Development Platform for Controlling the Infrastructure of the Internet of Things

Alexander A. Yakimenko[1,2], Anton I. Belov[1], Pavel S. Goncharuk[1], Igor M. Stubarev[1]
[1]Novosibirsk State Technical University, Novosibirsk, Russia
[2]Institute of Computational Mathematics and Mathematical Geophysics SB RA, Novosibirsk, Russia

***Abstract* — The work presents a technology for remote monitoring and control of devices connected to the Internet. The architecture of the platform is proposed, which allows automating the process of monitoring and management. A prototype with an example of functioning was developed and demonstrated. This work was supported by RFBR grant №16-37-00240.**

***Index Terms* — Internet of things, platform, service, essence, restful api.**

I. Introduction

NOWADAYS, TECHNOLOGIES that allow automating the environment surrounding human are gaining popularity. However, the problem of managing multiple devices and monitoring their behavior and state is not trivial. These needs are met by the Internet of things industry (IoT). The Internet of Things (IoT) is a technology and perspective direction of market development based on the combination of everyday items among themselves. IoT allows you to create an ecosystem of smart applications and services that will improve and simplify the lives of individuals and societies in general [1].

II. Problem Definition

The IoT's underlying challenge is that there are no clear and agreed-upon architectures for building connected systems. Your light switch may have one level of data-security encryption, while your TV remote control has another. Wireless protocols may differ, too: One device might use ZigBee while others rely on Bluetooth or Wi-Fi. Bridges to connect across all these options will proliferate. And even if independent systems are secure, we will have to cobble them together—and the resulting chain will only be as strong as the weakest link [4]

The market has plenty of solutions for aggregation of IoT things. For example, there are projects based on PaaS technology. They have a large number of functionality, but at the same time they occupy the high-price segment on the market. Therefore, to compare our service with this kind of solution is not correct. We can only think that open source projects are trusted instruments, which could be analyzed by everybody interested in. This gives advantages in the form of rapid and multilateral development from the side of community.

Therefore, the purpose of this work is to develop a system that will manage devices and monitor their status through the Internet. It should be a flexible and, at the same time, easily customizable platform. To achieve this goals, a prototype platform was developed, this is a web server written in the Java language. It was called IOPT (Internet of Pretty Things). It is planned that the platform will be an open source project.

It will have a modular structure that will allow you to combine it with various third-party projects. This will provide the wide expansion and rapid development of this platform. Particular attention is paid to the scalability and speed of data exchange. Modularity will provide the flexibility of the project, and open source code will allow communities to develop user-friendly interfaces for various modules.

During the development of the platform, definitions "model" and "object" were introduced. The model in this context means the copy of the "Smart Organization". For example, an automated greenhouse in a platform is understood as a model of the "Smart Greenhouse". The object in turn is an isolated observable (and/or controlled) unit of the model. For example, a section or room in a greenhouse. Each object has a configurable set of properties that can be monitored and modified. In turn, a script can be attached to each property, which will be executed when certain events occur. Fig. 1 illustrates relational structure of the connections described in this paragraph.

III. Used Technologies

A. Technologies of the Platform's Core

The implementation of the platform is a RESTful web-service (REST, REpresentational State Transfer - architectural style of interaction between components of a distributed application on the network) [2].

Today, the IoT is an abstract collection of uses and products using a huge variety of protocols. It's imperative that we establish paradigms for effective implementation and use [4].

Web-services allow you to make the program functions available via the Internet, i.e., having developed some universal functionality, we can provide it to applications on other computers (and other computing devices such as tablets, smartphones, etc.). The main thing is to connect to the Internet. Consequently, web-service technology makes it

978-1-5386-7054-5/18/$31.00 ©2018 IEEE

possible to develop cross-platform solutions, and use different programming languages to write the service and its consumer (client) [5]. It is done so to avoid necessity of implementing compatibility of different messaging protocols between devices and server.

