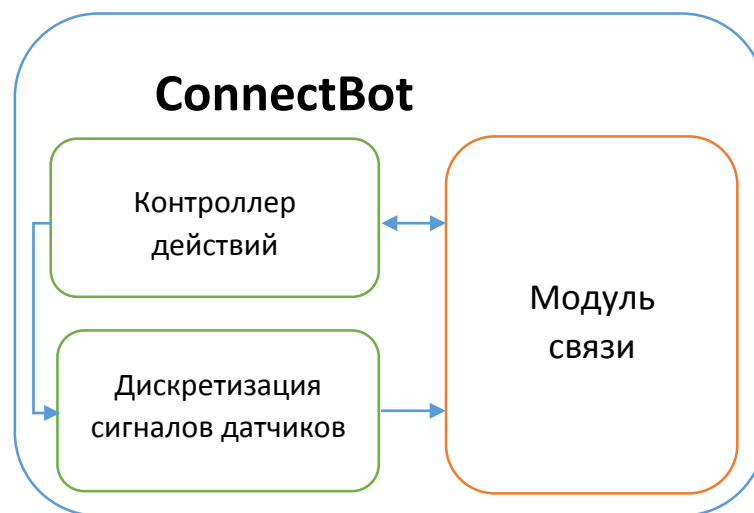


Рисунок 12. Схема коммуникации ConnectBot

На рис.12 представлено устройство программы ConnectBot. Сообщения, полученные от клиента, принимаются в модуле связи ConnectBot. После обработки сообщения, в зависимости от его содержания, вызывается один из контроллеров действий. Контроллеры действий управляют датчиками, двигателями и т.д. После того, как действие завершится, ответ отправляется обратно клиенту, сообщая ему, прошло ли все как ожидалось, или нет. Поток дискретизации сигналов датчиков автоматически применяет полученные данные на добавленные в веб-интерфейсе датчики. Сервис будет ожидать их подключения, если датчики не будут доступны.

Как уже было отмечено ранее, ConnectBot может иметь одновременно соединение только с одним клиентом (в данном случае, клиентом будет являться веб-сервер). Клиент должен поддерживать соединение с ConnectBot до тех пор, пока требуется получать доступ к EV3. Это необходимо для увеличения скорости выполнения некоторых операций. Операции типа добавления датчика могут занять много времени, и однажды добавленный датчик будет работать все время, пока программа запущена.

ConnectBot не сохраняет информацию после разрыва соединения с EV3. Приложение рассматривает каждое разъединение как перезагрузку системы,

поэтому все значения в памяти сбрасываются до повторного подключения клиента.

Для того, чтобы облегчить использование ConnectBot, были предусмотрены некоторые настройки. Когда ConnectBot не запущен и ни один клиент не подключен к нему, ConnectBot транслирует свое «свободное» состояние в виде мигания оранжевого светодиода в EV3. Когда клиент подключен – загорается зеленый цвет.

2.3.3. Связь компонентов

Для обмена сообщениями JSON, используется библиотека Gson²⁰. Gson – java-библиотека, предназначенная для сериализации и структуризации JSON в объекты Java и обратно. Использование библиотеки облегчает использование JSON в Java.

Базовый шаблон JSON-запроса рассмотрен в Приложении 1. Листинг 1.1 показывает минимальное требуемое сообщение, необходимое для корректного запроса. Дополнительные пары атрибут-значение могут быть использованы в зависимости от типа посылаемого запроса.

Ответ от Java-приложения на запрос всегда выглядит так, как показано в Приложении 1, листинг 1.2. Иных связок атрибутов и значений в ответе не предусмотрено, но некоторые пары могут быть опущены по причине отсутствия в них информации.

Порядковый номер – метод приложения ConnectBot, который позволяет многократным потокам данных запускаться на одном сожете. Так, каждый запрос, отправленный к ConnectBot, должен содержать порядковый номер. Тот же самый порядковый номер используется и в ответе на запрос.

²⁰ <https://code.google.com/p/google-gson/>

Следующий компонент – библиотека `Ev3-python`. Данная библиотека, которая дает пользователям программный контроль над EV3 через Java-приложение.

Автор библиотеки поясняет, что выбрал язык Python, потому что он является динамичным, интерактивным, производительным языком. Использование Python позволяет легко проводить эксперименты с подключенным блоком.

Базовая функция библиотеки – использование API для подключения и контроля над блоком. Все коммуникации между библиотекой и блоком EV3 в первую очередь проходят через асинхронный обработчик сообщений. Обработчик сообщений запускается в отдельном потоке и его задача – отправка и получение сообщений между библиотекой `ev3-python` и java-приложением. Асинхронный обработчик сообщений отслеживает, какие запросы ждут ответа, а какие можно заблокировать до получения ответа. Любое действие, которому нужно получить ответ от EV3 прежде чем продолжить работу (например, добавление датчиков) будет заблокировано до тех пор, пока не будет получен ответ. Некоторые действия, которые не требуют ответа, такие как вращение мотора и т.п., используют немедленные вызовы. Немедленные вызовы не ждут ответа и сразу приступают к исполнению.

Кроме того, у пользователя есть возможность назначить подписку на два разных потока данных. В этом случае API отправит сообщение `ConnectBot` для запуска потоковых услуг. Полученные потоковые сообщения будут непрерывно обрабатываться в асинхронном обработчике сообщений, который в свою очередь, отправит эти сообщения к модулю подписки. Модуль подписки отвечает за создание обратных вызовов назад к программе, когда получены потоковые сообщения разного типа.

На Листинге 1 отражено, как библиотека используется для подключения к EV3 и получению значений датчика цвета: после того как датчики были открыты, режим отображения цвета был выбран в обеих версиях, можно выводить на экран значения. Каждый сенсор библиотеки имеет свои собственные режимы работы. Данная часть кода показывает, как значения датчика цвета могут быть отображены на экране.

```
import ev3 ev3brick =
ev3.connect_to_brick(address='192.168.1.7')
colorsensor = ev3.EV3ColorSensor(ev3brick, port=3)
color = colorsensor.get_color_id_mode() for _ in
range(0, 4):
print color.get_color_id() # вывод номера цвета
```

Листинг 1. Чтение значения датчика цвета

Листинг 2 показывает, как использовать библиотеку для управления моторами. В программном коде подключаются два мотора – в портах А и В. Вызывается функция для движения вперед. После окончания движения, программа получает значения датчика угла поворота мотора.

Библиотеку так же можно использовать для непрерывного получения значений датчика. На листинге 3 показано, как можно «подписаться» на обновление значений гироскопического датчика.

```
import ev3 import time ev3brick =
ev3.connect_to_brick(address='192.168.1.7') port_a =
ev3.Motor(ev3brick, port='A') port_b =
ev3.Motor(ev3brick, port='B') port_a.forward()
port_b.forward() time.sleep(2) # движение вперед в
течение двух секунд print port_a.get_position(),
port_b.get_position()
```


Листинг 2. Управление моторами

```
import ev3 import time ev3brick =
ev3.connect_to_brick(address='192.168.1.7') gyro_sensor =
ev3.EV3GyroSensor(ev3brick, 4) def print_samples(data): print
data sub = ev3.Subscription( subscribe_on_sensor_changes=False,
subscribe_on_sample_data=True)
sub.subscribe_on_samples(print_samples)
brick.set_subscription(sub)
time.sleep(4) # получение значений в течение 4-х секунд
```

Листинг 3. Пример подписки на значения датчика

Таким образом, для того чтобы пользователь мог считать значение датчика или написать программный модуль, необходимо пройти цепочку из взаимосвязанных компонентов. Например, нажимая кнопку подключения датчика в веб-интерфейсе SenseBot, HTTP-запрос обрабатывается веб-серверов. Веб-сервер посылает JSON-запрос к приложению ConnectBot, которое запускает поток получения значений датчика с микроконтроллера EV3. Пройдя обратный путь, значения начинают отображаться в пользовательском интерфейсе.

