Journal of Mobile, Embedded and Distributed Systems, vol. VI, no. 4, 2014
ISSN 2067 – 4074

# IoT – Internet of Things Architecture for Context Aware Sensors Data Processing in Waste Management Solution

**Cristian TOMA, Marius POPA**
*Department of Economic Informatics & Cybernetics*
*The Bucharest University of Economic Studies*
*ROMANIA*
*cristian.toma@ie.ase.ro, marius.popa@ie.ase.ro*

***Abstract***: Internet of Things (IoT) refers interconnectivity of different devices and its increasing reasons aim Cloud Computing Services development, interconnectivity among personal smart devices and other devices, and significant development of the applications operate with this kind of connections and data provided by such connections. The biggest role is played by the devices with measuring capabilities helping the understanding of the world around by humans analyzing data generated in new points by these instruments. Data are securely stored and processes to be a viable source for real-time decisions. The paper provides an overview of this new data acquisitions paradigm together with short presentations of the communication protocols can be implemented in IoT infrastructure. Also, a possible solution architecture is provided for waste management.

***Key-Words:*** IoT – Internet of Things, IoT Communications protocol, MQTT, CoAP, REST

## 1. IoT Overview

An overview of technologies from IoT is described in [21] (fig.1). The IoT infrastructure is the support base for all solutions from "Smart Cities" including "Smart Waste Management".
In a typical IoT infrastructure, the logical data flow is the following:

- **IoT Sensors** are items used to obtain specific data indicators from environment like for RFID, Humidity, Temperature, images, video / multimedia, etc.;
- **IoT Nodes** (Embedded Devices – Smart Objects) collect data about various indicators from the system using **IoT Sensors**;

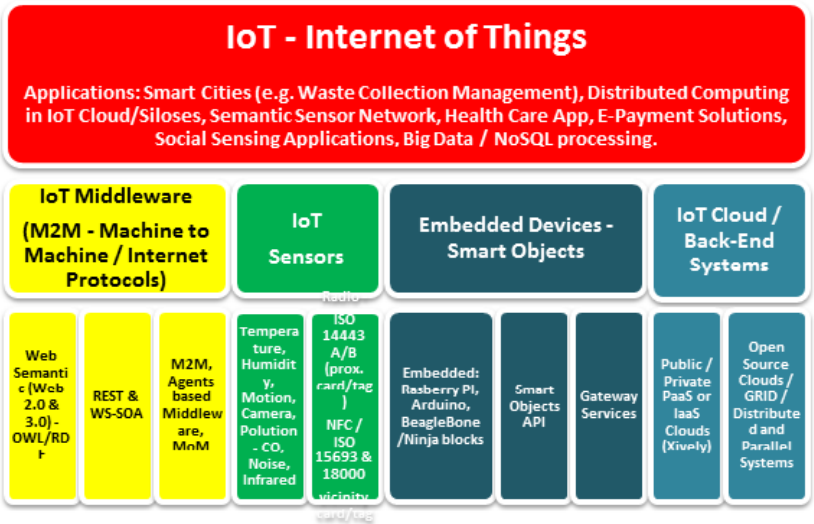


*Figure 1. IoT – Internet of Things technologies modified from [21]*

- The **IoT Gateways** (Embedded Devices – Smart Objects) processes the data collected from IoT Nodes may establish a local feed-back behavior and send to the Data-Center / Back-End system, in order

to process them with dedicated solutions for Big Data, Distributed and Parallel / GRID / Cloud computing;

- **IoT Cloud** / Back-End **Systems** like Public/Private or OpenSource Clouds (in PaaS – Platform as a Service or IaaS – Infrastructure as a Service approaches) for IoT are designed to provide frameworks and API for collecting data from IoT gateways and nodes and to compute them accordingly with the solution's business logic rules.
- **IoT Middleware** / Communications Protocols are used to link the IoT sensors with IoT nodes and Gateways (M2M – Machine to Machine protocols) and finally to connect the IoT nodes/gateways with cloud back-end servers.

A list of possible embedded boards that might be used as IoT Nodes / Gateways is in Table 1:

*Table 1. IoT Devices*

| IoT Boards | IoT Features – HW + SW | Tools + Developers Web Page |
|---|---|---|
| **Waspmote** | **HW characteristics:**<br>***Microcontroller:*** ATmega1281<br>***Frequency:*** 14MHz<br>***SRAM:*** 8KB<br>***EEPROM:*** 4KB<br>***FLASH:*** 128KB<br>***SD Card:*** 2GB<br>***Temperature range:*** [-10ºC, +65ºC]<br>***Clock:*** RTC (32KHz)<br>***GPS:*** Optional<br>***Wireless interfaces:*** 802.15.4/ZigBee, WiFi, 6LoWPAN / IPv6 Radio, 3G + GPS, GSM / GPRS, Bluetooh PRO, RFID/NFC, Expansion Radio Board, Waspmote Gateway<br>***Inputs/Outputs:*** 7 Analog (I), 8 Digital (I/O), 1 PWM, 2 UART, 1 I2C, 1USB, 1SPI<br>***Accelerometer:*** ±2g/±4g/±8g<br>***Sensor Boards:*** Gases, Events, Smart Water, Smart Cities, Smart Parking, Agriculture, Video Camera, Radiation, Smart Metering, Prototyping Sensor<br><br>**SW characteristics:**<br>***Operating system:*** No, embedded programs uploaded and run by board. | Programming Languages:<br>▪ C++, Waspmote code<br><br>Tools & SDK:<br>▪ Waspmote Pro IDE;<br>▪ Waspmote Pro API;<br>▪ OTA Shell<br><br>Web:<br>http://www.libelium.com/development/waspmote/documentation/?cat=programming |
| **Arduino Yún** | **HW characteristics:**<br>***Microcontroller:*** Atmega32u4 and Atheros AR9331<br>***Flash Memory:*** 32 KB (of which 4 KB used by bootloader)<br>***SRAM:*** 2.5 KB<br>***EEPROM:*** 1 KB<br>***Clock Speed:*** 16 MHz<br>***Digital I/O Pins:*** 20<br>***PWM Channels:*** 7<br>***Analog Input Channels:*** 12<br>***Linux microprocessor Atheros AR9331***<br>***Architecture:*** MIPS @400MHz<br>***Ethernet:*** IEEE 802.3 10/100Mbit/s<br>***WiFi:*** IEEE 802.11b/g/n<br>***USB Type-A:*** 2.0 Host/Device<br>***Card Reader:*** Micro-SD only | Programming Languages:<br>▪ C/C++, Arduino code<br><br>Tools & SDK:<br>▪ Arduino IDE;<br>▪ Arduino API;<br>▪ Linino images for the Yun;<br><br>Web:<br>http://arduino.cc/en/Main/Software |

ISSN 2067 – 4074

| | | |
|---|---|---|
| | ***RAM:*** 64 MB DDR2<br>***Flash Memory:*** 16 MB<br><br>**SW characteristics:**<br>***Operating system:*** Atheros processor supports a Linux distribution based on OpenWRT named Linino | |
| **Intel Galileo** | **HW characteristics:**<br>***Microcontroller:*** 32-bit Intel® Pentium® instruction set architecture (ISA)-compatible processor<br>***Frequency:*** 400 MHz<br>***Cache:*** 16 KB L1<br>***SRAM:*** 512 KB<br>***DRAM:*** 256 MB<br>***Legacy SPI Flash (bootloader):*** 8 MB<br>***Micro SD card (optional):*** up to 32 GB<br>***EEPROM:*** 11 KB<br>***Communication:*** 10/100 Mb Ethernet RJ45, USB 2.0 Client port, USB 2.0 Host port, RS-232 UART port and 3.5mm jack, Mini PCI Express (mPCIe) slot with USB2.0 Host support<br><br>**SW characteristics:** Linux based and Arduino API | Programming Languages:<br>▪ C/C++, Arduino code<br><br>Tools & SDK:<br>▪ Arduino IDE;<br>▪ Arduino API;<br>▪ Linux Image for SD for Intel Galileo;<br>▪ Little Linux Image Firmware Update for Intel Galileo;<br>▪ Board Support Package Sources for Intel Quark;<br><br>Web:<br>https://communities.intel.com/docs/DOC-22226 |
| **Raspberry Pi** | **HW characteristics (Model B):**<br>***Microcontroller:*** Broadcom BCM2835<br>***SDRAM:*** 512 MB (shared with GPU)<br>***USB 2.0 ports:*** 2 (via the built in integrated 3-port USB hub)<br>***Storage:*** SD / MMC / SDIO card slot<br>***Network:*** 10/100Mbps Ethernet USB adapter on the third port of the USB hub<br>***Video outputs:*** Composite RCA (PAL and NTSC), HDMI, raw LCD Panels via DSI<br>Audio outputs: 3.5 mm jack, HDMI, and I²S audio<br><br>**SW characteristics:** Raspbian, Pidora, XBCM distributions, other Linux based OSs. | Programming Languages:<br>▪ Python, C++, C#, Java;<br><br>Tools & SDK:<br>▪ Python IDE;<br>▪ Visual Studio IDE;<br>▪ Java SE Embedded;<br><br>Web:<br>http://docs.python.org/3/<br>http://www.oracle.com/technetwork/articles/java/raspberrypi-1704896.html |
| **BeagleBone Black** | **HW characteristics (Model B):**<br>***Microcontroller:*** AM335x ARM Cortex-A8<br>***Frequency:*** 1 GHz<br>***RAM:*** 512MB DDR3<br>***Storage:*** 2GB 8-bit eMMC on-board<br>***Connectivity:*** USB client, USB host, Ethernet, HDMI, 2x 46 pin headers<br><br>**SW characteristics:** Ångström Linux, Android, Ubuntu. | Programming Languages:<br>▪ C/C++, Javascript;<br><br>Tools & SDK:<br>▪ Cloud9 IDE;<br>▪ BoneScript Javascript library;<br><br>Web:<br>http://beagleboard.org/Support/Software%20Support |

