

Министерство образования и науки Российской Федерации
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

И. С. Дубков, П. С. Сташевский, И. Н. Яковина

РЕШЕНИЕ ПРАКТИЧЕСКИХ ЗАДАЧ НА БАЗЕ ТЕХНОЛОГИИ ИНТЕРНЕТА ВЕЩЕЙ

Утверждено редакционно-издательским советом университета
в качестве учебного пособия

НОВОСИБИРСК
2017

УДК 004.896:004.738.5(075)

ББК 32.973я7

Д794

Рецензенты:

доц. каф. АСУ НГТУ, канд. техн. наук, доцент *И. Н. Томилов*

доц. каф. АСУ НГТУ, канд. техн. наук *Д. Н. Достовалов*

Дубков И.С.

Д794 Решение практических задач на базе технологии интернета вещей: учебное пособие / И. С. Дубков, П. С. Сташевский, И. Н. Яковина. – Новосибирск: Изд-во НГТУ, 2017. – 80 с.

ISBN 978-5-7782-3161-0

В теоретической части учебного пособия рассматриваются основные понятия, модели и технологии организации интернета вещей, программные платформы и инструменты для разработки проектов. Практическая часть посвящена примеру решения конкретной задачи, в ходе которого поэтапно разбираются основные элементы технологии интернета вещей, особенности использования аппаратной платформы *Arduino* и платформы *ThingWorx*.

Учебное пособие разработано в студенческом конструкторском бюро «Робототехника и искусственный интеллект» НГТУ, практическая часть апробирована в ходе занятий для учащихся старших классов Инженерного лицея НГТУ и студентов 3-4 курса факультета автоматизации и вычислительной техники НГТУ в форме факультативного курса. Практическая часть может быть рекомендована как цикл лабораторно-практических занятий по дисциплинам, изучающим интеллектуальные системы и технологии для студентов среднего и высшего профессионального образования, а также как пособие для наставников и тренеров команд *JuniorSkills* и *WorldSkills* компетенции «Интернет вещей».

УДК 004.896:004.738.5(075)

ББК 32.973я7

ISBN 978-5-7782-3161-0

©Дубков И.С., Сташевский П.С., Яковина И.Н., 2017

© Инженерный лицей НГТУ, 2017

© Новосибирский государственный
технический университет, 2017

Оглавление

Оглавление.....	3
Введение.....	4
Глоссарий.....	5
Понятие интернета вещей.....	7
История развития интернета вещей.....	11
Модели и технологии организации.....	14
Платформы и инструменты.....	20
Краткий обзор аппаратных платформ.....	25
Умная теплица на базе технологий интернета вещей.....	28
Занятие 1. Основы работы с платформой <i>ThingWorx</i>	31
Занятие 2. Основы работы с платформой <i>Arduino</i>	48
Занятие 3. Взаимодействие с платформой <i>ThingWorx</i>	57
Занятие 4. Взаимодействие <i>Arduino</i> и <i>ThingWorx</i>	70
Список источников.....	77

Введение

По мнению многих аналитиков и специалистов, интернет вещей – это технология будущего. Появление интернета вещей – это ожидаемый шаг, который предсказывался фантастами и футурологами. Так, еще в 1926 году Никола Тесла сказал, что в будущем все вещи станут частью единого целого, а инструменты, благодаря которым это станет возможным, будут легко помещаться в кармане. И тот процесс, который мы наблюдаем сегодня, действительно соответствует этому описанию.

Термин, впервые введенный в 1999 году Кевином Эштоном, до сих пор расширяется и наполняется новыми понятиями и охватывает концепцию объединения устройств и сетей для мониторинга и управления.

Интернет вещей – это не просто множество различных приборов и датчиков, объединенных между собой проводными и беспроводными каналами связи и подключенных к сети Интернет, а более тесная интеграция реального и виртуального миров, в котором общение производится между людьми и устройствами.

В емком понятии «интернет вещей» сплелись и органично соединились многие информационные технологии. За несколько лет привычные нам приборы и устройства приобрели новые функциональные свойства, которые позволяют говорить о едином технологическом и информационном пространстве глобального масштаба.

Основные технологические аспекты интернета вещей рассматриваются в данном пособии на примере решения конкретной практической задачи. Прототип системы автоматизации технологического процесса выращивания растений, мониторинга и комбинированного управления или «умная теплица» является одним из популярных приложений. Пошаговая разработка этого решения, на наш взгляд, позволит составить технологически целостное представление о процессе создания различных приложений и устройств мира интернета вещей.

Глоссарий

Интеллектуальные объекты – устройства, осуществляющие коммуникацию в рамках интернета вещей. Как правило, это устройства, обычно имеющие значительные ограничения, такие как ограниченная мощность, память, ресурсы обработки или ширина диапазона.

Интернет вещей (Internet of Things, IoT) – взаимодействие большого количества устройств (интеллектуальных объектов) с использованием общеизвестных протоколов и минимальным участием человека в задачах сбора, обработки и передачи данных.

Проблема интероперабельности (стандартизации) – проблема отсутствия единых стандартов для сопряжения различных компонент в рамках интернета вещей.

M2M (межмашинные подключения) – взаимодействие устройств (интеллектуальных объектов) без участия человека на основе определенных стандартов.

Internet Protocol, IP – маршрутизируемый протокол сетевого уровня стека TCP/IP. Именно IP является протоколом, объединяющим отдельные компьютерные сети во всемирную сеть Интернет.

Representational State Transfer Application Programming Interface, REST API – архитектурный стиль взаимодействия компонентов распределенного приложения в сети посредством HTTP-протокола.

RESTful сервисы (Representational State Transfer) – веб-сервисы, поддерживающие взаимодействие между распределенными компонентами системы в сети и реализующие REST API (Application Programming Interface) для их взаимодействия.

JavaScript Object Notation, JSON – текстовый формат обмена данными, визуально легко читаемый людьми и часто используемый в реализации REST API.

Python – высокоуровневый интерпретируемый язык разработки, ориентированный на повышение производительности разработки и читаемости кода. Существует две основных версии: 2.x и 3.x.

HyperText Transfer Protocol, HTTP – протокол прикладного уровня передачи данных, предполагающий клиент-серверное взаимодействие.

Model Tags (теги модели) – метки, используемые для классификации сущностей.

Thing Shapes (формы вещей) – это базовые компоненты определения, которые содержат уникальные характеристики и правила поведения. В терминах программирования они представляют собой интерфейс.

Thing Template (шаблон вещи) – сущность, которая используется для представления набора схожих объектов. В терминах программирования шаблон – это абстрактный класс.

Thing (вещь) – это цифровое представление физического объекта. В терминах программирования – то же самое, что класс.

Data Storage (хранилище данных) – выделенное на сервере дисковое пространство для хранения данных.

Value Stream (поток значений) – одна из форм представления данных.

Services (сервисы) – функциональные скрипты.

Snippets (сниппеты) – небольшие законченные блоки кода, используемые для ускорения разработки приложения.

Event (событие) – изменение состояния или значения свойства.

Subscription (подписка) – это действие, которое выполняется, когда происходит событие.

Mashup (мэшап) – графическое представление приложения.

Layout (макет) – это адаптивный контейнер, который позволяет создавать разделы в вашем приложении.

Widget (виджет) – элемент графического интерфейса пользователя, виртуальный объект, репрезентирующий физическую сущность.

Понятие интернета вещей

Несмотря на активное обсуждение вопросов *интернета вещей* во всем мире, для этого термина отсутствует единое, общепринятое определение. Различные группы используют разные определения для описания понятия *IoT* и его основных характеристик [6]. Некоторые определения учитывают понятие Интернета или протокола *IP*, в то время как другие не упоминают их. Рассмотрим несколько часто употребляемых определений.

Комиссия по архитектуре Интернет (*IAB*) описывает определение в документе «Особенности архитектуры в сетях интеллектуальных объектов», следующим образом: термин «интернет вещей» (*IoT*) обозначает тенденцию, при которой большое число встроенных устройств использует услуги связи на основе протокола Интернет. Многие из этих устройств, часто называемые «интеллектуальными объектами», не управляются напрямую человеком, но существуют в виде компонентов зданий или транспортных средств или установлены в окружающей среде (рис. 1). В Рабочей группе проектирования Интернет (*IETF*) термин «сеть интеллектуальных объектов» обычно используется по отношению к интернету вещей. В этом контексте «интеллектуальные объекты» – это устройства, обычно имеющие значительные ограничения, такие как ограниченная мощность, память, ресурсы обработки или ширина диапазона (смартфоны, умные устройства и др.).

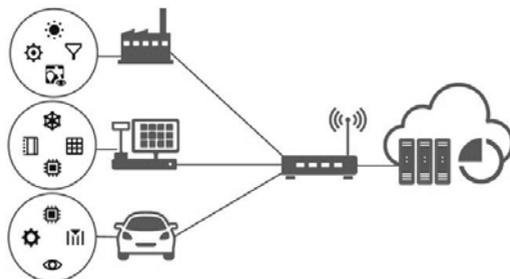


Рис. 1. Коммуникация устройств и объектов в рамках интернета вещей

Опубликованный в 2012 году Международным союзом электросвязи (*International Telecommunication Union, ITU*) краткий обзор интернета вещей, включает понятие взаимоподключаемости устройств, хотя и не связывает напрямую *IoT* с Интернетом. Тогда, интернет вещей – глобальная инфраструктура общества с развитой информационной технологией, обеспечивающая возможность предоставления расширенных услуг за счет взаимоподключения существующей и развивающейся функционально совместимой информации и технологий связи. При этом за счет использования функций идентификации сбора, обработки и передачи данных, *IoT* использует предметы для того, чтобы предлагать услуги во всех областях при соблюдении требований безопасности и конфиденциальности. Таким образом, в данной формулировке интернет вещей представляет собой сеть сетей, где все устройства (или почти все) имеют связь друг с другом.

Следующее определение связывает *IoT* с облачными услугами: интернет вещей – это концептуальная основа, в соответствии с которой все предметы представлены в Интернете и имеют определенное место в нем. Точнее, интернет вещей ставит своей задачей предлагать новые области применения и услуги, объединяющие физический и виртуальный мир, в котором межмашинная связь (*M2M*) является основной связью для взаимодействия вещей и приложений в облачных вычислениях.

Все эти определения описывают сценарии, в которых сетевое подключение и вычислительная способность распространяются на целую группу предметов, устройств, датчиков и повседневных предметов, которые обычно не считаются компьютерами; благодаря этому устройства могут генерировать и потреблять данные и обмениваться ими, часто при минимальном участии со стороны человека. Различные определения *IoT* не всегда противоречат друг другу – они скорее подчеркивают различные аспекты явления *IoT* с разных точек зрения и перспектив применения.

В рамках данного пособия термин «интернет вещей» или «*IoT*» определяется как расширение возможностей подключения к сети и вычислительных способностей для объектов, устройств, датчиков и других предметов, обычно не считающихся компьютерами. Эти

«интеллектуальные объекты» требуют минимального вмешательства со стороны человека для создания, использования и обмена данными; при этом часто они имеют возможность подключения к функциям удаленного сбора, анализа и управления данными.

Рассмотрим ряд факторов, которые способствуют развитию интернета вещей.

1. **Повсеместное подключение.** Повсеместное недорогое и высокоскоростное сетевое подключение позволяет подключить к сети практически любой предмет.

2. **Широкое применение сетей на основе протокола IP.** Протокол *IP* стал основным глобальным сетевым стандартом, обеспечивающим четко определенную и широко используемую платформу для программного обеспечения и инструментов, которая может быть легко и без больших затрат включена в широкий спектр устройств.

3. **Экономические тенденции в области вычислительных систем.** Под воздействием отраслевых инвестиций в исследования, разработки и производство, закон Мура продолжает обеспечивать все большие вычислительные возможности по все более низкой цене и с уменьшением энергопотребления.

4. **Миниатюризация.** Достижения в области производства позволяют применять самые современные технологии вычислений и связи в объектах очень маленького размера. В сочетании с более высокой экономичностью вычислений это послужило толчком для создания недорогих датчиков малого размера, на которых основано множество областей применения *IoT*.

5. **Достижения в области анализа данных.** Новые алгоритмы и быстрый рост вычислительной мощности, объема хранения данных и облачных услуг делают возможными агрегирование, корреляцию и анализ больших объемов данных; эти крупные и динамические наборы данных обеспечивают новые возможности получения информации и знаний.

6. **Развитие облачных вычислений.** Облачные вычисления, использующие удаленные сетевые ресурсы для обработки, управления и хранения данных, позволяют небольшим и распределенным

устройствам взаимодействовать с мощными функциями анализа и управления на сервере.

К наиболее интересным и перспективным сферам применения интернета вещей можно отнести:

1. «Умный дом», включая:

- Решения для создания интеллектуальных сервисов безопасности
- Решения для создания интеллектуальных сервисов оптимизации использования ресурсов домохозяйствами

2. «Умный транспорт», включая:

- Сервисы класса *fleet management* для индивидуальных перевозчиков (некий аналог *Uber* для грузового транспорта)
- Сервисы *UBI*-страхования
- Сервисы технического обслуживания по фактическому состоянию

3. Торговля и финансовые услуги:

- Решения для автоматической передачи и анализа данных с *POS*-терминалов, включая виртуальные
- Управление запасами домохозяйств как сервис.

4. Промышленный сегмент – перевод автоматизированных систем управления технологическим процессом на принципы *IoT*.

Как говорится в обзоре портала *Tadviser*, «человечество имеет шансы избавиться от фобий, типа «закрыл ли я дверь» или «выключил ли я утюг», потому что информация об этом будет в смартфоне. И если вдруг не закрыл и не выключил, все можно исправить из любой точки города и мира. Система наблюдения распознает лица всех, кто проходил мимо вашего дома или стоял около двери квартиры, и при повторном появлении того же человека сравнит его лицо с базой полиции. Холодильник, снабжённый набором камер, сообщит о конце срока годности продуктов и просто истощении запасов любимого мороженого, умный пылесос отправит сообщение о находке ювелирного украшения, завалившегося под диван» [1].

История развития интернета вещей

Термин «интернет вещей» (*IoT*) впервые использовал в 1999 году основатель исследовательской группы *Auto-ID* при Массачусетском технологическом институте Кевин Эштон на презентации *Procter & Gamble* для описания системы, в которой предметы физического мира могут подключаться к Интернету с помощью датчиков [1]. Эштон создал этот термин для того, чтобы проиллюстрировать потенциальные возможности подключения меток радиочастотной идентификации (*RFID*), используемых в корпоративных цепочках поставок, для подсчета и отслеживания товаров без необходимости вмешательства со стороны человека.

Несмотря на то, что термин «интернет вещей» является сравнительно новым, концепция объединения компьютеров и сетей для мониторинга и управления устройствами существует уже несколько десятилетий. Например, уже в конце 1970-х гг. осуществлялось коммерческое использование систем для удаленного мониторинга счетчиков электрической сети через телефонные линии [2]. В 1990-х гг. достижения в области беспроводной технологии сделали возможным широкое распространение корпоративных и производственных решений «машина-машина» (*M2M*) для мониторинга и управления оборудованием. Однако многие из этих ранних решений *M2M* были созданы на основе закрытых специализированных сетей на фирменных или отраслевых стандартах, а не на сетях на основе протокола Интернета (*IP*) и стандартах Интернета.

Идея использования *IP* для подключения к Интернету устройств, не являющихся компьютерами, не является новой. Первое устройство с подключением к Интернету – тостер с поддержкой протокола *IP*, который можно было включать и выключать через Интернет – был представлен на интернет-конференции в 1990 году [3]. В течение следующих нескольких лет появились другие предметы с поддержкой протокола *IP*, включая автомат прохладительных напитков [4] в университете Карнеги-Меллона в США и кофеварка [5] в Троянском зале в Кембриджском университете в Великобритании (которая оставалась подключенной к Интернету до 2001 года). С самых первых

шагов упорная работа в области исследований и разработок привела к созданию «интеллектуальной сети объектов», которая стала основной для сегодняшнего интернета вещей. В 2004 году в *Scientific American* опубликована обширная статья, наглядно показывающая возможности концепции в бытовом применении. В статье приведена иллюстрация, показывающая как бытовые приборы (будильник, кондиционер), домашние системы (система садового полива, охранная система, система освещения), датчики (тепловые, датчики освещенности и движения) и «вещи» (например, лекарственные препараты, снабженные идентификационной меткой) взаимодействуют друг с другом посредством коммуникационных сетей (инфракрасных, беспроводных, силовых и слаботочных) и обеспечивают полностью автоматическое выполнение процессов (включают кофеварку, изменяют освещенность, напоминают о приеме лекарств, поддерживают температуру, обеспечивают полив сада, сберегают электроэнергию). Тем не менее, в отчете Национального разведывательного совета США (*National Intelligence Council*) 2008 года интернет вещей фигурирует как одна из шести потенциально разрушительных технологий, указывается, что повсеместное и незаметное для потребителей превращение в интернет-узлы таких распространенных вещей, как товарная упаковка, мебель, бумажные документы, может нанести урон национальной информационной безопасности.