2.4. Логическая структура приложения

Внутренний сервер ConnectBot состоит из двух основных частей – одна отвечает за установку соединения, другая – за обработку и выполнение действий, запрошенных внешним сервером.

Главный класс приложения ConnectBot – Startup. В нем, как и в остальных классах подключены классы из коллекции Ev3classes, которая предоставляется вместе с LeJOS. Класс Startup – начальная точка входа в приложение, он отвечает за работу с экраном и подсветкой блока EV3, парсинг и выполнение JSON запросов (см. Прил.3). При обработке запросов используются методы библиотеки Gson *toJSON()* и *fromJson()*.

Следующий важный класс – `Communication`, работа с ним начинается при запуске метода `startUpCommunication()` класса `Startup`. Класс `Communication` отвечает за установку соединения. Для работы сервера выделяется порт. К этому порту должен будет обращаться клиент. Клиент для связи с портом хоста, который соединен в свою очередь с нужным сервером, создает сокет²¹.

Для создания сокетов и управления ими в Java есть специальные классы `java.net.Socket` и `java.net.ServerSocket`. Первый для клиента, второй для сервера. Клиентский сокет представляет собой конечную точку для связи между двумя машинами. Серверный сокет ожидает запросы, приходящие по сети от клиентов и может, при необходимости отправлять ответ.

Соединение может осуществляться по беспроводным каналам связи `wifi` и `Bluetooth`, за это отвечают классы `TCPThread` и `BTThread` соответственно.

Получение значений датчиков описано двумя классами – `SensorControl` и `RecieveSample`. Первый из них осуществляет работу по открытию портов и установления соответствия с присоединенными датчиками, контролирует количество их количество, определяет какие датчики перестают отвечать. Класс `SampleThread` работает с интерфейсом `SampleProvider`, который предоставляет `leJOS`. Интерфейс предназначен для получения сэмплов — значений, получаемых с датчика в отдельный момент времени. Сэмпл может состоять из одного или нескольких элементов, это зависит от типа датчика и режима его работы.

Класс `MotorControl` представляет собой набор стандартных методов, таких как `forward()` и `backward()` – движение вперед и назад, для работы с подключенными моторами. (см. Прил.3).

²¹ Сокет (англ. socket — разъём) — название программного интерфейса для обеспечения обмена данными между процессами.

На Рис.13. изображена логическая структура, иллюстрирующая отдельные классы приложения ConnectBot и связь между ними.

SenseBot

Приложение SenseBot состоит из четырех компонентов: веб-сервер, сервер веб-сокетов, модуль управления блоком, модуль управления блоком (рис.14).

Веб-сервер SenseBot запускается выполнением скрипта `server.py`. SenseBot позволяет пользователям подключаться к статическим файлам вебинтерфейса, используя браузер. Большое количество коммуникаций вебприложения требует от клиента установку стабильного соединения с SenseBot через веб-сокеты.

Работу веб-сервера обеспечивает микрофреймворк Flask. Когда клиенты выбирают один из блоков EV3, происходит подключение к серверу веб-сокетов (используется SimpleWebSocketServer). После этого все коммуникации между данным клиентом и SenseBot, такие как получение значений датчиков и редактирование кода, проходят через веб-сокеты.

Основное назначение модуля управления блоком – мониторинг клиентов, подключенных к блокам. Можно сказать, это модуль маршрутизации, который следит за тем, чтобы запросы и ответы достигали нужных получателей.

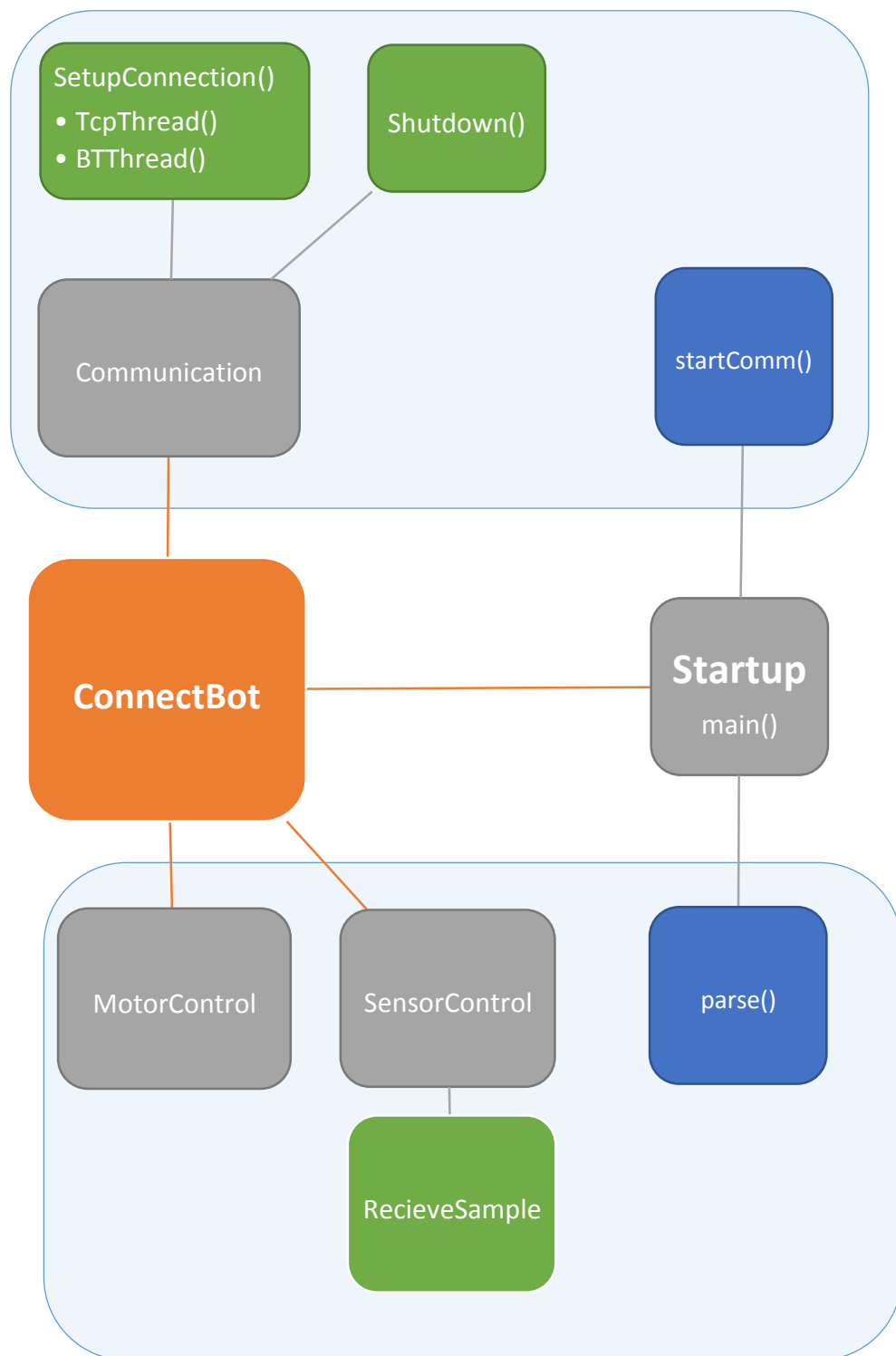


Рисунок 13. Логическая структура ConnectBot

Модуль управления блоком обновляет программный код у подключенных пользователей. Для оптимизации пропускной способности между внешним сервером и клиентом было создано правило, согласно которому менеджер блока не посылает значения датчиков, если оно не изменилось по сравнению с предыдущим.

