CISTER
Research Center in
Real-Time & Embedded
Computing Systems

Technical Report

A Visual Programming Framework for Wireless Sensor Networks in Smart Home Applications

Maria Serna*

Cormac J. Sreenan

Szymon Fedor

*CISTER Research Center

CISTER-TR-150207

2015/04/07

A Visual Programming Framework for Wireless Sensor Networks in Smart Home Applications

Maria Serna*, Cormac J. Sreenan, Szymon Fedor

*CISTER Research Center

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: mdlam@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

In this paper, we build upon the Internet of Things (IoT) paradigm, with aim of delivering networked solutions that enable to connect not only single sensors, but also whole wireless sensor networks (WSN) to the Internet in a secure, simple and efficient way, and describe the design and implementation of a smart-home management system. The system is composed of a lightweight tool with an intuitive user interface for commissioning of IP-enabled WSN with constrained capabilities. The solution includes a visual programming interface with a common framework for discovering smart home services on the constrained WSN, and a code analysis and translation engine to generate python code. This engine analyses the application rules defined with the graphical user interface and translates them into distributed application scripts. The system also includes modules to plan the optimization of the deployment, and deploy and start the generated code. A prototype of the system, with the visual programming solution and code generation module developed is presented in this paper.

A Visual Programming Framework for Wireless Sensor Networks in Smart Home Applications

M. Ángeles Serna[1*], Cormac J. Sreenan[1]
[1] Dept. of Computer Science
University College Cork
Ireland
m.serna@cs.ucc.ie, cjs@cs.ucc.ie

[*] CISTER/INESC TEC
ISEP, Polytechnic Institute of Porto
Porto, Portugal
mdlam@isep.ipp.pt

Szymon Fedor[2†]
[2] United Technologies Research Center
(UTRC)
Cork, Ireland
szymon.fedor@gmail.com

***Abstract*—Most of the currently deployed integrated home management products require an experienced technician to install and configure the system. In this paper, we build upon the Internet of Things (IoT) paradigm, with the aim of delivering networked solutions that enable multi-node wireless sensor networks (WSNs) to connect to the Internet in a secure, simple and efficient way. We also describe the design and implementation of a smart-home management system. The system is composed of a lightweight tool with an intuitive user interface for commissioning of IP-enabled WSNs. The solution includes a visual programming interface with a common framework for discovering smart home services on the WSN, and a code analysis and translation engine to generate Python code. This engine analyses the application rules defined with the graphical user interface and translates them into distributed application scripts. The system also includes modules to plan the optimization of the deployment, and deploy and start the generated code. In this paper we present a prototype of the system, with the visual programming solution and code generation module.**

Keywords—IoT; WSN; Macroprogramming; CoAP

I. Introduction

In today's homes we can see more and more smart devices. They can enable functionalities such as: automatic heating and air-conditioning based on multiple sensors, as well as environmental or alarm systems that cooperate with other smart devices. With this diversity of smart devices around the house, programming all of them becomes a difficult task. The increasing number of devices is also being enabled by the trend of the so-called Internet of things (IoT), which among other factors, is relying in a set of standards to enable cooperation between these different smart devices.

Most of the currently available integrated home management products require considerable technical know-how to install and configure the system. However, do it yourself (DIY) smart home systems are emerging at very affordable costs, and therefore their adoption is increasing significantly. We can already observe novel solutions for home energy management with a very intuitive commissioning process available in the market [1], [2]. But we are lacking technology enabling DIY *multi-application* smart home systems where a user can discover available services provided by the wireless sensor networks (WSNs) and configure/reconfigure them using intuitive tools as the system evolves. Another example is when a property changes ownership and the new owner needs to have tools that make it possible to learn about the deployed devices, and easily configure the system preferences.

In this paper, we address the complexity of managing the increasing number of smart devices in the home by developing a solution to enable easy configuration and upgrade of WSNs using a graphical programming interface. We target novice users, requiring minimal learning of the system to be used, and aim at enabling users to easily define application high level behaviour, avoiding installation of client applications to commission the WSN and complex configuration. The solution builds upon previously developed technologies: IoT-enabled sensor and actuator networks, and a code deployment and runtime environment for Python scripts [3]. Our solution demonstrates the ability to use simple intuitive interfaces to configure and manage IoT-based WSNs.

