Design of a domain specific language and IDE for Internet of things applications

A.Salihbegovic*, T. Eterovic * , E. Kaljic* and S.Ribic*
* Faculty of Electrical engineering, University of Sarajevo, Bosnia Herzegovina
asalihbegovic@etf.unsa.ba , teo.eterovic@gmail.com, ekaljic@etf.unsa.ba, sribic@etf.unsa.ba

Abstract - With the goal of relieving the Internet of things (IoT)application designers of the complexities and heterogeneity of wireless sensory networks (WSNs), devices, communication media, protocols and operating systems, the development of higher level domain specific language has been undertaken. The result is DSL-4-IoT Editor- Designer, which is based on high level visual programming language, established on the class of visual domain specific modeling languages (VDSMLs). DSL-4-IoT is using formal presentations and abstract syntax in a metamodel. The visual front-end of the Editor has been developed in JavaScript language. The runtime execution of generated IoT application configuration files is done by the open source project "OpenHAB" runtime engine.
In order to demonstrate the viability and usability of the developed DSL-4-IoT visual model based language, an experimental IoT testbed including 15 heterogeneous wireless sensory devices spanning two application domain (smart home and remote patient monitoring), has been designed and deployed.

I. INTRODUCTION

Application development challenges, diversity of devices (physical variables sensors, biometrics sensors), different software implementation (Android, iOS, Linux), different interaction modes (pub/sub, request/response, command) and different engineering units (deg. C, deg. F, bar, N, etc.) are the ecosystem in which the designers of The Internet of Things (IoT) applications are creating their projects[1].

Designers of IoT applications should be relieved of most of these heterogeneity and specifics (wide range of hardware and software entities running on specific platforms, middleware specific features). They need to use integrated development environment (IDE) based on domain specific high level language which in its entities would abstract most of these intricacies and specifics in hardware, software, communication media and protocols. Furthermore, the language should be capable to support large scale design of these systems by combining several design blocks and saving them in application libraries, importing them, reconfiguring and parameter setting with ease for new tags and locations, enabling reusability and scalability [2].

II. DOMAIN SPECIFIC LANGUAGES IN IOT APPLICATIONS DEVELOPMENT

In this area of very intensive research and design, a short overview of the state of the art and approaches to empower IoT designers with higher level languages for their design and deployment is presented.

A. *PervML Language*

PervML is a Domain Specific Language (DSL) designed with the aim of providing the pervasive IoT system developers with a set of constructs that allow them to precisely describe the pervasive system in a technology independent way [3]. PervML promotes the separation of roles where IoT application developers can be categorized as system analysts and system architects.

B. *DiaSuite: a Tool Suite to Develop/Compute/Control Applications Language*

DiaSuite focuses on a specific domain, namely Sense/Compute/Control (SCC) applications [4]. It comprises of:

- Domain-specific design language,
- Compiler producing a Java programming framework,
- 2D-renderer to simulate an application,
- Deployment framework.

DiaSuite provides a design language, named DiaSpec that is dedicated to the SCC paradigm. The design language consists of two layers: taxonomy layer and application design layer.

The taxonomy layer allows a class of entities to be described. An entity is defined as a set of data sources and actuating capabilities, abstracting over devices, whether hardware or software.

Application Design Layer of DiaSpec allows the application logic to be decomposed into context and control operators. Sensors and actuators are the two facets of an entity described in the taxonomy.

C. *Node-RED*

Node-RED open source IBM Emerging Technology software is visual tool for wiring the Internet of things application configurations. It enables the designer to create IoT applications wiring together hardware devices, APIs and online services. Node-RED provides a browser based flow editor that supports wiring the flows using the wide range of sensory nodes in the palette. Created flows can then be deployed and executed in runtime with a single mouse click.

The paper has been prepared and written on the results achieved within the project that was financially supported by the Federal ministry for education and science, Federation of Bosnia Herzegovina.