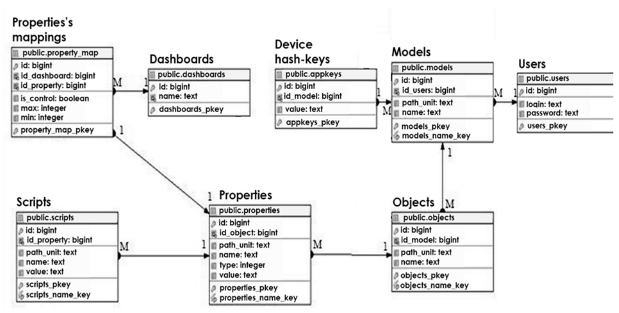


Fig. 1. Relational data model.A simplified scheme of the implementation of the REST architecture is depicted in Fig. 2

In general, REST is a very simple information management interface without the use of any additional internal layers. A global identifier, such as a URL, uniquely identifies each piece of information. Each URL in turn has a strictly defined format. The absence of additional internal layers means the transfer of data in its original form. I.e. data is not wrapped in XML, as SOAP and XML-RPC do, we do not use AMF, as Flash does, etc. Sending original data [2].

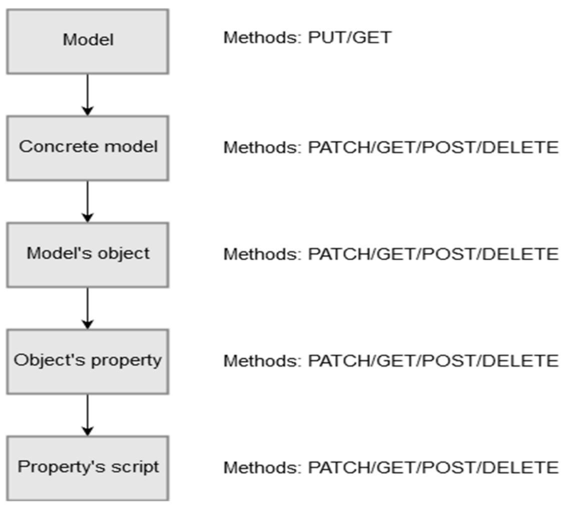

Model
Methods: PUT/GET
Concrete model
Methods: PATCH/GET/POST/DELETE
Model's object
Methods: PATCH/GET/POST/DELETE
Object's property
Methods: PATCH/GET/POST/DELETE
Property's script
Methods: PATCH/GET/POST/DELETE

Fig. 2. Simplified scheme of the REST implementation.

As stated above - the URL uniquely identifies each unit of information - this means that the URL is essentially the primary key for the data unit. E.g., the third book from the bookshelf will look like /book/3, and the 35 page in this book is /book/3/page/35. The result is a strictly defined format. And it does not really matter what format the data is at /book/3/page/35 - it can be HTML, a scanned copy as a jpeg-file, and a Microsoft Word document.

A way of management organizing of the service's information fully depends on the data transfer protocol. The most common protocol is of course HTTP. So, for HTTP, the action on data is specified by methods: GET (get), PUT (add, replace), POST (add, change, delete), DELETE (delete). Thus, the actions CRUD (Create-Read-Update-Delete) can be performed as with all 4 methods, and only with the help of GET and POST.

REST is considered an alternative to SOAP. REST is not strictly defined standard; it is like a style of constructing and programming. SOAP, on the contrary, is a documented standard. The advantage of SOAP is the ability to use the service without documentation, documentation in the case of SOAP is automatically generated. The advantage of REST is a cost-effective use of bandwidth..

The RESTful service itself is developed with the help of Java EE technologies: Jersey, Glassfish, HTTP Servlets, etc. To this service, you can perform HTTP requests from any application that has access to the Internet using the REST-API. There is no built-in GUI. It is possible to develop your own graphical interface. It is assumed that there will be open source implementations. This ensures that there is no binding to a specific graphical interface.

As already mentioned, Java was used for development. It was chosen as one of the most widely developed and rapidly improving tools, system applications, web services and other network (and not only) information systems.

