УДК 004.031.43:004.75

РАЗРАБОТКА ПЛАТФОРМЫ ДЛЯ УПРАВЛЕНИЯ ИНФРАСТРУКТУРОЙ ИНТЕРНЕТА ВЕЩЕЙ

© 2017 А.А. Якименко[1,2], А.И. Белов[1], П.С. Гончарук[1], И.М. Стубарев[1]

[1] Новосибирский государственный технический университет
[2] Институт вычислительной математики и математической геофизики СО РАН, Новосибирск

Статья поступила в редакцию 11.12.2017

В статье представлена технология для удаленного мониторинга и управления устройствами, имеющими подключение к интернету. Предложена архитектура платформы, которая позволяет автоматизировать процесс взаимодействия устройств и платформы. Разработан прототип экспериментальной системы, работающей на основе данной платформы.
Ключевые слова: Интернет Вещей, платформа, веб-сервис, Java, REST, API.

Работа выполнена при поддержке гранта РФФИ 16-37-00240 мол_а.

1. ВВЕДЕНИЕ

В современном мире технологии, позволяющие автоматизировать окружающий человека мир, набирают популярность. Однако проблемы управления множеством устройств и мониторинга за ними не является тривиальной, а индустрия интернета вещей испытывает недостаток технологий и стандартов. Интернет Вещей – это технология, основанная на совмещении цифровых устройств и простых (бытовых, профессиональных, научных и др.) предметов и вещей в повсеместной жизни [1].

Основная проблема Интернета Вещей заключается в том, что нет общепринятых стандартов или согласованных архитектур для построения инфраструктуры. Одно устройство может работать с одним набором протоколов и средств обеспечения безопасности, другое устройство, соответственно, с другим набором. Некоторые устройства используют Wi-Fi, другие Bluetooth, а для третьих могут быть разработаны специфические способы взаимодействия. В случае, если отдельные фрагменты инфраструктуры (устройство или группа устройств) будут обеспечены надлежащими средствами обеспечения безопасности, мы должны будем сопрягать их работу с другими менее безопасными устройствами, что в результате приведет к падению уровня защищенности устройств до уровня самого уязвимого устройства [4].

Якименко Александр Александрович, кандидат технических наук, доцент кафедры вычислительной техники.
E-mail: yakimenko@corp.nstu.ru
Белов Антон Игоревич, магистрант.
E-mail: antonbelov1509@gmail.com
Гончарук Павел Сергеевич, магистрант.
E-mail: goncharuk.p.s@gmail.com
Стубарев Игорь Михайлович, магистрант.
E-mail: igor.stubarev@yandex.ru

На рынке уже много решений для создания инфраструктуры Интернета Вещей. Например, проекты, основанные на технологии PaaS. Такие решения имеют широкий функционал, но, как правило, занимают нишу дорогостоящих услуг. Платформа, рассматриваемая в данной статье, позиционируется как проект с открытым исходным кодом. Сравнивать его с платными платформами PaaS будет не корректным. Однако, нельзя не отметить наличие ряда преимуществ проектов с открытым исходным кодом перед платными продуктами. Ввиду того, что код открыт, всегда есть возможность проверить его безопасность с точки зрения отсутствия уязвимостей, а при их обнаружении такие уязвимости легко устранить. Также можно быть уверенным, что проект не закроется внезапно, ведь развитием занимается целое сообщество разработчиков, а не одна организация. Следствием этого является быстрое развитие за счет вклада сообщества разработчиков. Конечно, все эти преимущества играют роль только в случае эффективной архитектуры платформы, которая способна составить конкуренция другим продуктам.

Следовательно, целью работы является разработка платформы, которая будет управлять устройствами и следить за состоянием этих устройств через интернет. Предполагается, что платформа будет гибкой и, в тоже время, легко настраиваемой. Для достижения этих целей был разработан прототип платформы в виде веб-сервера, написанного на языке программирования Java. Прототип получил название IOPT (Internet Of Pretty Things). Планируется, что платформа будет проектом с открытым исходным кодом.

Прототип (как и сама платформа) имеет модульную структуру, что позволяет использовать сторонние проекты в качестве модулей. Эта особенность позволяет улучшить свойство рас-

ширяемости проекта, а также способствует развитию платформы. Модульность архитектуры также обеспечивает гибкость и многозадачность платформы. Особенное внимание при построении архитектуры уделяется масштабируемости и скорости обмена данными.

Рис. 1 иллюстрирует реляционную структуру связей между сущностями, описанными в данном абзаце.

В процессе разработки платформы, были введены понятия «модели» и «объекта» как части иерархии систем в платформе. Они являются основными инструментами в построении «умной» организации. Модель в этом контексте понимается как отображение самой организации, для которой планируется использовать платформу. Например, автоматическая теплица понимается, как модель «умной теплицы». Объект, в свою очередь, это наблюдаемый (и/или контролируемая) единица в данной модели. Например, секция или комната в теплице. Каждый объект имеет настраиваемый набор «свойств», за состоянием которых можно наблюдать и которые можно изменять. «Скрипт» еще одна сущность иерархии платформы, которая может быть привязана к любому свойству, а ее содержимое (в простейшем случае – код на языке JavaScript), может исполняться при наступлении некоторых событий (например, при изменении значения свойства).