Runtime engine is built on Node.js, making use of its event-driven, non-blocking model. Since this runtime module is of small footprint, it can run on low-cost hardware such as Arduino or Raspberry Pi as well as in cloud [5].

D. OpenHAB

OpenHAB represents general-purpose framework for smart home gateways and is fully based on the Eclipse SmartHome project, which allows building smart home solutions as the most common application area of IoT. The framework consists of a set of Java OSGi bundles that can be deployed on an OSGi runtime and which defines OSGi services as extension points, acting as middleware on the sever side of WSN (wireless sensory) network [6].

The project focuses on services and APIs for the following topics [7]:

- *Data Handling*: This includes a basic but extensible type system for smart home data and commands that provides a common ground for an abstracted data and device access, as well as event mechanisms to send this information around. Integration with other systems is done through so called bindings, which are a special type of extension.
- *Declarative User Interfaces*: A framework with extensions for describing user interface content in a declarative way. This includes widgets, icons, charts etc.
- *Rule Engines*: A flexible rule engine that allows changing rules during runtime and which defines extension types that allow breaking down rules into smaller pieces like triggers, actions, logic modules and templates.
- *Persistence Management*: Infrastructure that allows automatic data processing based on a simple and unified configuration persistence services are pluggable extensions, which can be anything from a log writer to an IoT cloud service.

Beside the runtime framework and implementation, Eclipse SmartHome has also its text Editors for editing configuration models and rules.

III. STRUCTURE AND FUNCTIONALITY OF DSL-4-IOT LANGUAGE

The vast versatility of the types of sensors and actuators as well as heterogeneity of communication media and protocols that are immanent when the standards are lagging behind, are facing design engineer within the domains of Internet of things applications. Understandably, all this is posing burden on designer to acquire knowledge and learn many specifics that are far beyond the functionality and application requirements that he/she wants to build into the system.

Therefore, there are many efforts going on among researchers and developers to develop high-level domain specific language that will abstract many of these specifics and peculiarities into building blocks and library modules. Then, he/she is designing its application, specifying the devices and hardware needed for designed system acquisition and deployment on the application site [8].

In this research project, we have opted for the development of one visual domain specific language that shall empower IoT application designer with a visual high level language and based on it, a design Editor named DSL-4-IoT. The Editor shall enable application designer to configure the system structure, select devices, sensors and actuators either from built-in library modules available in Designer itself, or enter the ones that he/she selects, parametrizing the metadata available from manufacturer's design literature. These device modules can then be saved and included in built-in library modules for future use, although for these later actions, more specific knowledge is required, and shall be accomplished by someone that shall be named DSL-4-IoT developer.