Fig. 3 demonstrates structural scheme, expressed by the components of Java EE. Client Machine illustrates an endpoint client (or middleware broker) using http protocol to for communication with server. The server has a public part represented by a set of web-interfaces. Jersey library is a main part of web interfaces. These web interfaces are API's endpoints. Client perform HTTP request, add specific attributes to the request and send it to the appropriate endpoint (URL). Inside the server placed business logic modules. They developed under such technologies like EJB, CDI, JSP, etc.

B. Other Used Technologies

The data is transferred in the form of JSON-structures. JavaScript Objective Notation (JSON). JSON is a young format based on a subset of the JavaScript language. From XML, it differs in the simplicity of data description, both for humans and for the parser program, the lack of syntax redundancy inherent in markup, and also the direct orientation to data exchange.

JSON is one of the most common formats that offer different web-APIs. The data in JSON is one of two structures: a set of key pairs: a value is an object, or an ordered set of values is an array. The key can be only a string, the value is any type, including a nested structure [3].

The JSON format allows you to reduce the amount of data transferred due to the lack of framing tags and service information. All the service information is posted to the documentation, and not sent every time along with the data. To build a graphical interface, it is enough to parse the model's data and build a visual representation. Fig. 4 shows an example of a structure reduced to a compact form used in the platform.

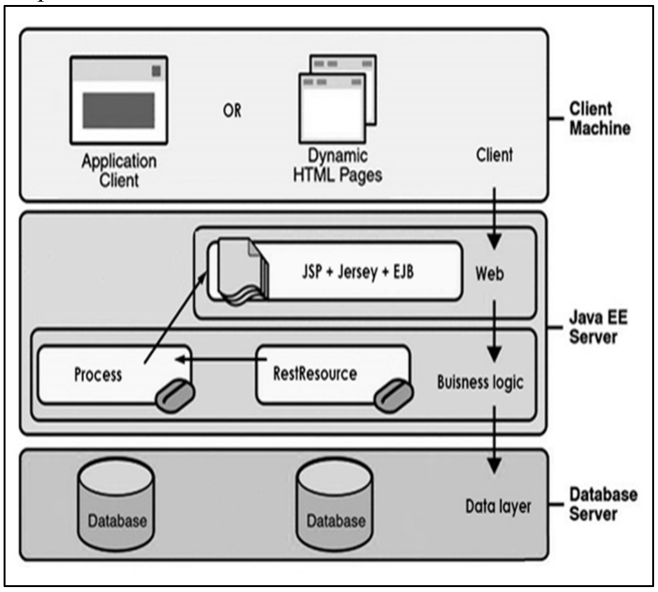
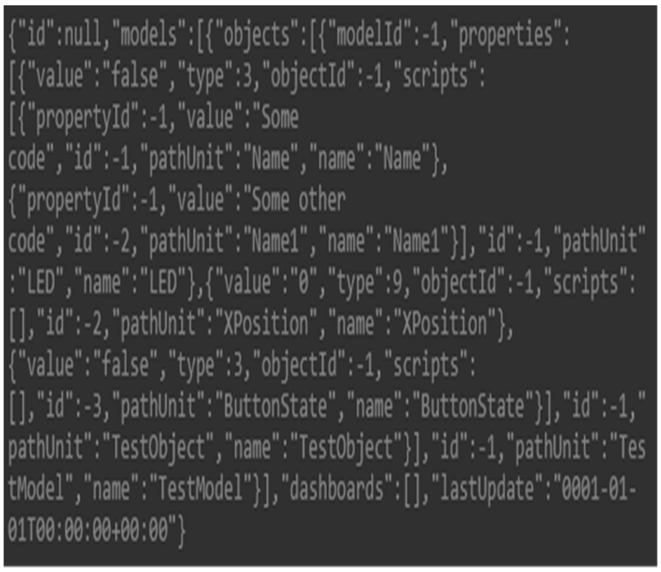


Fig. 4. JSON-structure of the platform's model.



```
{"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"}
```

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

Creating a basic application for working with IOPT is a simple process. First, create classes that describe the model, object, property, script. Then create a data access layer that will receive the current root model at certain intervals. This model already includes all submodels with their objects/properties/scripts. It is also necessary to map add/delete/change/etc in local model to PUT/DELETE/POST/etc requests. Finally, create a graphical interface that work with this local model.