2. ПРИМЕНЯЕМЫЕ ТЕХНОЛОГИИ

Сегодня интернет вещей представляет собой абстрактную коллекцию видов использования и продуктов [4].

В качестве реализации платформы была выбрана технология построения RESTful вебсервисов (REST, REpresentational State Transfer – архитектурный стиль взаимодействия между компонентами распределенного приложения в сети) [2].

Веб-сервисы позволяют разрабатывать функциональные инструменты, доступные через интернет. Например, стандартизированные функциональные интерфейсы достаточно легко использовать для разработки клиентских приложений под различные пользовательские устройства (например, планшеты, смартфоны и т.д.).

Единственное, что необходимо – доступ в интернет. Вследствие этого, технологии вебсервисов делают возможным разработку кроссплатформенных решений и использование любые языки программирования для разработки как модулей в виде сервисов, так и потребителей услуг, предоставляемых сервисами. Например, один модуль может быть написан на языке Java, другой – на C++. Связующим звеном здесь является HTTP-протокол, что позволяет избежать ограничений, связанных с совместимостью тех или иных средств разработки, а также протоколов общения между устройствами и платформой.

Упрощенная схема реализации архитектуры REST изображена на рис. 2.

В общем, REST – это простой интерфейс для управления информацией без применения дополнительных внутренних слоев. Глобальный идентификатор, такой как URL-адрес, однозначно идентифицирует каждую часть информации. Каждый URL-адрес в свою очередь имеет определенный формат. Отсутствие дополнительных

