Developing of energy resources accounting and controlling system for the Internet of Things

Sergey Popov, Evgeny Chernyy
NRU ITMO
Saint-Petersburg, Russia
sp, ech@winghouse.ru

***Abstract*—Currently there are actively developing services implementing Internet of Things [1]. Such services allow to delegate controlling and monitoring tasks over heterogeneous devices to the cloud. Because of interconnectivity, connecting devices to single cloud opens up new possibilities in controlling complex systems based on real-time data analysis, predicting accidents, etc. In this paper we will describe our experience on developing and introduction to the manufacturing facility of accounting and controlling of energy resources system.**

***Keywords*—Internet of Things, M2M, Energy savings, Utility meter data collecting, Smart Grid.**

I. Introduction

Accounting of energy resources is a vexed problem for enterprises in Russia. For instance, according to latest government regulations [2], all companies having peak electricity consumption above 670 kW are mandated to switch to an hourly rate of pay. The problem is that electricity meters available on the market do not provide per-hour measurements. So, such companies have to use third-party solutions for automatic accounting and control of energy resources or to develop their own. In our company we are working on such a solution for enterprises.

In this paper we discuss the system that we developed for "LenPoligraphMash" [3] facilities. It is based on our research in connecting and controlling of devices over the Internet. We have developed hardware and software that can poll data from many kinds of utility meters, transmit it to the server for further processing and receive control commands from it.

II. IoT in the energy sector

According to authors [4] "IoT becomes the natural enabling architecture for the deployment of independent federated services and applications, characterized by a high degree of autonomous data capture, event transfer, network connectivity and interoperability". Implementing such services on big manufacturing facilities and city environments will allow to move from passive observation of to active participation in production processes. Also IoT enterprise services based on open standards will allow to apply "horizontal" approach in M2M development [5], where key components of M2M application/device development (e.g. hardware or software interfaces) are standardized and hardware or software products by different vendors can be interchangeable. Some IoT technologies like OWL [6] and RDF [7] are the good candidates to be a unified software interfaces for M2M systems.

First step to deploy such service at facility is to implement energy flow monitoring system. It will show ineffective parts of energy system and allow to improve them by eliminating flaws produced by the human factor (e.g. incorrect electricity meter connection).

Second step is introducing of controlling functions to the existing monitoring system. It will allow to implement SmartGrid analog for local facility to prevent energy consumption overheads. For example, central server could disable least important electricity consumers or disable non-critical lights in the offices when overall day electricity consumption is getting to the upper limit.

Next, after several facilities implement monitoring and controlling functions, they will be able to communicate with electricity supply companies. On this step facilities can send streams of energy consumption data in real-time to these companies for more accurate energy consumption planning. Also electricity suppliers, when some failures occurs that leads to lowering levels of generated energy, can send alarm messages to all users with recommendations to temporary lower energy consumption and avoid failure of entire energy network.

III. High level architecture

Our system implements (Fig. 1) common scheme of plugging devices to the server, also used by some SmartGrid [8] and M2M [9], [10] systems. It supposes that standard equipment (utility meters, home appliances, etc.) is connected to the server through a hardware agent that can poll data from and control the equipment.

Such scheme, where independent hardware agent is connected to equipment, allows to quickly react to abnormal situations by sending alarming messages to the server without waiting for the next server request.

Today for resource consumption accounting often used another scheme, where utility meters, equipped with GPRS modules, are remotely requested by the server to get per-day or per-month data. Such scheme is easier to deploy on facilities, because devices already have tools to work with them. For example, some electricity meter manufacturers provide special software for viewing state of their devices, equipped with GPRS. The main disadvantage of this scheme is impossibility of obtaining important information without creating request