In IoT infrastructures the communications protocols are important and there are in the market already implemented versions of middleware in WSN (Wireless sensor Network) and communications protocols such as: CoAP, MQTT and REST/SOA-SOAP, (optional other new protocols such as XMPP). These protocols can operate over radio or wire. For wireless / radio connections, it is taken into account the following standards: ZigBee (IEEE 802.15.4), Wi-MAX (IEEE 802.16), Wi-Fi (IEEE 802.11 variants), Bluetooth (IEEE 802.15.1), UWB (IEEE 802.15.3a), Flash OFDM (IEEE 802.20), 6LoWPAN - IPv6 over Low power Wireless Personal Area

Networks (IETF RFC 6282 & 4919), but not limited to them. Another important issue regarding technological approaches is security and quality of the solution. In the next section an overview of IoT communications protocols and middleware is presented.

## 2. IoT Communications Protocols

IoT communication protocols aim specifications adapted to resource-constrained devices architectures and infrastructures. Below, the MQTT, CoAP and REST specifications are presented.

### 2.1 MQ Telemetry Transport (MQTT)

MQTT is a communication protocol aiming a very simple and lightweight messaging model according to following principles [25]:

- Minimization of device resource requirements as limited processor or memory;
- Minimization of network bandwidth, cost and unreliability;
- Ensuring reliability of the communication model;
- Assurance of message delivery.

There are two main specifications for MQTT:

1. ***MQTT v3.1 specification*** – implements a publish/subscribe messaging model for remote locations requiring small code footprint and/or premium network bandwidth;
2. ***MQTT-SN v1.2 specification*** – is a MQTT protocol version for Sensor Networks aiming embedded devices in non-TCP/IP networks.

MQTT v3.1 protocol specification is stated by International Business Machines Corporation (IBM) and Eurotech. It is a communication protocol adapted to constrained environments. The main sections of the protocol aim the following issues [26]:

- The message format is common to all packet types;
- The packet types are described in specific details;
- The packets flow and the packets exchanges between client and server.

The MQTT command message can contain [26]:

- A fixed header – the header format is composed by two bytes:
  - *Byte 1* – contains the Message Type (bits 7 to 4) and the flags DUP (bit 3), QoS level (bits 2 to 1) and RETAIN (bit 0); Message Type takes one of 16 possible values and encodes the command mnemonic; DUP flag is set when there are re-deliver attempts of some messages; QoS is used when a PUBLISH message is delivered and the flag indicates the level of delivery assurance; RETAIN is a flag used only for PUBLISH messages when these are sent by client to a server, setting the value 1 for this flag;
  - *Byte 2* – contains the Remaining Length field; indicates the number of bytes remaining for the current message (variable header, payload) less the length of the fixed header;
- A variable header – is placed between the fixed header and the payload;
- Payload – the MQTT command messages types are:
  - *CONNECT* – contains one or more UTF-8 encoded strings; the content is set by the flags from the variable header; UTF-8 is an encoding technique for Unicode strings and it is used in text-based communication;
  - *SUBSCRIBE* – contains UTF-8 encoded strings assigned to topic names to which the client can subscribe and the QoS level;
  - *SUBACK* – contains a list of granted QoS levels in the same order like the topic names from correspondent *SUBSCRIBE* message;
- Message Identifier – is present in messages where the QoS level in the fixed header indicates level 1 or 2; it is stored in the variable header of some MQTT messages.

MQTT-SN v1.2 protocol specification is designed for small devices constrained by cost, battery life, limited processor and memory resources. The MQTT-SN

ISSN 2067 – 4074

architecture is described in [27] and depicted in figure 2.

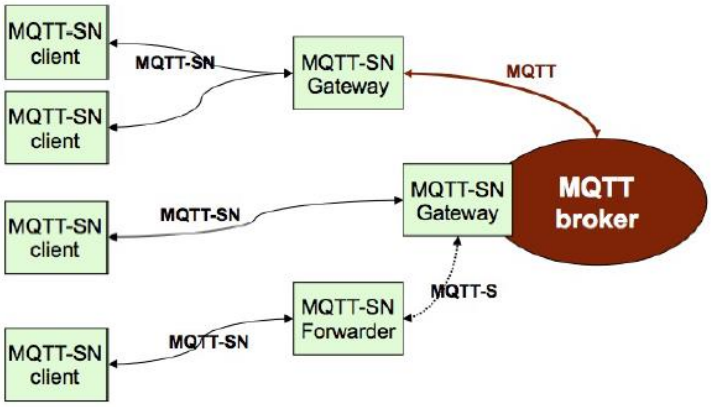


*Figure 2. MQTT-SN architecture [27]*

The MQTT-SN architecture emphasizes three types of components:

- MQTT-SN client – connects itself to a server via a MQTT-SN gateway using the MQTT-SN protocol;
- MQTT-SN gateway – possible to be integrated with a MQTT server; when it is not integrated, the gateway uses the MQTT protocol for communication with MQTT server; a translation between the MQTT and MQTT-SN is made by the MQTT gateway;
- MQTT-SN forwarder – is the connection component between the client and gateway when the gateway is not directly attached to client's network; forwards the MQTT-SN frames from client to gateway without frame changing.