Модуль управления кодом предназначен для запуска модулей поведения. Для непрерывного обновления и запуска модулей используется библиотека Subsumption. Модуль управления кодом обрабатывает модули и выполняет их используя выражение *exec* (см. Прил.3).

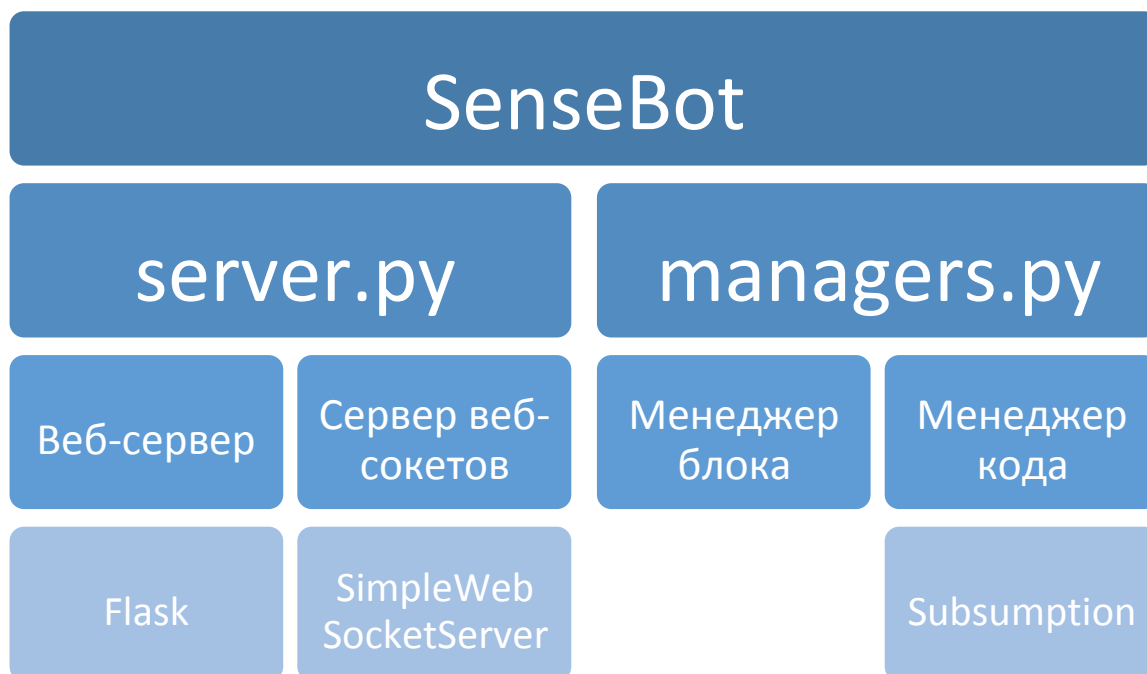


Рисунок 14. Логическая структура приложения SenseBot

ГЛАВА 3. ВНЕДРЕНИЕ И ИСПОЛЬЗОВАНИЕ ПРИЛОЖЕНИЯ SENSEBOT

3.1. Ввод приложения в эксплуатацию

Этап введения приложения в эксплуатацию является одним из самых ответственных этапов в цикле производства веб-приложения. В конечном счете, именно от него зависят доступность приложения, сохранность исходных кодов и надежность приложения в целом.

Для использования веб-приложения в рамках одной организации, достаточно использовать корпоративную сеть. Требования, предъявляемые к оборудованию клиента:

Одна из операционных систем:

- Windows XP и выше;

- Любой дистрибутив Linux с поддержкой графического режима; □ MacOS X и выше.

Интернет проводник (браузер) поддерживающий:

- HTML5;
- JavaScript.

Схема сетевой работы приложения изображена на рисунке 15.



Рисунок 15. Общая схема с сетевыми адресами

В данном случае, клиент и сервера находятся в разных сегментах корпоративной сети, требуется настроить маршрутизацию на клиентской компьютере или на маршрутизаторе, который использует клиент.

Для этого в командной строке Windows необходимо ввести следующую команду, тем самым добавив сетевой путь:

```
route -p add 172.16.0.0 mask 255.255.0.0 172.21.16.1
```

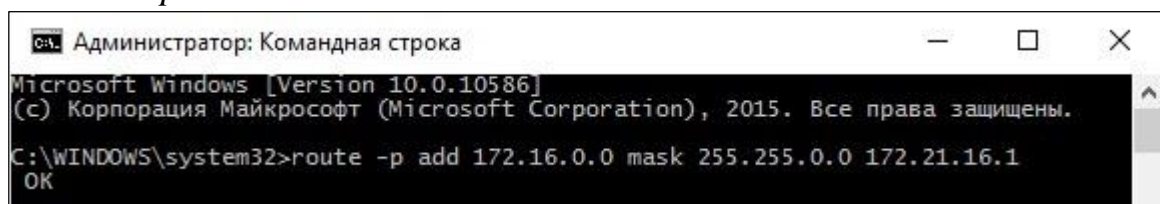


Рисунок 16. Добавление статических маршрутов

После применения, клиент, имеющий ip-адрес 172.21.106.50 сможет получить доступ к веб-приложению, запущенному на сервере 172.16.5.244 и работать с блоками EV3, запущенными в сети 172.16.5.0.

Очень часто интернет-провайдеры при подключении к сети предоставляют внешний динамический ip-адрес. Подавляющему большинству пользователей этого достаточно, однако для доступа к своему компьютеру из вне, необходим внешний статический адрес. Эту услугу предоставляют далеко не все провайдеры, а если и предоставляют, то за дополнительную плату. Для получения доступа к веб-приложению через глобальную сеть предлагается использовать сервисы, предлагающие технологию динамического DNS (DDNS²²), позволяющую связать внешний динамический ip-адрес и постоянное доменное имя.

Сервис DDNS состоит из двух компонентов. Первый — специальное ПО, работающее на удаленном компьютере. Второй — клиентская программа, устанавливаемая на рабочее место. Первый компонент отвечает за взаимодействие со своим DNS, настройкой и поддержкой пользовательских аккаунтов. Клиентская часть обеспечивает связь с серверной, передает ей текущее значение выделенного для данного соединения адреса, обеспечивает некоторые дополнительные настройки. При изменении адреса, сведения, записанные в DDNS, обновляются, после этого набрав в строке браузера адрес домена, предоставленного сервисом, пользователь попадет на ваш компьютер.

Конечной целью размещения приложения в сети Интернет является создание полностью работоспособной копии приложения, которая может бесперебойно предоставлять услуги по средствам сети Интернет.

3.1.1. Настройка блока EV3

Прежде чем блок EV3 сможет использоваться в системе, требуется выполнить некоторые предварительные действия. Как описано в предыдущем

²² *Динамический DNS — технология, позволяющая информации на DNS-сервере обновляться в реальном времени, в автоматическом режиме. Она применяется для назначения постоянного доменного имени устройству (компьютеру, сетевому накопителю) с динамическим IP-адресом.*

параграфе, для работы SenseBot необходимо использовать оболочку leJOS вместо стандартного встроенного программного обеспечения (рис.17).

С помощью leJOS запускается разработанное java-приложение ConnectBot, которое позволяет внешним системам подключаться и управлять EV3. Поэтому необходимо, чтобы leJOS и ConnectBot были запущены на EV3, после чего они могут использоваться в системе.