Если идея подключения объектов друг к другу и к Интернету не нова, следует спросить, почему интернет вещей вновь отличается такой популярностью? В более широкой перспективе слияние нескольких тенденций рынка и технологии позволяет просто и недорого подключить большее число устройств меньшего размера. В отчете консалтинговой компании *McKinsey* «*Unlocking the Potential of the Internet of Things*» описывается широкий спектр возможных областей применения с точки зрения условий, в которых *IoT* будет обеспечивать преимущества для отрасли и пользователей.

По мере увеличения числа устройств, подключенных к Интернету, ожидается существенное увеличение трафика. Например, согласно прогнозам *Cisco*, интенсивность сетевого обмена данными между устройствами (не включая ПК) увеличится с 40% в 2014 г. до почти

70% в 2019 г. [7]. *Cisco* также прогнозирует, что число «межмашинных» подключений *M2M* (включая промышленные, домашние, медицинские, автомобильные и другие вертикали *IoT*) увеличится с 24% от всех подключенных устройств в 2014 г. до 43% в 2019 г. Результатом этих тенденций станет то, что через десять лет понятие подключения к Интернету существенно изменится. Приведем слова профессора Массачусетского технологического института Нил Гершенфельда: «...Возможно, стремительный рост Всемирной паутины был лишь искрой, которая приведет к настоящему взрыву теперь, когда вещи начинают подключаться к сети».

В сознании обычных людей Всемирная паутина уже практически стала синонимом Интернета. Сетевые технологии способствуют взаимодействию между людьми и контентом, делая его определяющей характеристикой нынешнего Интернета. Использование Интернета во многом характеризуется активным участием пользователей, скачивающих и создающих контент через компьютеры и смартфоны. Если прогнозы относительно роста *IoT* окажутся верными, может произойти сдвиг в сторону более пассивного взаимодействия с Интернетом между пользователями и такими объектами, как автомобильные компоненты, бытовая техника и устройства с самодиагностикой. Эти устройства отправляют и получают данные от имени пользователя, при минимальном участии человека, а иногда даже без его ведома [6].

Развитие *IoT* может привести к изменению образа мышления, если наиболее распространенное взаимодействие с Интернетом и данными, полученными на основе этого взаимодействия, будет происходить в результате пассивного взаимодействия с подключенными предметами вокруг. Потенциальное достижение этого результата – «гиперподключенного мира» – является подтверждением общего назначения Интернета без ограничения области применения или услуг, в которых может применяться эта технология.

Модели и технологии организации

В марте 2015 г. Комиссия по архитектуре Интернет выпустила директивный документ по архитектуре для сетевого подключения интеллектуальных объектов, в котором определяется концептуальная основа четырех общих моделей связи, используемых устройствами *IoT*. Перейдем к рассмотрению их особенностей.

Подключение от устройства к устройству.

Модель связи от устройства к устройству представляет два или несколько устройств, подключенных и осуществляющих связь друг с другом напрямую, а не через промежуточный сервер приложений. Эти устройства осуществляют связь через различные типы сетей, в том числе, сети на основе протокола *IP* или Интернет.

Однако, эти устройства часто используют такие протоколы, как *Bluetooth*, *Z-Wave41* или *ZigBee42* для установления прямой связи от устройства к устройству, как показано на рисунке 2.



Рис. 2. Модель взаимодействия устройств через беспроводную сеть

Эти сети со связью *M2M* позволяют устройствам, поддерживающим определенный протокол, осуществлять связь и обмен сообщениями для выполнения своих функций. Эта модель связи обычно применяется в таких приложениях, как домашние системы автоматизации, в которых обычно используются пакеты данных малого размера для установления связи между устройствами с низким уровнем требований в области скорости передачи данных. Бытовые устройства *IoT*, такие как лампочки, выключатели, термостаты и дверные замки, в домашней системе автоматизации обмениваются малым

объемом информации (например, сообщение о состоянии дверного замка или команда включения света).

Эта связь от устройства к устройству наглядно демонстрирует многие проблемы интероперабельности (способности к взаимодействию). Согласно описанию в статье, опубликованной в *IETF Journal*, «эти устройства часто находятся в непосредственной связи, обычно они оснащены встроенными механизмами безопасности, но также используют определенные модели данных для каждого устройства, требующие дополнительных усилий в разработке». Это означает, что производители устройств должны вкладывать средства в разработку определенных форматов данных для каждого типа устройств вместо использования открытой платформы для стандартных форматов.

С точки зрения пользователей, это часто означает, что используемые протоколы передачи данных от устройства к устройству несовместимы, и в результате пользователь вынужден выбирать другие устройства, поддерживающие тот же протокол. Например, устройства, использующие протокол *Z-Wave*, несовместимы с устройствами семейства *ZigBee*. Несмотря на то, что эта несовместимость ограничивает выбор пользователя устройствами, принадлежащими к определенному семейству на основе одного и того же протокола, пользователь знает, что продукты определенного семейства работают надлежащим образом.

Подключение от устройства к облаку.

В модели связи «от устройства к облаку» устройство *IoT* подключается напрямую к облачной интернет-службе для обмена данными. При таком подходе часто используются существующие механизмы связи, такие как традиционные проводные соединения *Ethernet* или *Wi-Fi* для установления соединения между устройством и сетью *IP*, которая, в свою очередь, подключается к облачной службе. Этот подход показан на рисунке 3.



Рис. 3. Модель взаимодействия устройств с облачными сервисами (поставщиками услуг аренды приложений)

Эта модель соединения используется некоторыми популярными потребительскими устройствами *IoT*, такими как самообучающийся термостат *Nest Labs* [8] и *SmartTV* производства *Samsung* [9]. В случае самообучающегося термостата *Nest* устройство передает данные в облачную базу данных, где эти данные могут использоваться для анализа потребления электроэнергии дома. Это облачное подключение позволяет пользователю получать удаленный доступ к своему термостату через смартфон или веб-интерфейс, а также поддерживает обновления программного обеспечения термостата. Аналогичным образом, в случае технологии *SmartTV* производства *Samsung*, телевизор использует подключение к Интернету для передачи информации о просматриваемых пользователем программах в *Samsung* для анализа и подключения интерактивной функции распознавания голоса на устройстве телевизора. В этих случаях модель подключения устройства к облаку обеспечивает дополнительную ценность для конечного пользователя за счет расширения стандартных функций устройства.

Тем не менее, проблемы интероперабельности могут возникнуть при попытке интеграции устройств различных производителей. Чаще всего используются облачные услуги и устройство одного производителя. Если для связи между устройством и облачными службами используются патентованные протоколы данных, владелец или пользователь устройства может пользоваться лишь определенной облачной службой, что ограничивает его возможность пользоваться

услугами других поставщиков. Такая ситуация обозначается термином «зависимость от поставщика», которая охватывает различные аспекты отношений с поставщиком, такие как владение данными и доступ к ним. В то же время пользователи обычно могут быть уверены в возможности интеграции устройств, созданных для определенной платформы.

Подключение от устройства к шлюзу.

В случае модели подключения устройства к шлюзу или, чаще всего, в модели подключения устройства к шлюзу прикладного уровня устройство *IoT* подключается через службу *ALG (Application-level gateway)* в качестве канала для использования облачной службы. Проще говоря, это означает, что прикладное программное обеспечение функционирует на устройстве локального шлюза, которое играет роль посредника между устройством и облачной службой и обеспечивает безопасность и другие функции, такие как преобразование данных или протоколов. Эта модель показана на рисунке 4.



Рис. 4. Модель взаимодействия устройств через шлюз

В пользовательских устройствах присутствуют различные варианты этой модели. Во многих случаях в качестве локального шлюза используется смартфон с приложением для связи с устройством и передачи данных в облачную службу. Эта модель часто используется с популярными потребительскими устройствами, такими

как браслеты для занятий спортом. В этих устройствах отсутствует функция прямого подключения к облачной службе, поэтому они часто используют приложения смартфона для работы в качестве шлюза подключения.

Другой разновидностью этой модели подключения устройства к шлюзу являются устройства, выполняющие роль концентратора в приложениях домашней автоматике. Они используются в качестве локального шлюза между отдельными устройствами *IoT* и облачной службой, но также могут заполнять пробелы совместимости между самими устройствами. Другими словами, эта модель связи часто используется для интеграции новых интеллектуальных устройств в традиционную систему с устройствами, которые изначально не могут с ними взаимодействовать. Недостаток этого подхода состоит в том, что необходимость разработки системы и шлюза прикладного уровня увеличивает сложность и стоимость системы в целом. Системы, использующие модель соединения устройства со шлюзом, и их роль в решении проблем совместимости устройств *IoT* до сих пор находятся в процессе развития.

Модель совместного использования данных на сервере

Модель совместного использования данных на сервере соответствует архитектуре, позволяющей пользователям экспортировать и анализировать данные интеллектуальных объектов из облачной службы в сочетании с данными из других источников. Такая архитектура поддерживает «...желание [пользователей] предоставлять доступ третьим сторонам к загруженным данным датчиков». Архитектура совместного использования данных на сервере позволяет объединять и анализировать потоки данных, полученных от одного устройства *IoT*.

Например, корпоративный пользователь, ответственный за офис, может быть заинтересован в объединении и анализе данных о потреблении электроэнергии и других коммунальных услуг, получаемых всеми датчиками *IoT* и системами инженерного обеспечения с подключением к Интернету. В модели подключения отдельных устройств к облачным службам данные каждого датчика или системы *IoT* находятся в отдельной базе данных. Эффективная архитектура совместного использования данных на сервере должна

позволять компании с легкостью получать доступ и анализировать облачные данные, полученные от всех устройств в здании. Кроме того, этот тип архитектуры позволяет обеспечить переносимость данных. Эффективная архитектура совместного использования данных на сервере позволяет пользователям перемещать свои данные при переключении между услугами *IoT*, преодолевая барьеры традиционных отдельных баз данных.

Модель совместного использования данных на сервере предполагает объединенный подход к облачным услугам. На рисунке 5 показано графическое представление этой модели. Данная модель архитектуры представляет собой подход для обеспечения совместимости между этими системами на базе сервера.

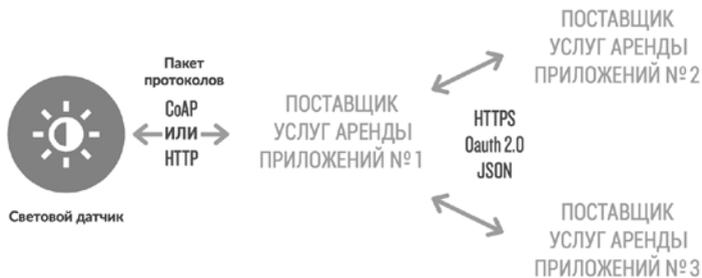


Рис. 5. Модель взаимодействия устройств для совместного использования данных на сервере

Платформы и инструменты

Принцип *IoT* подразумевает взаимодействие привычных для нас в быту вещей с помощью вычислительных сетей. В широком понимании, это не просто множество различных приборов и датчиков, объединенных между собой проводными и беспроводными каналами связи и подключенных к сети Интернет, а это более тесная интеграция реального и виртуального миров, в котором основную роль играет общение между людьми и всевозможными устройствами.

Желание многих пользователей почувствовать себя в роли создателей подтолкнуло некоторые компании к разработке специальных программируемых платформ. В итоге оказалось, что подобные разработки позволили справиться с различными задачами, начиная решением инфраструктурных концепций и заканчивая созданием интерактивных объектов. Рассмотрим наиболее популярные *IoT* платформы с точки зрения обычного пользователя и разработчика, а также выясним особенности каждой из них [13].

Samsung Artik Cloud

Платформа от *Samsung Artik Cloud* (рис. 6) была продемонстрирована на конференции разработчиков *Samsung* в мае 2016 г. Идея разработки заключается в том, чтобы соединить каждое *IoT* устройство со всеми облачными сервисами, сенсорами и любыми типами данных так, чтобы у пользователей не возникало проблем и путаницы с умными устройствами.



Рис. 6. Платформа *Artik Cloud* от *Samsung*

Для пользователей: Новая платформа от *Samsung* хорошо масштабируема. *Samsung* совместно с компанией *Legrand* (имеет более чем 200 миллионов умных сенсоров и других *IoT*-гаджетов по всему миру от бытовой техники и до смартфонов) представила первый в мире световой выключатель для *IoT*, работающей на платформе *Artik*. Платформа предоставляет удобные открытые программные интерфейсы и инструменты для безопасного сбора, хранения и обработки данных с любых подключенных устройств или из облачных сервисов.

Для разработчиков: *Samsung Artik* предоставляется разработчикам по доступной цене, а также имеется пробная бесплатная версия. Новая платформа позволяет обеспечить более быструю и упрощённую разработку новых пользовательских и корпоративных приложений. *Artik Cloud* – это открытая платформа, содержащая один из лучших в своем сегменте набор интегрированных и готовых к использованию модулей, программного обеспечения, плат, драйверов, инструментов.

Windows 10 IoT

В 2015 году компанией *Microsoft* было выпущено семейство встраиваемых операционных систем *Windows 10 IoT* (рис. 7), пришедшее на смену *Windows Embedded* и активно развивающее концепцию Интернета вещей.

Для управления *IoT* устройствами было выпущено три версии *Windows 10 IoT*:

- *Enterprise* – полностью совместимая с ОС для десктопов и применима для широкого круга аппаратных решений, таких как банкоматы, *POS*-устройства, медицинские и промышленные устройства и т.д.;
- *Mobile* – ориентирована на производителей мобильных устройств;
- *Core* – применима на устройствах, которые вообще не имеют дисплея, начиная различными робототехническими изделиями, системами домашней автоматизации и заканчивая всевозможными приборами с датчиками.

Для пользователей: *Windows 10 IoT Core* работает с микрокомпьютерами *Raspberry Pi 2*, *MinnowBoard Max* и *Intel Galileo*. Компания *Microsoft* активно разрабатывает большинство компонентов

для создания своей *IoT*-экосистемы: клиентские устройства, носимую электронику, инструментарий и облачный сервис *Azure IoT*. Среди уже существующих проектов на основе новой платформы можно отметить системы управления с обратной связью на базе *Raspberry Pi 3* [14], метеорологическую станцию с *Windows 10* [15] и комплект «Робот».

Для разработчиков: Основным преимуществом платформы *Microsoft IoT* для разработчиков является ее универсальность. Так как в основе всех операционных систем лежит одно ядро, то единожды написанное приложение (*Universal App*) будет одинаково функционировать на любых устройствах с ОС *Windows 10*. Технология *Universal Driver* позволяет также быстро создавать универсальные драйверы и инструменты, подходящие для любых устройств на *Windows 10*.



Рис. 7. *Windows 10 IoT*

Intel IoT Platform

Для компании *Intel* интернет вещей является одним из ключевых направлений деятельности. В 2014 году компанией была представлена *IoT* платформа (рис. 8), которая облегчила жизнь разработчикам в создании, тестировании и обеспечении безопасности интеллектуальных устройств. По словам старшего вице-президента *Intel IoT Group* Дугласа Л. Дэвиса, компания упрощает для клиентов переход от масштаба отдельных «умных» вещей до облаков с использованием *Intel Quark SE* и специализированных ОС для микроконтроллеров. Сейчас новая платформа *Intel* для *IoT* решает две основные задачи: упрощает интеграцию «умных» устройств и

обеспечивает улучшенный контроль над обменом данными между ними. На основе платформы от *Intel* уже было создано много интересных проектов, среди которых можно отметить голографический музыкальный автомат, умную систему контроля за ребенком [16], устройство визуальных оповещений *Instagram*.