The MQTT-SN message format includes two main components [27]:

- Message Header (2 or 4 bytes) – is composed by two parts:
  - Length (1 or 3 bytes) – total number of bytes contained in the message, including the Length field itself;
  - Message Type (1 byte) – specifies the message type encoded on 1 byte;
- Message Variable Part (n bytes) – depends on the message type stored in Message Header; the following fields are defined [27]:
  - ClientId – unique identifier of the client used by server; contains a 1-23 character long string;
  - Data – corresponds to the payload of MQTT PUBLISH message;
  - Duration – number of seconds stored in 2-byte long field;
  - Flags – have 1 byte length for the following flags: DUP, QoS, Retain, Will, CleanSession, TopicIdType;
  - GwAdd – contains the address of a MQTT gateway; has a variable length depending the network over which the MQTT-SN operates;
  - GwId – unique identifier of MQTT gateway; has 1 byte length;
  - MsgId – corresponds to Message Identifier in MQTT protocol specification; is stored on 2-byte long area;
  - ProtocolId – is coded 0x01, other values being reserved; it is 1-byte long; it is specified only in a CONNECT message;
  - Radius – stores the value of the broadcast radius; it is 1-byte long;
  - ReturnCode – is given by the encoded value stored by this field; the meanings of the codes are detailed in MQTT-SN protocol specification [27];
  - TopicId – has 2-byte length and the values 0x0000 and 0xFFFF are reserved;
  - TopicName – has a variable length and it is specified by a UTF-8 encoded string;
  - WillMsg – contains the Will message; the length is variable;
  - WillTopic – contains the Will topic name; the length is variable.

The MQTT-SN protocol specification provides the formats for each MQTT-SN

message. Also, the protocol specifies the functional procedures between the actors of the MQTT-SN architecture [27].

## 2.2 Constrained Application Protocol (CoAP)

CoAP is a communication protocol used in constrained nodes and networks. CoAP is a *web protocol* designed for constrained environments. The main features of the CoAP aim [28]:

- Web protocol for constrained networks;
- User Datagram Protocol (UDP) binding;
- Asynchronous message exchanges;
- Low complexity of header structure and parsing operation;
- Support for Uniform Resource Identifier (URI) and Content-type;
- Capabilities regarding the proxy and caching implementation;
- Access to CoAP resources via HTTP made by proxy;
- HTTP interfaces implemented over CoAP;
- Security binding to Datagram Transport Layer Security (DTLS).

The CoAP communication model is based on message exchange over UDP between network nodes. The CoAP message is stored in data section of one UDP datagram. Also, CoAP may use DTLS, SMS, TCP or SCTP to implement the message exchange between the endpoints.

CoAP messages are encoded in binary format and have the following structure [28]:

- Fixed-size header (4-byte long) – includes values organized in fields like: Version, Type, Token Length, Code, Message ID;
- Variable-length Token value (0 to 8 bytes) – used in correlation of requests and responses;
- CoAP options – is given in Type-Length-Value (TLV) format and can be followed by another options; each option is structured in the fields Option Number, length of the Option Value, and the Option Value;
- Payload – is preceded by the Payload Marker.

The transaction model is presented in [28] and it is depicted in figure 3.

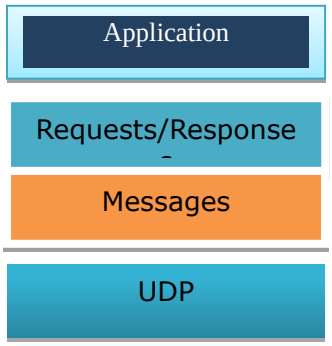


*Figure 3. CoAP transaction model [28]*

The messaging model uses the UDP to exchange messages between CoAP nodes. The messages are simple and consist of a fixed-size binary header and an eventual binary option section and a payload. The CoAP messages are shared by requests and responses.

Each message has a 2-byte long Message ID field. Using the default parameters, the CoAP can transmit to about 250 messages per second from one endpoint to other CoAP node.

The message transmission reliability is implemented using particular messages as Confirmable (CON), Acknowledgement (ACK) and Reset (RST). When a CoAP endpoint responses with an ACK message containing the same Message ID like the request message, then the message transmission is reliable. Otherwise, the response message is RST message that means the endpoint is not able to process the CON message.