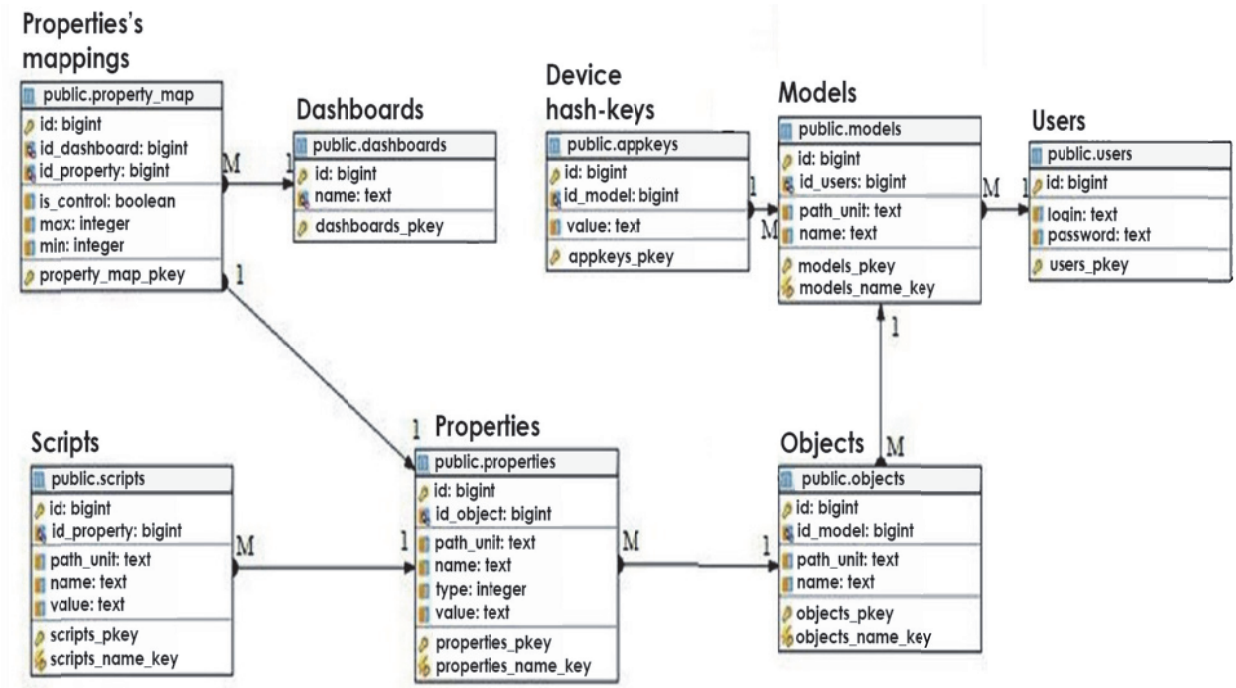

Рис. 1. Реляционная модель данных

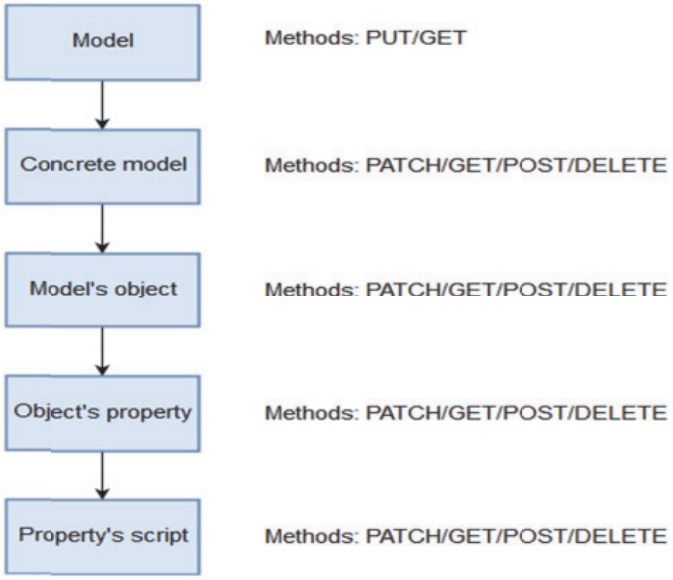

Рис. 2. Упрощенная схема реализации архитектуры REST

внутренних слоев означает передачу данных в его первоначальной форме. То есть данные не обернуты в XML, так как это делается в случае SOAP и XML-RPC, не используется AMF, как во Flash, и т. д. Отправка исходных данных происходит в том же виде, в котором они представлены изначально [2].

Как сказано выше, URL является уникальным идентификатором единицы информации, можно сравнить его с первичным ключом единицы данных в реляционной СУБД. Рассмотрим следующий пример – третья книга с книжной полки будет представлена как ссылка /bookshelf/book/3, а третья страница в этой книге тогда будет иметь вид: /bookshelf/book/3/page/35. По этому адресу может находиться HTML-страница, отсканированная копия в виде набора файлов JPEG или текстовый документ.

Способ организации управления информацией сервиса полностью зависит от протокола передачи информации. Часто для этих целей используется протокол HTTP. Таким образом, для HTTP, действия с данными определены методами GET (получить), PUT (добавить, заменить), POST (добавить, изменить, удалить), DELETE (удалить). Хотя так называемые действия CRUD (Create-Read-Update-Delete) могут быть выполнены только лишь с использованием двух методов – GET и POST.

REST считается альтернативой стандарту SOAP. REST - не строго предопределенный стандарт, это скорее архитектурный и программный стиль. SOAP же, напротив, хорошо документированный согласованный стандарт. Преимущества SOAP в том, что его можно применять без разработки документации к его интерфейсам, т.к. SOAP является автодокументируемой технологией. REST, в свою очередь, хорош тем, что он не использует прослоек и не обертывает данные в дополнительные теги, что позволяет сокращать размеры трафика при обмене данными.

RESTful веб-сервис разработан с помощью технологий Java EE: Jersey, Glassfish, EJB, CDI, HTTP Servlets и т.д. Из любого приложения, имеющего доступ в интернет, можно осуществлять HTTP-запросы к этому сервису с помощью разработанного REST API. Встроенного графического интерфейса нет. Вместо этого предоставлена возможность разрабатывать собственные реализации графического интерфейса. Планируется разработка графического интерфейса в виде смежного проекта-модуля. Но также предполагается, что появятся реализации графического интерфейса с открытым исходным кодом. Тем самым избегается привязка к конкретному интерфейсу.

Java была выбрана в качестве основного языка разработки, т.к. она кроссплатформенная, широко распространена в различных сферах жизни человека, в которых применяются информационные технологии и динамично развивается, шагая в ногу со временем.

Рис. 3 демонстрирует структурную схему, выраженную компонентами Java EE. Клиентская машина является отображением пользовательским интерфейсом, использующим протокол HTTP для связи с сервером. Сервер предоставляет публичный доступ к набору веб-интерфейсов.

Библиотека Jersey является главной частью веб-интерфейсов. Эти веб-интерфейсы являются функциональными интерфейсами REST API. Клиент формирует HTTP-запросы, добавляет необходимые атрибуты к запросам и отправляет их по соответствующим адресам. Внутри сервера находятся модули, отвечающие за бизнес-логику. Эти модули разработаны с применением технологий JSP, EJB, CDI и т.д.