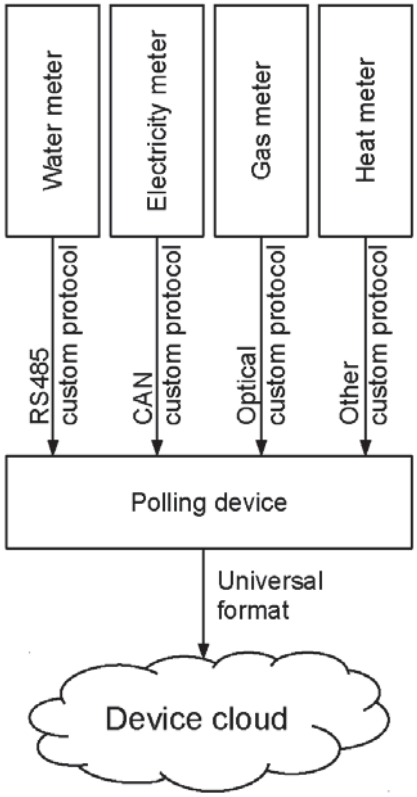

Fig. 1. Our implementation of common device connecting scheme

from the server. In our implementation hardware agent can autonomously define conditions where server should be notified about the situation in current facility and send only important data.

IV. Collecting data from the devices

A. Overview

Our system for automatic metering and control of energy resources (Fig. 2) consists of the following components: electric meters, polling devices (Fig. 3), the server and its generated DBF files. We use RS485 or CAN for communicating between meters and the polling device and Ethernet for sending collected data to the server. One detail that confused us during development of a polling device is that some electricity meters has CAN transmitter, but in fact does not support CAN 2.0B [11] and all its features of like collision resolution and detection, checksums, addressing, etc. Following the standard, a CAN transmitter should send echo to the controller for collision discovery or a confirmation bit, so that the receiver returns that echo from transmitter back to controller.

We also had to make some modifications to regular plugging scheme (Fig.4) of CAN transmitter to use it on the simple RS485 bus.

B. Polling data

Polling devices constantly poll data from meters on the bus one by one, using counter's custom protocol. Because polling device is built on a simple 8-bit controller with 2 kB of RAM, polled data is converted to JSON as it comes to the device and is uploaded to the server over TCP.

We used JSON for sending data from polling devices to the server, because it is not memory expensive to work with on 8-bit controllers, human-readable and flexible enough to transmit data from many meters in one packet. Before JSON, we used a key-value format but it is not usable when dealing with more than one meter on the bus. We do not use XML because it is costly to process on 8-bit controllers (need open and close tags, etc).

To ensure stability of the system and provide debug information we collect additional data from the meters. Among the most important indicators is the uptime, which is represented by a similarly named field that ends each JSON packet. We also add the meter's time readings to the packet, because polling device doesn't have a power-independent clock. Manufacturing date of the meter indicates the time of its next maintenance check. Serial number serves as a unique key on the bus and identifies physical location. In addition, in the packet there is debug information, for example, a flag indicating whether or not connection to meter was completed successfully. If some error occurs while reading data from the meter, or a checksum is incorrect, polling device sends low-level debug information to the server with all bus metadata for logging.

Since electricity bills are issued on a monthly basis, it is important to summarize meter readings by the end of the month. Counters we work with store all data in its internal memory over the year, but it is not efficient to transmit all the data on each request. Instead, when polling device connects to the server for the first time, in its first JSON packets all data from meters is transmitted to the server, so that any server will have data from all meters before receiving current metrics. This feature is very valuable during initial installation of our system when we need to preload the database with data on earlier periods or when our system had been offline.

C. Extending JSON packet format

Facing the need to scale our system, it became clear that our JSON format was not detailed enough despite its already large packet size. We decided to make each packet a self-sufficient unit, so that by reading it, the server could determine all the information needed for further processing. Each packet should contain the ID number of polling device, meter model and customer data (e.g. name of organization). This data may be very useful in debugging, so that we can localize the problem just by analyzing logs.