CoAP does not require that all messages to be reliably transmitted. In this case, the server can respond only with RST message, and the client request is a Non-confirmable message (NON), including also the Message ID.

The request/response model implemented by CoAP uses the concept of Token as separate concept from the Message ID. The request message includes the Message ID and Token values and the server responses with the same Message ID and Token, containing information about the response content and the proper content. The request/response matching is made on Token field.

ISSN 2067 – 4074

Within request/response model, the following scenarios are available [28]:

1. The server response carries the information to a CON message, when this information is immediately available;
2. The server response is and empty ACK and sends back the information in a separate message; the empty ACK is necessary to stop the client to resend the CON message; when the response information is ready, the server sends a CON message to client having a different Message ID as against the initial CON and ACK messages, but the same Token to match the request to response; the client sends back a ACK message as response to CON message requested by server and containing the information requested by client in the first CON message;
3. The client request is a NON message; the server sends back a NON or CON message, having different Message ID, but the same Token for request/response matching; the server response has the information requested by client.

The methods used by CoAP (GET, PUT, POST, DELETE) have a similar applying way like Hypertext Transfer Protocol (HTTP) for a better mapping between the CoAP and HTTP. There are significant differences between two protocols, but the HTTP experience helps for a better applying of CoAP. CoAP implement caching to make more effective the request fulfillments. The cache is placed in a CoAP node that can be an intermediary or endpoint node.

Also, CoAP implement proxy when the network is constrained. The constraints aim the following issues [28]:

- Limitation of the network traffic;
- Performance improvement;
- Accessing resources of sleeping devices;
- Security issues.

The CoAP message transmission implements asynchronous message exchanges and uses the UDP. As effect, the CoAP messages have issues regarding the transmission reliability. The reliability features of the CoAP are [28]:

- A simple retransmission reliability for CON messages;
- Detection of the duplicated CON and NON messages.

The message transmission considers the following elements in its implementation, as it is stated in [28]:

- Messages – are encoded in binary format and have a structure; the type is specified in the Type field of the message header; also, messages are parts of the request/response model and they may carry a request, response, or be Empty, depending on the Request/Response Code field of the message header.
- Endpoints – represents the sources or destinations of the CoAP messages; a security mode may be implemented for each endpoint;
- Reliable transmission – is implemented on CON messages; the destination endpoint must acknowledge the CON message with an ACK message, or reject it by a RST message;
- Non-Reliable transmission – is used when the message does not require an acknowledgment; a such message is marked as Non-confirmable message;
- Message correlation – is made on Message ID generated for CON and NON messages; it has 2-byte size and it is generated by the sender, being included in the message header;
- Message de-duplication – the receiver endpoint must acknowledge each duplicated CON message which is processed only once;
- Message size – a CoAP message must avoid the IP fragmentation, and it must encapsulated in a single IP packet;
- Congestion control – aims limitation of simultaneous interactions with a server when an ACK message is expected to be received as response to a CON message by using NSTART parameter; the default value of NSTART is 1;
- Transmission parameters – the message transmission parameters and their default values are presented in table 2.

*Table 2 The message transmission parameters [28]*

| Parameter name | Default value |
|---|---|
| ACK_TIMEOUT | 2 seconds |
| ACK_RANDOM_FACTOR | 1.5 |
| MAX_RETRANSMIT | 4 |
| NSTART | 1 |
| DEFAULT_LEISURE | 5 seconds |
| PROBING_RATE | 1 Byte/second |

## 2.3 Representational State Transfer (REST)

REST is a style of communication architecture between systems that implements a request/response model over the Internet. REST is a simple alternative for Remote Procedure Calls and Web Service. REST cannot provide asynchronous message exchanges. The stateless model supported by HTTP requires including additional information in every request that must be processed by the server. This issue introduces additional time in request processing and additional needs regarding the available resources [29].

In the REST style, the message exchanges are made in plain text because the HTTP is used as application layer transfer protocol between client and server. Also, the SSL and TLS may be used for security assurance.

REST uses HTTP request to manage data through Create/Read/Update/Delete operations, implemented by the request methods GET, POST, PUT and DELETE.

According to [30], the REST architecture components aim:

- Resources – are identified by URIs and represent the key component of the architecture; related information about a resource is available by links;
- Client/Server model – any node can issue a request to any of the other nodes; the request issuer is the client and the receiver is the server; thus, a server may be a client of other node;
- Stateless model – the server response cannot use information resulted from previous interactions; thus, the server needs all appropriate information to complete the client request; that information is included in the client request;
- Resource caching – implemented to increase the effectiveness of resource use; decision regarding what resources and how long are cached is made by server;
- Proxy servers – are implemented to improve the performance and scalability of the REST architecture.

Service-Oriented Architecture (SOA) provides application functionality as services to other applications. SOA is software architecture and makes the computers connected over a network to cooperate in order to provide the complete functionality of large application software. SOA can be implemented by using web services standards like Simple Object Access Protocol (SOAP). SOAP is a protocol specification for exchanging structured messages in computer networks as web services. The message format is defined on Extensible Markup Language (XML) Information Set, and message exchanges rely on application protocols like HTTP or Simple Mail Transfer Protocol (SMTP).