The key contribution of this paper is a common framework for discovering the smart home devices and find available services. The tool includes an engine that analyses and translates the user-defined rules binding the smart home services into the distributed application code. To the best of our knowledge, this set of combined features is not available in previous smart home systems, and their integration significantly help making these systems more usable.

This paper is organized as follows. Section 2 describes the system overview and the architecture. Section 3 shows a prototype of the visual programming environment. Section 4 performs a preliminary evaluation of this interface. Section 5 overviews related work in programming WSNs. Finally, Section 6 presents our conclusions.

[*] This work was completed while the author was with University College Cork, Ireland.
[†] The author is now with MIT Media Lab, 75 Amherst St, Cambridge, MA 02139

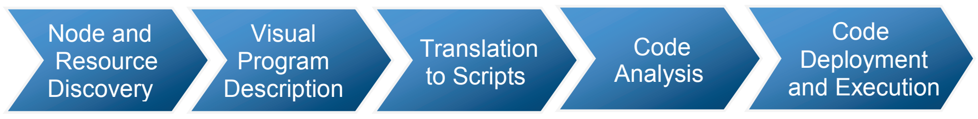
Node and Resource Discovery
Visual Program Description
Translation to Scripts
Code Analysis
Code Deployment and Execution

Fig. 1. System Workflow.

II. System Overview and Architecture

Smart home systems are still very cumbersome for the novice user, and require a significant installation effort. To address this problem, we have built a novel WSN system for the smart home that allows users to visually define the behaviour of the WSN, including automated support for device and resource discovery.

The workflow of the system is depicted in Fig. 1. First, node and resource discovery makes it possible to show the user the available functionalities, then the user defines a program visually in a graphical user interface. This user-defined program is then translated into a distributed application composed of several scripts, which are analysed with the purpose of optimizing resource usage. Finally, the system interacts with a WSN via an HTTP-CoAP proxy to deploy and execute the application. The system relies on a code (Python scripts) deployment solution developed in [3] (note that other similar solutions exist [4]), which makes use of the Constrained Application Protocol (CoAP) [5].

The visual user interface is based on Blockly [6], a web-based visual block programming environment developed by Google, where users manipulate and connect blocks that look like puzzle pieces to build their programs. Blockly is very extensible, allowing a person to define custom blocks and code generators. There are a few important advantages of Blockly when compared to predecessors. It executes in a web-browser, with no need for downloads or plugins. And it is open source, allowing for free use and modification. Blockly proved to be an attractive option as it is suitable for novice users to define small scripts that define the behaviour of a WSN. It allows for the definition of custom blocks (including for sensors/actuators), generates Python code out-of-the-box, and additionally allows for necessary customization of this generator.

It is important to note that each sensor/actuator block is a self-contained block that might encapsulate considerable complexity, as they can reference groups of nodes/resources which need to be discovered and eventually configured.

Blockly can generate code for JavaScript and Python. In our work we have leveraged the code generation features of Blockly to produce Python code compatible with the code deployment solution developed in [3]. Because the code generated by the block is tailored to fit a specific functionality, and a typical program will be composed of a few blocks, the resulting generated code avoids inefficiencies and redundancy usually associated with automatically generated code.

Finally, for execution and deployment, Blockly needs to interact with the constrained WSN using a CoAP-based interface of the runtime executing in each node.

The workspace presented to the user is dynamically constructed according to the nodes and resources discovered in the network. At load time, Blockly triggers a node and resource discovery that allows mapping the nodes available in the WSN with the blocks presented to the user. This mapping is done automatically, based on the CoAP resource descriptions of the nodes.

The overall architecture is depicted in Fig. 2, which shows example nodes on the WSN (the light, presence, temperature sensors) and depicts the proxy server, which translates HTTP requests from the Internet network to CoAP. The system comprises the following main elements:

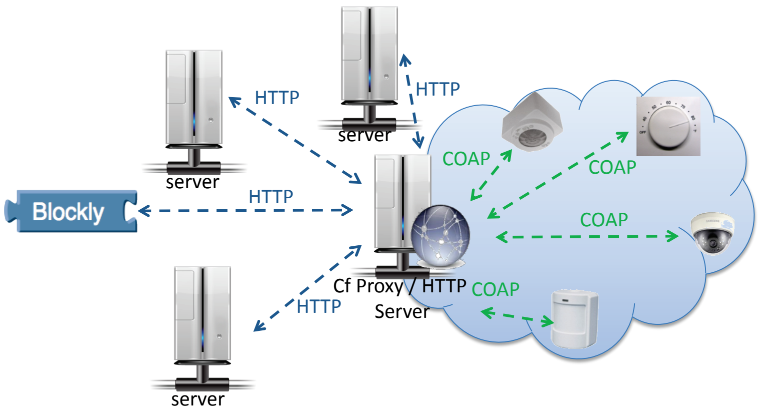

Fig. 2. Architecture Overview.

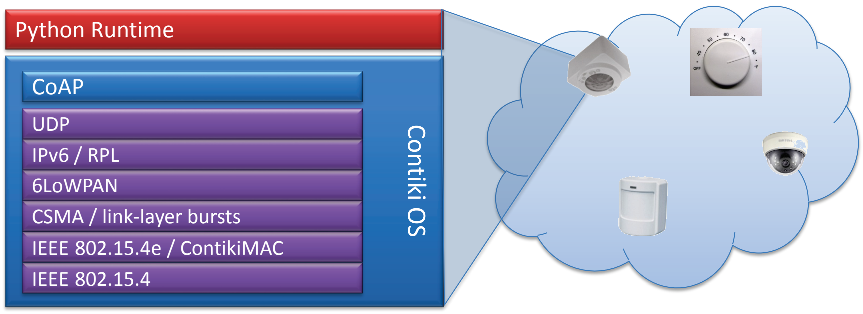

Fig. 3. WSN Node Embedded Protocol Stack.

- A WSN, composed of nodes running an embedded protocol stack as depicted in Fig. 3, which includes a runtime for Python scripts managed using a CoAP-based interface.
- A proxy server that translates HTTP requests to CoAP. We used the available implementation of proxy server Californium Proxy Server [7].
- A web application based on Blockly, which allows end users to specify programs using a visual editor.
- A web server that serves the web application and provides basic webservices (such as translation of scripts into the format used by runtime).

III. Prototype

We have built a prototype of the visual programming environment which presents a workspace that allows the user to visually compose programs using a set of pre-defined blocks. The prototype includes basic code generation features and is able to interact with the WSN.

The workspace presented to the user is dynamically constructed according to the nodes and resources discovered in the network. Fig. 4 presents a sample script in the user workspace that turns on the light depending on the value of luminance, concretely if the value of luminance sensor is less than 50.

An example of a simple alarm system to detect intrusions into the house is presented in Fig. 5. When the alarm is active (we have omitted the activation conditions for simplicity) some motion is detected or any door is open the alarm is triggered. The script shows that several actions could be performed, in this case when the alarm is triggered a LED red is set ON, the alarm siren sounds and an instant message is sent to the user (turning on the red LED). For advanced users, the activation/deactivation options can be shown by using the block modifier. This, for example, could allow the user to define that only a subset of the doors is checked at activation time.

IV. Evaluation

We have performed a preliminary evaluation of the interface using the well-established cognitive dimensions framework [8], with the results presented in Table 1. We refer the interested reader to [8] for definitions of the various dimensions, but in summary we conclude from the qualitative

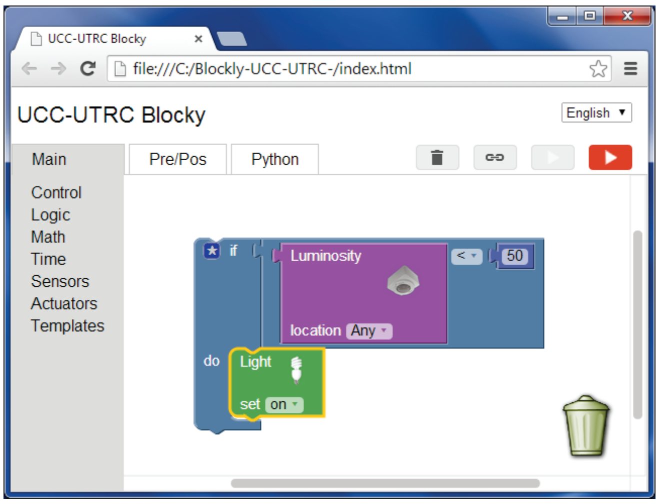

Fig. 4. Workspace: Sample Script.