Diagram of classes illustrated at Fig. 5. Classes called "Script", "Property", "Object" and "Model" represents entities from real world and their properties. "Snapshot" is a class, which contains the state of all "Pretty Organization". Classes with postfix "Resource" are the connectors between endpoints and business logic. Classes with postfix "Proc" implements business logic. "JDBCConnection" is a class adapter between JDBC resource and server.

Resource classes are the implementation of Jersey application resources. They are annotated with @Path("/url_to_resource") annotation which make mapping between Java class and http document. Processors classes (classes with Proc postfix) are written in style of Statless Local Bean. It provides new instance for each request, all requests executes in their own thread. Processors connected with Resources via CDI mechanism.

The platform provides the use of configurable data stores, it can be standard relational databases for your choice, such as PostgreSQL DBMS, or it can be distributed data warehouses. This will help to avoid the restriction of particular database architecture, thereby expanding the range of consumers.

Depending on the way the platform is used, a particular data store may be a bottleneck and will not scale. For example, a platform in a cluster that operates with a large amount of data. Thus, this implementation feature does not limit the user in choosing a DBMS. This was achieved by using the JDBC driver.

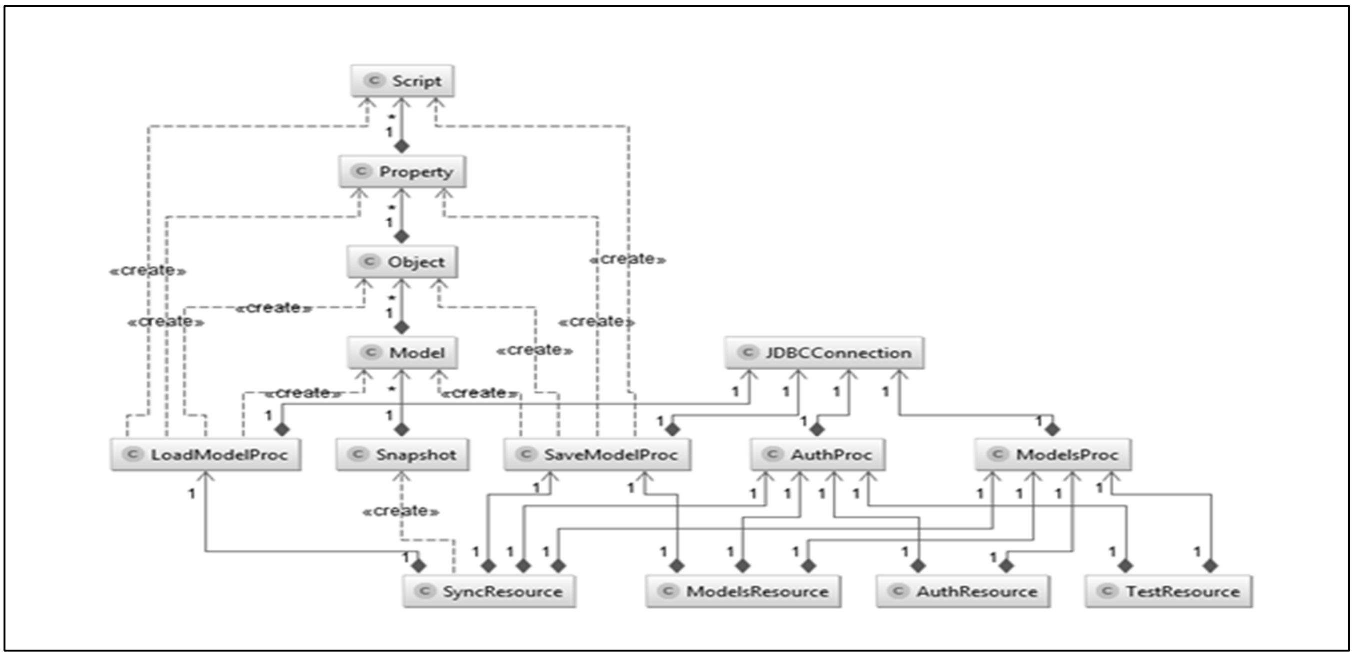


Fig. 5. Relational data model.

For convenient interaction with the platform, you need to implement complex scripts or develop your own application in any programming language that supports networking. At the same time, for the purpose of demonstration, an application with a graphical interface was written. This application refers to this system as a third-party module. Its appearance is shown in Fig. 6.

Fig. 7 shows the architecture of the platform. The basis of the platform is a module called IOPT-Server. This module is the central part of the platform. It handles all requests - from devices, from user applications and from third-party modules. It also interacts with the configured data store. This module will include a data warehouse; it can be run on any server with a pre-installed Java distribution. It provides coverage to a wide audience and transparency of the platform (due to open source and the ability to deploy on isolated servers).

It is planned to cluster and scale the platform, which will make it possible to use the platform both in small enterprises with a small number of "smart" objects, and on large corporations that serve a large number of "smart" organizations that generate impressive traffic of messages and data.

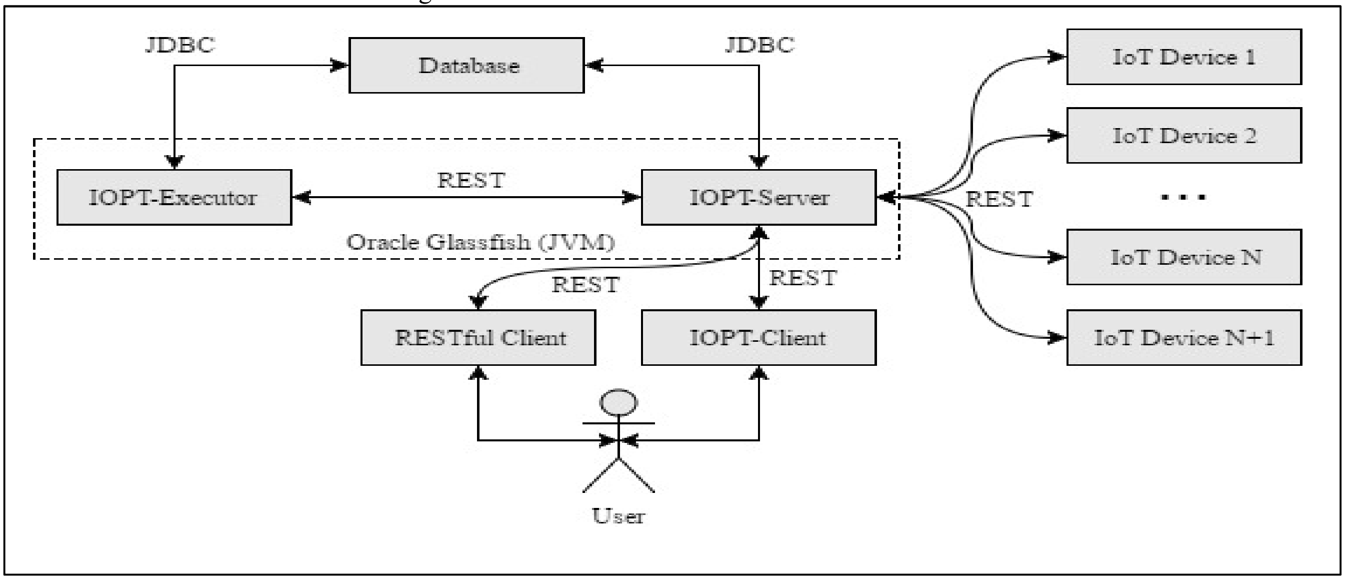


Fig. 6. Scheme of the platform's architecture.