SOAP provides request/response encapsulation and transmission over the network using the HTTP. Concepts like addressing, security, discovery or service composition are descripted in the Web Services (WS-*) standards and they are adapted to meet the needs of the resource-constrained devices [31].

SOA can use REST as service-based technology to provide a greater interoperability of the application software. Thus, uniform interfaces may be created to couple services for different smart devices.

ISSN 2067 – 4074

## 3. IoT Architecture for Processing Sensors Data in Waste Management Solution

A solution proposal for smart waste management solution has been submitted for SSE research call. The solution was so called Eco-SMART. The one of the core task is to obtain data from a deployed IoT – Internet of Things and embedded devices infrastructure with sensors (two main types: in the vehicle-embedded and on the containers / cans), RFID/NFC tags, IoT smart-objects and gateways within garbage collection company, in order to dynamically re-route garbage/waste vehicles in traffic. The main tasks from the data-flows of the solution consist of:

- Collecting necessary info from the sensors, such as: cans and containers emptiness percentage; info from public smart recycling bins; clients-citizens/companies dynamic requests regarding waste pick-up; info about the garbage/waste selection pre-processing (glass, plastic, paper, etc.); vehicles fleet management and availability info (including the gas level – from the vehicle sensors); weather conditions (humidity, pollution, etc.) info for environment monitoring; snow level info on the streets for enhanced routing of the vehicles; etc. The data is collected from the IoT sensors through WSNs (Wireless Sensors Networks) using IoT gateways deployed near a set of containers or embedded/IoT devices deployed into vehicles.
- Dynamically re-routing the garbage/waste vehicles in traffic (real time) based on the IoT collected data. The re-routing is based on optimized critical paths obtained according with the consortium's implemented mathematical models, which will take as input data collected from IoT sensors and it will integrate a telematics fleet-vehicles management system.

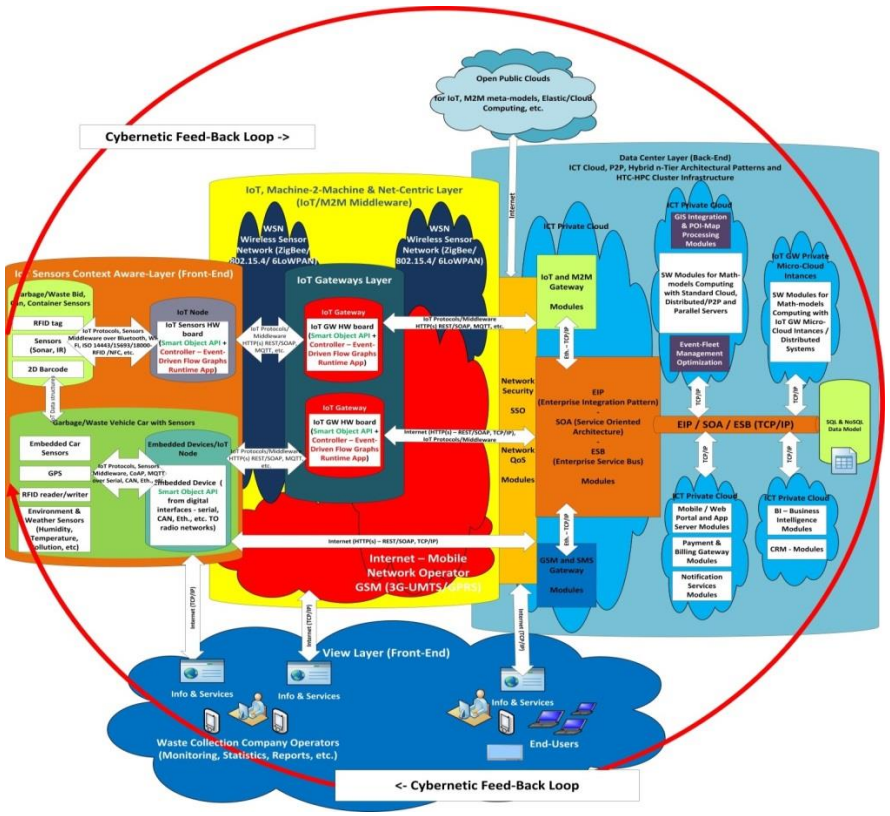


*Figure 4. IoT Components System Architecture for Waste Collection Management System*

- After the garbage/waste collection procedures are executed for the end-user (citizens and companies), it is necessary to offer the possibility to integrate a "pay as you throw" system, taking into account secure payment solutions available in the market.
- All the data collected from the sensors, it will ensure the prerequisites for an enhanced and optimized garbage collection selection process and management, for providing environmental protection mechanisms, as well as obtaining an environmental monitoring system ($CO_2$ emissions, pollution indicators, etc.).