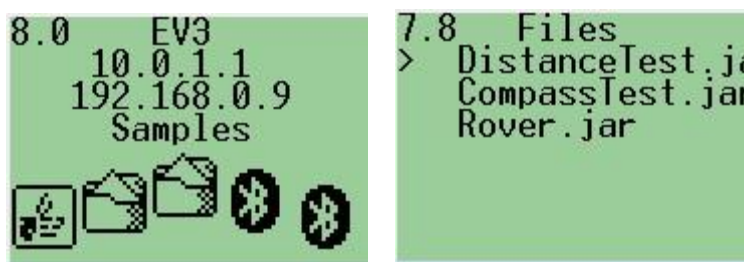


Рисунок 17. Интерфейс программной оболочки leJOS

Когда блок EV3 правильно настроен, платформа SenseBot может начать управление через библиотеку ev3-python.

Полный внешний контроль над ConnectBot осуществляется через ev3python библиотеку. Библиотека предоставляет пользователям API с более широкими возможностями, чем предлагаются в рамках программы ConnectBot. SenseBot использует библиотеку, чтобы управлять блоком ev3 через вебинтерфейс.

Для работы с системой нужно выбрать способ подключения к блоку EV3. Соединение может быть установлено через USB, Bluetooth или Wi-Fi. Если использовать USB или Bluetooth для подключения, веб-приложение SenseBot должен быть запущен в непосредственной близости с EV3. При подключении через wi-fi, блок EV3 и SenseBot могут располагаться в любом месте, до тех пор, пока соединение с EV3 доступно. Для возможности работать по wi-fi к блоку EV3 требуется присоединить USB-адаптер беспроводной сети и выполнить аутентификацию к точке доступа.

3.2. Описание пользовательского интерфейса

SenseBot позволяет пользователю осуществлять контроль над роботом через веб-интерфейс. В веб-приложение можно войти откуда угодно, используя веб-браузер, но пользователю необходимо иметь визуальный контакт с роботом. Если использовать веб-камеру, которая будет направлена на робота, то, имея удаленный доступ к ней, можно говорить о том, что SenseBot получится использовать из любой точки с доступом в глобальную сеть.

На рис.18 можно видеть вкладку браузера с доступом к системе. На снимке экрана отображается пустая страница с панелью навигации вверху, кнопки позволяют пользователю добавить блок EV3 в систему по его адресу в сети и выполнить соединение. Пользователь может иметь одновременно соединение только с одним блоком. Если требуется отображать информацию с разных блоков в одно и то же время, можно создать новую вкладку в браузере.

После выполнения операции присоединения блока EV3 к серверу, вебприложение готово к сбору информации и управления роботом.

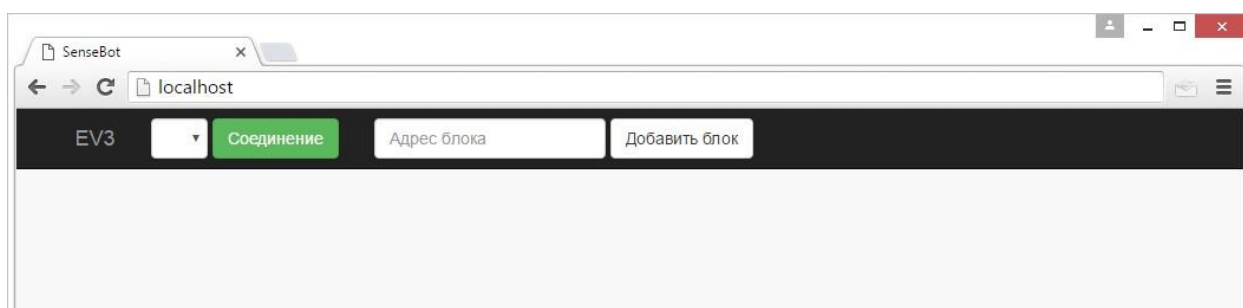


Рисунок 18. Шапка страницы приложения

На данном этапе пользователь может получить доступ к основным функциям: считывание данных с датчиков в реальном времени, разработка и запуск модулей управления роботом. К сожалению, пользователю придется самостоятельно выбирать датчик и порт, к которому он присоединен, для отображения информации. Подсистема автоматического распознавания датчиков в EV3 очень ограничена – многие датчики используют одни и те же драйверы и практически не представляется возможным определить какой тип

присоединен в данный момент. По этой причине, пользователю предлагается нажать кнопку «Открыть датчик», выбрать его тип и номер порта на блоке EV3.

Для добавления новых модулей управления роботом пользователю нужно ввести название создаваемого модуля и нажать кнопку «Открыть».

Добавленные на экран датчики и программные модули сохраняются в памяти SenseBot. Новые пользователи, подключенные к EV3, будут видеть все сохраненные данные. Кроме того, если один пользователь добавит новый или изменит существующий модуль управления, все остальные пользователи немедленно узнают об этом. В тот момент, когда EV3 отключается от вебприложения, вся информация и изменения удаляются из памяти, а подключенные веб-интерфейсы получают уведомление об отключенном блоке. Основная причина, почему все наработки удаляются из памяти – отсутствие точного способа убедиться, что вновь подключенный по данному сетевому адресу блок является тем же, что использовался прежде: название блока, его идентификатор и ip-адрес могут изменяться.

Веб-интерфейс с отображением значений датчиков и открытой областью для написания модулей показан на рис.19. Значения открытых датчиков отображаются в больших прямоугольниках в левой части. Вверху каждого прямоугольника выводится полное название датчика, порт, к которому он присоединен, и режим, в котором работает датчик. Чуть ниже, отображается последнее полученное с датчика значение. Для сокращения коммуникаций, только изменения посылаются из SenseBot в веб-интерфейс.

Если пользователь хочет получить контроль над другим блоком EV3, он должен в первую очередь разорвать текущее соединение. Это позволяет сделать соответствующая кнопка в верхней панели.