Для пользователей: В рамках платформы *Intel* создал полную линейку масштабируемых процессоров *Intel Quark*, которые широко покрывают сегмент устройств для систем умный дом и умный офис.

Для разработчиков: На данный момент аппаратная часть *Intel IoT* работает под управлением открытой ОС *Wind River Pulsar Linux* [19] – прямым конкурентом *Google Brillo*. Основные преимущества ОС от *Intel* – это глубокая оптимизация и интегрированная поддержка облачной модели *SaaS*, которая позволяет разрабатывать приложения для *IoT* в виртуальной среде, а также едино управлять всеми устройствами *IoT* с упрощенным контролем доступа.

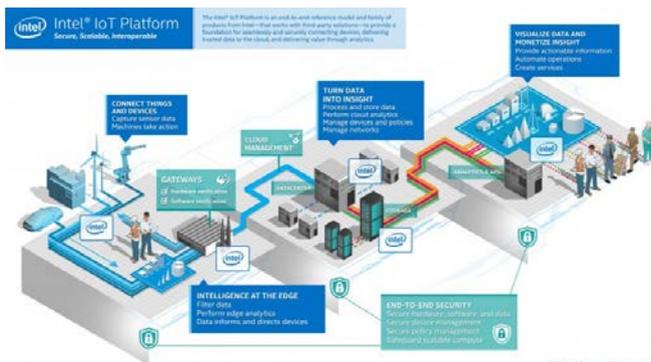


Рис. 8. Intel IoT Platform

Google Brillo

В мае 2015 года на конференции для разработчиков компания *Google* представила новую операционную систему *Brillo* (рис. 9), предназначенную для интернета вещей и умного дома. Отличительной особенностью платформы является функция *Weave*, которая позволяет устройствам в умном доме с системой *Brillo* и смартфонам общаться друг с другом напрямую, без использования промежуточного облака.

Краткий обзор аппаратных платформ

Основа стека технологий интернета вещей – устройства, собирающие данные с физических объектов и отправляющие их в сеть Интернет для хранения, обработки и анализа. Ключевым моментом здесь является способность устройства подключаться к сети с помощью беспроводных интерфейсов и обмениваться данными с удаленными серверами. Средствами подключения к сети обычно являются *Wi-Fi* и *GPRS*, как наиболее развитые и распространенные технологии.

Стремление к миниатюризации в электронике дало толчок к появлению *SoC* (*System on Chip*) – систем, которые интегрируют на одном чипе различные функциональные блоки, которые образуют законченное изделие для автономного применения в электронной аппаратуре [24]. *SoC* имеют богатый функционал, благодаря своей блочной структуре, и даже могут включать в себя радиочастотные преобразователи. Компактные системы позволили создать миниатюрные решения с малым энергопотреблением для решения задач управления физическими объектами.

Производители электронного оборудования, уловив тренд *IoT*, реализуют на рынке огромное количество продуктов, так или иначе применимых в задачах этой сферы. Практически еженедельно появляются новые платформы, которые стараются занять свое место в приложениях интернета вещей. Среди явных лидеров по распространенности можно выделить платформы *Intel Edison*, *MediaTek LinkIt*, *Raspberry Pi*, *Arduino MKR1000*. Среди менее популярных, но стоящих внимания платформ, можно указать *Adafruit Feather*, *Arduino 101*, *BeagleBone Black*, *C.H.I.P.*, *NXP ConnectCore i.MX6UL*, *DragonBoard 410c*, *Intel Curie*, *Intel Galileo*, *Linino One*, *Parallella Board Embedded Platform*, *Particle Photon*, *PINE A64*, *Seeed Arch*, *Spark Core*, *TI LaunchPad CC3200*. [23] Все эти устройства решают задачи в своей области и делятся на классы по производительности от простейших контроллеров ввода-вывода с поддержкой беспроводного интерфейса до мощных мультимедийных систем, поэтому сравнивать их напрямую не имеет смысла.

Появление в 2012 году платформы *Raspberry Pi* задало один из стандартов на встраиваемые устройства [25]. В свое время полноценный компьютер размером с кредитную карту с частотой ядра процессора порядка 1 ГГц и широким набором физических интерфейсов по цене 20-30 долларов совершил настоящий прорыв. В результате возникло множество подражателей, и некоторые из которых до сих пор конкурируют с оригинальным проектом *Raspberry Pi*, например: *Banana Pi*, *ODROID*, *CubieBoard*. *Raspberry Pi* выбирается многими разработчиками встраиваемых приложений и приложений Интернета вещей благодаря развитому сообществу, которое позволяет получить поддержку по любым проблемам, возникающим в процессе работы. Ее конкуренты в большинстве своем гораздо менее документированы и хуже поддерживаются, однако подкупают ценой на устройства.



Рис. 10. Плата *Raspberry Pi 3*

Еще одну из недавних революций совершила компания *Espressif Systems*, выпустив *Wi-Fi* чипсет *ESP8266*, стоимость которого начинается всего от 1 доллара США. Благодаря своей низкой цене он получил звание «народного *Wi-Fi*», поскольку значительно снизил планку вхождения в разработку мобильных систем и сделал технологию *Wi-Fi* широкодоступной. Для разработки устройств на базе *ESP8266* необходима минимальная внешняя обвязка, что положительно сказывается на стоимости и размерах устройств. Модули, созданные на основе *ESP8266* могут выполнять роль самостоятельного микроконтроллера или моста *UART–Wi-Fi* для любого стороннего устройства.

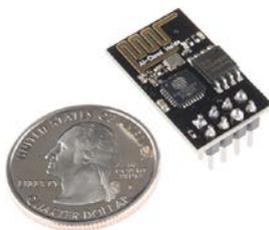


Рис. 11. Модуль *ESP-01* на основе чипа *ESP8266*

Существуют прошивки, специально предназначенные для решения задач интернета вещей, например, *ESP Easy*, которая поддерживает множество датчиков и исполнительных устройств (датчики температуры, влажности, освещенности, дисплеи, реле) и *WiFi-IoT.ru* – онлайн конструктор прошивок для систем домашней автоматизации [26, 27].

Максимально распространенным инструментом для создания проектов по автоматизации в пользовательском сегменте является платформа *Arduino*. Первые модели платформы оснащались простыми 8-битными микроконтроллерами с ядром *AVR*, затем стали появляться более производительные устройства на базе 32-разрядных чипов с ядром *ARM*, а сейчас в линейке *Arduino* присутствуют специальные решения для Интернета вещей (*Arduino YUN*, *Arduino TIEN*, *Arduino MKR1000*), в которые интегрирована поддержка *Wi-Fi* благодаря встроенным чипам производства *Atheros* [28].

Изначально в *Arduino* была заложена идея поддержки широкого спектра подключаемых модулей, и ее появление породило волну «*Arduino-совместимости*»: многие производители выпускают датчики или исполнительные устройства, которые были специально разработаны для сопряжения с *Arduino*. Это позволяет пользователю с минимальными затратами выполнить прототипирование системы автоматизации: согласование блоков обычно не требуется, схемы подключения любых компонентов и модулей к *Arduino* широко известны, прошивки для конкретных решений часто доступны в сети Интернет.

Умная теплица на базе технологий интернета вещей

В этом и последующих разделах приводится пример и разбор задания компетенции «Интернет вещей» Регионального отборочного Чемпионата «Молодые профессионалы» (*WorldSkills*) Новосибирской области 2017 года, которое предусматривало решение задачи организации мониторинга и управления в системе устройств, связанных через информационную сеть, путем создания приложения, обеспечивающего их взаимодействие. На основе данного задания был разработан комплекс лабораторно-практических занятий для студентов и школьников. В ходе данного цикла занятий поэтапно рассматриваются все технологические компоненты, которые используются для большинства устройств, работающих в соответствии с концепцией интернета вещей на платформах *ThingWorx* и *Arduino*.

Лабораторный стенд: «Умная теплица»

«Умная теплица» – это производственная система, нацеленная на обеспечение автоматизации технологических процессов выращивания растений, удаленного мониторинга этих процессов, а также возможности ручного вмешательства в процессы при необходимости. «Умная теплица» непрерывно находится в состоянии подключения к облаку через сеть Интернет и предоставляет пользователю графический интерфейс для наблюдения за показателями системы и их изменения, а также программный интерфейс для выгрузки и анализа данных. Система состоит из объектов, которые делятся на три группы:

1. Сенсоры – предназначены для сбора данных с реального объекта управления (в данном случае - теплицы для выращивания растений).
2. Исполняющие устройства – предназначены для создания управляющего воздействия на контролируемые параметры
3. Управляющие и интерфейсные модули – предназначены для координации работы сенсоров и исполняющих устройств, обменом информации с облаком

Пример задания на разработку «Умной теплицы»

Модуль 1: Разработка проекта системы мониторинга и управления технологическим процессом для заданного производственного модуля

В данном модуле необходимо разработать концепцию организации взаимодействия технологических единиц заданного производящего модуля (разрабатываемый проект является прототипом систем производства сельскохозяйственной продукции различных групп растений в закрытом грунте), а также способ ее реализации, а так же представить свой проект в форме презентации, выполненной в формате *Power Point*.

Технический проект должен содержать описание проблемы, стратегию и модель ее решения, включая организацию взаимодействия с пользователем (*UX*), пользовательского интерфейса (*UI*), информационной архитектуры (*IA*) и другие необходимые детали.

Презентация должна включать: (а) Изображения и минимальное количество текста, необходимые для понимания предлагаемой в проекте стратегии решения задачи; (б) Изображения и минимальное количество текста, представляющие техническую реализацию предложенной стратегии; (в) Изображения и минимальное количество текста, представляющие предложения по организации интерфейсов и веб-страниц приложения; (г) Изображения, схемы и другие иллюстративные материалы с минимальным количеством текста, касающиеся конкретных систем проекта (сбора и передачи данных / управления устройствами / процедур обработки информации и пр.).

Для пользователя необходимо создать эскиз дизайнера веб-страницы приложения, которая будет обеспечивать ему вывод получаемых значений отслеживаемых параметров в режиме реального времени и задание целевых значений для исполняющих устройств.

Модуль 2: Работа с информационной моделью

В данном модуле необходимо:

- Создать информационную модель системы на облачной платформе, задать ее свойства и службы.

- Организовать необходимую обработку данных в зависимости от задаваемого типа культур (параметров).

Модуль 3: Организация работы веб-приложения на симулированных данных, получаемых от клиента.

В данном модуле необходимо:

- Создать веб-интерфейс для обеспечения взаимодействия пользователя с ранее созданной информационной моделью.
- Обеспечить вывод получаемых значений с использованием веб-страницы отслеживаемых параметров в режиме реального времени и задание целевых значений для исполняющих устройств.

Модуль 4: Организация локальной системы управления

В данном модуле необходимо:

- Организовать сбор данных с датчиков температуры, влажности почвы и освещенности системы.
- Организовать вывод данных через последовательный порт на ПК.
- Организовать выдачу управляющих команд с ПК исполняющим устройствам теплицы.
- Отладить полученную локальную систему управления.

Модуль 5: Сопряжение веб-интерфейса приложения и локальной системы управления

В данном модуле необходимо:

- Настроить локальную систему управления для отправки реальных данных на сервер приложения.
- Настроить локальную систему управления для выдачи управляющих команд реальным исполняющим устройствам с сервера приложений.

Модуль 6: Демонстрация работоспособности системы и определение ее технико-экономических показателей

В данном модуле необходимо:

- Отладить систему, построенную в результате выполнения всех предыдущим модулей.
- Представить систему в форме презентации, выполненной в формате Power Point, с демонстрацией реальной работы проекта.

Занятие 1. Основы работы с платформой *ThingWorx*

В занятии рассматривается один из вариантов создания приложения на платформе *ThingWorx*, которое интегрирует данные, получаемые от устройств умного дома (лампы, термостата и электросчетчика).

Для работы с платформой *ThingWorx* необходимо зарегистрироваться по ссылке <https://developer.thingworx.com/> и подтвердить адрес электронной почты.

Для создания приложения предлагается использовать инструмент *ThingWorx Composer*. Для его запуска необходимо зайти на веб-страницу <https://developer.thingworx.com/> и нажать зеленую кнопку *Launch Foundation Server* (запуск основного сервера). Откроется новое окно с инструментом *ThingWorx Composer*. В данном окне расположен весь основной функционал для создания и настройки приложения.

Создание модели данных

Основной сервер использует модельный подход для создания функциональных блоков проекта, которые могут быть использованы многократно. Это позволяет в ходе разработки использовать принципы масштабируемости и гибкости.

В приложениях платформы *ThingWorx* создается *Data Model* (модель данных), которая содержит компоненты, называемые формы вещей и шаблоны вещей, используемые для структурирования данных, а также для определения свойств (характеристик) и сервисов (правил поведения) вещей.

Прежде чем создавать различные сущности на платформе, необходимо создать тэги модели, которые нужны для группировки объектов и разбиения их на категории по всей платформе. Тэги модели состоят из словаря и термина – *Vocabulary:Term*. Например, *Color:Blue* или *Department:Finance*. Как правило, для работы необходимо создать свой словарь и термины.

Создание словаря тэга модели. Нажмите на пиктограмму с домиком в верхнем левом углу, в разделе *Modeling* (моделирование) на панели навигации слева наведите мышь на пункт *Model Tags* (тэги

модели) и нажмите на значок «+», в поле *Name* (имя) введите *FoundationTutorial*, сохраните результат.

Создание термина тэга модели. Откройте словарь *FoundationTutorial* и выберите пункт *Edit* (редактировать), выберите пункт *Manage Terms* (управление терминами), введите в поле *Home Application* и нажмите кнопку *Add Term* (добавить термин), сохраните результат.

На рисунке 12 представлен интерфейс инструмента при создании термина тэга модели.



Рис. 12. Создание термина тэга модели

Создание форм вещей

Формы вещей – это базовые компоненты определения, которые содержат уникальные характеристики и правила поведения. В терминах программирования они являются интерфейсом.

Создание формы электрической лампочки. Щелкните на иконку домика в верхнем левом углу (см.рис.13), на панели слева наведите курсор на строку *Thing Shapes* (формы вещей) и нажмите на кнопку со знаком «+». Задайте имя формы *LightShape* и укажите тэг *FoundationTutorial:HomeApplication*. Также вы можете использовать автодополнение этого поля, чтобы найти и выбрать *HomeApplication*.

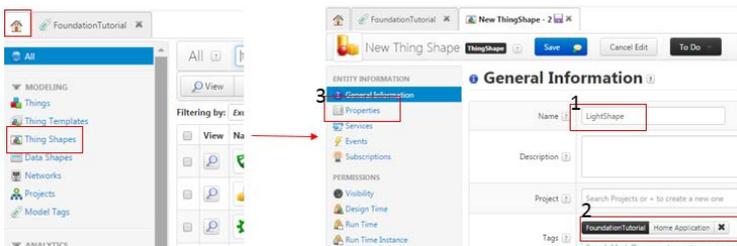


Рис. 13. Создание формы вещей

Выберите вкладку *Properties* (свойства) слева. Нажмите на кнопку *Add My Property* (добавить мое свойство) вверху. Если вы добавляете лишь одно свойство, то после заполнения полей нажмите кнопку *done* (закончить), а если вы добавляете несколько свойств, то можете использовать кнопку *done and add* (закончить и добавить).

Создайте свойства для формы электрической лампочки (см.рис 14; все поля, не указанные в данном примере, можно игнорировать).

Name	Base Type	Persistent*	Logged^	
lightID	String	✓	✓	
watts	Integer	✓	✓	
dimLevel	Number	✓	✓	
runtime	Integer	✓	✓	
onTooLong	Boolean	✓	✓	
status	Boolean	✓		
totalRunTime	Integer	✓		

Рис. 14. Создание свойств формы вещей

Важно:

- Когда выбран флаг *Persistent* (сохраняемый), то значение свойства сохранится при перезагрузке системы или Вещи. Свойства, которые не являются сохраняемыми, будут сброшены в свое изначальное состояние при перезагрузке системы или Вещи.
- Когда выбран флаг *Logged* (регистрируемый), то значение свойства автоматически будет записываться в поток значений, когда данные будут изменяться.
- Убедитесь, что названия свойств в точности совпадают с тем, что здесь написано. Через несколько этапов вы будете запускать симуляцию, которая требует точного совпадения названий свойств.