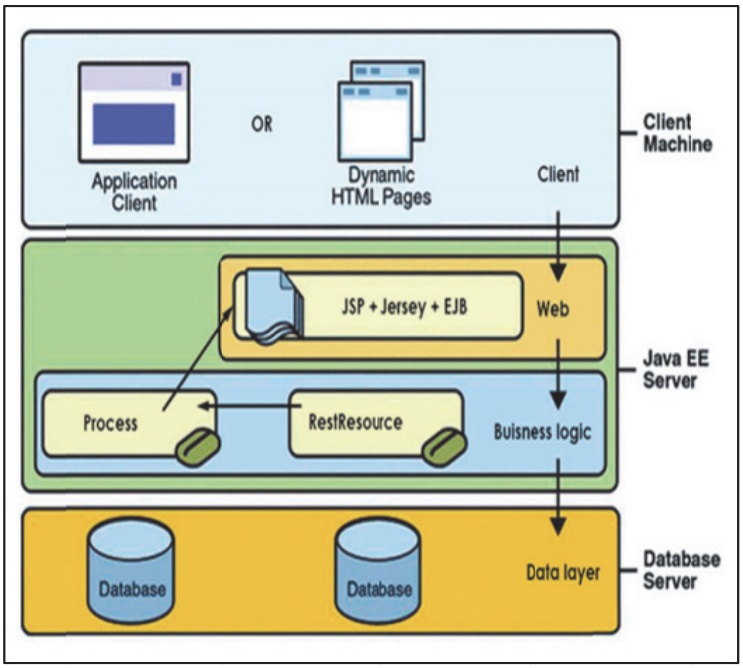

Рис. 3. JSON-structure of the platform's model

3. ДАННЫЕ. ХРАНЕНИЕ, ОБМЕН И ОБРАБОТКА

Данные передаются в формате JSON-структур. JSON (JavaScript Objective Notation) это относительно молодой формат, произошедший от языка JavaScript. От XML он отличается простотой описания данных, как для людей, так и для программных парсеров. JSON не имеет такой избыточности, как XML. Таким образом, JSON хорошо подходит для операций обмена данными благодаря своей компактности.

Сейчас JSON один из самых популярных форматов, который присутствует в многих web-API. Данные в формате JSON могут быть представлены в виде одной из двух структур: набор ключ-значение или упорядоченный набор значений в массиве. Ключ может быть строкой, значение любым поддерживаемым типом данных, включая вложенные структуры [3].

Формат JSON позволяет уменьшить размер передаваемых данных благодаря отсутствию служебной информации и различных тегов. Вся служебная информация при этом выносится в документацию, а не отправляется каждый раз совместно с данными.

Для построения графического интерфейса достаточно просто обработать данные, отвечающие за отображение модели и хранящиеся в виде JSON-структуры. Рис. 4 демонстрирует пример такой JSON-структуры в компактом виде.

Создание основного приложения для работы с платформой IOPT является не сложным процессом. Сначала были реализованы классы, отвечающие за основные сущности (модель, объект, свойство, скрипт). Потом разработана прослойка для доступа к данным, этот слой


```
{"id":null,"models":[{"objects":[{"modelId":-1,"properties":
[{"value":"false","type":3,"objectId":-1,"scripts":
[{"propertyId":-1,"value":"Some
code","id":-1,"pathUnit":"Name","name":"Name"},
{"propertyId":-1,"value":"Some other
code","id":-2,"pathUnit":"Name1","name":"Name1"}],"id":-1,"pathUnit"
:"LED","name":"LED"},{"value":"0","type":9,"objectId":-1,"scripts":
[],"id":-2,"pathUnit":"XPosition","name":"XPosition"},
{"value":"false","type":3,"objectId":-1,"scripts":
[],"id":-3,"pathUnit":"ButtonState","name":"ButtonState"}],"id":-1,"
pathUnit":"TestObject","name":"TestObject"}],"id":-1,"pathUnit":"Tes
tModel","name":"TestModel"}],"dashboards":[],"lastUpdate":"0001-01-
01T00:00:00+00:00"}
```


Рис. 4. JSON-структура модели платформы

получает текущую корневую модель в текущий промежуток времени. Эта модель уже включает в себя все вложенные модели с их набором объектов, свойств и скриптов. Также необходимо сопоставить действия add/delete/change (и т.д.) в коде с запросами PUT/DELETE/POST (и т.д.). Наконец, создать графический интерфейс исходя из собранной локальной копии модели. Таким образом, разработку моделей можно осуществлять с помощью средств версионной разработки, таких как, например, git.

На рис. 5 изображена диаграмма классов. Классы «Script», «Property», «Object» and «Model» представляют сущности реального мира и их свойства. «Snapshot» – класс, который содержит данные о текущем состоянии всей «умной» организации пользователя (что-то вроде массива моделей). Классы с суффиксом «Resource» (ресурсы) являются связующим звеном между пользовательскими веб-интерфейсами и внутренней бизнес-логикой. Классы с суффиксом «Proc» (Processors – процессоры) представляют собой реализацию бизнес-логики для каждого набора задач. «JDBCConnection» – класс-адаптер между ресурсами JDBC и сервером.

Ресурсы (в данном случае – классы с суффиксом «Resource») реализованы в виде ресурсов, предоставляемых библиотекой Jersey. Они обозначены как аннотации вида @Path(“/url_to_resource”), которые позволяют соотносить классы Java с генерируемыми документами HTML. Классы-процессоры (в данном случае – классы с суффиксом «Proc») реализованы с помощью технологии EJB и являются локальными «бинами» без сохранения состояний (один раз обработали в рамках одного запроса и полностью освободили память). Это позволяет на каждый запрос пользователя иметь управляемый сервером поток, при этом нет необходимости разрабатывать многопоточную модель, т.к. она уже встроена в технологию EJB и полностью поддерживается сервером Glassfish. Процессоры и ресурсы взаимодействуют посредством механизма CDI (Context Dependency Injection).