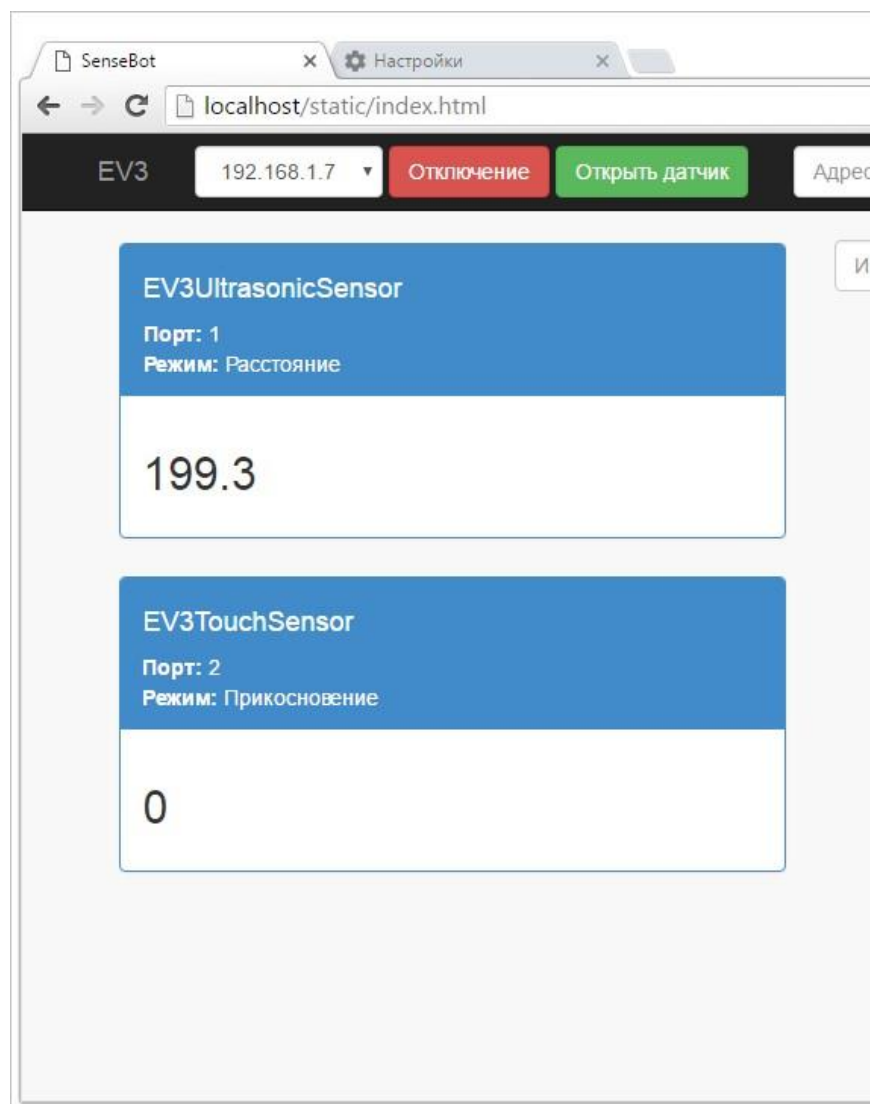


Рисунок 19. Область отображения значений датчиков

Рисунок 20 демонстрирует этап, когда добавлены два модуля поведения. Пользователь может добавить новый модуль, присвоив ему имя в поле для ввода, располагаемого чуть ниже панели навигации. Для удаления модуля, нужно нажать на крестик справа от имени модуля. Ниже списка модулей располагается редактор кода для выбранного модуля. Текущий модуль может иметь темносиний или темно-зеленый цвет в зависимости от состояния. Модуль управляется тремя кнопками: Сохранить, Пуск, Стоп. Кнопка сохранения записывает все изменения для модуля на сервер. Кнопка запуска активирует модуль на сервере.

Кнопка остановки останавливает все модули на сервере. При повторном нажатии кнопки Пуск, все модули продолжают работать с момента остановки.

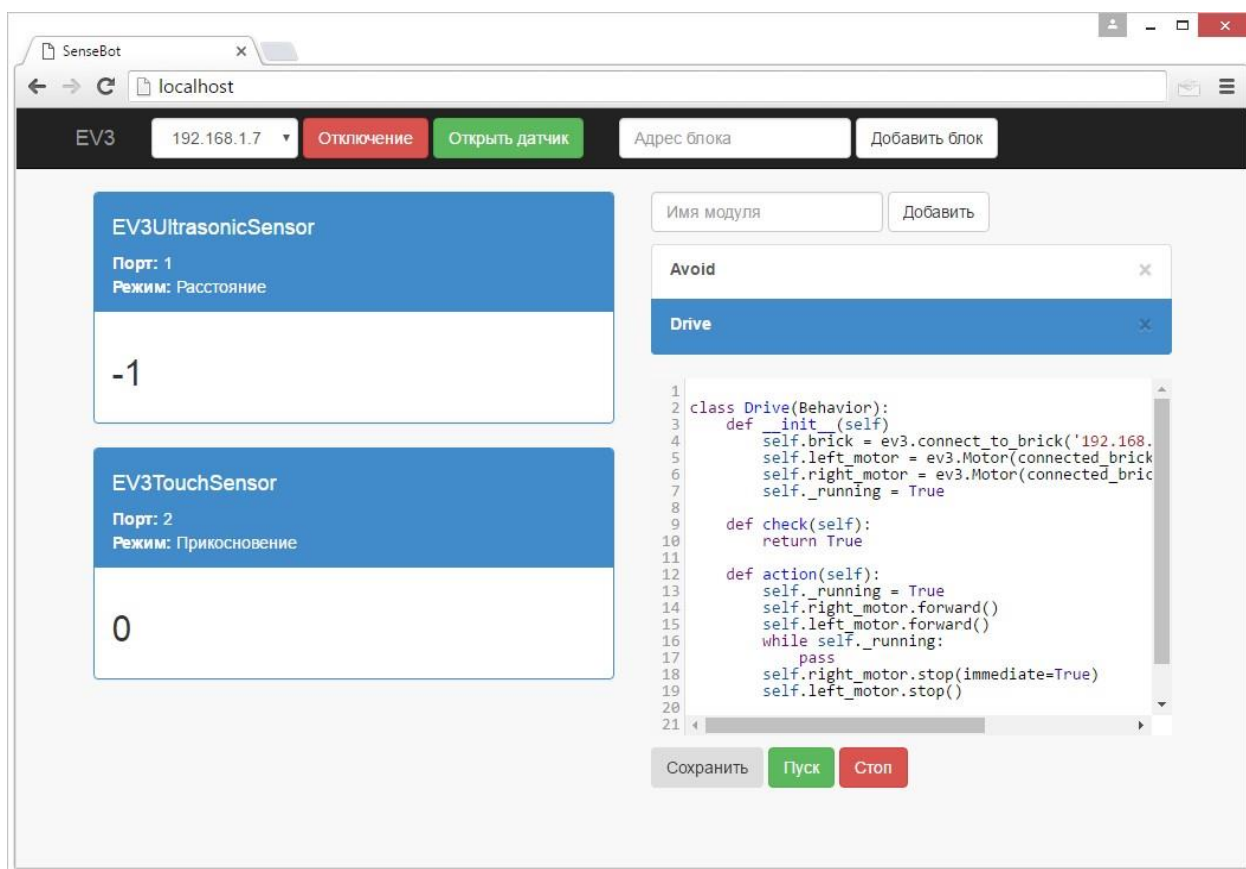


Рисунок 20. Пользовательский интерфейс во время работы с EV3

Стороннему наблюдателю может показаться, что модули исполняются в веб-интерфейсе, на самом же деле, код пересылается на SenseBot, компилируется и запускается.

3.3. Инструкция по программированию в SenseBot

Очередность модулей определена *категориальной архитектурой* (описана в параграфе 1.5) с небольшими изменениями для простоты использования. Вместо параллельного запуска, все модули запускаются по очереди. Очередь формируется в зависимости от приоритета. Модули, расположенные выше, имеют больший приоритет. Очередность может измениться, несмотря на то, что уже запущенный модуль не завершил свою работу. В тот момент, когда модуль с приоритетом выше текущего хочет

запуститься, модуль выполняемый в данный момент получает сигнал. В этом случае, модуль должен отдать контроль над роботом и позволить завестись модулю с приоритетом выше.

Модуль поведения должен быть записан на языке Python в виде, представленном на Листинге 4. для возможности запуска. Модуль поведения представляет собой класс, который состоит из трех инструкций: `check`, `action`, `suppress`.

```
Class SomeName(Behavior):
    Объявление переменных класса, инициализация датчиков и
    моторов

    Def check(self):
        Хочет ли этот модуль начать работу? Для запуска должен
        вернуть истинный результат.

    Def action(self):
        Действия модуля при его запуске.

    Def suppress(self):
        Должен остановить работу метода action() и передать
        контроль модулю с приоритетом выше.
```

Листинг 4. Структура модуля поведения

Метод *check* проверяет текущее положение модулей, должен ли модуль запускаться или нет. Обычно это происходит с помощью проверки значений доступных датчиков, от которых зависит запуск модуля.

Метод *action* целиком состоит из действий, заданных для робота. Чаще всего, программный код управляет моторами и делает различные действия, такие как поворот колес вперед и пр. Модуль управляет роботом пока все действия метода *action* не будут выполнены.

Метод *suppress* вызывается, когда модуль должен остановить свою работу. Этот метод должен остановить любое запущенное в данный момент действие и выполнить выход из метода *action*. При выходе из метода *action*,

модуль теряет контроль над роботом и позволяет другим очередным модулям запуститься.

Ошибки, обнаруженные в коде, отлавливаются и отображаются в вебинтерфейсе, без исправления ошибок программа не будет запущена. Примеры программирования модулей поведения приведены в Приложении 1, подробное руководство по использованию категориальной архитектуры можно найти на официальном сайте leJOS[7].