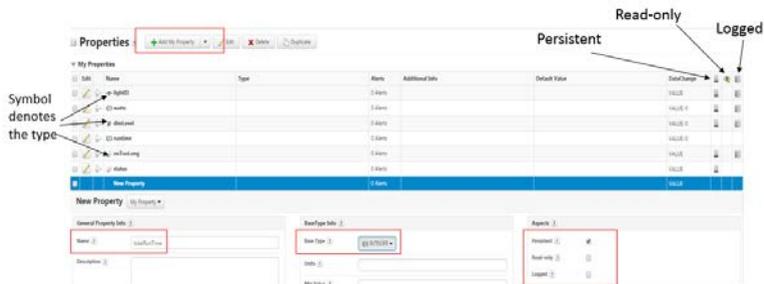


Рис. 15. Свойства формы вещей

Создание формы термостата. Напишите в панели поиска в левом верхнем углу *+thing* и нажмите клавишу *Enter* для создания новой формы вещей. Это сокращение может использоваться для создания чего угодно при нахождении на домашнем экране. Назовите форму *ThermostatShape* и выберите тэг *FoundationTutorial:HomeApplication*. Выберите вкладку *Properties* слева, нажмите на кнопку *Add My Property* сверху. Создайте свойства формы термостата (см. рис.16).

Name	Base Type	Persistent*	Logged^
thermostatID	String	✓	✓
temperature	Number	✓	✓
measurementScale	String	✓	✓

Рис. 16. Свойства формы вещей для термостата

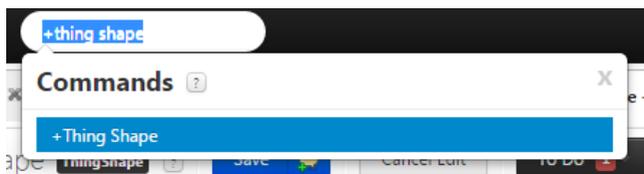


Рис. 17. Использование сокращения

Создание формы электросчетчика. Создайте новую форму вещи, дайте ей название *ThermostatShape* и укажите тэг *FoundationTutorial:MeterApplication*. Выберите вкладку *Properties*

слева, нажмите на кнопку *Add My Property* сверху. Создайте свойства, перечисленные на рис.18.

Name	Base Type	Persistent*	Logged^	
meterID	String	✓	✓	
measurement	Number	✓	✓	
currentCost	Number	✓	✓	
costPerHour	Number	✓	✓	

Рис. 18. Свойства формы вещей для электросчетчика

Создание шаблона вещей

Шаблоны вещей схожи с формами вещей в плане задания характеристик и правила поведения вещи. Разница в том, что шаблон может содержать несколько форм. Шаблоны обычно используются для представления набора схожих объектов. В терминах программирования шаблон – это абстрактный класс.

Создание шаблона вещи. Нажмите на пиктограмму домика в левом верхнем углу, на левой панели навигации наведите мышью на строку *Thing Templates* (шаблоны вещей) и нажмите на «+». Задайте имя шаблона *HouseGateway* и тэг *FoundationTutorial:HomeApplication*. В поле *Base Thing Template* (базовый шаблон вещи) введите *GenericThing*, в поле *Implemented Shapes* (задействуемые формы) нажмите на волшебную палочку. Добавьте три созданные формы: *MeterShape*, *ThermostatShape*, *LightShape*.

На предыдущих шагах были добавлены свойства в формы вещей, эти свойства могут быть также добавлены на уровне шаблона. Свойства доступны для всех вещей, которые созданы с использованием этого шаблона.

Для добавления свойств выполните следующие действия: выберите вкладку *Properties* слева, нажмите на кнопку *Add My Property* сверху и создайте свойства, перечисленные на рис.19.

Name	Base Type	Persistent*	Logged^
houseID	String	✓	
address	String	✓	
city	String	✓	
state	String	✓	
house_lat_long	Location	✓	

Рис. 19. Добавление свойств шаблона

Важно:

Обратите внимание, как и где расположены свойства уровня шаблона, а также свойства отдельных форм.

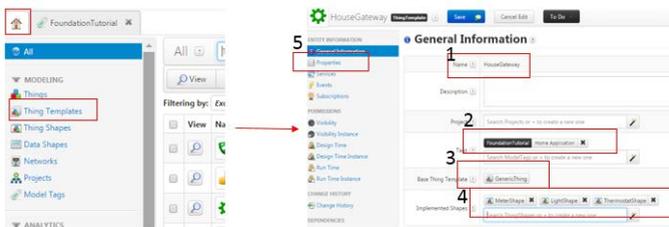


Рис. 20. Свойства уровня шаблона и отдельных форм

Создание вещи

Вещь – это цифровое представление физического объекта. В терминах программирования это то же самое, что класс.

На основе шаблона вещи, созданной на предыдущих шагах можно создавать вещь для конкретного дома. Дополнительные дома могут быть созданы вручную или программно с использованием этого же шаблона, и, таким образом, не нужно создавать свойства заново каждый раз.

Создание вещи, представляющей собой дом. Щелкните на пиктограмму домика в левом верхнем углу, на левой навигационной панели наведите мышью на строку *Things* (вещи) и нажмите «+». Задайте имя вещи *HouseThing* и укажите тэг *FoundationTutorial:HomeApplication*, в поле *Thing Template* (шаблон вещи) введите *HouseGateway*, сохраните результат.

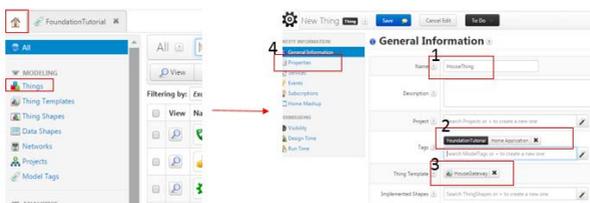


Рис. 21. Свойства вещи

Создание хранилища данных

Когда создана модель данных, нужно создать *Data Storage* (хранилище данных), в которое будут передаваться данные с устройства дома. Основной сервер предлагает несколько способов хранения данных. В данном варианте реализации предлагается использовать *Value Stream* (поток значений), который является быстрым и простым способом хранить последовательные по времени данные.

Создание потока значений. Нажмите на пиктограмму с домиком в левом верхнем углу, в секции *Data Storage* на левой панели навигации наведите мышь на *Value Streams* и нажмите «+», выберите шаблонную опцию *ValueStream* (см. рис.22). Задайте имя Потока *Platform_Quickstart_ValueStream*, добавьте тэг *FoundationTutorial:HomeApplication*, сохраните результат.



Рис. 22. Создание потока данных

Обновление шаблона вещи. Откройте шаблон вещи под названием *HouseGateway* (вы можете использовать поле поиска сверху окна, если нужная вкладка закрыта), если вы не находитесь в режиме редактирования, нажмите *Edit* в поле *Value Stream*, найдите или введите *Platform_Quickstart_ValueStream*, сохраните результат.

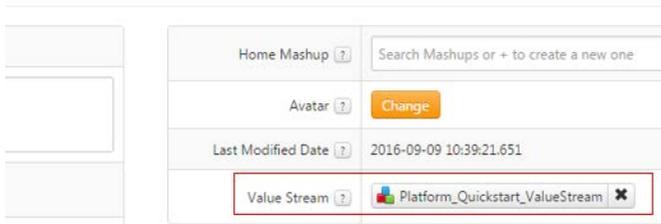


Рис. 23. Обновление шаблона вещи

Сервисы

Логика работы приложения создается путем написания сервисов на *Javascript*. Сервисы могут быть определены на уровне формы вещи, шаблона вещи или в самой вещи.

Платформа *ThingWorx* предлагает множество преднастроенных сервисов, а также отдельных функций, которые называются снippets. Пример преднастроенного сервиса – это *GetPropertyValues*, который получает все значения свойств для указанной вещи. Пример снippetsа – *dateFormat* – функция, которая обрабатывает дату в указанном формате.

Создание сервиса. Откройте форму под названием *MeterShape*, нажав на строку *Thing Shapes* (формы вещей) на левой панели навигации, перейдите в режим редактирования, если вы в нем не находитесь, нажав на кнопку *Edit*. Выберите пункт *Services* (сервисы) в столбце слева, нажмите кнопку *Add My Service* (добавить мой сервис) сверху, в секции *Name* введите *calculateCost*.

Теперь вам нужно создать *JavaScript* сервис, который умножает текущее показание электросчетчика на стоимость часа и сохраняет это в свойство, которое отслеживает текущую стоимость.

Выберите вкладку *Me*, затем внутри нее выберите вкладку *Properties* в левой панели. Заметьте, что там указаны несколько свойств, включая *costPerHour*, *currentCost* и *measurement*. Поместите ваш курсор в поле *Script* (скрипт) справа, выберите свойство *currentCost*, нажав на голубую стрелку. Это добавит свойство в поле скрипта. Дополните скрипт свойствами так, чтобы получился следующий код: $me.currentCost = me.measurement * me.costPerHour$. Сохраните результат.

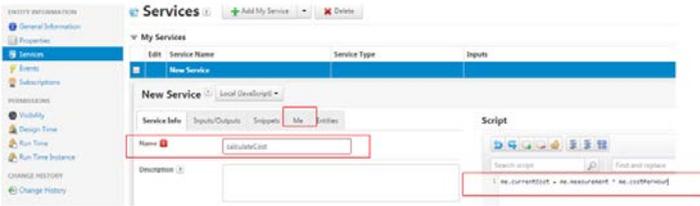


Рис. 24. Создание сервиса

События и подписки

Событие – это то, что запускает сервис. Например, изменение состояния или значения свойства. Другие вещи в том же приложении могут подписаться на событие. Подписка – это действие, которое выполняется, когда происходит событие.

Создание события. Откройте форму *LightShape* и выберите пункт *Properties* в столбце слева, нажмите на иконку карандаша напротив свойства *runtime*. Нажмите на кнопку *Manage Alerts* (управление уведомлениями) внизу панели редактирования, разверните список *New Alert* (новое уведомление) вверху и выберите пункт *Above* (превышение). Введите *runTimeTooHigh* в поле *Name*, Введите 14400 в поле *Limit*. Это заставит уведомление срабатывать, когда прошло 4 часа (или 14400 секунд) работы. Сохраните работу.



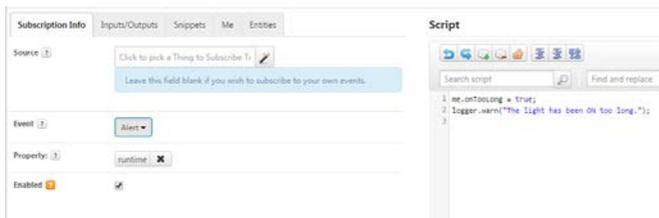
Рис. 25. Создание события

Создание подписки. Нажмите на *Subscriptions* (подписки) на левой панели формы *LightShape*. Нажмите на кнопку *Add My Subscription* (добавить мою подписку), выберите *Alert* (уведомление) из выпадающего списка *Event* (событие). Выберите свойство *runtime* в поле *Property*, нажав на волшебную палочку. Выберите *Enabled* (включено).

Создание сервиса, который устанавливает булеву переменную *'ontooLong'* и выводит предупреждающее сообщение в лог. Выберите вкладку *Me*. Нажмите на строку *Properties* в левой навигационной панели. Выберите свойство *'onTooLong'*, нажав на синюю стрелку.

Выставьте значение *True* для свойства *'onTooLong'*, используя *JavaScript: me.onTooLong = true;*

Запись сообщения в лог *ThingWorx*. Выберите вкладку *Snippets*, которая предоставляет несколько встроенных функций. Введите *warn* в поле поиска или пролистайте список вниз, пока не найдете его. Нажмите на стрелку вправо. Блок кода *JavaScript* добавится в окно скриптов. Введите предупреждающее сообщение между кавычек *logger.warn(«The light has been ON too long.»);*. Сохраните работу.



Создание приложения

Платформа *ThingWorx* дает возможность создания веб-приложений для визуализации получаемых данных. Визуализация называется мэшап и создается с помощью инструмента *Mashup Builder*.

Создание мэшапа. Нажмите на пиктограмму домика в правом верхнем углу. В секции *Visualization* (визуализация) на левой панели навигации наведите мышь на пункт *Mashups* (мэшапы) и нажмите «+». Оставьте все как есть и нажмите *Done*. Для того, чтобы открыть инструмент *Mashup Builder*, необходимо нажать кнопку *Edit*,

Важно:

Responsive (адаптивный) макет меняет свой размер в зависимости от размера экрана устройства, которое получает доступ к странице. *Static* (статический) макет сохраняет фиксированные размеры и позицию. Нажмите на кнопку *Info* (информация) вверху, чтобы просмотреть информацию о мэшапе. Задайте имя для вашего мэшапа, например, *'HomeConnectionMashup'*, и тэги. Сохраните работу.

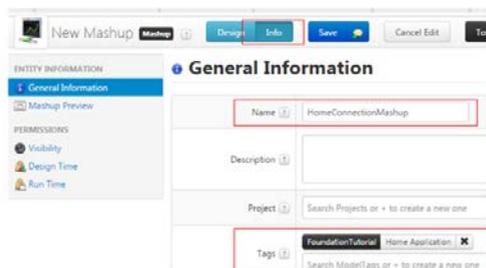


Рис. 27. Создание мэшапа

Создание макета. Нажмите кнопку *Design* (дизайн), чтобы переключиться обратно на экран дизайна мэшапа. Во вкладке *Widget* (виджет) выберите *Layout*. *Layout* (макет) – это адаптивный контейнер, который позволяет создавать разделы в вашем приложении. Макеты могут содержать другие макеты, так что вы можете создать приложение любого вида, какого хотите. Перетащите виджет *Layout* на полотно. Выберите *Vertical* (вертикальный) макет вверху. Нажмите кнопку *Done* в появившемся окне.

Важно:

Вы должны увидеть разделение полотна на два адаптивных ряда, которые обозначаются голубыми стрелками в каждом разделе. Перетащите еще один виджет *Layout* на верхний ряд. Нажмите кнопку *Done*. Повторите этот процесс для нижнего ряда.

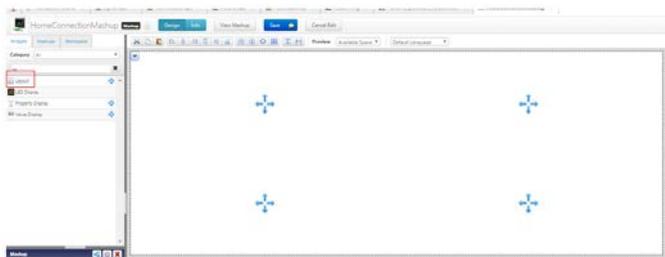


Рис. 28. Создание макета

Визуализация света

Наша вещь *HouseThing* содержит свойства, ассоциированные с лампочкой. Пользователь будет иметь возможность видеть состояние

лампочки (*ON/OFF*) и уровень яркости, а также сможет самостоятельно менять уровень яркости.

Визуализация лампочки. На вкладке *Widget* найдите *Panel* (панель) и перетащите его в верхний левый квадрант.

Важно:

После перетаскивания виджета на полотно он автоматически становится активным, и в нижнем левом углу экрана *Mashup Builder* появляется список его свойств. Каждый виджет имеет набор свойств, которые могут быть заданы и изменены.

На вкладке *Widget* найдите *Label* (надпись) и перетащите его в верхний левый квадрант. Впишите '*Living Room Light*' в поле свойства *Text* (текст). Во вкладке *Widget* найдите *Value Display* (дисплей значения). Виджет *Value Display* используется для отображения связанного значения. Обычно это значение берется из свойств вещи. Перетащите этот виджет в верхний левый квадрант. Впишите '*Light Status (ON/OFF):*' в поле свойства *Label*. Перетащите еще один виджет *Label* в верхний левый квадрант и впишите '*Dim Level*' в поле свойства *Text*.

Яркостью лампочки можно управлять из приложения. На вкладке *Widget* найдите *Slider* (ползунок) и перетащите его в верхний левый квадрант. Этот виджет предназначается для отображения значения свойства, но в комбинации с кнопкой он позволит нам его задавать.

На вкладке *Widget* найдите *Button* (кнопка) и перетащите его в верхний левый квадрант. Введите '*Set Dim Level*' в поле свойства *Label*.