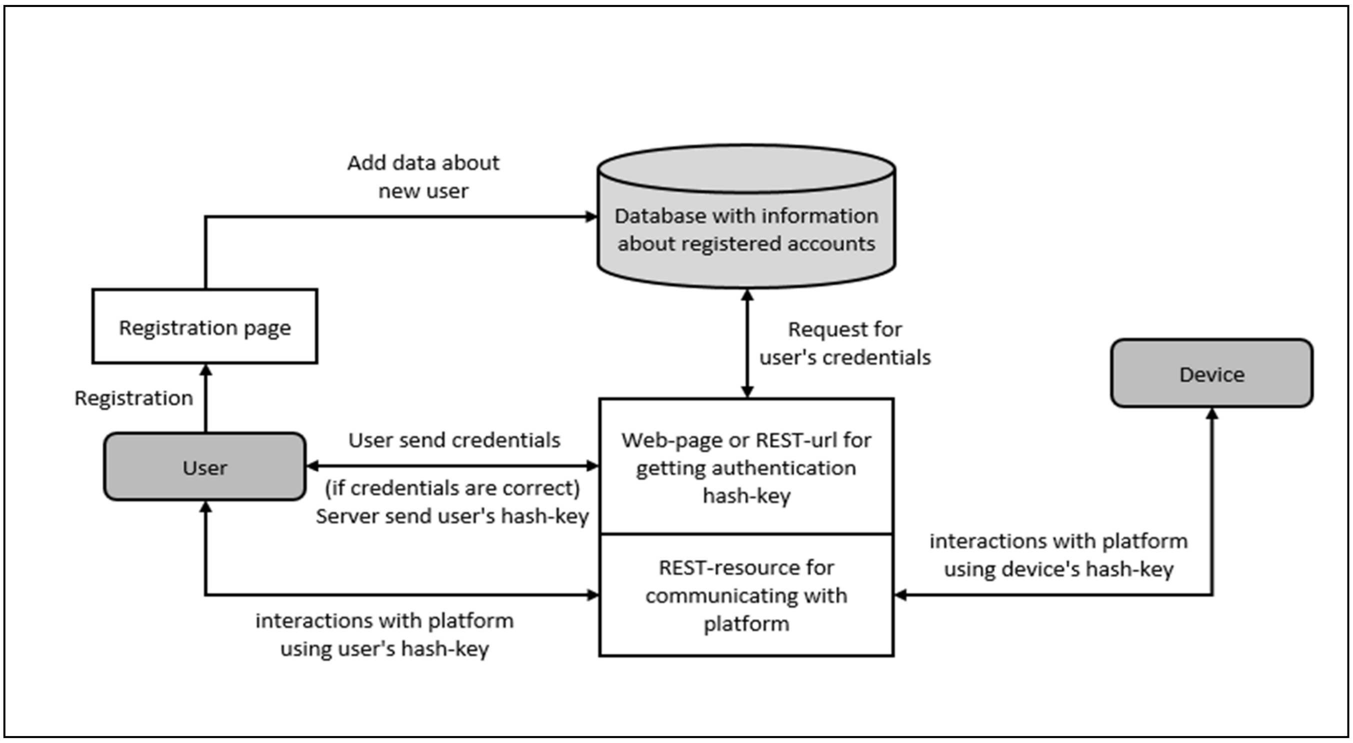

Add data about new user
Database with information about registered accounts
Registration page
Request for user's credentials
Device
Registration
User
User send credentials
Web-page or REST-url for getting authentication hash-key
(if credentials are correct) Server send user's hash-key
interactions with platform using device's hash-key
REST-resource for communicating with platform
interactions with platform using user's hash-key

Fig. 7. Scheme of the authentication and further interactions with the platform.

In general, the query structure is represented as links to resources. In particular, for this project, the reference is a nested entity (object).