Платформа предполагает настраиваемые и взаимозаменяемые хранилища данных. В качестве хранилища данных можно использовать как обычные реляционные СУБД, такие как PostgreSQL, или это могут быть распределенные хранилища данных, поддерживающие JDBC. Это позволяет избежать привязки к одному конкретному хранилищу данных, тем самым расширяя сферу применения, ведь в зависимости от применения, одно конкретное хранилище данных может оказаться узким местом с точки зрения масштабирования. Например, платформа, работающая в кластере, обрабатывающем большие объемы данных.

Для удобства взаимодействия с платформой потребуется разработать сложные скрипты или полноценное сетевое клиентское приложение. В качестве примера был разработан вариант графического интерфейса в виде стационарного приложения под операционную систему Windows. Это приложение является внешним модулем по отношению к платформе. Внешний вид приложения представлен на рис. 6.

Рис. 7 демонстрирует архитектурную схему платформы. Основа платформы – модуль под названием IOPT-Server. Этот модуль является ядром платформы. Он обрабатывает все запросы: устройств; пользовательских приложений и сторонних модулей. Также этот модуль взаимодействует с хранилищем данных, он может быть запущен на любом сервере с установлен-

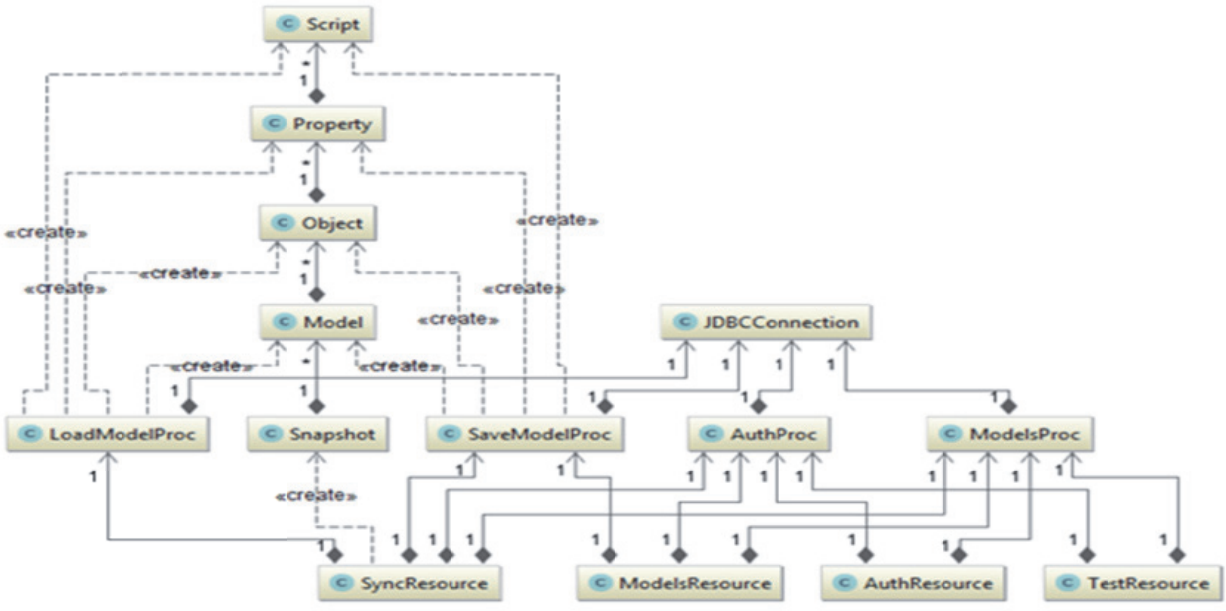

Рис. 5. Диаграмма классов

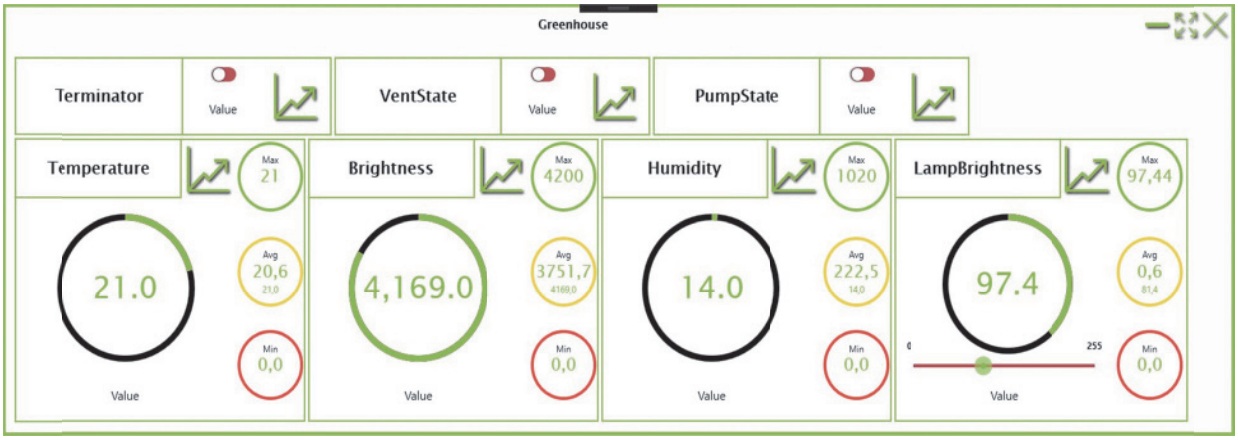

Рис. 6. Внешний вид демонстрационного графического интерфейса

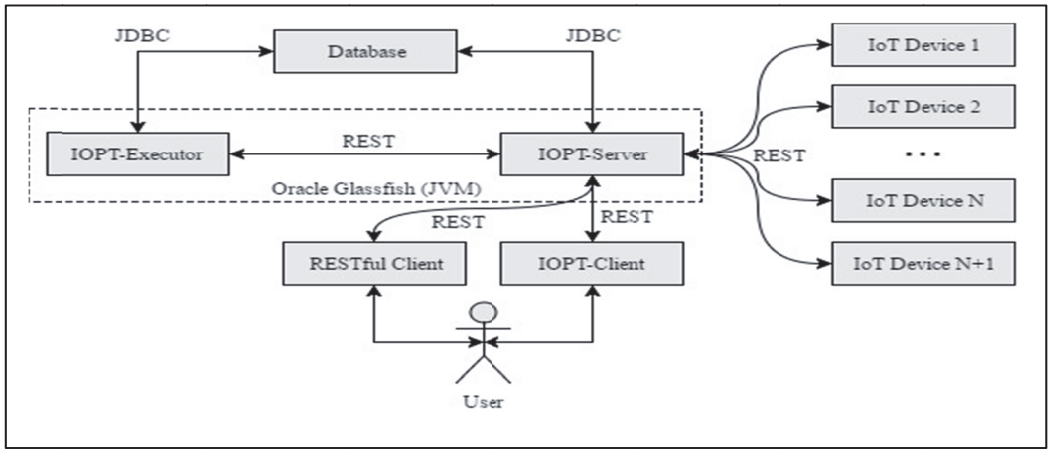

Рис. 7. Схема архитектуры платформы

ной средой и средствами разработки Java. Это предоставляет широкую целевую аудиторию и прозрачность платформы (т.к. исходный код открыт и есть возможность установки на изолированных серверах).

Планируется приспособить платформу для кластеризации, что позволит использовать платформу как на малых предприятиях с небольшим числом «умных объектов», так и в крупных корпорациях, которые включают в себя большое число «умных организаций», генерирующих внушительный трафик.

В общем случае структура запроса представляется в виде ссылок на ресурсы. В данном проекте ссылка представляет собой вложенную сущность (объект).

Например, http: // url / models / farm / garden1 / temp_prop, где ферма является моделью фермы, garden1 - это теплица с номером 1, а temp_prop - температура воздуха. Команда формируется по типу HTTP-заголовка (например, получить данные о температуре HTTP заголовка GET и ссылку, которая приведена выше).