Figure 4 represents the overview for the IoT Waste Collection system components architecture and interactions between them.

The proposed components system architecture lays on IoT (Internet of Things) infrastructure with cybernetic feed-back loop approach in design (sensor-data -> "brain" processing -> "real-time" behavior changing).

The solution presents an IoT Smart System architecture based on the following n-tier design pattern (fig. 4):

- S1: Sensors and IoT Gateways / Smart Objects Sub-system (**Front-End – Context Aware Tier**)
- S2: Mobile and Vehicle Embedded Devices Sub-system (**Front-end**)
- S3: IoT Silos/Micro-cloud Infrastructure Sub-system – containing modules for (**Back-end**):
  - Heterogeneous Sensors Data Acquisition Collection + M2M Proxy and Gateway + Micro-cloud IoT instances for HPC (High Performance Computer) and HTC (High Throughput Computing) green computing
- S4: IT Cloud, Data and Processing Center Sub-system – containing modules for (**Back-end**)
  - GIS (Geographical Information Systems), POI (Point of Interests) - Maps and Graph Routing + IT Cloud/P2P (Peer2Peer) for implementing the mathematical models in order to obtain dynamic minimal paths/routes calculation in HTC (High Throughput Computing) and HPC (High Performance Computing) approach, according with the received data from the sensors, vehicles and citizens.
  - Traffic and Fleet Management
  - BI – Business Intelligence for automated decisions + ERP – Enterprise Resource Planning for resources management + ESB – Enterprise Service Bus and SOA – Service Oriented Architecture Modules for data flow orchestration + Payment and Billing for citizens and companies
  - Mobile (SMS/Smart-Mobile-App) and Web for the project web-services, web-portal and mobile convergence access (from the end-users access)
- S5: Secure Middleware Sub-system contains modules in Front-End and Back-End, which are responsible with data communications protocols implementation (REST, WS, etc.), M2M (Machine to Machine) data structures exchange, and secure messages routing.

## 4. Conclusions

Deploying a proper IoT infrastructure and architecture for waste management solution, a lot of possible applications are showing market potential. Large number of practical established routes might be transformed, due to the implementation of the proposed architecture, into an integrated **flexible structure of waste collection of garbage.**

The **first applicable solution** is for **Optimization of the Procedures and Management of Garbage/Waste Collection** in major cities from Romania / E.U. Most of the companies from Romania that are responsible with waste collection management; they are also responsible with snow removal. The real challenge is to be determined after the moment of reaching at the finest possibility of efficiency made in other environment

ISSN 2067 – 4074

structures, such as **snow removal routes**. If the road that is to be applicable for snow removal is going to be announced to the stakeholders that are staying on both sides, the local citizens will also remove their cars and vehicles from that particular road. The final result of cleaning the snow from the road will be faster, cheaper implementation of the procedures made by snow removal company and also a **large procedure of resizing of damages** made in the process of snow-removal at the citizens cars because of the lack of cars left on the road at the moment of cleaning due to the prior announcement made through using software massive computing technologies for processing "Big Data", such as combination of IT Cloud, HTC (High-Throughput Computing with P2P or GRID systems), HPC (High Performance Computing – with server accelerators boards – each with 60 cores/240 virtual cores).

The **second applicable solution with market potential** is to use the same IoT (Internet of Things) infrastructure for ***Real Time Environment Monitoring*** within Smart Cities systems. The waste/garbage collection vehicles have environment sensors for: Wind Speed, Wind Direction, Rainfall, Temperature, Humidity, Barometer, etc., and reports at a configurable amount of time to the data center, in order to have a ***real time environment monitoring system***.

Also, it is possible as **third possible potential application** to develop a **Smart City Sensors Data Map** (*as a tool to study "smartness" of a city*) from university campuses, public network services, institutions buildings, hospitals, etc. sensors, in order to have a better management for: *City development* (Openness and communality availability of digital services); *City Offices and enterprises productivity and security* (with fire-detection and other buildings sensors), and Information Technology (IT City infrastructure, know-how and semantic).

## Acknowledgement

Parts of this paper were presented by the authors at "13th International Conference on Informatics in Economy (IE2014) – – Section 1: Cloud and Distributed / Parallel Computing", Bucharest, Romania, 15 – 18 May 2014.