В качестве примера рассмотрена простая программа для движения вперед, обнаружения и объезда препятствий. Робот должен ехать вперед, поворачивать налево при опасности столкновения.

Для создания такого поведения потребуется создать два модуля – Avoid и Drive. Первый модуль имеет наибольший приоритет и имеет возможность запуститься всякий раз, когда это необходимо. Запускаться ему нужно в том случае, когда ультразвуковой датчик в передней части конструкции робота обнаружит что-либо на расстоянии ближе 25 см. Второй же модуль должен запускаться всегда, когда имеет это предоставляется возможным. Модель робота, использованного для написания примеров показана на Рис.21.

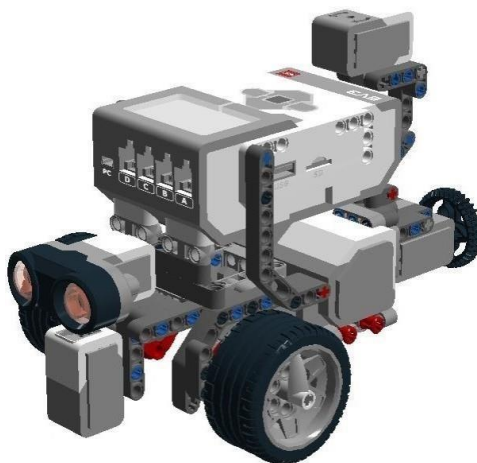


Рисунок 21. Модель робота, использованного в примерах Avoid – пер.с англ.: избегать. Данный модуль запускается в случае, если робот близок к столкновению с чем-либо впереди. Когда модуль получает

контроль, начинается поворот на 90°. Затем модуль завершает свою работу и другие модули могут запуститься. Если после поворота робот снова оказывается в ситуации близкой к столкновению, действия повторяются. Программный код модуля *Avoid* представлен на Листинге 5.

```
class Avoid(Behavior):
    def __init__(self):
        self._brick = ev3.connect_to_brick('192.169.1.3')
        self.left_motor = ev3.Motor(self._brick, ev3.
            MOTOR_PORTS.PORT_D) self.right_motor =
        ev3.Motor(self._brick, ev3.
            MOTOR_PORTS.PORT_A) self.ultrasonic =
        ev3.EV3UltrasonicSensor(self.
            _brick, ev3.SENSOR_PORTS.PORT_1).
            get_distance_mode()

    def check(self):
        distance = self.ultrasonic.fetch_sample()[0] if
        distance < 0.25 and distance != -1:
            return True

        return False

    def
action(self):
        self.left_motor.rotate(600, immediate_return=True)
        self.right_motor.rotate(-600)

    def
suppress(self):
        pass
```

Листинг 5. Модуль поведения Avoid

Модуль поведения *Drive* включает моторы для езды вперед. Он запускается всегда, когда это возможно, но имеет меньший приоритет по сравнению с модулем *Avoid*. Это означает, что модуль теряет контроль всегда, когда модуль избегания столкновений посылает сигнал. Перед тем, как передать контроль, движение останавливается. Код модуля *Drive* представлен на Листинге 6.

```

class Drive(Behavior):
    def __init__(self):
        self._brick = ev3.connect_to_brick('192.169.1.3')
        self.left_motor = ev3.Motor(self.brick, ev3.
            MOTOR_PORTS.PORT_D) self.right_motor =
            ev3.Motor(self.brick, ev3.
                MOTOR_PORTS.PORT_A)
        self._running = True
    def
    check(self):
        return True
    def action(self):
        self._running = True
        self.right_motor.forward()
        self.left_motor.forward() while
        self.running:
            pass
        self.right_motor.stop(immediate=True)
        self.left_motor.stop()
    def suppress(self):
        self.running = False

```

Листинг 6. Модуль поведения Drive

ЗАКЛЮЧЕНИЕ

В данной работе описывается архитектура, проектирование и разработка SenseBot – веб-приложения для программирования робота, основанного на поведенческих правилах. Приложение потребовалось создать в рамках работы над интегрированным курсом «Программирование в робототехнике». Благодаря веб-приложению, пользователи могут добавлять или редактировать существующие модули поведения, не прерывая при этом работу и действия робота.

SenseBot позволяет эффективно использовать мобильных роботов для обучения программированию на занятиях, экономя при этом значительное

количество учебного времени. Программирование роботов на занятиях курса стало более динамичным, а эффективность проведения занятий повысилась.

Веб-приложение было разработано для использования совместно с программируемым конструктором Lego Mindstorms EV3. В SenseBot реализована полнофункциональная среда для программирования на языке Python. В ходе тестирования определено, что веб-приложение выполняет все поставленные задачи.

Тематическое планирование курса приведено в приложении 1. Программа курса и разработанное приложение успешно прошли проверку методическим объединением по информатике и робототехнике города Тюмени, рекомендованы к использованию в образовательных целях в ГАОУ ТО «Физико-математическая школа».

В перспективе планируется доработка приложения и модификация интегрированного курса для обеспечения возможности использования дополнительных роботизированных платформ.

СПИСОК ЛИТЕРАТУРЫ

1. Рейнерс П. Роботы, лабиринты и архитектура поглощения [Электронный ресурс]. – Режим доступа: <http://www.ibm.com/developerworks/ru/library/j-robots/index.html?ca=drs-ru>
2. Люгер Д. Искусственный интеллект: стратегии и методы решения сложных проблем [Текст] / Пер.с англ – М.: Издательский дом «Вильямс», 2011. — 864 с.
3. Brooks R. Intelligence Without Reason, Proceedings of 12th Int. Joint Conf. on Artificial Intelligence, Sydney, Australia, pp. 569-595.
4. Сиерра К. Изучаем Java [Текст] / К. Сиерра, Б.Бейтс. – М.: Эксмо, 2012. — 720 с.

5. Лутц М. Изучаем Python [Текст] / – М.: Символ-плюс, 2011. — 1280 с.
6. Лоусон Б. Изучаем HTML5. Библиотека специалиста [Текст] / Б. Лоусон, Р. Шарп. – СПб.: Питер, 2011 — 272 с.
7. William J. Rust. Learning to Program in Java Using Lego™ Mindstorms® Robots And LeJOS. – MA: Syngress Publishing, 2013.
8. Програмируем работа LEGO Mindstorms EV3 на Java [Электронный ресурс]. - Режим доступа: <http://www.proghouse.ru/articlebox/55-lejos-ev3>. – (Дата обращения 04.06.2016).
9. Twitter Bootstrap: цикл статей [Электронный ресурс]. - Режим доступа: <http://itchief.ru/lessons/bootstrap-3/19-introduction-to-twitter-bootstrap-3>. – (Дата обращения 04.06.2016).
10. Документация микрофрейворка Flask [Электронный ресурс]. - Режим доступа: <https://flask-russian-docs.readthedocs.io/ru/latest/>. – (Дата обращения 04.06.2016).
11. Русская документация jQuery [Электронный ресурс]. - Режим доступа: <http://jquery-docs.ru/ajax/jquery-getjson/>. – (Дата обращения 04.06.2016).
12. Официальный сайт проекта leJOS: Программирование модулей поведения [Электронный ресурс]. - Режим доступа: <http://www.lejos.org/nxt/nxj/tutorial/Behaviors/BehaviorProgramming.htm>. – (Дата обращения 04.06.2016).
13. Форум сообщества leJOS [Электронный ресурс]. - Режим доступа: <http://www.lejos.org/forum>. – (Дата обращения 04.06.2016)