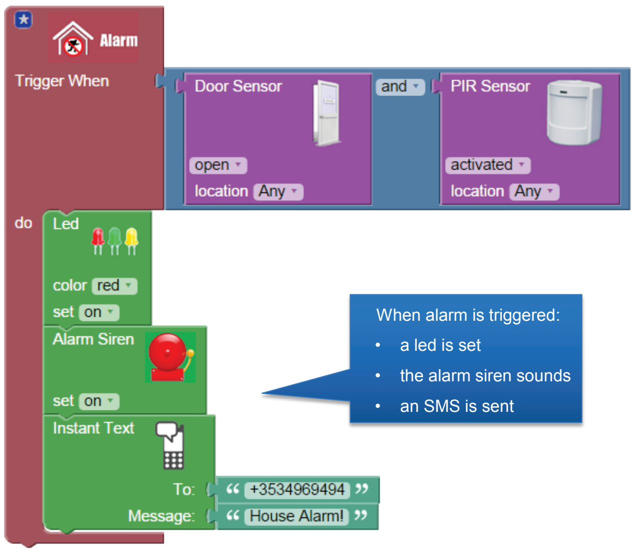

Fig. 5. Example Blockly Script.

Table 1. Comparative Table.

Cognitive Dimension	***Evaluation***
Abstraction Gradient	*High initial levels of abstraction, but new abstractions are not supported.*
Closeness of mapping	*Medium, not many 'programming games' need to be learned.*
Consistency	*High, when some of the language has been learnt, the rest can be inferred.*
Diffuseness	*Low, very few icons or graphic entities are required to express a meaning.*
Error-proneness	*Very low, the design of the notation does not induce 'careless mistakes'.*
Hard mental operations	*Low, the user does not need to resort to fingers or penciled annotation to keep track of what's happening.*
Hidden dependencies	*Some, not all the dependencies are fully visible due to the high initial level of abstraction.*
Premature commitment	*Low, initial design decisions do not limit the final result.*
Progressive evaluation	*No, the user cannot execute a partially-complete program to obtain feedback on 'How am I doing'.*
Role-expressiveness	*Easy to understand, the reader can see how each component of a program relates to the whole.*
Secondary notation	*No, the user cannot add comments.*
Viscosity	*Low, the user does not require much effort to perform a single change.*
Visibility	*Good, every part of the code is simultaneously visible.*

analysis that the proposed system satisfies many of the key facets of good interface design. In our case, the high level of abstraction produces hidden dependencies and for this reason the closeness of mapping is classified as being medium. However, the user does not need to reason with difficult mental operations. The program structure maps closely onto the problem structure by choosing the right abstractions. The diffuseness (the number of icons) is very low, for example, if we have one hundred doors in our system the interface only will show to the user one component.

We have conducted manual evaluation of the code generated by our tool, by comparing it with what would be produced by an expert developer, to find that both generally match very closely. As a future work, we would like to develop a more systematic evaluation of the code generated.

V. Related Work

Programming WSN has attracted the attention of many researchers that recognized long ago that their use in the real-world would only be possible if enabled by the existence of simple and easy to use programming support. Essentially, previous work in this area can be classified as: (i) node-level approaches, that attempt to provide a unified framework for developing applications by abstracting from network and operating-system details (such approaches include query-based frameworks [9] [10], or middleware approaches [11]), and (ii) macro scale approaches, which rely on higher level network abstractions such as logical neighbourhoods [12] or spatially-distributed data streams [13]. The general area of programming WSN is not the focus of our work, and the reader is referred to an extensive survey of programming approaches for WSN [14].

In this document, we are focusing on visual programming approaches for WSN and we will now summarize such approaches. We distinguished four main categories: (i) Work developed specifically for TinyOS, (ii) Dataflow-based and web-based approaches, (iii) one Business Process Modelling inspired approach and (iv) approaches that use puzzle like pieces to define programs. We will describe each one in the remainder of this section.