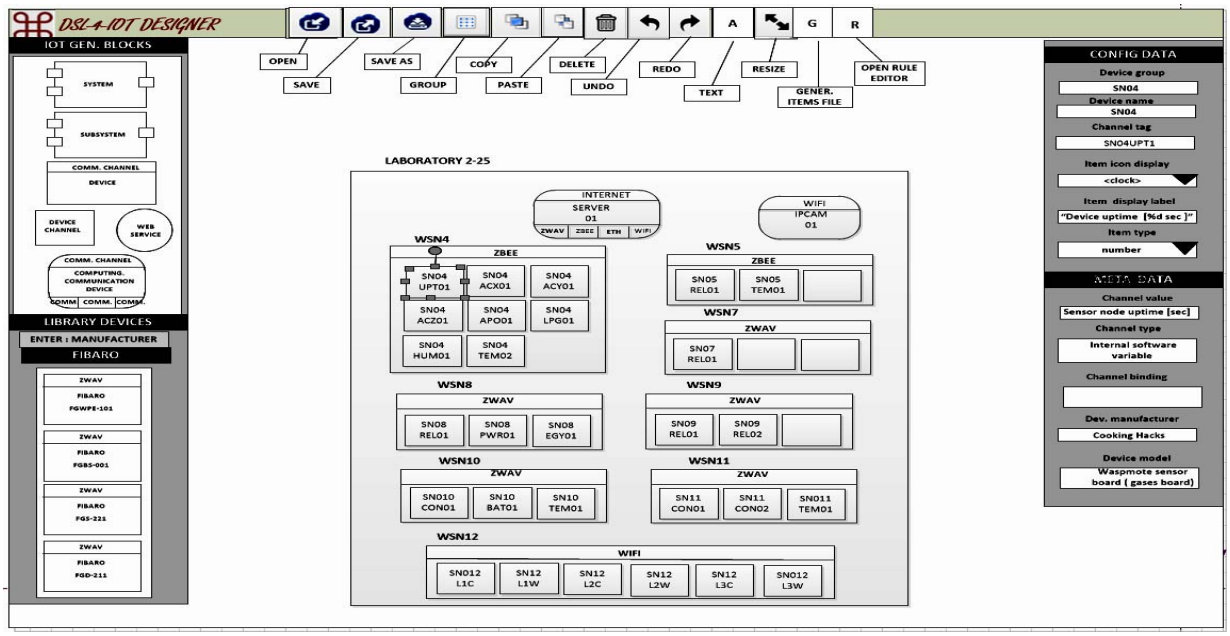

Fig.1 The layout of DSL-4-IoT Editor

The development of the DSL-4-IoT language is based on the class of visual domain-specific modeling languages (VDSMLs) that are using formal presentations and abstract syntax in a metamodel. The front end of the Editor has been developed in JavaScript. The layout of DSL-4-IoT Editor-Designer is shown on Fig. 1.

In the command bar, the Editor is providing the necessary tools for opening and saving the applications configuration, together with additional commands for manipulation of individual or groups of block structures (group, copy, paste, delete, undo, redo, resize, etc.).

Labeling and commenting of the block structures on the hierarchical level of generic blocks (system, subsystem, devices, device channels, etc.) is enabled using the "Text" button.

Two buttons further right, are intended for generation of textual configuration files with extension "items", and "sitemap" that are used directly in the runtime execution.

The last button is used to call an external program named "Rule editor", that enables the Designer to configure any combinatorial, sequential or temporal relationships between "items" created in previous step of entering configuration program. These „items" are the logical representation of physical channels of the configured devices (sensor inputs or actuator or command outputs), or variables associated with messages obtained via web services, SMS messages or emails. In broader terms, items are readable or write-able objects, in order to interact with them. Items can be bound to bindings, i.e. for reading the value or status from device channel input or for updating output channel going to some actuator device. More on this can be found in documentation on OpenHAB project [7].

Bellow the higher abstract layer of DSL-4-IoT domain specific language, additional glue code that is running in Openhab runtime is executing:

- Interfacing software components and middleware

- Interfacing with disparate hardware and software components.

- Mapping code for device and software components

This glue code running in runtime and executing DSL-4-IoT tools developed configuration files, has been selected as Java based OpenHAB 1.x open source project, widely popular in the IoT community.

In order to abstract mentioned heterogeneity of various IoT devices, sensing and actuating physical devices are described as entities in high level manner. These resources are instantiated as many times as necessary with the distinct tag names, to represent all physical and/virtual devices included in the particular project application.

To address scalable operations within the IoT system application logic that connects "items" in interconnected wireless sensory networks of devices and users, hierarchical clustering has to be specified consisting of:

- System
- Subsystem
- Device (autonomous and powered)
- Physical or virtual channel

Use of this hierarchical structure enables: project partitioning at physical level; project partitioning at functional level; defining scope from which software components shall produce or consume data. This multi-stage model driven approach for design of IoT applications supports all stages of the life cycle of these systems. Selected VDSML, model driven approach applied in development of DSL-4-IoT visual design environment, contributes to ongoing search for design tools that shall allow IoT application designers to model their systems at a higher abstraction level, providing them with automatic model transformations to incrementally refine abstract models into more concrete ones.