СПИСОК ИЛЛЮСТРАЦИЙ

Рисунок 1. Интеллектуальные блоки различных поколений	8
Рисунок 2. Короткая программа, написанная на стандартном языке EV3-G. ..	9
Рисунок 3. Электронные компоненты набора EV3	10
Рисунок 4. Встроенное программное обеспечение Lego Mindstorms EV3	11
Рисунок 5. Блок-схема, отражающая вид структурного программирования .	12
Рисунок 6. Схема программирования модулей поведения	15
Рисунок 7. Общая схема архитектуры системы	17
Рисунок 8. Схема работы клиентской части	18
Рисунок 9. Макет пользовательского интерфейса	19
Рисунок 10. Общая схема работы приложения.	25
Рисунок 11. Архитектура работы библиотеки ev3-python	26
Рисунок 12. Схема коммуникации ConnectBot	27
Рисунок 13. Логическая структура ConnectBot	34
Рисунок 14. Логическая структура приложения SenseBot	35
Рисунок 15. Общая схема с сетевыми адресами	36
Рисунок 16. Добавление статических маршрутов	37
Рисунок 17. Интерфейс программной оболочки leJOS	38
Рисунок 18. Шапка страницы приложения	39
Рисунок 19. Область отображения значений датчиков	41
Рисунок 20. Пользовательский интерфейс во время работы с EV3.....	42

ПРИЛОЖЕНИЯ

Приложение 1.

Тематическое планирование 18-недельного курса поведенческой робототехники:

Неделя	Тема занятия	Примечания
1	Что такое поведение? Основные парадигмы управления роботами	

2	Модели поведение животных	
3	Адаптивное поведение животного и робота	
4	Датчики как органы чувств	
5	Классическая архитектура, классические роботы	
6	Управление роботом с помощью поведенческих принципов	
7	Архитектура категоризации	
8	Проблемы архитектуры Брукса	
9	Колесные системы передвижения роботов	
10	Приложение SenseBot	
11	Язык программирования Python	
12	Переменные и константы	
13	Состояния и события. Сенсоры	
14	Циклы	
15	Программирование модулей поведения.	
16	Конструирование, программирование роботов	
17	Работа над проектом	
18	Презентации проектов	

Приложение 2.

```
{
  "cla" : "", // тип команды, обращение к мотору, датчику и пр.
  "cmd" : "", // команда
  "seq" : 0, // порядковый номер }
```

Листинг 1.1. Шаблон JSON-запроса

```
{
  "msg" : "", // тип ответа, ответное сообщение
  "data" : 0, // данные, полученные от моторов, датчиков и пр.
  "seq" : 0, // порядковый номер, такой же, как и в запросе
  // поля датчиков
  "sample" : [], // массив дискретных значений датчика
  "samples" : [ [], [], [], [] ], // массив всех значений }
```

Листинг 1.2. Примерный вид JSON-ответа

Приложение 3.

Класс «Startup» приложения ConnectBot

```
import com.google.gson.Gson; import
com.google.gson.GsonBuilder; import
lejos.hardware.Battery; import
lejos.hardware.Button; import
lejos.hardware.Sound; import
src.lib.BroadcastThread; import
src.motor.MotorControl; import
src.sensor.SensorControl; import
src.util.Request;
import src.util.Response;

import java.io.IOException;

public class Startup {

    public static void main(String[] args) throws IOException {

        System.out.println("ConnectBot started");
        Runtime.getRuntime().addShutdownHook(new Thread() {
public void run() {          Button.LEDPattern(0);
        }
    });
        communication = new Communication();
        Button.LEDPattern(9);
        communication.setUpConnection();
        Button.LEDPattern(1);
        while(running){
            try {
                command = communication.receive();
                if (command == null){
                    throw new IOException();
                }
            }
        }
    }
}
```

```

        }
        data = gson.fromJson(command, Request.class);
        response.reset();
response.seq = data.seq;
response.msg = "response";
parseAndRunCommand();
        communication.send(gson.toJson(response));
    } catch (Exception e) {
sensorControl.reset();
motorControl.closeMotors();
communication.close();
startUpCommunication();
        motorControl.openPorts();
    }
}
communication.shutdown();
}

static void parseAndRunCommand() throws IOException {
    if (data.cla.equals("motor")){
if (data.cmd.equals("forward")){
        motorControl.forward(data.motor_port);
    }
    else if (data.cmd.equals("backward")){
motorControl.backward(data.motor_port);
    }
    else if (data.cmd.equals("stop")){
motorControl.stop(data.motor_port, data.immediate);
.....
.....
.....
    else if(data.cmd.equals("subscribe_on_stream_data")){
sensorControl.startSampleThread();
    }
    else if(data.cmd.equals("close")){
        sensorControl.resetThreads();
    }
}
    else if (data.cla.equals("ping")){
        response.sample_string = "pong";
    }
}
}
}

```

Класс «Communication»

```
public class Communication {
```

```

public Communication(){
    try{
        serverSocket = new ServerSocket(9200);
        btConnector = new BTConnector();
        btConnector.setUpSocket();

        tcpThread = new TcpThread();
        tcpThread.start();        btThread =
new BTThread();
        btThread.start();

        // запущь udp
        BroadcastThread bt = new BroadcastThread();
        bt.start();
    }catch(IOException e){
        e.printStackTrace();
    }
}

class TcpThread extends Thread{
    @Override
    public void run() {
        try{
            client = serverSocket.accept();
            synchronized (lock){        if
(connectionEstablished){
                return;
            }
            bufferedReader = new BufferedReader(new    InputStreamReader
(client.getInputStream()));
            printWriter = new PrintWriter(client.getOutputStream());
            connectionEstablished = true;
            lock.notifyAll();
        }

        }catch(IOException e){
            e.printStackTrace();
        }

        .....
        .....
        .....

        public void setUpConnection(){
            connectionEstablished = false;

```

```

        if (!tcpThread.isAlive()){
            tcpThread = new TcpThread();
            tcpThread.start();
        }
        if(!btThread.isAlive()){
            btThread = new BTThread();
            btThread.start();
        }
    try {
        synchronized (lock){
            System.out.println("Wait client");
            lock.wait();
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

    public void shutdown(){
        this.close();    try {
            serverSocket.close();
            btConnector.cancel();    }
        catch (IOException e) {
        }
    }
}
}

```

Класс SensorControl

```

public class SensorControl {
    SensorEventListener sensorEventListener;
    SensorDiscovery sensorDiscovery;
    SampleThread sampleThread;

    public SensorControl(SensorEventListener sensorEventListener){
        this.sensorEventListener = sensorEventListener;
        this.sampleThread = new SampleThread(sensorTracker, sensorEventListener);
    }

    public boolean openSensorByNameOnPort(String sensorName, int portNumber){
        synchronized (openSensorLock){
            if (temp !=null){
                synchronized (sensorTracker.openLock){
                    sensorTracker.setSensorAtPort(portNumber, temp);
                }
            }
            setSensorModes(portNumber, 0);
            return true;
        }
    }
}

```



```

        return false;    }
    }

    public void setSensorModes(int port, int mode){
        sensorTracker.sensorMode[port] =
        sensorTracker.getSensorAtPort(port).getMode(mode);
        sensorTracker.sampleProvider[port]
        = new float[sensorTracker.sensorMode[port].
        sampleSize()];
    }
    .....
    .....
    }
}

```

Класс MotorControl

```

public class MotorControl {

    public MotorControl(){
        openPorts();
    }

    public void openPorts(){
        motors.put("A", new
        NXTRegulatedMotor(BrickFinder.getDefault().getPort("A")));
        motors.put("B", new
        NXTRegulatedMotor(BrickFinder.getDefault().getPort("B")));
        motors.put("C", new
        NXTRegulatedMotor(BrickFinder.getDefault().getPort("C")));
        motors.put("D", new
        NXTRegulatedMotor(BrickFinder.getDefault().getPort("D")));
    }

    public void forward(String motorPort){
        motors.get(motorPort).forward();
    }
    .....
    .....
    .....

    public int getPosition(String motorPort){
        return motors.get(motorPort).getPosition();
    }

    public boolean isStalled(String motorPort){
        return motors.get(motorPort).isStalled();
    }

    public float getMaxSpeed(String motorPort){
        return motors.get(motorPort).getMaxSpeed();
    }
}

```

```

    public void closeMotors(){
motors.get("A").close();
motors.get("B").close();    motors.get("C").close();
    motors.get("D").close();
    }

    public void reset(){
closeMotors();    openPorts();
    }

}

```

Приложение 4.