For example, http://url/models/farm/garden1/temp_prop, where farm is the farm model, garden1 is the greenhouse with the number 1, and temp_prop is the air temperature. The action is indicated via the type of the HTTP header (e.g., get the HTTP temperature data of the GET header and the link, which is presented above).

This provides a human-like form of query, which reduces the number of errors and makes it easier to understand the operating principle of the software interface. The only inconvenience, for example, in comparison with SOAP, is the need to develop documentation for the REST API.

IV. Features of the Platform

Due to the fact that interaction with the platform occurs only through the program interface RESTful API, excluded direct interaction with databases, so an erroneous query will not damage its structure. Protection against XSS attacks and data theft is configured similarly to the protection of any website, for example, using TLS/SSL and HTTPS certificates [7].

To automate the behavior of devices, the platform provides the ability to program logic by developing custom scripts, which have already been mentioned. Now only the JavaScript programming language is supported, it is planned to implement support for Python and other popular scripting languages. Now, this functionality is implemented as an internal module (IOPT-Executor in Fig. 7).

The IOPT-executor is written with Java Scripting API, which first came with Java 6. Nowadays new engine called Nashorn is implemented in Java 8, an implementation of the ECMAScript Edition 5.1 Language Specification. It comes bundled with Java SE 8. It can be used as a scripting tool along with Java to create polyglot applications.It gives wide variety of actions. It gives the access to Java from scripts and other way round. IOPT-Server listen for property change event and when it happens, IOPT-Server sends to IOPT-Executor command to execute all script subscribed for current property’s change event.

One of the sensitive problems of the Internet of things is the weak security of the endpoint devices and the whole architecture from hacking. To solve this problem, a system for authentication and authorization of users and devices was developed. This system allows you to use a third-party authentication module or develop your own if the required functionality is missing [7].

Fig. 8 shows a diagram of interaction of users and devices with the platform. To date, only basic authentication by login and password has been developed. The implementation of other popular authentication methods (e.g., Kerberos, LDAP and Active Directory) is planned.

User cannot modify or view the models of other users. At the first access, the user provides a login and password to the server. In case of success, the server sends a hash key to the user, which will later allow you to access the server without using a username and password, the similar algorithm implemented on web sites. The only difference is that the user should send the hash key manually, not automatically, as the web-browser does. Using the REST API, the user can generate a special hash key, which is required for the devices to communicate with the server. Each device is associated with only one model, so a separate hash key is generated for each model.

V. Conclusions

At the moment, the system architecture is being improved and upgraded, and the platform is tested. It is planned to improve the composition of technologies and components used in developing platform, which will reduce memory usage and simplify the procedure for deploying the application on servers, and provide scalability that is more flexible. This is planned using containerization technology, which is implemented in a different product - Docker.

“Heavy and bulky” application server Glassfish is planned to be replaced by Jetty in conjunction with Weld. This will make it easier to launch the platform and reduce memory consumption.

The Docker is an open source platform for the development, delivery and running applications. Docker is designed to quickly deploy your applications. Using the Docker, you can separate your application from your infrastructure and handle the infrastructure as a managed application.

Docker is lightweight and fast. It provides a stable, cost-effective alternative to virtual machines based on the hypervisor. It is especially useful in high load environments, for example, when creating your own cloud or platform-as-service. However, it is also useful for small and medium-sized applications when you want to get more out of available resources.

Docker helps spread code faster, faster testing, faster deploy applications, and reduce the time between writing code and running code. Docker does this with a lightweight container virtualization platform, using processes and utilities that help manage and deploy your applications.

Integration with the docker will allow you to easily scale to an unlimited number of servers or platforms. As the balancer acts Kubernetes.

Kubernetes is an open source project designed to manage a cluster of Linux containers as a single system. Kubernetes manages and runs Docker containers on a large number of hosts, and provides the co-location and replication of a large number of containers. The project was launched by Google and is now supported by many companies, including Microsoft, RedHat, IBM and Docker.