A. Visually Combining TinyOS modules

Many visual development environments were developed to provide an overview of the node-level program and construct programs by visually combining (wire) TinyOS components together [15][16]. Both VipTOS and RaPTEX target a scenario of application prototyping, where users will develop complex applications, and use these tools to improve the design and validate the application. This is different from our work, which enables the development and deployment of small applications, developed by non-experts. TOSDev and YETI [17][18] are essentially code editors with some (NesC/TinyOS specific) added functionality, and do not go a long way in facilitating the task of application development for WSN. Noticeably, these environment focus on node-level programming, and do not include any feature particularly directed for interactions between nodes.

B. Dataflow-based and Web-based Approaches

Dataflow is one appealing model that has been also been employed in visual programming of WSN. In such model, users specify diagrams of blocks that are graphical representations of functions. At runtime, a block executes once it receives all required inputs. When a block executes, it produces output data and passes the data to the next block in the dataflow path. The execution order is therefore determined by the flow of data through the block diagrams. Two different dataflow approaches have been identified. ClickScript [19] and LabVIEW WSN Module [20], and will be described next.

ClickScript is a Web application that allows to visually build scripts from the browser. The user specifies a block diagram made up of components (the blocks) and wires which connect the components. Each component represents a function (symbolized by the picture on the component) and has input and output sockets. The different functionalities of a component are symbolized by the picture on it [19][21]. Several extensions to connect ClickScript to WSN have been made, notably a recent work has used ClickScript to generate JavaScript scripts that would be executed on the WSN by Actinium [22], a RESTful (CoAP-based) runtime container that allows dynamic installation, update, and removal of JavaScript apps [23]. Because ClickScript is a web-based application and cannot communicate directly using CoAP, the authors have built a Web Socket/CoAP proxy to enables communication between ClickScript and Actinium.

ClickScript is an interesting approach that as shown to be effective as a visual programming interface for WSN. Nevertheless, while it facilitates programming, it is not targeting the novice user of smart home systems, as it presents a great number of features that are not application specific, which require significant computer knowledge by the user.

LabVIEW includes a Wireless sensor Module that allows using LabVIEW graphical programming interface to embed applications on WSN measurement nodes [20]. LabVIEW is a widely used tool, however it is proprietary, and it is not targeted for the novice user of smart home systems, as it does not target a specific domain, and requires the user to know about particularities of LabVIEW and of the WSN.

IFTTT [32] this web tool allows to connect different data sources and websites to automate tasks based on "if" conditions. This tool provides an interesting approach for novice users, but relies on web connectivity (our solution is based on web standards, but can be based on a local installation), and the programming model based on if conditions is quite restricted.

C. *Programming WSN using Business Process Modelling*

One large effort for facilitating programming of WSN and integrating them in business processes was carried out by the MakeSense Project [24]. MakeSense is a unified programming framework and a compilation chain that, from high-level business process specifications, generates code ready for deployment on WSN nodes. The architecture of MakeSense has three different layers: i) an application layer concerned with business processes and their modelling; ii) a macro programming layer concerned with the distributed execution of activities within the WSN; iii) a run-time layer concerned with the low-level aspects supporting the above and enabling self-optimization [24]. This is an interesting project that has many interesting ideas, some that could be of use for our work (e.g. the concept of including performance annotations, such as reliability even, or minimum lifetime), but is mainly targeting experts that can specify processes using the BPMN.

D. *Using Puzzle Pieces to Build Programs*

There are a number of visual programming tools based on

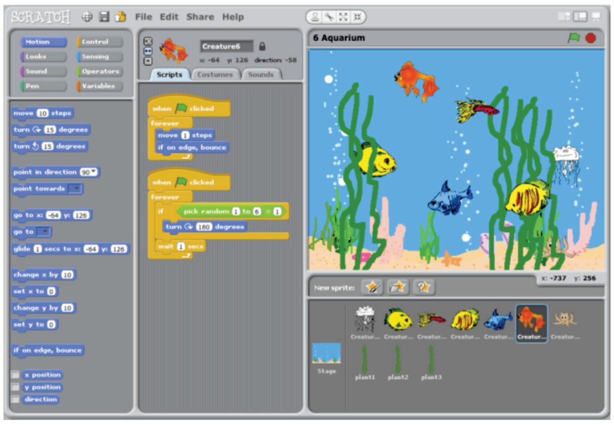

Fig. 6. Scratch [25].

the idea of using puzzle pieces to build programs. This idea was originally developed as an educational programming language that allows people of any age to experiment with programming by putting together blocks to control images, music, and sounds [25], and has been also applied in many other environments such as Scratch [26] presented in Fig. 6, OpenBlocks [27] (a graphical programming editor later adopted by Google for Android's App Inventor), TinyInventor [28], Blockly [6], and many other.