На вкладке *Widget* найдите *Validator* (валидатор) и перетащите его в верхний левый квадрант. Виджет *Validator* используется для выполнения логических операций в интерфейсе пользователя. В выпадающем меню валидатора выберите пункт *Configure Validator* (настроить валидатор) и нажмите *Add Parameter*. Введите '*lightStatus*' в поле *Name* и выберите базовый тип *Boolean* (булевый), после чего нажмите *Done*. Введите строку *JavaScript* в поле *Expression* (выражение) валидатора, чтобы преобразовать значение *True* в '*On*', а *False* в '*Off*'. Строка выглядит следующим образом: *Output = lightStatus ? "On" : "Off"*; Выберите чекбоксы свойств *AutoEvaluate* и *Output*, после чего нажмите *Save*.

Визуализация электросчетчика

Для визуализации электросчетчика используется виджет *Gauge* (измеритель). Перетащите виджет *Label* в нижний левый квадрант. Впишите '*Energy Usage*' в поле свойства *Text*. На вкладке *Widget* найдите *Gauge*. Виджет *Gauge* показывает текущее значение свойства. Также он может быть настроен на индикацию различных диапазонов – красный, желтый, зеленый. Перетащите виджет *Gauge* в нижний левый квадрант.

Визуализация местоположения

В дополнение к статусу лампочки и значению электричества можно реализовать просмотр местоположения дома с помощью виджета *Google Map* (карты *Google*). Он использует стандартные координаты широты и долготы для местоположения на карте. На вкладке *Widget* найдите *Google Map*. Перетащите виджет *Google Map* в верхний правый квадрант.

Визуализация термостата

Зависимость температуры термостата от времени можно изобразить на графике. *Time Series Chart* (временной график) – это один из типов графиков, доступных для создания приложения, он может быть настроен на отображение как одного, так и нескольких параметров свойств от времени. На вкладке *Widget* найдите *Time Series Chart* и перетащите его в нижний правый квадрант. Сохраните работу.

Соединение сервисов с виджетами

Добавление сервисов в мэшап. После настройки визуальной части приложения, к ней можно присоединить сервисы, которые содержат данные, необходимые для мэшапа. В правой верхней части экрана *Mashup Builder* есть вкладки *Data* (данные), *Session* (сессия) и *User* (пользователь). Сейчас выберите вкладку *Data*. Нажмите на зеленый символ «+» справа. В поле *Search Entities* (поиск сущностей) введите *House*. Нажмите на *HouseThing*. В поле *Filter* (фильтр) справа от *Select Services* (выбор сервисов) введите *GetPropertyValues*. Нажмите на синюю стрелку справа от названия сервиса, чтобы его добавить. Выберите пункт *Mashup Loaded* (мэшап загружен).

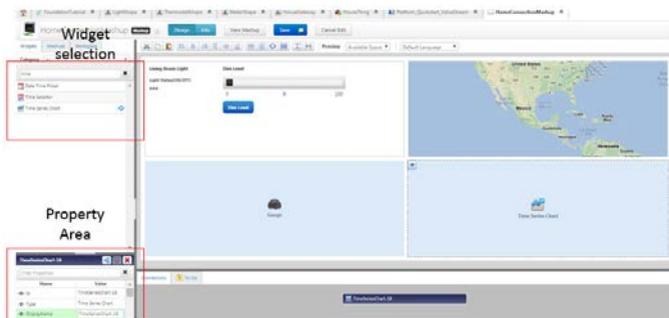


Рис. 29. Добавление виджетов

Добавьте сервис *QueryPropertyHistory*. Выберите чекбокс *Mashup Loaded*. Добавьте сервис *SetProperties*. Оставьте чекбокс *Mashup Loaded* невыбранным. Сохраните работу.

Важно:

При выбранной опции *Mashup Loaded* сервис выполняется, когда мэшап запускается или обновляется. Иногда сервис может запускаться только по определенному событию, поэтому этот чекбокс выбирается не всегда.

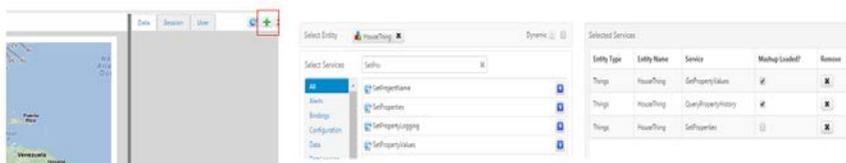


Рис. 30. Добавление сервисов в Мэшап

Соединение данных с виджетами. Добавленные сервисы присоединяются к виджетам.

Статус лампы. В секции *Things_HouseThing* справа нажмите на значок «+» напротив сервиса *GetPropertyValues*. В подсекции *Returned Data* (возвращаемые данные) разверните секцию *All Data*. Перетащите *Status* (статус) на виджет *Validator* в верхнем левом квадранте. В диалоговом окне с названием *Select Binding Target* (выберите цель связывания) выберите свойство *lightStatus*. Наведите курсор на стрелку вниз на виджете состояния света и выберите пункт *Binding Sources* (источники связывания). Откроется диалоговое окно *Add Data Binding*

(добавление связывания данных), нажмите на пункт *Output* (вывод) под именем вашего виджета валидатора. Нажмите *Done* для закрытия обоих диалоговых окон.

Уровень яркости для ползунка. В секции *Things_HouseThing* справа нажмите на значок «+» напротив сервиса *GetPropertyValues*. В подсекции *Returned Data* разверните секцию *All Data*. Перетащите *dimLevel* на виджет *Slider* в верхнем левом квадранте. При появлении диалогового окна *Select Binding Target* выберите *# Value*. Наведите курсор на виджет кнопки в верхнем левом квадранте. Нажмите на стрелку вниз для открытия выпадающего списка. Перетащите событие *Clicked* (нажато) на сервис *SetProperties* справа. Нажмите на значок «+» напротив сервиса *SetProperties* справа, чтобы развернуть его. Нажмите на виджет *Slider* в верхнем левом квадранте. Нажмите на стрелку вниз для открытия выпадающего меню. Перетащите *# Value* на *dimLevel* в сервисе *SetProperties* справа.

Значение измерителя. Разверните сервис *GetPropertyValues*, а также секции *Returned Data* и *All Data*, если это еще не сделано. Перетащите свойство *measurement* (измерение) на виджет *Gauge*. В открывшемся диалоговом окне *Select Binding Target* выберите *# Data*.

Месторасположение дома. Разверните сервис *'GetPropertyValues'* и секцию *Returned Data*, если это еще не сделано. Перетащите *All Data* на виджет *Google Map*. В открывшемся диалоговом окне *Select Binding Target* выберите *Data*.

Нажмите на виджет *Google Map* в верхнем правом квадранте. Левая нижняя часть экрана содержит секцию со свойствами виджета *Google Map*. Задайте свойство *Location Field* (местоположение) как *house_lat_long* через выпадающее меню.

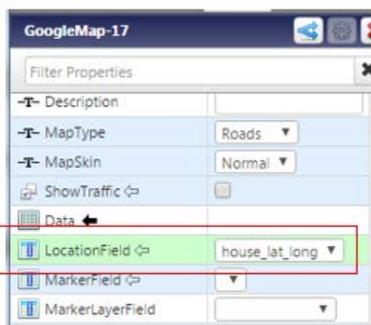


Рис. 31. Месторасположение дома

Потоковые значения для временного графика. Разверните сервис *QueryPropertyHistory* и секцию *Returned Data*, если это еще не сделано. Перетащите *All Data* на виджет *Time Series Chart*. В открывшемся диалоговом окне *Select Binding Target* выберите *Data*. Убедитесь, что в секции свойств в поле *DataField1* установлено *temperature*, а в поле *XAxisField* установлено *timestamp*. Сохраните работу. Вы можете убедиться, что связи установлены, нажав на виджет и просмотрев диаграмму внизу экрана.



Рис. 32. Установленные связи

Симулятор данных

Воспользуйтесь сущностями, включающими в себя сервис *JavaScript*, который симулирует данные, приходящие от дома и таймер, который запускается через заданный интервал времени.

Импорт сущностей симуляции данных. Скачайте на ПК файл https://github.com/skbrii/Thingworx/blob/master/Foundation_Tutorial_Services.xml. В окне *Composer* выберите вкладку *Import/Export* (импорт/экспорт) вверху страницы. Выберите *Import>From File*

(импорт из файла). Оставьте все значения по умолчанию и выберите файл, который вы только что скачали.

Просмотр импортированных сущностей. Выберите вещь *Foundation_Tutorial_Services*, используя панель поиска вверху экрана. Нажмите на *Subscriptions* на панели навигации слева. Нажмите на иконку карандаша слева от *ThirtySecondTimer*. Эта подписка основана на таймере. Каждые 30 секунд выполняется скрипт, который симулирует данные для вещи *HouseThing* и обновляет ее свойства.

Выберите вкладку *Subscription Info* (информация подписки). Активируйте пункт *Enabled*, чтобы запустить симулятор. Нажмите *Done* и *Save*.

Проверка выполнения симуляции. Откройте вещь *HouseThing*. Вы должны заметить, что большинство свойств теперь имеет значения.

Важно:

Если симуляция не работает, убедитесь, что названия всех свойств, которые вы создали, в точности совпадают с указанными в данном руководстве.

Запуск приложения

Перейдите на ваш мэшап и запустите приложение, нажав на кнопку *View Mashup* (просмотреть мэшап) вверху окна. Убедитесь, что данные правильно отображаются на мэшапе.

Занятие 2. Основы работы с платформой *Arduino*

Вводные замечания

Arduino – программно-аппаратная платформа, изначально основанная на микроконтроллерах серии *Atmega*. Самыми распространенными платами в семействе являются *Arduino UNO* и *Arduino Nano*, в основе которых лежит чип *Atmega328*. Это 8-битный контроллер с ядром *AVR*, который работает на частотах до 20 МГц, имеет порты ввода-вывода, аналого-цифровой преобразователь (АЦП), аппаратную поддержку распространенных цифровых интерфейсов (*UART*, *SPI*, *I2C*), 6-канальную широтно-импульсную модуляцию (ШИМ).

Arduino содержит 14 цифровых пинов, которые обозначены номерами от 0 до 13, и 6 аналоговых пинов, обозначенные от *A0* до *A5* (на некоторых платах до *A7*). Цифровые пины могут быть настроены в трех режимах:

OUTPUT (выход) – в этом режиме пин позволяет управлять своим логическим состоянием (устанавливать логическую единицу или логический ноль), также он подключается к драйверу тока, что позволяет управлять небольшой нагрузкой (светодиодом, транзистором, реле);

INPUT (вход) – в этом режиме пин может быть опрошен с целью определения его текущего логического состояния (например, нажата или не нажата кнопка);

INPUT_PULLUP (вход с подтяжкой) – данный режим аналогичен предыдущему, но в дополнение к этому, к пину подключается высокоомный резистор, подтягивающий вход к питанию (это необходимо в определенных схемах для устранения помех).

Аналоговые пины представляют собой входы аналого-цифрового преобразователя (АЦП), который дискретизирует входные аналоговые напряжения в диапазоне от 0 до 5В (по умолчанию) и выдает соответствующие им числа от 0 до 1023.

Кроме того, 6 пинов имеют функцию аппаратной широтно-импульсной модуляции (ШИМ). Эта техника позволяет управлять

мощностью нагрузки путем изменения скважности импульсов при постоянном периоде. ШИМ чрезвычайно распространена в системах управления двигателями и яркостью подсветки экранов. Пины, на которых присутствует аппаратная ШИМ, помечены на плате дополнительными символами ~ или #, и это пины 3, 5, 6, 9, 10, 11.

Arduino программируется на языке C++, код собирается компилятором *avr-gcc*. В ядре находится функция *main()*, которая при старте программы выполняет настройки микроконтроллера, затем вызывает функцию *setup()* один раз, после чего в бесконечном цикле вызывает функцию *loop()*. Таким образом, пользователь *Arduino* не пишет код для контроллера целиком, а пишет скетч (набросок), в котором описывает семантику функций *setup()* и *loop()* согласно логике своего встраиваемого приложения. Кроме того, пользователь может объявить и использовать любое количество собственных функций, а также функций из подключаемых библиотек (пока не закончится память размером 32 КБ). Для написания кода и его загрузки используется среда *Arduino IDE*, которую можно скачать с официального сайта <https://www.arduino.cc/en/Main/Software/>

Перечисленные выше режимы и функции пинов можно свести к использованию четырех функций, названия которых говорят сами за себя:

digitalWrite(pin, value); – запись высокого или низкого логического состояния в пин;

digitalRead(pin); – чтение логического состояния пина;

analogWrite(pin, value); – установка длительности импульсов ШИМ (от 0 до 255);

analogRead(pin); – чтение аналогового значения с помощью АЦП.

При использовании цифровых режимов (вход\выход) требуется единая настройка пина, для чего используется функция *pinMode(pin, mode)*. При использовании аналоговых функций дополнительная настройка не требуется.

Управление нагрузками

На рисунке 33 изображен внешний вид лабораторного стенда и принципиальная схема электронной составляющей теплицы. В схеме

отсутствуют линии *VCC* и *GND*, а также элементы подтяжки. Цифрами отмечены отдельные модули, которые будут использованы в работе.

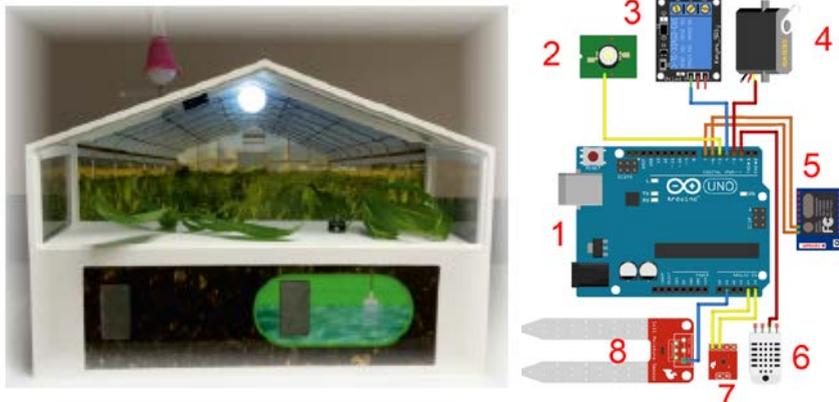


Рис. 33. Внешний вид лабораторного стенда и принципиальная схема электронной составляющей стенда

Примечание: 1 – *Arduino UNO*, 2 – светильник, 3 – реле, 4 – сервопривод, 5 – *Wi-Fi* модуль, 6 – температурный датчик, 7 – датчик освещенности, 8 – датчик влажности почвы.

Светильник подключен к цифровому пину 5. Заставьте его включаться и выключаться через каждую секунду. Для этого напишите следующий скетч:

```

// задаем макрос для пина, к которому подключен светильник
#define LIGHT_PIN 5
// функция setup() обязательна и выполняется один раз
void setup(){
    // задаем режим работы пина - цифровой выход
    pinMode(LIGHT_PIN, OUTPUT);
}
// функция loop() обязательна и выполняется бесконечно
void loop(){
    // запись высокого логического состояния в пин
    digitalWrite(LIGHT_PIN, HIGH);
    // пауза 1000 миллисекунд
    delay(1000);
    // запись низкого логического состояния в пин
    digitalWrite(LIGHT_PIN, LOW);
    // пауза 1000 миллисекунд
    delay(1000);
}

```

Реле управляется аналогично светильнику – высоким и низким логическим состоянием. Самостоятельно напишите скетч для одновременного управления светильником и реле. При этом светильник должен включаться и выключаться через каждую секунду, а реле – через каждые десять секунд.

Для управления сервоприводом можно использовать встроенную библиотеку *Servo*. Сервопривод можно установить в положение от 0 до 180 градусов, используя метод *write()*, аргументом которого является угол в градусах.