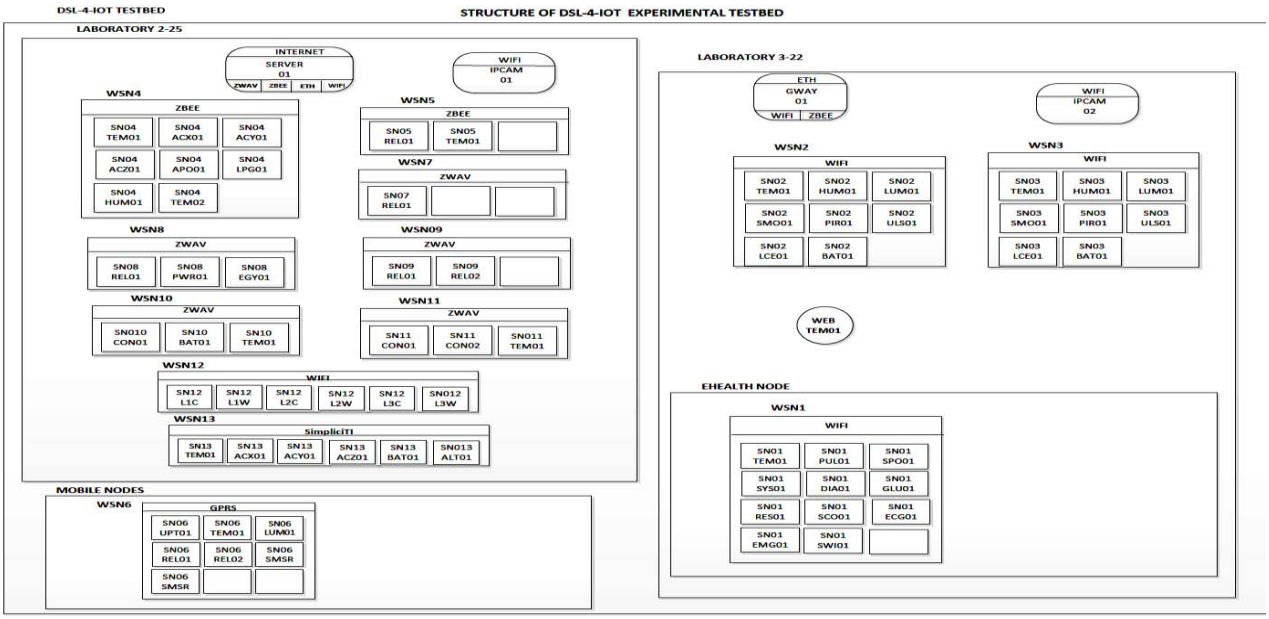

Fig. 2 The structure of experimental testbed generated with DSL-4-IoT Editor

In order to demonstrate the viability and usability of above described DSL-4-IoT model based language and tools, an experimental IoT testbed spanning two domain applications (smart home and remote patient monitoring), has been designed and deployed using the DSL-4-IoT language. The output configuration generated as the result of entering the structure of the experimental testbed is displayed on the Fig. 2.

DSL-4-IoT compiler within Designer is mainly composed of a parser and configuration files generator.

Configuration files generator (items, rules, sitemap) generates formatted text outputs that can be directly downloaded and executed within OpenHAB runtime engine.

Both tools rely on declarative domain-specific language used to graphically represent IoT system topology and "items" embedded in this topology.

Runtime libraries are JavaScript coded based on OpenHAB runtime engine used to execute application with generated configuration files.

DSL-4-IoT for configured IoT system structure, exports the data into one JSON array. This array keeps information about position of all the items within configuration, relationships between items and groups, value of all configured fields associated with items and of data types. JSON configuration array includes OpenHAB specific fields and associated rules. This array is typically huge in size, and is useful as container from which it is possible to restore the state for further editing and setting up of sensory network configuration.