Это обеспечивает человечески-читабельную форму запроса, что уменьшает количество ошибок и упрощает понимание принципа работы программного интерфейса. Единственным неудобством, например, по сравнению с SOAP, является необходимость разработки документации для REST API.

4. ОСОБЕННОСТИ ПЛАТФОРМЫ

В связи с тем, что взаимодействие с платформой происходит только через интерфейс программы RESTful API, исключается прямое взаимодействие с базами данных, поэтому ошибочный запрос не повредит их структуру. Защита от атак XSS и кражи данных настраивается аналогично защите любого веб-сайта, например, с использованием сертификатов TLS / SSL и HTTPS [7].

Для автоматизации поведения устройств платформа имеет возможность программно изменять их логику. Путем создания настраиваемых сценариев, о которых ранее уже было упоминание. На сегодняшний день поддерживается только язык программирования JavaScript, планируется реализовать поддержку Python и других популярных скриптовых языков. Сейчас эта возможность реализована как внутренний модуль (IOPT-Executor на рисунке 7).

IOPT-Executor написан с помощью Java Scripting API, который впервые появился с Java

6. В настоящее время используется движок под названием Nashorn, который был внедрен в Java 8. В нем реализована спецификация языка ECMAScript версии 5.1. Он поставляется в комплекте с Java SE 8. Он может использоваться в качестве инструмента для создания скриптов наряду с Java для создания приложений polyglot. Он предоставляет широкий спектр действий. Он предоставляет доступ из скриптов к Java и наоборот. IOPT-Server прослушивает события изменения свойств, и по происшествию которых, IOPT-сервер отправляет команду IOPT-Executor для выполнения всех скриптов, прикреплённых к изменению состояния у данного свойства.

Одной из чувствительных проблем Интернет вещей является слабая безопасность устройств в конечных точках, как и всей архитектуры от взлома. Для решения этой проблемы была разработана система аутентификации и авторизации пользователей и устройств. Эта система позволяет использовать сторонний модуль аутентификации или разрабатывать свои собственные, если требуемая функциональность отсутствует [7].