Напишите скетч, который выставляет сервопривод в положение 170 градусов, а затем, через 2 секунды, в положение 10 градусов, и обратно:

```

// подключаем библиотеку Servo
#include <Servo.h>
// задаем макрос для пина, к которому подключен сервопривод
#define SERVO_PIN 3
// создаем объект myservo
Servo myservo;

void setup(){
    // подключаем пин с сервопривод к объекту
    myservo.attach(SERVO_PIN);
}
void loop(){
    // устанавливаем сервопривод в положение 170 градусов
    myservo.write(170);
    delay(2000);
    // устанавливаем сервопривод в положение 10 градусов
    myservo.write(10);
    delay(2000);
}

```

Вывод информации на ПК

Arduino использует популярный интерфейс *UART* (он же *Serial*) для обмена информацией с ПК. На плате присутствует микросхема, выполняющая роль моста *USB-UART*, и общение ПК с *Arduino* происходит через виртуальный *COM*-порт.

В *Arduino IDE* есть инструмент, который называется «Монитор последовательного порта», который позволяет принимать и отправлять данные с/на *Arduino*. Для того чтобы общение состоялось, необходимо указать скорость соединения на обеих сторонах одинаковой. Набор скоростей фиксирован, самые популярные из них: 9600, 19200, 38400, 57600, 115200 бод.

Напишите скетч, который выводит в последовательный порт сообщения вида *count = xx*, где *xx* – инкрементирующееся значение переменной-счетчика.

```

// объявляем переменную-счетчик
int count = 0;

void setup(){
  // запускаем последовательный интерфейс на скорости
19200
  Serial.begin(19200);
}
void loop(){
  // печатаем текст без перевода строки
  Serial.print("count = ");
  // печатаем значение переменной-счетчика с переводом
строки
  Serial.println(count);
  // инкрементируем значение переменной-счетчика
  count++;
  delay(10);
}

```

Чтобы увидеть передаваемые данные на ПК, откройте монитор порта, щелкнув на значок «Увеличительное стекло» в правом верхнем углу *Arduino IDE*.

Обратите внимание на различие методов *print()* и *println()*: первый выводит содержимое без перевода строки, а второй – с переводом строки. Кроме этого, при необходимости можно использовать метод *write()*, который передает строго один байт.

Опрос датчиков

В работе используется датчик температуры *DTH11*. Для работы с ним можно воспользоваться существующими библиотеками. В *Arduino IDE* есть встроенный механизм скачивания библиотек из репозитория. Чтобы перейти в менеджер библиотек, выберите в меню пункт «Скетч», затем «Подключить библиотеку», затем «Управлять библиотеками...».

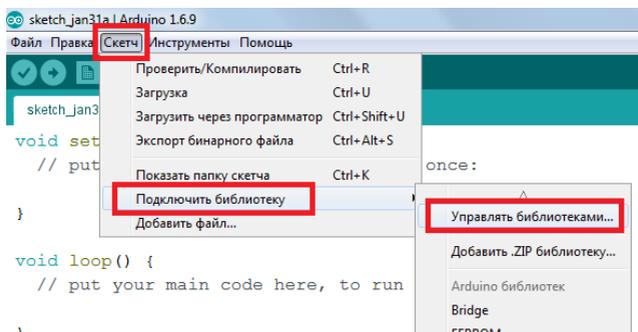


Рис. 34. Подключение библиотеки

Найдите и установите библиотеку *SimpleDHT*. После ее установки в *Arduino IDE* появятся примеры использования библиотеки, их можно найти через «Файл» → «Примеры» → «*SimpleDHT*». Откройте пример под названием *DHT11Default*, и скорректируйте его таким образом, чтобы пин, к которому подключен датчик, совпадал со схемой. Также обратите внимание на скорость соединения – она должна совпадать в скетче и в мониторе последовательного порта.

Следующий используемый датчик – это датчик освещенности *BH1750FVI*. Он имеет цифровой двухпроводной интерфейс I^2C и подключается к выходам *A4* и *A5* *Arduino*. Интерфейс I^2C позволяет одновременно подключить до 127 устройств на одни и те же пины, благодаря тому, что каждое устройство имеет собственный адрес на шине. Для опроса этого датчика тоже можно воспользоваться готовой библиотекой. Она может отсутствовать в репозитории, в таком случае ее можно скачать с *github* и установить вручную.

Скачайте библиотеку по ссылке <https://github.com/claws/BH1750>, распакуйте архив и скопируйте папку в «Документы/*Arduino*/libraries». Аналогично предыдущему заданию запустите пример «*BH1750test*» и проверьте его работу.

Последний используемый датчик – датчик влажности почвы. Он аналоговый, и подключен ко входу АЦП *A1*. Для чтения данных с этого датчика используется просто *analogRead(pin)*.

Напишите скетч для получения данных с датчика влажности и вывода в последовательный порт:

```

// задаем макрос для пина, к которому подключен датчик
влажности
#define HUM_SENSOR_PIN A1
// объявляем переменную для хранения получаемых значений
int humidity = 0;

void setup(){
  // запускаем последовательный интерфейс на скорости 19200
  Serial.begin(19200);
}
void loop(){
  // считываем значение с сенсора
  humidity = analogRead(HUM_SENSOR_PIN);
  // печатаем значение влажности в монитор порта
  Serial.println(humidity);
  // инкрементируем значение переменной-счетчика
  delay(100);
}

```

Создание управляющей логики

Напишите скетч, который будет выполнять управление устройствами в зависимости от показаний датчиков.

Управляющую логику задайте следующей:

- Если температура выше заданного верхнего предела – повернуть сервопривод так, чтобы открыть форточку.
- Если температура ниже заданного нижнего предела – закрыть форточку.
- Если влажность ниже заданного нижнего предела – включить реле насоса, чтобы наполнять водой емкость.
- Если влажность выше заданного верхнего предела – выключить реле насоса, чтобы перестать наполнять емкость.
- Если яркость ниже заданного нижнего предела – включить свет.
- Если яркость выше заданного верхнего предела – выключить свет.

На примере влажности код будет выглядеть примерно следующим образом:

```

#define HUM_SENSOR_PIN A1
#define RELAY_PIN 4
#define HUM_LOW_LIMIT 300
#define HUM_HIGH_LIMIT 800

int humidity = 0;

void setup(){
  pinMode(RELAY_PIN, OUTPUT);
  Serial.begin(19200);
}
void loop(){
  humidity = analogRead(HUM_SENSOR_PIN);
  Serial.print("humidity = ");
  Serial.println(humidity);
  if (humidity > HUM_HIGH_LIMIT){
    digitalWrite(RELAY_PIN, LOW);
    Serial.println("PUMP OFF");
  }
  if (humidity < HUM_LOW_LIMIT){
    digitalWrite(RELAY_PIN, HIGH);
    Serial.println("PUMP ON");
  }
  delay(100);
}

```

Подстройте значения пределов *HUM_LOW_LIMIT* и *HUM_HIGH_LIMIT* таким образом, чтобы система реагировала адекватно изменению входных параметров в соответствии с правилами.

Дополните код реакциями на изменение двух остальных параметров.

Занятие 3. Взаимодействие с платформой *ThingWorx*

Основные компоненты JSON:

JSON (JavaScript Object Notation) – простой формат обмена данными, удобный для чтения и написания как человеком, так и компьютером. Он основан на подмножестве языка программирования *JavaScript*. С помощью простого синтаксиса можно легко хранить все, что угодно, начиная от одного числа до строк, массивов и объектов, в простом тексте; связывать между собой массивы и объекты, создавая сложные структуры данных. Наиболее частое распространенное использование *JSON* – пересылка данных от сервера к браузеру. Во многих отношениях *JSON* может рассматриваться как альтернатива *XML*, по крайней мере, в сфере веб-приложений. Хотя *XML* является проверенной технологией, которая используется в достаточном количестве приложений, преимуществами *JSON* являются более компактный и простой для распознавания формат данных [29, 30]. Многие языки программирования имеют функции и библиотеки для чтения и создания структур *JSON*. Рассмотрим основные компоненты.

Объект – неупорядоченный набор пар ключ/значение. Объект начинается с открывающей фигурной скобки и заканчивается закрывающей фигурной скобкой. Каждое имя сопровождается двоеточием, пары ключ/значение разделяются запятой.

Массив – упорядоченная коллекция значений. Массив начинается с открывающей квадратной скобки и заканчивается закрывающей квадратной скобкой. Значения разделены запятой.

Значение может быть строкой в двойных кавычках, числом, *true*, *false*, *null*, объектом или массивом. Эти структуры могут быть вложенными.

Строка – коллекция, содержащая ноль или больше символов *Unicode*, заключенная в двойные кавычки. В качестве символа экранирования используется «\» – обратная косая черта. Символ представляется как односимвольная строка. Похожий синтаксис используется в *C* и *Java*.

Число представляется так же, как в *C* или *Java*, кроме того, что используется только десятичная система счисления.

Простой пример *JSON* файла – структура меню. В данном объекте содержатся атрибуты и массив, который включает другие объекты строки меню.

```
{
  "menu": "Файл",
  "commands": [
    {
      "title": "Новый",
      "action": "CreateDoc"
    },
    {
      "title": "Открыть",
      "action": "OpenDoc"
    },
    {
      "title": "Закрыть",
      "action": "CloseDoc"
    }
  ]
}
```

REST

REST (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем, таких как *WWW*, который, как правило, используется для построения веб-служб. Термин *REST* был введен в 2000 году Роем Филдингом, одним из авторов *HTTP*-протокола. Системы, поддерживающие *REST*, называются *RESTful*-системами. В общем случае *REST* является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как *URL*. Каждая *URL* в свою очередь имеет строго заданный формат. *REST* не связан строго с *HTTP*, но чаще всего его ассоциируют именно с ним.

Принципы *REST*:

- **Ресурсы** позволяют легко понять структуру каталогов *URI*.
- **Представления** передают *JSON* или *XML* в качестве представления данных объекта и атрибутов.

- **Сообщения** используют *HTTP* методы явно (например, *GET*, *POST*, *PUT* и *DELETE*).
- **Отсутствие состояния** взаимодействий не сохраняет контекст клиента на сервере между запросами [31].

Методы *REST*:

GET. Метод позволяет получать информацию. *GET* запросы должны быть безопасны и идемпотентны, т.е. независимо от того, как долго они повторяются с теми же самыми параметрами, результат останется тем же самым. Они могут иметь побочные эффекты, но пользователь не ожидает их, поэтому они не могут критичными для функционирования системы. Запросы могут быть также частичными или условными.

Получение адреса по *ID*, равному 1: *GET /addresses/1*

POST. Запрос с помощью этого метода позволяет что-либо сделать с ресурсом по *URI* с предоставлением сущности. Часто *POST* используется для создания новой сущности, но также возможно использовать и для обновления существующей.

Создание нового адреса: *POST /addresses*

PUT. Метод сохраняет сущность по *URI*. *PUT* может создавать новую сущность или обновлять существующую. *PUT* запрос идемпотентен. Идемпотентность – главное отличие в поведении между *PUT* и *POST* запросом.

Изменение адреса по *ID*, равному 1: *PUT /addresses/1*

PUT заменяет существующую сущность. Те элементы сущности, которые не представлены в запросе, будут очищены или заменены на *null*.

DELETE. Это запрос, который удаляет ресурс; кроме того, ресурс не должен быть удален немедленно. Он может быть асинхронным или "долгоиграющим" запросом.

Удаление адреса по *ID*, равному 1: *DELETE /addresses/1* [32].

Создание вещи и задание ее свойств

Для создания вещи и знания ее свойств необходимо войти в платформу *ThingWorx*, создать новый шаблон вещи с названием *testTemplate* и в качестве основного шаблона указать *GenericThing*.

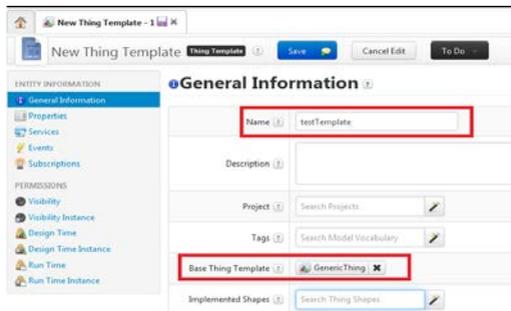


Рис.35. Создание шаблона вещи

Далее перейдите во вкладку *Properties* и создайте шесть свойств, которые будут означать температуру, влажность, яркость, состояние форточки, состояние насоса, состояние лампы, согласно таблице 1.

Таблица 1. Свойства вещи

Название свойства	Тип	<i>Persistent</i>	<i>Logged</i>
<i>temperature</i>	<i>number</i>	+	+
<i>humidity</i>	<i>number</i>	+	+
<i>brightness</i>	<i>number</i>	+	+
<i>ventState</i>	<i>boolean</i>	+	+
<i>pumpState</i>	<i>boolean</i>	+	+
<i>lightState</i>	<i>boolean</i>	+	+

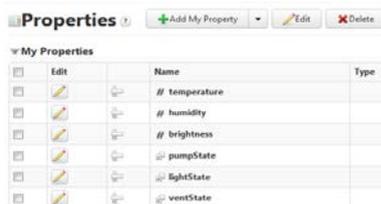


Рис. 36. Свойства вещи

Создайте вещь с именем *testThing* на основе шаблона *testTemplate*.

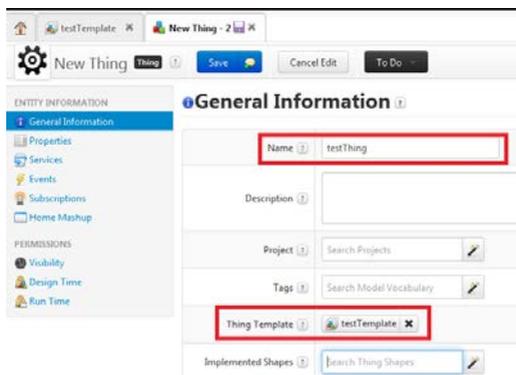


Рис. 37. Создание вещи

При создании вещи было создано несколько ресурсов со своими унифицированными идентификаторами (*URI*). Доступ к ресурсам может быть получен с помощью запросов (*HTTP*, *REST* и т.д.). Например, можно получить доступ к списку всех свойств созданной вещи, набрав в строке браузера *URL* <http://34.248.123.234/Thingworx/Things/testThing/Properties/>:

Property Listing For testThing 

name	value
brightness	12.0
description	
humidity	30.0
lightState	true
name	testThing
pumpState	true
tags	
temperature	80.0
thingTemplate	testTemplate
ventState	false

Рис. 38. Список свойств созданной вещи

Разберем *URL*:

34.248.123.234 – IP адрес сервера (приведен для примера, реально ваш адрес будет другим);

/Thingworx – название приложения, на котором работает платформа;

/Things – список всех вещей;

/testThing – название вещи;

/Properties – ресурс «свойства».

У созданной вещи есть другие ресурсы, которые можно запросить, например:

/ServiceDefinitions – список всех сервисов, которые присущи вещи;

/EventDefinitions – список всех событий, которые присущи вещи и т.д.

Создание Application key (Ключ приложения)

Ключ приложения необходим для обеспечения безопасности вашего приложения. Ключ используется для авторизации вашего запроса к серверу и является более простым способом, чем задание имени пользователя и пароля.

На панели слева выберите строку *Application Key* и нажмите на значок «+». В поле *Name* напишите имя ключа *testKey*, в поле *User Name Reference* (имя пользователя) выберите *Administrator* – это пользователь, на имя которого создается ключ. После нажатия кнопки *Save* сгенерируется ключ, который состоит из нескольких шестнадцатеричных значений. Скопируйте текст ключа для дальнейшего использования.

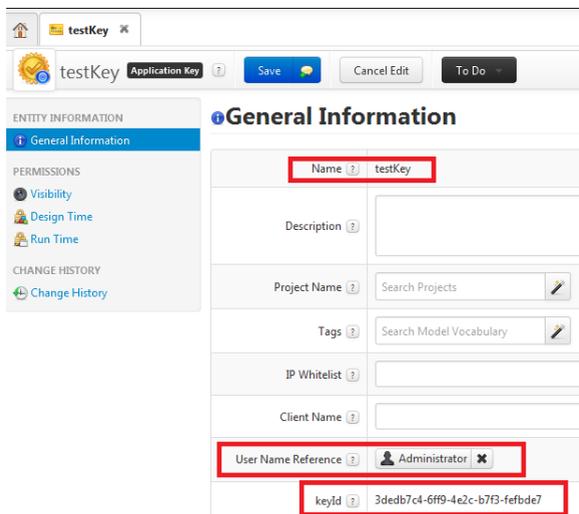


Рис. 39. Создание ключа приложения

Написание скриптов для взаимодействия с сервером

Для быстрого прототипирования скриптов, которые будут отправлять запросы серверу, предлагается использовать *Python 2.7*. Скачайте и установите интерпретатор *Python 2.7* с сайта <https://www.python.org/>, если он еще не установлен. Для написания кода можно использовать интегрированную среду разработки *PyCharm* (<https://www.jetbrains.com/pycharm/>), которая бесплатна в варианте *Community Edition*, или более простой текстовый редактор *SublimeText* (<https://www.sublimetext.com/3>), который поддерживает запуск скриптов *Python* по комбинации клавиш *Ctrl+B*.