Another reason for using the extended packet format is that each polling device could work with heterogeneous devices, e.g. electric and water meters produced by different vendors. This unifies the way to process data on the server and delegates all responsibility for using proprietary device protocols to polling devices.

At present time, this extension is under development.

D. Examples of JSON packets

The following two messages are examples of JSON packets that server receives from polling devices. The first

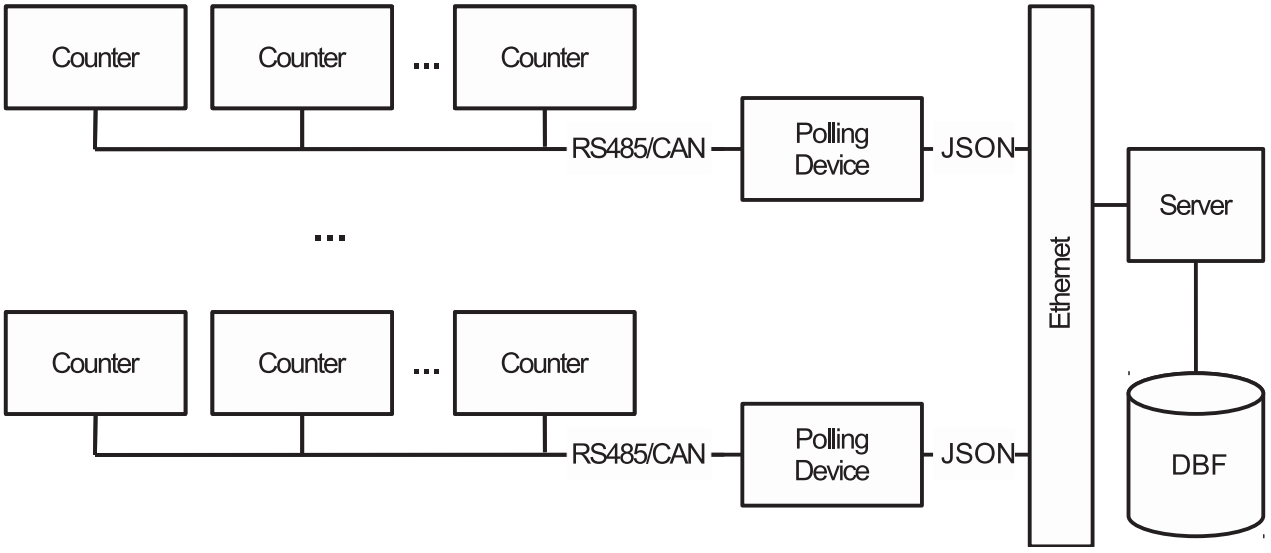

Fig. 2. Schematic representation of system for automatic counting and controlling of energy resources