На рис. 8 показана схема взаимодействия пользователей и устройств с платформой. На сегодняшний день была разработана только базовая аутентификация по логину и паролю. Планируется реализация других популярных методов аутентификации (например, Kerberos, LDAP и Active Directory).

Пользователь не может изменять или просматривать модели других пользователей. При первом доступе к серверу пользователь предоставляет свой логин и пароль. В случае успеха сервер отправляет пользователю хэш-ключ, который позже позволит ему получить доступ к серверу без использования логина и пароля, аналогичный алгоритм реализован на веб-сайтах. Единственное различие заключается в том, что пользователь должен отправить хеш-ключ вручную, а не автоматически, как это делает веб-браузер. Используя API REST, пользователь может создать специальный хеш-ключ, который требуется для взаимодействия устройств с сервером. Каждое устройство связано только с одной моделью, поэтому для каждой модели создается отдельный хеш-ключ.

5. ПЛАНЫ РАЗВИТИЯ

На данный момент идет обновление и улучшение архитектуры системы и тестирование платформы. Планируется улучшить состав технологий и компонентов, используемых в платформе разработки, что уменьшит использование памяти и упростит процедуру развертывания приложения на серверах и обеспечит более гибкую масштабируемость. Это планируется сделать благодаря использованию технологии контейнеризации, которая реализуется в другом продукте - Docker.

Ресурсоемкий сервер приложений Glassfish планируется заменить на Jetty с интегрированным фреймворком Weld. Это упростит запуск платформы и сократит потребления памяти.

Docker - это платформа с открытым исходным кодом для разработки, доставки, запуска и быстрого развертывания приложений. Используя Docker, мы можем отделить свое приложение от своей инфраструктуры и обработать инфраструктуру как управляемое приложение.

Docker - легкий и быстрый. Он обеспечивает стабильную, экономически эффективную альтернативу виртуальным машинам на основе гипервизора. Это особенно полезно в средах с высокой нагрузкой, например, при создании собственного облака или платформы в качестве службы. Тем не менее, это также полезно для малых и средних приложений, когда надо получить больше эффективности от доступных ресурсов.

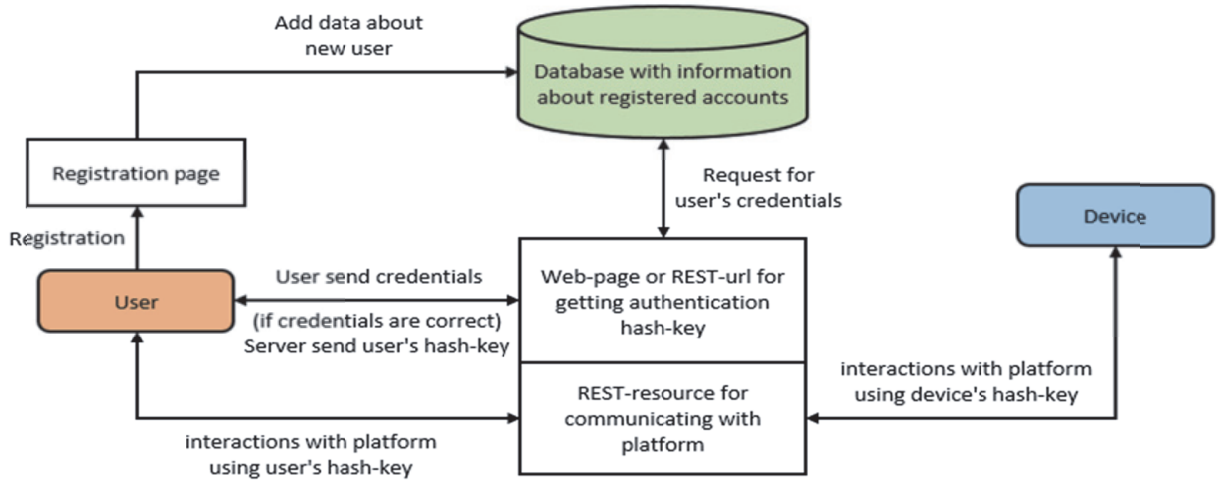

Рис. 8. Схема авторизации и взаимодействий с платформой

Docker помогает быстрее распространять код, тестировать, развертывать приложения и сокращать время между написанием кода и его исполнением. Docker делает это с помощью платформы виртуализации контейнеров, используя процессы и утилиты, которые помогают управлять и развертывать приложения.

Интеграция позволит легко масштабироваться до требуемого количества серверов или платформ, поскольку балансировка контейнеров действует с помощью Kubernetes.

Kubernetes - проект с открытым исходным кодом, предназначенный для управления кластером Linux-контейнеров в виде единой системы. Kubernetes управляет и запускает контейнеры Docker на большом количестве хостов и обеспечивает совместное размещение и тиражирование большого количества контейнеров. Проект был запущен Google и теперь поддерживается многими компаниями, включая Microsoft, RedHat, IBM и Docker.