Для упрощения работы с *REST API* в *Python* существует библиотека *requests*, которая может быть установлена через пакетный менеджер *pip*.

Для взаимодействия с ресурсами *Thingworx* используются три типа запросов:

- с помощью *PUT* запроса можно изменить значение любого свойства;
- с помощью *GET* запроса можно получить значение любого свойства;
- с помощью *POST* запроса можно запускать сервисы.

Напишите скрипт, который отправляет на сервер значение температуры, используя *PUT* запрос. Запрос состоит из *URL*, заголовков и тела. *URL* для отправки выглядит следующим образом: *http://34.248.123.234/Thingworx/Things/testThing/Properties/temperature*, однако, вместо того, чтобы указывать его напрямую, более корректным подходом будет вынести отдельные составляющие части в отдельные строковые переменные, а затем собирать *URL* с помощью конкатенации. В заголовках укажите тип контента *application/json*, чтобы отправить *JSON* объект на сервер, а также ключ, который вы получили на предыдущем шаге. Тело запроса представляет собой *JSON* объект, который состоит из словаря с ключом “*temperature*” и значением температуры.

```
# -*- coding: utf-8 -*-
import requests
import json
# IP адрес сервера
tws_IP = '34.248.123.234'
# ключ
app_key = '3dedb7c4-6ff9-4e2c-b7f3-fefbde7'
# название вещи
thing_name = 'testTing'
# свойство, значение которого меняем PUT запросом
put_property_name = 'temperature'
# URL для PUT запроса
put_url = 'http://' + tws_IP + '/Thingworx/Things/' +
thing_name + '/Properties/' + put_property_name
# заголовки для PUT запроса
put_headers = {'Content-Type' : 'application/json',
'apikey': app_key}
# данные для отправки в виде JSON (температура = 100)
put_data = '{"temperature": 100.0}'
# PUT запрос - отправляем значение параметра
put_r = requests.put(put_url, headers=put_headers,
data=put_data)
# печатаем ответ сервера
print "PUT RESPONCE:", put_r.json
```

Ответ сервера содержит код, который либо означает, что все выполнилось успешно, либо объясняет ошибку.

Ответ OK: *<bound method Response.json of <Response [200]>>*

Ответ «Ошибка авторизации»: *<bound method Response.json of <Response [401]>>*. Проверьте, что вы используете действующий ключ.

Ответ «Ресурс не найден»: *<bound method Response.json of <Response [404]>>*. Проверьте, что в *URL*, на который вы отправляете запрос, нет ошибок.

Напишите скрипт, который получает состояние форточки, используя *GET* запрос. Этот запрос состоит из *URL* и заголовков, тело у него отсутствует. Итоговый *URL* для отправки запроса должен выглядеть следующим образом:

http://34.248.123.234/Thingworx/Things/testThing/Properties/windowState.

В заголовках укажите принимаемый формат, который вы ожидаете от сервера: *application/json*, а также свой ключ.

```
# -*- coding: utf-8 -*-
import requests
import json
# IP адрес сервера
tws_IP = '34.248.123.234'
# ключ
app_key = '3dedb7c4-6ff9-4e2c-b7f3-fefbde7'
# название вещи
thing_name = 'testThing'
# свойство, значение которого меняем PUT запросом
get_property_name = 'windowState'
# URL для GET запроса
get_url = 'http://' + tws_IP + '/Thingworx/Things/' +
thing_name + '/Properties/' + get_property_name
# заголовки для GET запроса
get_headers = {'Accept' : 'application/json', 'apikey':
app_key}
# GET запрос - получаем значение параметра
get_r = requests.get(get_url, headers=get_headers)
print get_r.json()
```

В ответ на запрос сервер отдает *JSON* следующего содержания:

```
{ "dataShape" :
  { "fieldDefinitions" :
    { "windowState" :
      { "name" : "windowState",
        "description" : "" ,
```

```

        "baseType": "BOOLEAN",
        "ordinal": 1,
        "aspects":
            { "isReadOnly": false,
              "isPersistent": true,
              "isLogged": true,
              "dataChangeType": "VALUE",
              "cacheTime": 0.0}
            }
        },
        "rows": [{"windowState": true}]
    }

```

Для получения конкретного параметра (состояния форточки) из *JSON*, как видно, нужно выбрать значение по ключу *rows*, которое содержит массив, в этом массиве нужно взять нулевой элемент, и в нем по ключу *windowState* получить интересующее значение. Чтобы это сделать, допишите следующую строку и выполните скрипт еще раз.

```
print "GET RESPONCE:", get_r.json()['rows'][0]['windowState']
```

Напишите скрипт, который запускает сервис, используя *POST* запрос. Состав запроса зависит от того, какой сервис запускается. Полный список сервисов вы можете посмотреть по *URL* <http://34.248.123.234/Thingworx/Things/testThing/ServiceDefinitions/>. Запустите сервис под названием *GetPropertyValues*, который возвращает все значения всех свойств в виде *JSON* объекта. *URL* для отправки запроса должен выглядеть следующим образом: <http://34.248.123.234/Thingworx/Things/testThing/Services/GetPropertyValues>.

```

# -*- coding: utf-8 -*-
import requests
import json
# IP адрес сервера
twS_IP = '34.248.123.234'
# ключ
app_key = '3dedb7c4-6ff9-4e2c-b7f3-fefbde7a7c60'
# название вещи
thing_name = 'testThing'
# сервис, который запускается POST запросом

```

```
post_service_name = 'GetPropertyValues'
# URL для POST запроса
post_url = 'http://' + tws_IP + '/Thingworx/Things/' +
thing_name + '/Services/' + post_service_name
# заголовки для POST запроса
post_headers = {'Accept' : 'application/json', 'Content-
Type' : 'application/json', 'appkey' : app_key}
# POST запрос - запускаем сервис
post_r = requests.post(post_url, headers=post_headers)
# печатаем ответ сервера
print "POST RESPONSE:", post_r.text
```

В заголовках укажите в качестве отправляемого и принимаемого формата *application/json*, а также свой ключ. В данном случае у запроса нет тела, но, например, если требуется запустить сервис *SetPropertyValues*, то в теле запроса должен содержаться *JSON* объект.

Обратите внимание, что сервер вернул *JSON* объект, который содержит значения всех свойств вашей вещи. Таким образом, запустив сервис единственным запросом, можно сразу получить полную информацию о вещи.

Создание скрипта исполняющей логики

Найдите вещь *testThing* и перейдите к ее свойствам (строка *Properties* на левой панели). Напротив свойства *temperature* нажмите на кнопку с колокольчиком, чтобы открыть *Alert Manager* (менеджер предупреждений).

Создайте два предупреждения. Нажмите на кнопку *New Alert* (новое предупреждение) и выберите в выпадающем списке пункт *Above* (больше). После чего в поле *Name* напишите *temperatureTooHigh*, а в поле *Limit* напишите 40. Нажмите на кнопку *New Alert* еще раз и выберите в выпадающем списке пункт *Below* (меньше). После чего в поле *Name* напишите *temperatureTooLow*, а в поле *Limit* напишите 20. Нажмите *Update*, а затем *Save*. Теперь при значении температуры больше 40 или меньше 20 будут генерироваться предупреждения, на которые необходимо оформить подписку.

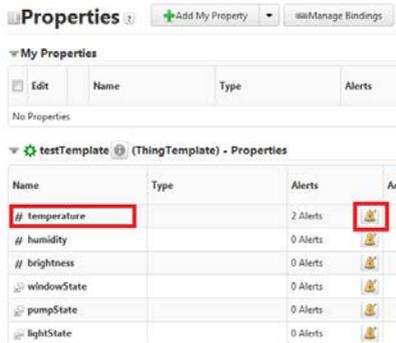


Рис.40. Открытие менеджера предупреждений



Рис. 41. Создание предупреждений

Для оформления подписки выберите на левой панели строку *Subscriptions*. Нажмите кнопку *Add My Subscription* (добавить мою подписку). Выберите в поле *Event* значение *Alert*, в поле *Property* впишите *temperature*, поставьте галочку в пункте *Enabled*. После чего, напишите в поле *Script* следующий текст:

```

if ( eventData.name == "temperatureTooHigh" ) {
    me.ventState = true;
    logger.warn( "Open window" );
}
if ( eventData.name == "temperatureTooLow" ) {
    me.ventState = false;
    logger.warn( "Close window" );
}

```

Нажмите *Done*, затем *Save*.

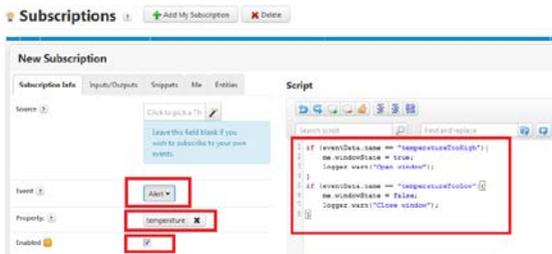


Рис. 42. Оформление подписки на предупреждение

Теперь создайте из двух скриптов на *Python*, которые отправляют *PUT* и *GET* запросы, один скрипт, который обрабатывает эти запросы по очереди. Выполняйте его, задавая различные значения температуры в отправляемом *JSON* объекте.

Обратите внимание, что после отправки значения больше 40, возвращается состояние форточки *True*, а после отправки значения меньше 20, возвращается состояние форточки *False*. Таким образом, логика работы приложения была реализована в облаке. Информация обо всех произошедших событиях сохраняется, и вы в любой момент можете просмотреть историю. Для этого, находясь в окне *Composer*, нажмите на вкладку *Monitoring* (мониторинг) на верхней панели и выберите пункт *Alert History* (история предупреждений).

Name	Property	Source	Source Type	Timestamp	Alert Type	Event Name	Location
temperatureBelow	temperature	testThing	Thing	2017-01-30 17:04:17.756	Above	Alert	0:0000 : 0:0000
temperatureAbove	temperature	testThing	Thing	2017-01-30 22:06:53.710	Above	Alert	0:0000 : 0:0000
temperatureAbove	temperature	testThing	Thing	2017-01-30 22:06:52.630	Below	Alert	0:0000 : 0:0000
temperatureAbove	temperature	testThing	Thing	2017-01-30 22:06:52.629	Above	AlertReset	0:0000 : 0:0000
temperatureAbove	temperature	testThing	Thing	2017-01-30 22:06:49.630	Below	AlertReset	0:0000 : 0:0000
temperatureAbove	temperature	testThing	Thing	2017-01-30 22:06:49.629	Above	Alert	0:0000 : 0:0000
temperatureAbove	temperature	testThing	Thing	2017-01-30 22:06:47.630	Below	Alert	0:0000 : 0:0000
temperatureAbove	temperature	testThing	Thing	2017-01-30 22:05:12.939	Below	AlertReset	0:0000 : 0:0000
temperatureAbove	temperature	testThing	Thing	2017-01-30 22:05:12.937	Above	Alert	0:0000 : 0:0000
temperatureAbove	temperature	testThing	Thing	2017-01-30 22:05:10.931	Below	Alert	0:0000 : 0:0000
temperatureAbove	temperature	testThing	Thing	2017-01-30 21:12:13.069	Below	AlertReset	0:0000 : 0:0000
temperatureAbove	temperature	testThing	Thing	2017-01-30 21:12:13.062	Above	Alert	0:0000 : 0:0000
temperatureAbove	temperature	testThing	Thing	2017-01-30 21:38:17.208	Below	Alert	0:0000 : 0:0000
temperatureAbove	temperature	testThing	Thing	2017-01-30 21:38:17.202	Above	AlertReset	0:0000 : 0:0000

Рис. 43. История предупреждений

Занятие 4. Взаимодействие *Arduino* и *ThingWorx*

Реализация REST запросов в Arduino

Изученный ранее *REST API* может быть использован для взаимодействия реальной физической системы с платформой *Thingworx*. *Arduino* может использовать запросы *PUT* для отправки значений измеряемых параметров на сервер и запросы *GET* для получения от сервера целевых значений для управления исполняющими устройствами.

Arduino подключена к беспроводной сети *Wi-Fi*, имеющей выход в сеть Интернет, через модуль *ESP8266*. Этот модуль содержит радиointерфейс 2,4 ГГц и собственный микроконтроллер, который может быть прошит различными прошивками для получения разнообразного функционала. В данном случае он запрограммирован как мост *UART–Wi-Fi* и способен отправлять на заданный *IP* адрес и заданный порт пакет данных, в который будет помещено содержимое входного буфера, а ответ сервера он запишет в выходной буфер, содержимое которого можно считать.

Arduino управляет модулем с помощью команд, отправляемых по интерфейсу *UART*. Аппаратный *UART* в *Arduino UNO* всего один, но на любых двух пинах можно организовать этот интерфейс программно, воспользовавшись встроенной библиотекой *SoftwareSerial*. Командой считается последовательность символов, завершающаяся символами возврата каретки '\r' и перевода строки '\n'. В таблице ниже перечислены некоторые команды. Полный перечень команд доступен по ссылке:

<https://github.com/skbrii/Thingworx/blob/master/ESPBridge/README.md>

Для того чтобы отправить запрос на сервер, следует действовать по следующему алгоритму:

1. задать сервер, к которому производится подключение;
2. задать порт на сервере;
3. очистить буфер;
4. сложить в буфер текст запроса;
5. подключиться к серверу и отправить содержимое буфера;
6. получить ответ.

Таблица 2. Команды, отправляемые по интерфейсу UART

Команда	Пример	Описание
<i>h</i>	<i>h192.168.1.1\r\n, hserver.skbrii.ru\r\n</i>	Выбрать 192.168.1.1 как сервер, к которому будет производиться подключение.
<i>p</i>	<i>p80\r\n</i>	Выбрать порт 80 как порт сервера, к которому будет производиться подключение. Порт по умолчанию - 88.
<i>a</i>	<i>aGET /test HTTP/1.1\r\n</i>	Добавить к буферу строку <i>GET /test HTTP/1.1\r\n</i> (<i>a\r\n</i> добавит к буферу строку <i>\r\n</i>).
<i>?sw</i>	<i>?sw\r\n</i>	Возвращает статус подключения <i>ESPBridge</i> к <i>wi-fi</i> -сети в виде <i>rFAIL_CONN2WIFI\r\n</i> (не подключен) или <i>rOK_CONN2WIFI\r\n</i> (подключен).
<i>?ss</i>	<i>?ss\r\n</i>	Возвращает статус подключения <i>ESPBridge</i> к серверу в виде <i>rOK_CONN2SERVER\r\n</i> (подключение установлено) или <i>rFAIL_NOTCONNECTEDTOSERVER\r\n</i> (не подключен).
<i>?bl</i>	<i>?bl\r\n</i>	Возвращает длину содержимого буфера в виде <i>rOK_BL:25\r\n</i> , где 25 - длина содержимого буфера.
<i>?bc</i>	<i>?bc\r\n</i>	Возвращает содержимое буфера <i>rOK_BC:<СОДЕРЖИМОЕ БУФЕРА>\r\n</i> , где <i><СОДЕРЖИМОЕ БУФЕРА></i> - содержимое буфера
<i>?ip</i>	<i>?ip\r\n</i>	Возвращает <i>ip ESPBridge</i> в виде <i>rOK_IP:192.168.1.104\r\n</i> .
<i>i</i>	<i>i\r\n</i>	Очистить буфер. Возвращает <i>rOK_BUFFER_CLEANED\r\n</i> .
<i>c</i>	<i>c\r\n</i>	Подключиться к заранее выбранному серверу.
<i>C</i>	<i>C\r\n</i>	Подключиться к заранее выбранному серверу, отправить содержимое буфера. К ответу сервера добавляется символ <i>g</i> в начале и <i>\$</i> в конце.