From the configuration stored in JSON array format, it is possible to export data to wider scope of integration platforms for home automation, one of them being OpenHAB. OpenHAB runtime engine requires five kinds of configuration files: Items, Rules, Persistence, Scripts and Sitemaps. These text files differ by file extension and internal structure.

At the time of writing this paper, DSL-4-IoT Designer with visual Editor, parser and configuration files generator for two configuration files: items and sitemaps has been completed and tested, while development of visual Editor for rules configuration is underway.

Items in OpenHAB 1.x runtime can be defined using configuration files located in subfolder configurations/items, relative to OpenHAB installation directory. Item definition files have the file extension *.items. Groups are also defined in the *.items file.

A special conversion program scans the JSON document generated from the visual DSL-4-IoT configuration Editor. Each cell in this document, which is of type iot.device, becomes item, and cells which are recognized as embedding cells, became a group. The items are stored inside groups.

Sitemaps are used to create elements of a user interface. They are stored in OpenHAB runtime directory configurations/sitemaps. Each file has *.sitemap extension. The structure is similar to above described items configuration file. The very first line has an element called Sitemap, and it is followed in next lines by some of elements: Colorpicker, Chart, Frame, Group, Image, List, Selection, Setpoint, Slider, Switch, Text, Video, and Webview [7]. All of them, except Frame, describe user interface elements in single line.

The file which defines rules for IoT application configuration has extension *.rules. A rule file can contain multiple rules, and consists of three sections:

Imports, Variable Declarations and Rules.

The Imports section contains import statement, used to make the imported types available without need of fully qualified name for them. The Variable declaration section declares global variables (with or without initial values), accessible to all rules in this file. The Rules section contains a list of rules. After configuration files are generated, they can be transferred to the respective OpenHAB runtime directory manually or automatically downloaded, using simple web service. Manual transfers are carried out using Blob interface and download links [9].

Configuration files generator within DSL-4-IoT Designer creates these files in broader a more generic way, hence these configuration files are not confined only to OpenHAB runtime for deployment and execution. Since for development and coding of DSL-4-IoT Designer functionality, the interpreted language of JavaScript is used, it is possible to simply replace corresponding JavaScript file to generate and export configuration files for other IoT integration platforms and their runtime engines.

IV. EXPERIMENTAL TESTBED FOR PROOF OF DEVELOPED IDE

For proof of concept and versatility of developed visual domain specific language and development environment based on DSL-4-IoT Designer, the experimental testbed has been built and implemented composed of the variety of wireless static and mobile sensory nodes with multiplicity of sensors, actuators, communication channels and protocols from various manufacturers. From the point of view of IoT application domains, two most frequent domains were chosen to be presented, by selection of the types of sensors and their functionality: smart home and remote patient monitoring.

Sensors and actuators used in build-up of experimental testbed have been either autonomous with integral communication transceiver channel, or hooked up to large number of wireless devices (nodes), based on different communication technologies (Wi-Fi, ZigBee, Z-wave, Bluetooth, NFC) and protocols (HTTP, TCP/IP, UDP, MQTT, Modbus, SimpliciTI, serial protocols, etc.).

The topology of experimental testbed wireless sensor network is depicted on the following Fig. 3:

Experimental testbed is physically divided in several domains:

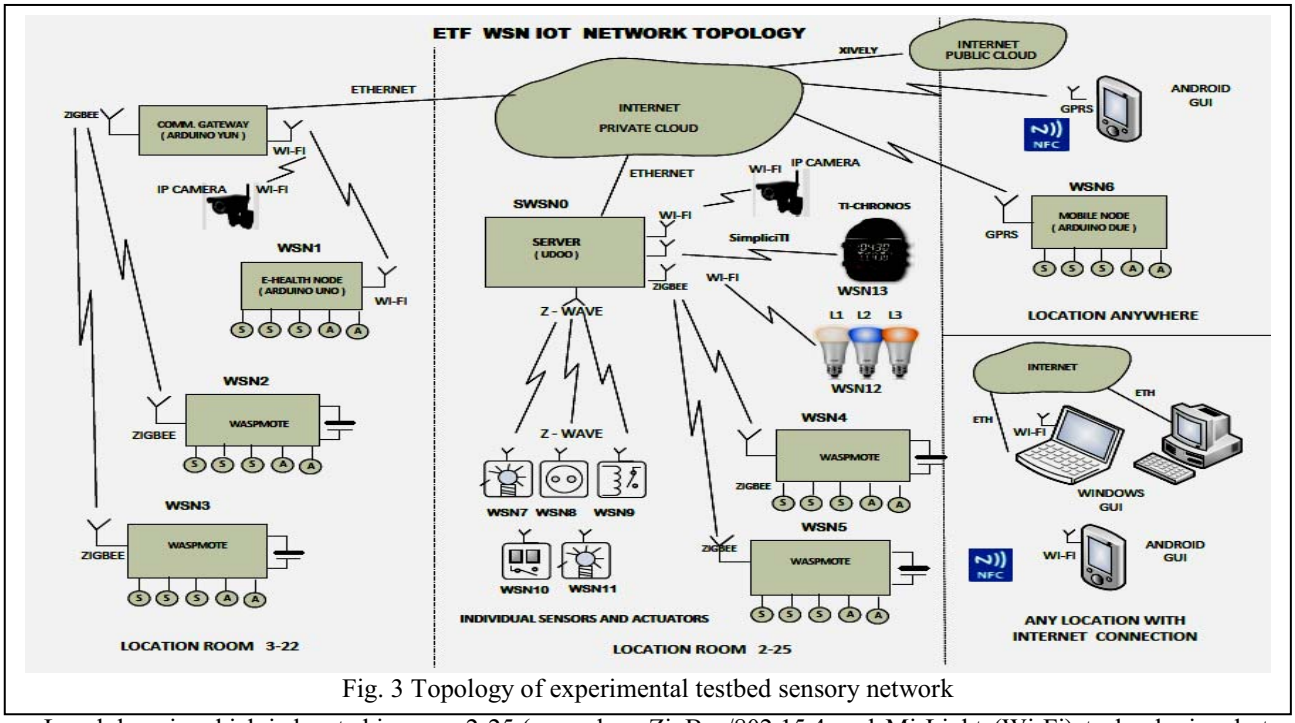

Fig. 3 Topology of experimental testbed sensory network

- Local domain which is located in room 2-25 (second floor within faculty building),

- Remote domain which is located in room 3-22 (third floor within faculty building)

- Mobile domain that can be located anywhere within the reach of mobile network signal.

Local domain (room 2-25) consists of several wireless sensor networks implemented using a variety of wireless accessing technologies (e.g. Wi-Fi, ZigBee, Z-Wave, etc.) and core server (SWSN0).

Z-Wave sensor network consists of a light dimmer (WSN7), window sensors (WSN10), relays (WSN9), outlet with electrical consumption metering (WSN8), and multi-purpose binary sensor (WSN11) connected to the door (open-close) and knock-knock sensor. ZigBee/ 802.15.4 wireless sensor network is implemented using Libelium Waspmote nodes (WSN4 & WSN5) [10]. On the input of the Waspmote nodes are connected various analogue and digital sensors from smart home variety. The sensors perform sensing and measurement of a large number of environmental parameters (e.g. temperature, humidity, luminosity, presence, position, proximity, air pollution with various pollutants, movement, weight, opened, closed, flood, etc.). They are capable, as well, for actuating and controlling functions (switching ON and OFF various electrical devices like lights, fans, heaters, solenoid valves, opening and closing or positioning in any intermittent position blinds, windows, doors, etc.). For control of LED lights including the RGB spectrum and intensity, the Mi-Light sensor network that consists of three RGB light bulbs is used in (WSN12) [11].