Scheme of interaction between Docker and Kubernetes products with a platform IOPT illustrates Fig. 8.

Also there are plans to analyze data coming from sensors, for anomalous behavior. To do this in the database will introduce a new table which will store the logs data from sensors. And to check the deviation will apply specially trained for this neural network in a separate module.

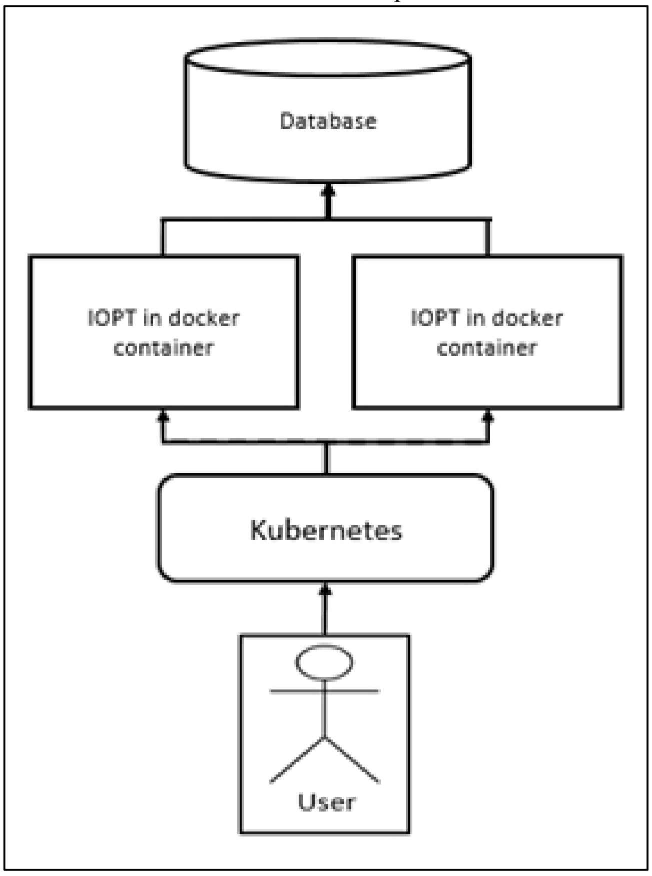


Fig. 8. Scheme of the planning integration with Docker and Kubernetes.

Monitoring and analyzing system would be a module developed in the same style as whole platform. It would be the RESTful web service with its own API. For storing data could be used time series databases (for example, InfluxDB) and for visualization in the pair with Influx could be used Grafana. Grafana is an open platform for analytics and monitoring.

References

[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-of-things-roadmap-to-a-connected-world/

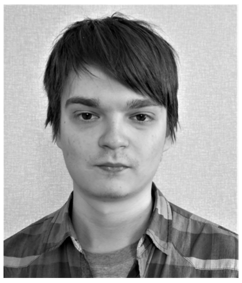
[5] Y.A. Vorontsov, “Standarts of web-services,” Age of Quality – Moscow: Sientific journal “Age of Quality”, 2015. p.56.

[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

Yakimenko Alexander, Ph.D., Associate Professor, Department of Computer Science of NSTU, researcher at the Institute of Computational Mathematics and Mathematical Geophysics SB RAS. Research interests - information technology, computer systems, computer simulation, parallel computing. Author of more than 30 scientific papers.

Goncharuk Pavel - master of computer science at the NSTU.
The developer for the company "VEON (Vimpelcom Ltd)".
Research interests: development and implementation of distibuted systems and web-applications, BI-systems.
Author 4 scientific and educational works.



Belov Anton - master of computer science at the NSTU.
The developer of Infor-CRM for the company "FB Consult".
Research interests: Data Mining, development and implementation of CRM-systems, BI-systems.

Stubarev Igor - master of computer science at the NSTU. The developer of Infor-CRM for the company "FB Consult". Certified business analyst and data architect. Research interests: Data Mining, development and implementation of CRM-systems, BI-systems.