## References

[1] Top 50 IoT – Internet of Things Sensor Applications: http://www.libelium.com/top_50_iot_sensor_applications_ranking/http://www.libelium.com/smart_cities

[2] Ian G Smith (Editor), "The Internet of Things 2012 New Horizons", ISBN hard cover: 978-0-9553707-9-3, published in Halifax, UK, www.internet-of-things-research.eu | IERC – European Research Cluster on the Internet of Things - report on 2012: http://www.internet-of-things-research.eu/pdf/IERC_Cluster_Book_2012_WEB.pdf

[3] US Boston implementation of solar powered waste compacting and recycling bins: http://bostinno.streetwise.co/2011/02/02/bigbelly-doubles-revenues-in-2010-boston-due-for-a-citywide-trash-makeover/

[4] David Boswarthick (Editor), Omar Elloumi (Editor), Olivier Hersent (Editor): M2M Communications: A Systems Approach, WILEY, Publication Date: April 30, 2012 | ISBN-10: 1119994756 | ISBN-13: 978-1119994756 | Edition: 1

[5] Olivier Hersent: The Internet of Things: Key Applications and Protocols, WILEY, Publication Date: February 6, 2012 | ISBN-10: 1119994357 | ISBN-13: 978-1119994350 | Edition: 2

[6] Dieter Uckelmann (Editor), Mark Harrison (Editor), Florian Michahelles (Editor): Architecting the Internet of Things, Springer, Publishing Date: April 3, 2011 | ISBN-10: 3642191568 | ISBN-13: 978-3642191565 | Edition: 2011

[7] Hakima Chaouchi (Editor): The Internet of Things: Connecting Objects (ISTE), WILEY, Publication Date: June 14, 2010 | ISBN-10: 1848211406 | ISBN-13: 978-1848211407 | Edition: 1

[8] Honbo Zhou (Author): The Internet of Things in the Cloud: A Middleware Perspective, CRC Press, Publication Date: October 29, 2012 | ISBN-10: 1439892997 | ISBN-13: 978-1439892992 0

[9] JavaScript Libraries / Frameworks used in IoT nodes/gateways and back-end of IoT infrastructures:
*. jQuery (old school):
http://learn.jquery.com/
http://jquery.com/download/
*. Angular JS (MVC + Data-binding) / Knockout JS (Data-binding + UI auto-refresh):
http://angularjs.org/
http://knockoutjs.com/
*. Dojo:
http://www.dojotoolkit.org/
*. Prototype:g
http://www.prototypejs.org/
*. Yahoo YUI:
http://developer.yahoo.com/yui/
*. extJS - Sencha:
http://www.sencha.com/products/touch/
*. MooTools:
http://mootools.net/
http://mootools.net/download
http://mootools.net/docs/core
*. NODE-Red (Node.js) (IoT Graph engine) - www.nodered.org
*. Parse.com (Backbone.js) (REST Services) - www.parse.com
[10] IoT Smart Cities Technologies, http://observatorio.cenatic.es/index.php?option=com_content&view=article&id=807:open-smart-cities-i-open-internet-of-things&catid=94:tecnologia&Itemid=137
[11] Waspmote board: http://www.libelium.com/products/waspmote
[12] Arduino board: http://www.arduino.cc/
[13] Intel Galileo board: http://www.intel.com/content/www/us/en/do-it-yourself/galileo-maker-quark-board.html | http://arduino.cc/en/ArduinoCertified/IntelGalileo
[14] Raspberry Pi board, online at http://www.raspberrypi.org/
[15] BeagleBone Board, online at http://beagleboard.org/bone
[16] Ninja Blocks: www.ninjablocks.com
[17] IOT Toolkit: http://iot-toolkit.com/ | http://iot-datamodels.blogspot.ro/
[18] AspireRFID: http://www.llrp.org | http://wiki.aspire.ow2.org/xwiki/bin/view/Main/WebHome
[19] DASH7: http://www.dash7.org/
[20] BigBellySolar Smart Trash Bin/Can:http://bigbellysolar.com/solutions/stations/smartbelly/
[21] Cristian TOMA, Cristian CIUREA, Ion IVAN – *Authentication Issues for Sensors in IoT Solutions*, Proceedings of the 6th International Conference on Security for Information Technology and Communications (SECITC'13), June 25-26, 2013, Bucharest, Romania, ASE Printing House, ISSN 2285-1798, ISSN-L 2285-1798.
[22] Internet of Things and M2M application platform: http://www.thingworks.com
[23] Public Cloud for Internet of Things: https://xively.com/whats_xively/
[24] Open Source Cloud for Internet of Things: https://sites.google.com/site/opensourceiotcloud/
[25] MQ Telemetry Transport Protocol: http://mqtt.org/
[26] MQTT Protocol Specifications v 3.1: http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/MQTT_V3.1_Protocol_Specific.pdf
[27] MQTT-SN (Sensor Networks) Protocol Specifications v 1.2: http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf
[28] CoAP (Constraint Application Protocol) specifications:
http://tools.ietf.org/html/draft-ietf-core-coap-18
https://datatracker.ietf.org/doc/draft-ietf-core-coap/?include_text=1 draft-ietf-core-coap-18.pdf
[29] Messaging Technologies, http://www.prismtech.com/sites/default/files/documents/MessagingComparsion_190913.pdf
[30] REST Protocol Intro: http://rest.elkstein.org/
[31] REST versus WS comparison from developers perspective: http://vs.inf.ethz.ch/publ/papers/dguinard-rest-vs-ws.pdf