Минимальный скетч, который отправляет *PUT* запрос на сервер, выглядит следующим образом (замените *IP* и ключ приложения на свои):

```
// подключаем библиотеку для программного UART
#include <SoftwareSerial.h>
// задаем макросы для пинов, к которым подключен модуль
ESP
#define RX_ESP 6
#define TX_ESP 7
```

```

// задаем адрес сервера
const char serveraddr[] = "34.248.123.234";
// создаем объект класса SoftwareSerial для UART
SoftwareSerial esp(RX_ESP, TX_ESP);
// строка JSON для отправки PUT запросом
String jsondata;

void setup() {
  // запускаем интерфейс для ESP
  esp.begin(19200);
  // запускаем интерфейс для ПК
  Serial.begin(115200);
  // сообщаем адрес сервера модулю ESP
  esp.println("h34.248.123.234 ");
  // сообщаем порт на сервере модулю ESP
  esp.println("p80 ");
  // очищаем буфер
  esp.println("i ");
  // заполняем буфер текстом запроса
  esp.print(F("aPUT
/Thingworx/Things/testThing/Properties/temperature HTTP/1.1
"));
  esp.print((String)"aHost: " + serveraddr);
  esp.println(" ");
  esp.println("aappKey: 3dedb7c4-6ff9-4e2c-b7f3-fefbde7a
");
  esp.println("aContent-Type: application/json ");
  jsondata = "{\"temperature\": ";
  jsondata += tempValue;
  jsondata += "}";
  esp.print((String)"aContent-Length: " +
(String)jsondata.length()-1);
  esp.println(" ");
  esp.print("a");
  esp.print(jsondata);
  esp.println(" ");
  // отправляем запрос на сервер
  esp.println("C ");
}
// внутри loop делаем зеркало HardwareSerial <->
SoftwareSerial
void loop(){
  while (esp.available() > 0)
  {
    Serial.write(esp.read());
  }
}

```

```

}
while (Serial.available() > 0)
{
  esp.write(Serial.read());
}
}

```

Аналогичным образом напишите скетч для получения значения переменной *ventState* с помощью *GET* запроса. Ответ сервера на запрос содержит кроме *JSON* объекта дополнительный длинный текст, который сохранять в памяти *Arduino* чересчур затратно, поэтому *Arduino* в этом случае может получить из всего объекта лишь нужный ей словарь, который отдаст *ESP8266* по ключу, находясь в специальном режиме.

Таблица 3.

Команда	Пример	Описание
<i>Mjson</i>	<i>Mjson\r\n</i>	Установка режима, при котором после соединения и отправки данных возвращается содержащийся в ответе от сервера <i>JSON</i> -объект, или <i>rFAIL_NO_JSON:XXX</i> , где <i>XXX</i> - <i>HTTP</i> -код ответа. Возвращает <i>rOK_MODE_JSON\r\n</i> .
<i>Mjson</i>	<i>Mjson:YYYY\r\n</i>	Установка режима, при котором после соединения и отправки данных возвращается содержимое содержащегося в ответе от сервера <i>JSON</i> -объекта по ключу <i>YYYY</i> , или <i>rFAIL_NO_JSON_ENTRY:YYYY:XXX</i> , где <i>YYYY</i> - запрошенный ключ и <i>XXX</i> - <i>HTTP</i> -код ответа, или <i>rFAIL_NO_JSON:XXX</i> , где <i>XXX</i> - <i>HTTP</i> -код ответа. Возвращает <i>rOK_MODE_JSON_ENTRY\r\n</i> .

Для получения целевого значения управляемого устройства можно, например, написать следующую функцию:

```

String getData(){
  String Data;
  esp.println(F("i"));
  esp.println(F("Mjson:"));
  esp.print(F("aGET
/Thingworx/Things/TestT/Properties/ventState HTTP/1.1"));
}

```

```

esp.print(F("aHost: "));
esp.println(F("h34.248.123.234"));
esp.println(F("aAccept: application/json"));
esp.print(F("aappKey:      3dedb7c4-6ff9-4e2c-b7f3-fefbde7a
")););
esp.println(F("a"));
esp.println(F("C"));
return Data;
}

```

Обратите внимание на выражение $F(...)$ – это специальный макрос, который помещает строку во *FLASH* памяти контроллера вместо *RAM*. Этот макрос работает только со строками и позволяет экономить оперативную память.

Создание графического интерфейса приложения на платформе Thingworx

Создайте новый мэшап и назовите его *testMashup*. Выберите для него тип *Page* и свойство полотно *Responsive*. После перехода в инструмент *Mashup Builder* найдите в списке виджетов *Layout*, переместите его на полотно и укажите количество колонок – 3. Найдите в списке виджетов *Panel* и перенесите этот виджет в каждую колонку.

Создайте секцию графического интерфейса для управления температурой. Она будет включать в себя виджеты *Label*, *Gauge*, *Value Display*, *Validator*, *Button* (2 шт.).

Разместите указанные виджеты в первой колонке. В поле *Text* для *Label* напишите *Temperature*, в поле *Label* для *Value Display* напишите *Vent state*, в полях *Label* для кнопок напишите *Open vent* и *Close vent*. В результате должно получиться следующее:



Рис.44. Вид мэшапа

Для управления состоянием форточки с помощью кнопок, их необходимо привязать к сервисам, которые нужно создать. Для этого перейдите в редактирование вещи *testThing*. Выберите вкладку *Services* на левой панели и создайте два сервиса: *openVent* и *closeVent*, напишите для них скрипты с текстом *me.ventState = true;* и *me.ventState = false;* соответственно.

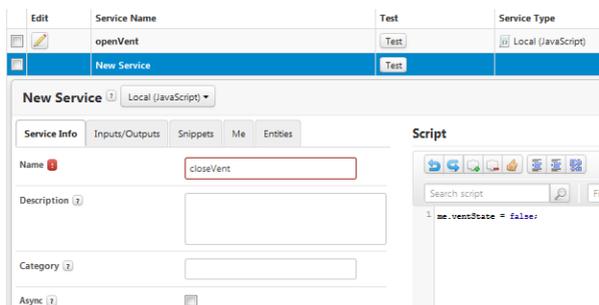


Рис. 45. Создание сервисов

Вернитесь в режим редактирования мэшапа. Нажмите кнопку со значком «+» на правой панели. В открывшемся окне в поле *Search entities* напишите *testThing*. Найдите следующие сервисы и добавьте их в колонку справа: *GetPropertyValues*, *SetPropertyValues*, *openVent*, *closeVent*. Отметьте чекбокс *Mashup Loaded* для пунктов *GetPropertyValues* и *SetPropertyValues*. Нажмите *Done*.

Add Data

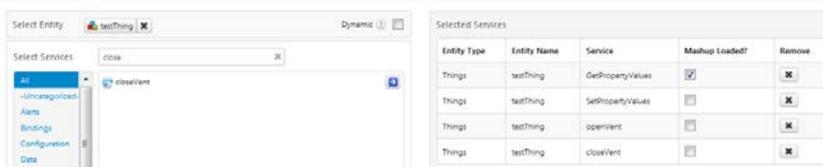


Рис. 46. Добавление сервисов в мэшап

Раскройте подпункт *All Data* в пункте *GetPropertyValues* и перетащите строку *temperature* на виджет *Gauge*. В выпадающем списке *Select Binding Target* выберите *Data*.

Выберите виджет валидатора и в его выпадающем списке выберите пункт *Configure Validator*. В открывшемся окне добавьте параметр

ventState и укажите тип *BOOLEAN*, сохраните результат. В левой нижней панели напишите в строке *Expression* выражение *output = ventState ? "Open" : "Close"*; отметьте чекбоксы *AutoEvaluate* и *Output*. Перенесите с правой панели пункт *ventState* на виджет валидатора, в выпадающем списке *Select Binding Target* выберите *ventState*. Откройте выпадающий список виджета *Value Display* и выберите пункт *Configure Bindings*. Нажмите *Binding Sources* и выберите пункт *Output* валидатора. Выберите кнопку *Open vent*, перенесите из выпадающего списка пункт *Clicked* на строку на сервис *openVent* на панели справа. Выполните аналогичное действие для второй кнопки.

По умолчанию мэшап обновляется только при перезагрузке страницы. Чтобы добавить автообновление, нужно использовать виджет *Auto Refresh*. Найдите этот виджет на панели слева и перенесите его на полотно. Задайте ему в поле *RefreshInterval* значение 1 и удалите отметку чекбокса *Visible*. В выпадающем списке виджета выберите пункт *Refresh* и перенесите его на сервис *GetPropertyValue* на правой панели.

Часть графического интерфейса, управляющая температурой, готова. Нажмите *View Mashup*, чтобы просмотреть приложение интерфейса.

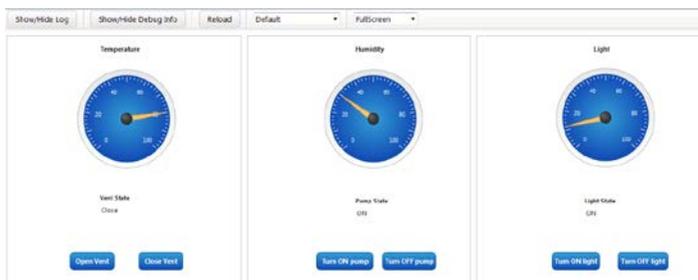


Рис. 47. Графический интерфейс управления температурой

Повторите все процедуры для создания двух оставшихся секций графического интерфейса для управления влажностью почвы и светом.

Протестируйте отправку всех трех измеряемых параметров с *Arduino* на платформу *ThingWorx* и получение целевых состояний управляемых устройств.

СПИСОК ИСТОЧНИКОВ

1. Radio-Frequency Identification. Wikipedia, the Free Encyclopedia: [Электронный ресурс], URL: https://en.wikipedia.org/wiki/Radio-frequency_identification. (Дата обращения: 12.12.2016)
2. Machine to Machine. Wikipedia, the Free Encyclopedia: [Электронный ресурс], URL: https://en.wikipedia.org/wiki/Machine_to_machine. (Дата обращения: 12.12.2016)
3. The Internet Toaster. Living Internet: [Электронный ресурс], URL: http://www.livinginternet.com/ia_myths_toast.htm. (Дата обращения: 01.10.2016)
4. The “Only” Coke Machine on the Internet. Carnegie Mellon University Computer Science Department, n.d. Web: [Электронный ресурс], URL: https://www.cs.cmu.edu/~coke/history_long.txt. (Дата обращения: 01.10.2016)
5. Stafford-Fraser, Quentin. “The Trojan Room Coffee Pot.” N.p.: [Электронный ресурс], URL: <http://www.cl.cam.ac.uk/coffee/qsf/coffee.html>. (Дата обращения: 01.10.2016)
6. Интернет вещей: краткий обзор. Вопросы и проблемы использования сети Интернет в более глобальном масштабе. Internet Society.: [Электронный ресурс], URL: <https://www.internetsociety.org/sites/default/files/report-InternetOfThings-20151221-ru.pdf>. (Дата обращения: 01.02.2017)
7. Cisco Visual Networking Index: Forecast and Methodology, 2014-2019 гг. Cisco: [Электронный ресурс], URL: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf (Дата обращения: 05.01.2017)
8. Meet the Nest Thermostat | Nest. Nest Labs. Интернет.: [Электронный ресурс], URL: <https://nest.com/thermostat/meet-nest-thermostat/>. (Дата обращения: 31.08.2015)

9. Samsung Privacy Policy - SmartTV Supplement. Корпоративная сеть Samsung: [Электронный ресурс], URL: <http://www.samsung.com/sg/info/privacy/smarttv.html>. (Дата обращения: 29.08.2016)
10. Язык разработки Python. [Электронный ресурс], URL: <https://www.python.org>. (Дата обращения: 01.02.2017)
11. Библиотека requests для обработки http-запросов на языке Python. [Электронный ресурс], URL: <http://docs.python-requests.org> (Дата обращения: 01.02.2017)
12. Формат обмена текстовыми JSON в веб-среде. [Электронный ресурс], URL: <http://www.json.org/json-ru.html> (Дата обращения: 01.02.2017)
13. Режепа В. «Интернет вещей» для потребителей и разработчиков: сравнение IoT-платформ. [Электронный ресурс], URL: <http://gagadget.com/science/22055-internet-veschej-dlya-potrebitelej-i-razrabotchikov-sravnenie-iot-platform/> (Дата обращения: 01.02.2017)
14. Closed Loop Control, Remote Sensors and Remote UX on RPi3. [Электронный ресурс], URL: <https://www.hackster.io/windows-iot/closed-loop-control-remote-sensors-and-remote-ux-on-rpi3-ef3ed0> (Дата обращения: 01.02.2017)
15. Hands-on-lab IoT Weather Station using Windows 10. [Электронный ресурс], URL: <https://www.hackster.io/windowsiot/build-hands-on-lab-iot-weather-station-using-windows-10-5b818f> (Дата обращения: 01.02.2017)
16. Intel Edison smart wearable baby monitor. [Электронный ресурс], URL: <http://www.instructables.com/id/Intel-Edison-smart-wearable-baby-monitor/?ALLSTEPS> (Дата обращения: 01.02.2017)
17. Wind River Pulsar Linux. [Электронный ресурс], URL: <https://github.com/WindRiver-OpenSourceLabs/wr-core> (Дата обращения: 01.02.2017)
18. Клементьев И.П., Устинов В. А.: Введение в Облачные вычисления.- УГУ, 2009, 233 стр.
19. Джордж Риз: Облачные вычисления.- BHV-СПб, 2011, 288 стр., ISBN: 978-5-9775-0630-4

20. Питер Фингар: «DOT. CLOUD. Облачные вычисления - бизнес-платформа XXI века», Акварина Книга, 2011, 256 стр., ISBN:978-5-904136-21-5
21. Gillam, Lee Cloud Computing: Principles, Systems and Applications / Nick Antonopoulos, Lee Gillam – L.: Springer, 2010. – 379 p. – (Computer Communications and Networks). – ISBN 9781849962407
22. Rittinghouse J.W., Ransom J.F. Cloud Computing - Implementation, Management, and Security. // Taylor and Francis Group, 2010, 174 pp.
23. IoT Hardware Guide [Электронный ресурс], URL: <http://www.postscapes.com/internet-of-things-hardware/> (Дата обращения: 01.02.2017)
24. Шагурин И. Системы на кристалле. Особенности реализации и перспективы применения // Электронные компоненты. — Издательский дом Электроника, 2009. — № 1
25. Официальный сайт Raspberry Pi [Электронный ресурс], URL: <https://www.raspberrypi.org/> (Дата обращения: 01.02.2017)
26. ESP Easy [Электронный ресурс], URL: <https://sourceforge.net/projects/espeasy/> (Дата обращения: 01.02.2017)
27. Конструктор прошивок [Электронный ресурс], URL: <http://wifiiot.com/> (Дата обращения: 01.02.2017)
28. Arduino Products [Электронный ресурс], URL: <https://www.arduino.cc/en/Main/Products> (Дата обращения: 02.02.2017)
29. Введение в JSON [Электронный ресурс], URL: <http://www.json.org/json-ru.html> (Дата обращения: 03.02.2017)
30. JSON: основы использования [Электронный ресурс], URL: <http://ruseller.com/lessons.php?rub=28&id=1212> (Дата обращения: 03.02.2017)
31. Понимание REST [Электронный ресурс], URL: <http://spring-projects.ru/understanding/rest/> (Дата обращения: 03.02.2017)
32. REST. Введение [Электронный ресурс], URL: <http://blogger.sapronov.me/2014/02/rest.html> (Дата обращения: 03.02.2017)

**Илья Сергеевич Дубков,
Павел Сергеевич Сташевский,
Ирина Николаевна Яковина**

**РЕШЕНИЕ ПРАКТИЧЕСКИХ ЗАДАЧ
НА БАЗЕ ТЕХНОЛОГИИ ИНТЕРНЕТ ВЕЩЕЙ**

Учебное пособие

В авторской редакции

Компьютерная верстка *Н. А. Баннова*

Подписано в печать 07.03.2017. Формат 60 x 84 1/16. Бумага офсетная.
Тираж 100 экз. Уч.-изд. л. 4,65. Печ. л. 5. Изд. № 58.
Заказ № 410. Цена договорная

Отпечатано в типографии
Новосибирского государственного технического университета
630073, г. Новосибирск, пр. К. Маркса, 20