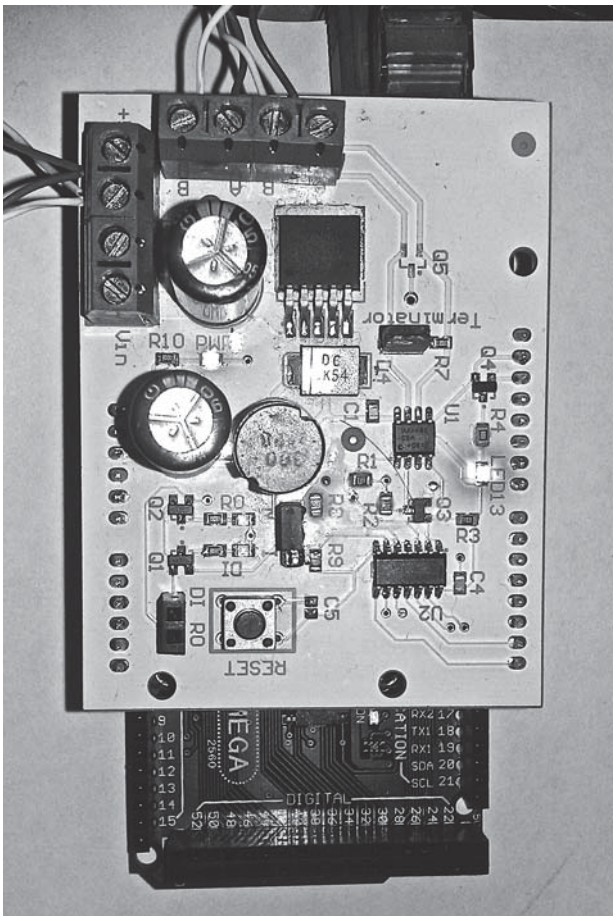

Fig. 3. Arduino-based polling device. RS485 shield developed by our team

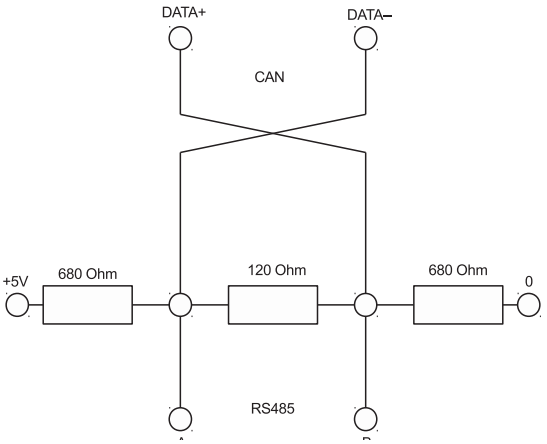

Fig. 4. Example of connection CAN transmitter to RS485 bus

message exemplifies a regular message and the second is a debug message sent by polling device when it fails to find a meter.


```
{
``CounterAddress__`` : 95,
"LoginAction__" : "Logged in",
"SerialNumber__" : 13535895,
"ManufactureDate__" : "5-2-13",
"Timestamp__Date" : "14-8-13",
"Timestamp__Hour" : 18,
"Timestamp__Min" : 25,
"Timestamp__Sec" : 28,
"Now_Power_P_SUM_____value" : 28.82,
"Now_Power_Q_SUM_____value" : -26.2,
```