Схема взаимодействия продуктов Docker и Kubernetes с платформой IOPT показана на рис. 9.

Также планируется анализ данных, поступающих с датчиков, для выявления аномального поведения. Для этого в базе данных будет представлена новая таблица, в которую будут сохраняться храниться данные от датчиков для дальнейшего анализа. А для проверки отклонения будут применяться специально обученные для этой нейронной сети в отдельном модуле.

Система мониторинга и анализа будет модулем, разработанным в том же стиле, что и вся платформа. Это будет веб-сервис RESTful со своим API. Для хранения данных могут использоваться базы данных временных рядов (например, InfluxDB) и для визуализации в паре с Influx можно использовать Grafana, открытую платформу для аналитики и мониторинга.

Система мониторинга и анализа будет модулем, разработанным в том же стиле, что и вся платформа. Это будет RESTful веб-сервис со своим API. Для хранения данных могут использоваться базы данных временных рядов (например, InfluxDB) и для визуализации в паре с Influx можно использовать Grafana, открытую платформу для аналитики и мониторинга.

ЗАКЛЮЧЕНИЕ

Разработан прототип платформы для управления инфраструктурой интернета вещей. Предложена и реализована архитектура платформы. Проведены начальные этапы тестирования, которые показали работоспособность системы. Обозначены цели и планы по развитию платформы.

СПИСОК ЛИТЕРАТУРЫ

1. *A.V. Leonov*, “Internet of things as the basis for smart applications,” Young Scientist – Kazan: Publishing house Young Scientist, 2015. – vol. 2. – pp.141-142.
2. *V.L. Chugreev*, “Development of service-oriented architecture in ISEDT RAS,” Young Scientist – Chita: Publishing house Young Scientist, 2013. – vol. 1. – pp.23-25.
3. *F.M. Kurilov*, “Visualization tools for structured data in client web applications,” Technical science in Russia and abroad – Moscow: Buki-Vedi, 2014. – p.17.
4. Internet of Things [Electronic resource] // MIT Technology Review, 2017. URL: https://www.technologyreview.com/s/601013/the-internet-

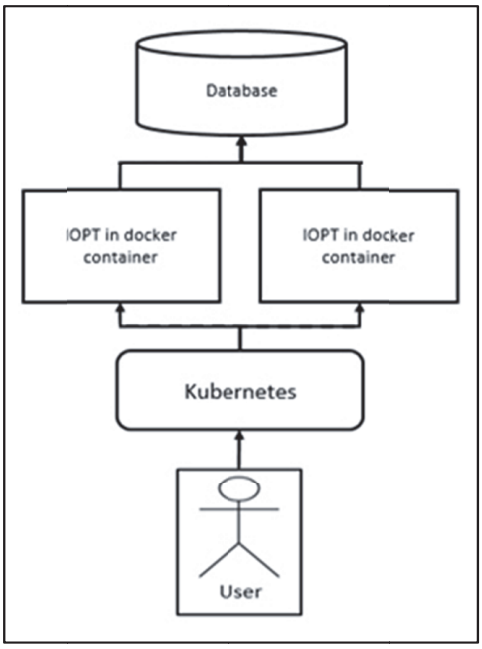

Рис. 9. Схема работы платформы с использованием Docker и Kubernetes

of-things-roadmap-to-a-connected-world/ (14.09.2017)
5. *Y.A. Vorontsov*, "Standarts of web-services," Age of Quality – Moscow: Sientific journal "Age of Quality", 2015. p.56.
6. *Y.A. Vorontsov, A.V. Kozinets* "Standarts of web-services," Age of Quality – Moscow: Sientific journal "Age of Quality", 2015. – vol. 3. – p.56.
7. *R. Hairetdinov,* "Quality of code in business applications: problems and solutions," Age of Quality – Moscow: Sientific journal "Age of Quality", 2015. – vol. 2. – p.49-50.

DEVELOPMENT PLATFORM FOR CONTROLLING THE INFRASTRUCTURE OF THE INTERNET OF THINGS

© 2017 A.A. Yakimenko[1,2], A.I. Belov[1], P.S. Goncharuk[1], I.M. Stubarev[1]

[1] Novosibirsk State Technical University
[2] Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Novosibirsk

The paper presents a technology for remote monitoring and control of devices connected to the Internet. The architecture of the platform is proposed. The platform allows automating the process of monitoring and management. A prototype of this platform with an example of functioning was developed.
Keywords: Internet of Things, platform, web service, Java, REST, API.

The work was supported by the RFBR grant 16-37-00240 Mol_a.

Alexandr Yakimenko, Candidate of Technics, Associate Professor at the Computing Machinery Department.
E-mail: yakimenko@corp.nstu.ru
Anton Belov, Master Student.
E-mail: antonbelov1509@gmail.com
Pavel Goncharuk, Master Student.
E-mail: goncharuk.p.s@gmail.com
Igor Stubarev, Master Student.
E-mail: igor.stubarev@yandex.ru