Core server (SWSN0) is built on Arduino compatible UDOO module based on embedded computer with Freescale i.MX 6 ARM Cortex-A9 1 GHz quad-core processor and 1 GB of RAM. UDOO server is connected to the communication modules for Z-Wave, ZigBee/802.15.4 and Mi-Light (Wi-Fi) technologies, but also includes the communication modules that make it also a gateway for non-IP devices. UDOO based server executes open source OpenHAB runtime software which is based on the OSGi container. To support the connected communication module protocols, OSGi plug-ins called bindings are used. For Z-Wave and Mi-Light there were already available bindings which have to be included as add-ons. However, for ZigBee/802.15.4 there was no binding readily available, so the support was provided within the Project implementation, developing a dedicated Java application that uses OpenHAB REST API on one side (server), and a serial connection to the ZigBee/802.15.4 module on the other side.

To test and experiment with wearable devices as IoT nodes, one Texas Instrument wristwatch model EZ430-Chronos was integrated with the UDOO server using proprietary SimpliciTI protocol over wireless RF link [12]. This wearable wireless sensor node offers the data on temperature, altimeter (absolute pressure), and three-axis accelerometer.

Remote domain, located in room 3-22, consists of ZigBee wireless sensor nodes (WSN2 & WSN3), of the same type as in the room 2-25 (Libelium Waspmote), Libelium e-Health wireless node (WSN1) and communication gateway (implemented on Arduino Yun embedded processor module). E-Health wireless node is based on the Arduino Uno board, e-Health sensor shield and Wi-Fi module [13]. The e-Health sensor shield empowers biometric and medical applications where human patient local and remote monitoring is needed, using 10 different biomedical sensors. They are: pulse (heartbeat rate), oxygen in blood (SPO2), airflow (breathing), body temperature, electrocardiogram (ECG), glucose meter, galvanic skin response (GSR-sweating), blood pressure (sphygmomanometer), patient position (accelerometer) and muscle/electromyography sensor (EMG). To enable connection of wireless nodes from a

remote domain in room 3-22 to the server in the local domain (room 2-25), the communication gateway is used. Communication gateway is implemented using Arduino Yún with Wi-Fi and ZigBee/802.15.4 communication modules [14].

Mobile domain is based on the GPRS mobile node, and Android based mobile phones and tablets. GPRS mobile node is implemented using Arduino Due board and GPRS shield. For communication between GRPS mobile node and server, OpenHAB REST API was used, utilizing my.openHAB cloud service [15].

Local and remote domains are covered by video surveillance serving as visual feedback of the activities performed in these two rooms, implemented with IR Led night vision IP cameras. Video streams from IP cameras are taken in MJPEG format and presented to the user inside openHAB web interface on desktop and laptop computers, or HABDroid application on Android based mobile phones [16].

User interface that provides monitoring as well as full control of all devices is provided as web application in any web browser that has JavaScript runtime engine included, connecting to IP address of the server (SWSN0). This interface can be accessed either on desktop or laptop computer or any smart phone or tablet with Android or Apple iOS Operating system, downloading the utility program.

V. CONCLUSION

The development of IoT application design environment based on the high level visual programming language in form of Editor within the class of visual domain specific modeling languages (VDSML) using formal presentations and abstract syntax, has been presented in this paper. The front end of the Editor is based on JavaScript language.

The runtime execution of IoT application configuration files, generated with this Editor is carried out by OpenHAB runtime engine developed within the framework of "OpenHAB" open source project.

Work in progress is development of visual rule editor based on Google Blockly blocks for building web based Editors [17]. The entered rules structure shall be converted to text file, compatible with *.rules file within OpenHAB runtime engine. Chris Jackson also announces similar work for OpenHAB project [18], and there is specific version for Vesternet smart home applications as Zipato rule creator [19].