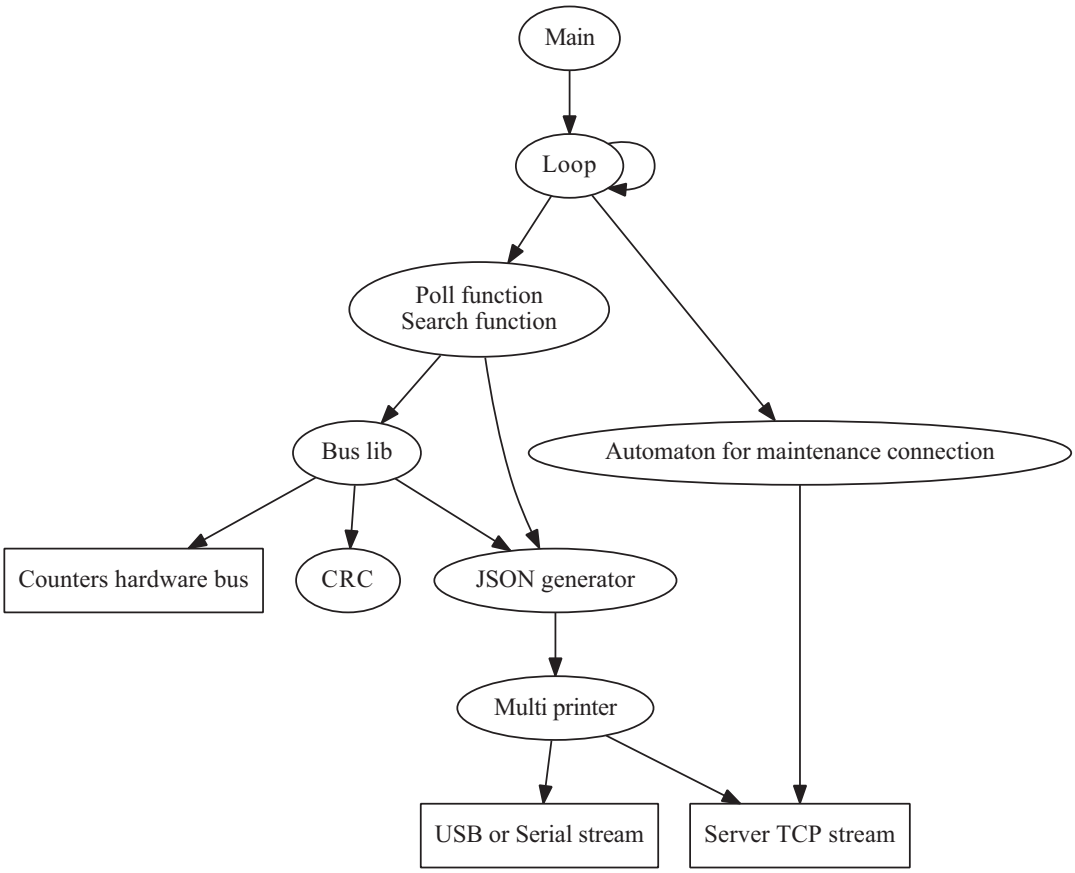

Fig. 5. High level architecture of polling device firmware


```
"Now_Power_S_SUM_____value" : 38.96,
"Now_Voltage_Phase_1_value" : 11.15,
"Now_Voltage_Phase_2_value" : 0.0,
"Now_Voltage_Phase_3_value" : 239.13,
"Now_PowerFactor_SUM_____value" : 0.739,
"Now_Frequency_value" : 50.01,
"Uptime" : "30039"
}
{
"type" : "debug",
"Address" : "0x5Eh",
"CountersSearch_SearchRequest_Error" : "No
answer from - 94",
"CountersSearch_SearchRequest_Buffer" :
"5E 0 39 D0 ",
"Uptime" : "27998"
}
```


V. Polling device firmware programming

Main challenges in programming polling devices were providing fault tolerance and correctness of collected data combined with ease of use and installation for the non-programmer service staff.

Critical parts of code, such as network connection manager, use automata-based programming principles. Robust implementation of states lowers the risk of programming error, increases stability of the program and makes debugging easier.

A. High level architecture of firmware

Polling device firmware consists of two main modules (Fig. 5), “Main” and “Loop”. The “Main” module is responsible for bootstrapping the device, including initialization of all outgoing connections such as serial port and Ethernet (by DHCP). Also this module initiates the search of meters, already discovered in previous sessions.

“Loop” module works in an infinite loop – this is where the state machine mentioned in previous section operates. Its main function is handling of TCP connections, polling and discovery of the meters.

Meters are polled on several levels. At top level of data processing, the device generates a request command, sends it and passes the response to JSON generator. The middle level is responsible for generation of polling packets. It composes a request packet, adds meter address and computes checksum. After that, the message is transmitted to the transport layer, and sent to target meter over the bus. After receiving a response from the meter, the device compares actual length of the received packet with the expected one. Since response length is known at the moment of sending the request, we may optimize network polling process and avoid 200 ms timeout while waiting for all network buffers to become full. Polling

device checks address of the responding meter and checksum of the packet. If validation errors occur, this fact will be logged by sending data to JSON handler and server.

JSON generator has three main functions: start new packet, transmit a value and finish packet. Every time a polling device starts polling of a meter, JSON generator starts new packet, and after receiving last chunk of the data from meter, JSON packet is finished. Only one meter can be polled at a time, embedded JSON packets are not allowed. When packet is closed, no other transmissions are allowed. Thanks to such rules, if some debugging information is generated while reading data from the meter, it is guaranteed that it will be included in the packet that belongs to the polled meter. JSON generator writes packets to a universal printer, which may be configured for outputting data over TCP or any serial port of the controller, so that we can debug the device even when network is not available.data though TCP or any serial port of the controller, so we can debug device even when network is not available.