Server.py – скрипт запуска внешнего сервера SenseBot

```

import functools
import sys import
threading import
flask import
jsonrpc import
managers
from lib.simplewebsocketserver import SimpleWebSocketServer

_brick_manager = managers.BrickManager()

app = flask.Flask(__name__)

def status(code):    response =
flask.make_response()
response.status_code = code
response.data = response.status
return response

@app.route('/') def
index():
    return flask.redirect(flask.url_for('static', filename='index.html'))

jsonrpc.register_remote_object('/brick_manager', _brick_manager, app)

def main():    print "Starting
websocket server"
    web_socket = functools.partial(managers.SubscriptionSocket, _brick_manager)
server = SimpleWebSocketServer("", 9999, web_socket)

```

```

    _thread = threading.Thread(name="receive_thread", target=server.serveforever, args=())
    _thread.daemon = True
    _thread.start()

    print "Starting flask main server"
    app.run(host='127.0.0.1', port=80)

if __name__ == "__main__":
    sys.exit(main())

```

managers.py – скрипт запуска модулей управления блоком EV3 и
программным кодом

```

class CodeManager(object):
    def __init__(self):
        self.subsumption_controller = subsumption.Controller(return_when_no_action=False)
        self.raw_code = { }
        self.behaviors_names_list = []

    def update_behavior(self, behavior_name, behavior):
        if behavior_name in self.behaviors_names_list:
            index = self.behaviors_names_list.index(behavior_name)
            self.subsumption_controller.update(behavior, index)
        else:
            self.subsumption_controller.add(behavior)
            self.behaviors_names_list.append(behavior_name)

    def add_behaviors(self, title, code):
        self.raw_code[title] = code
        try:
            exec code in self.expressions
        except Exception as e:
            return e
        for instance in set(self.expressions) - set(self.default):
            if inspect.isclass(self.expressions[instance]):
                self.update_behavior(title, self.expressions[instance]())
            break
        self.expressions = copy.copy(self.default)
        return None

    def remove_behavior(self, title):
        if title in self.behaviors_names_list:
            index = self.behaviors_names_list.index(title)
            self.subsumption_controller.remove(index)
            del self.behaviors_names_list[index]
            del self.raw_code[title]
        else:
            print "Ошибка при удалении модуля"

```

```

def code_package(self):
    data = []
    for index, behavior_name
in enumerate(self.behaviors_names_list):
        data.append({
            'title': behavior_name,
            'code': self.raw_code[behavior_name],
            'running': self.subsumption_controller.active_behavior_index == index
        })
    return json.dumps({'cmd': 'code_data', 'data': data})

class BrickManager(object):

    def tell_others_about_it(self, address, data, client):
    for other_clients in self._subscription_clients[address]:
    if other_clients != client:
        other_clients.send(data)

    def runnable_callback(self, address, index):
    print "Занято", index
    data =
    json.dumps({'cmd': 'running', 'title':
    self.code_managers[address].behaviors_names_list[index]})
    for client in self._subscription_clients[address]:
        client.send(data)

    def _run(self, address):
    try:
        self.code_managers[address].subsumption_controller.start()
    except
    Exception as e:
        self.tell_others_about_it(address, json.dumps({'cmd': 'error',
'data': str(e)}), None)

    def stop(self, address):
        self.code_managers[address].subsumption_controller.stop()

    def add_code_bulk(self, address, bulk, client):
        for code_module in bulk:
            exception = self.code_managers[address].add_behaviors(code_module['title'],
            code_module['code'])
            if exception:
                client.send(
                    json.dumps({'cmd': 'error', 'data': 'Модуль ' + code_module['title'] + ' -> ' +
                    str(exception)}))
        return
        self.tell_others_about_it(address, json.dumps({'cmd': 'code_data', 'data': bulk}), client)

    def add_code(self, address, name, code, client):
        exception = self.code_managers[address].add_behaviors(name, code)
    if exception:

```

```

        client.send(json.dumps({'cmd': 'error', 'data': 'Модуль ' + name + ': ' + str(exception)}))
self.tell_others_about_it(address,
                            json.dumps({'cmd': 'code_data', 'data': [{'title': name, 'code': code}]}),
client)

def remove_code(self, address, name, client):
    self.code_managers[address].remove_behavior(name)
    self.tell_others_about_it(address,
                            json.dumps({'cmd': 'remove_code', 'title': name})),
client)

def is_brick_connected(self, address):
    return address in self._connected_brick

def remove_old_subscription_from_brick(self, address, client):
self._subscription_clients[address].remove(client)    del
self._old_msg[client]

def remove_brick(self, address):
    print "Блок БЫЛ ОТКЛЮЧЕН ", address    for
client in self._subscription_clients[address]:
    client.close()

    self._subscription_clients[address] = []
del self._subscription_objects[address]

    self._connected_brick[address].close()
del self._connected_brick[address]

def add_new_subscriptions_to_brick(self, address, client):
if address not in self.code_managers:    code_manager
= CodeManager()    self.code_managers[address] =
code_manager
    code_manager.subsumption_controller.callback = functools.partial(self.runnable_callback,
address)    else:
    client.send(self.code_managers[address].code_package())

self._subscription_clients[address].append(client)

if address not in self._subscription_objects:
    sub = ev3.Subscription(False, True)
sub.subscribe_on_samples(functools.partial(self._callback_on_samples, address))
sub.subscribe_on_brick_disconnect(functools.partial(self.remove_brick, address))
self._connected_brick[address].set_subscription(sub)    self._subscription_objects[address]
= sub

def _callback_on_samples(self, address, samples):

```

```

    for client in list(self._subscription_clients[address]):
try:
    data = []          brick =
self._connected_brick[address]          for
port in ev3.SENSOR_PORTS:          if
port in brick.get_opened_ports:
    sensor = brick.get_opened_ports[port]
mode = sensor.get_selected_mode()          msg
= {'sensor': sensor.get_name(),
    'mode': mode.get_name(),
    'port': port,
    'sample': samples[port - 1]}
if self._old_msg[client][port] != msg:
data.append(msg)
self._old_msg[client][port] = msg          if data:
    client.send(json.dumps({'cmd': 'sensor_data', 'data': data}))
except socket.error:
    self._subscription_clients[address].remove(client)
except Exception as e:
    print "Ошибка при использовании веб-сокетов", type(e)
print e

```

```

def add_brick(self, address):          if not
self.is_brick_connected(address):
try:
    brick = ev3.connect_to_brick(address)
self._connected_brick[address] = brick
return True          except
ev3.BrickNotFoundException:          pass
    return False

```

```

def get_bricks(self):          return
self._connected_brick.keys()

```

```

def get_bricks_with_name(self):          bricks = []
for address, brick in self._connected_brick.iteritems():
bricks.append((address, str(brick)))
return bricks

```

```

def open_sensor(self, brick_address, sensor_name, port):
if brick_address in self._connected_brick:          brick =
self._connected_brick[brick_address]          sensor_class
= getattr(ev3, sensor_name)          try:
    sensor_class(brick, int(port))
return True          except (ev3.InvalidSensorPortException,
ev3.SensorNotConnectedException):          pass          return False

```