Further extension in the development of IDE is planned on adding new IoT devices into DSL-4-IoT library, which is activity done by IoT developer rather than by IoT application designer. This shall remove the burden from IoT application designers to add new devices not available in the library, and together with devices dynamic discovery feature [20], shall fully automate application library extension in future applications.

REFERENCES

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," Future Generation Computer Systems, vol. 29, pp. 1645-1660, 2013.

[2] J. Clerk J. Kiljander, J. Takalo-Mattila, M. Etelapera, J.-P. Soininen, and K. Keinanen. Enabling end-users to configure smart environments. In Applications and the Internet SAINT), 2011 IEEE/IPSJ 11th International Symposium on, pages 303–308, 2011.

[3] T. S. Lopez, D. C. Ranasinghe, M. Harrison, D. McFarlane, "Adding sense to the Internet of Things An architecture framework for smart object systems," Personal and Ubiquitous Computing, vol. 16, no. 3, pp. 291-308, 2012

[4] Benjamin Bertran, Julien Bruneau, Damien Cassou, Nicolas Loriant, Emilie Balland, et al.. ,"DiaSuite: a Tool Suite To Develop Sense/Compute/Control Applications. Science of Computer Programming", Fourth special issue on Experimental Software and Toolkits, Elsevier, 2012, <10.1016/j.scico.2012.04.001>

[5] Node- RED: Visual tool for writing the Internet of Things, http://nodered.org/ [Accessed on: 10.1.2015]

[6] Eclipse smarthome: A Flexible Framework for the smart home, https://www.eclipse.org/smarthome/ / [Accessed on: 10.1.2015]

[7] OpenHAB : open source project web site, http://www.openhab.org/ / [Accessed on: 10.1.2015]

[8] C. Perera, P. Jayaraman, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Dynamic configuration of sensors using mobile sensor hub in internet of things paradigm. In IEEE 8th International Conference on Intelligent Sensors, Sensor Networks, and Information Processing.

[9] Blob JavaScript API , http://www.javascripture.com/Blob [Accessed on: 10.1.2015]

[10] Waspmote: Libelium wireless sensory network , http://www.libelium.com/products/waspmote/ [Accessed on: 10.1.2015]

[11] Houselogix, Mi-Light Control4 https://www.houselogix.com/shop/limitlessled-milight-easybulb [Accessed on: 10.1.2015]

[12] I. S. SimpliciTI™ - RF software protocol - Texas Instruments http://www.ti.com/corp/docs/landing/simpliciTI/index.htm?DCMP=hpa_rf_general&HQS=NotApplicable+OT+simpliciTI, [Accessed on: 10.1.2015]

[13] e-Health Sensor Platform for Biometric and Medical applications http://www.libelium.com/130220224710/ , [Accessed on: 10.1.2015]

[14] Guide to Arduino Yun , http://arduino.cc/en/Guide/ArduinoYun [Accessed on: 10.1.2015]

[15] My.openHAB , https://my.openhab.org/ , [Accessed on: 10.1.2015]

[16] Y. HABDroid – Android application for openHAB , https://github.com/openhab/openhab/wiki/HABDroid, [Accessed on: 10.1.2015]

[17] Google Developers: Blockly : Library for building visual programming editors, https://developers.google.com/blockly/ [Accessed on: 10.1.2015]

[18] Chris Jackson : Rule designer : Overview, : https://github.com/cdjackson/HABmin/wiki/Rule-Designer%3A-Overview , [Accessed on: 10.1.2015]

[19] Vesternet : basic Guide to the Zipato rule creator, : http://www.vesternet.com/resources/application-notes/apnt-9#.VLzt-0fF_A0 , [Accessed on: 10.1.2015]

[20]] Charith Perera, Prem Prakash Jayaraman, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos," Context-aware Dynamic Discovery and Configuration of 'Things' in Smart environments, , Springer Berlin Heidelberg, Volume 546, Pages 215-241 (2014)