B. Counter discovery

To provide easy installation of new devices to already working system, simple algorithm of searching available counters on RS485 bus was implemented. For this purpose, the firmware reserves a permanent region of memory of 256 bits, one bit per each possible meter address.

Meter discovery is performed while polling data from already known meters. After each poll request device sends discovery requests to the bus, so that when new a meter appears on the bus, polling device would soon discover it and would start collecting data from it.

After powering on, the polling device first scans known addresses from last session and checks their availability on the bus – this facilitates a faster start compared to scanning all possible addresses every time. If polling device starts in a new location, previously known meters would not be found and would be deleted from the list. Installed meters would be found gradually.

If a meter stops responding, the polling device will continue to send it requests. This was designed intentionally, because there is no way for a device to know if a meter is disabled or removed by staff or it is out of service because of internal failure.

C. Automatic server connection

To implement a plug and play solution, we need to make polling devices capable of automatically connecting to the server. We considered several options for implementing this functionality:

- Using DHCP to determine the address of data collection server on current network, akin to the approach used in thin clients. Such a solution requires additional server configuration.
- Using domain name and local DNS server. There is a potential of packet delivery problems.
- Using broadcasting for server searching. Potential problems with discovery packets delivery.
- Augment each polling device with server and MAC addresses of configuration interface. Additional configuration required by service staff.

D. Counter addresses collision

Because counter bus address is encoded in one byte it is easy to find, but different counters may have the same address. Address collision related errors are not obvious. For the polling device, as first device on the bus, those errors appears as checksum errors. When address collision occurs polling device can access only that counter that physically located nearer than others with the same address. We also can't change counter address from the polling device, because changes will occur on all counters on the bus. In this situation we had to change counter address manually with help of manufacturer special equipment.

VI. DBF FILES MANAGEMENT

Current implementation of the system we developed for "LenPoligraphMash" collects all data from electricity meters to DBF files, which later processed by "1C Arenda" system [12]. The 1C system was chosen by the client as the most popular accounting system in Russia. DBF files are a convenient way for exporting data into the 1C system. Other capabilities, such as COM objects, were deemed unreliable due to low fault tolerance of the 1C system, which often leads to system freeze and data corruption. For large enterprises, the following indicators are most important:

- Hourly loads for power consumption forecasting.
- Power grid efficiency for monitoring electric transformer overloads.
- Meter readings for a period, for billing and monitoring of total consumption.

Our clients analyzing data we collected by themselves in 1C system (Fig. 6 and 7).

Energy network node	Average power coefficient	Electricity network efficiency
TP-1	0.7053	Average
TP-2	0.6933	Average
TP-3	0.4730	Poor
TP-4	0.6059	Poor
Overall	0.6031	Poor

Fig. 6. 1C report example, effectiveness of electricity consuming

VII. RESULTS

While working on the primary task of collecting electricity consumption data on "LefPoligrafMash" facilities we found that with our system we can detect a number of issues related to electric circuits in building. One of the main problems we found is loss of connection of one of the phases to the meter.

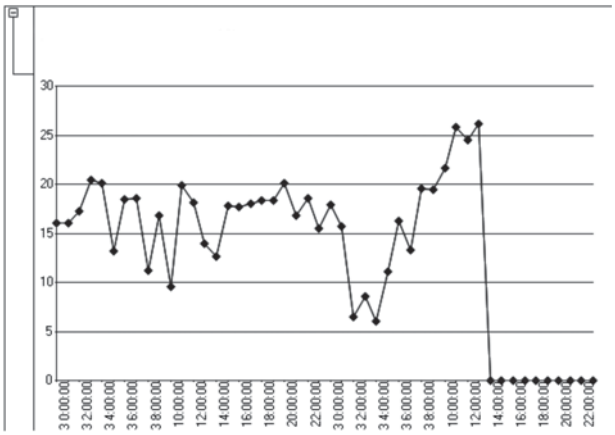

Fig. 7. 1C report example, hourly consumption maximums, kW * hr

Because meters are connected to electricity supply though a transformer, in case of absence even of a single phase connection to a meter, it will not collect data, but the consumer would still get electricity. In real situations service staff rarely pays attention to the indicator lamp signaling absence of phase connection. We may programmatically detect this by analyzing data in such situations (zero or low voltage on the meter and null angle between phases) and alarm staff about it. Such alarming extension is also under development.

Our electricity accounting system replaces 4 man-days of staff work – all data is collected from meters in only 30 seconds. Also, during development we discovered some issues in the energy grid on the facility, which lead to incorrect resource accounting. Our solution is several times cheaper than similar solutions provided by other companies in our country, because we did not resell proprietary products but develop hardware and software by ourselves.

VIII. Conclusion

In our work was developed the system for collecting, analyzing and presenting utility meters data, that integrates to our IoT service.

IX. Future work

We will continue to develop our system by improving JSON message protocol, implementing alarming system and adding water and heat metering to the accounting. Development of own counters and Web of Things infrastructure is our long term goal.

A. Ideas on improvement of accounting devices

During the development and deployment processes we came to some proposals for enhancement of existing utility meters. Most meters are passive – their main purpose is to count consumed resources. They can be polled, but polling methods are not well-documented and devices do not provide discovery and collision resolution capabilities. Also, existing meters do not notify about issues on the energy network – we need to wait for request packet from polling device. Obvious solution would be implementing CAN 2.0B on meters by manufacturers. With CAN we can create a network of intellectual agents on the facility, which can systematically monitor situation of energy flow on the facility and control energy overload and reactive power compensators. CAN uses the same bus as RS485, so introduction of this protocol will not complicate meter installation. Connecting such a network to the Internet could allow SmartGrid technology implementation on the facility.

References

[1] S. Popov, D. Mouromtsev, “Development of a Distributed Semantic Platform for Internet of Things and Internet of Devices”, *in Proc. of the 13th Conference of Open Innovations Association FRUCT and 2nd Seminar on e-Tourism for Karelia and Oulu Region Petrozavodsk, Russia, 22-26 April* , 2013 Publisher: State University of Aerospace Instrumentation (SUAI), ISSN 2305-7254, 226 p.

[2] Russian Federation government regulation #442 from 04.05.2012.

[3] LenPoligraphMash’s official website, Web: http://www.lenpoligraphmash.ru.

[4] L. Atzori et al., “The Internet of Things: A survey”, Comput. Netw. (2010), doi:10.1016/j.comnet.2010.05.010.

[5] A. Metnitzer, “The “horizontalization” of the machine to machine (M2M) world and how to market it”, MIPRO, 2012 Proceedings of the 35th International Convention, pp.603,606, 21-25 May 2012.

[6] Web Ontology Language (OWL) official site, Web: http://www.w3.org/2004/OWL/.

[7] Resource Description Framework (RDF) official site, Web: http://www.w3.org/RDF/.

[8] K. De Craemer, G. Deconinck, “Analysis of State-of-the-art Smart Metering Communication Standards”, *in Proc. of the 5th Young Researchers Symposium*, March 2010.

[9] AnyBridge M2M Platform, Web: http://www.anybridge-m2m.nl/anybridge-m2m-platform.

[10] Metrilog official website, End to end solutions for the M2M Market, Web: http://www.metrilog.at/site/index.php/en/products-a-services.html.

[11] CAN specification, ISO 11898-1 2003, Web: http://www.iso.org/iso /iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=33422.

[12] 1C Arenda official website, Web: http://solutions.1c.ru/catalog/rentestate.