Because the shape of the blocks makes clearer how they can be connected together, the user does not have to learn the syntax of the language, as it is embedded in the shape of the pieces. The user is also presented will all the pieces he can interact with, avoid making him remember all the keywords of the language and objects he can interact with.

The work in [29], while somewhat old, presents a good structuring and taxonomy of languages and environments designed to make programming more accessible to novice programmers. The work also points out some future work in novice programming environments and languages, which relates to Alice [30], an open source programming language with educational purposes. Alice was unique in the sense that it was specially designed for learning purposes. The evolution of Alice is called Storytelling Alice [31]. Scratch [26] also has similar ideas, and is a visual programming environment that allows users to learn computer programming while working on personally meaningful projects such as animated stories and games. Scratch has a puzzle-based interface and makes programming very accessible for novice users. Scratch however is targeted to be a standalone application, and not embedded in a smart home system.

TinyInventor [28] is based on Open Blocks [27], a java library for creating puzzle-based programming interfaces, where users can drag blocks together to build an Android application. In TinyInventor, the programs are defined as a collection of functional blocks where some blocks are running on sensors (nesC generated code) and some on personal computers (Python generated code). TinyInventor unifies the development of mote and PC code by requiring common cross-platform programming abstractions (thread based execution model and IPv6 communication primitives), which facilitates application development. However, TinyInventor focuses on the specification of node behaviour and requires considerable effort to program a full usable distributed application. Note also that according to the online code repository, TinyInventor was not updated since October 2011.

Our systems makes use of Blockly [6], and, as put by the authors , it "was influenced by App Inventor, which in turn was influenced by Scratch, which in turn was influenced by StarLogo".

VI. CONCLUSIONS

We have built a prototype that allows users to visually define the behaviour of a WSN. The user defines a program visually in a Blockly-based user interface, and then this program is translated into a distributed application composed of several scripts, that are deployed and executed in the WSN. The prototype includes relevant system features: IoT-enabled sensor and actuator network, user-friendly programming interface, support for device and resource discovery and over-the-air code generation and deployment.

Our proposed system exhibits many interesting features, previously not integrated in a single solution: (i) no need for installation or downloads, as the application is web based, requiring no installation of software by the end user; (ii) avoid errors, because the puzzle pieces approach eliminates syntax errors, and frees the user from such concerns (iii) our system provides a low barrier to entry as the block shapes give visual feedback about how they can be combined (iv) the system offers domain abstractions, the visual programming interface includes blocks tailored for the target applications, and these include representations to deal with groups, location and state of sensors; (v) the programming interface has a small number of elements, but supports the specification of a broad range of behaviours; (vi) as a result of the small number of elements in each program, the generated code greatly avoids inefficiencies and redundancy usually associated with automatically generated code, (vii) the solution leverages the features existing IoT-related protocols to facilitate deployment and resource discovery.

ACKNOWLEDGMENT

This work was sponsored by UTRC Ireland, and was also partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness') within project FCOMP-01-0124-FEDER-028990 (PATTERN) and by FCT and the EU ARTEMIS JU funding, within project ARTEMIS/0004/2013, JU grant nr. 621353 (DEWI , www.dewi-project.eu).

REFERENCES

[1] Nest Thermostat, http://www.nest.com/. 2015.

[2] Tado Thermostat, http://www.tado.com/en/. 2015.

[3] S. Bocchino, S. Fedor, and M. Petracca, "PyFUNS: A Python Framework for Ubiquitous Networked Sensors," in Proceedings 12th European Conference, EWSN 2015., 2015, pp. 1–8.

[4] D. Alessandrelli, M. Petracca, and P. Pagano, "T-Res : enabling reconfigurable in-network processing in IoT-based WSNs," in IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS), 2013, pp. 337 – 344.

[5] Constrained Application Protocol (CoAP), https://datatracker.ietf.org/doc/draft-ietf-core-coap/. 2013.

[6] Blocky a visual programming editor, http://code.google.com/p/blockly/. 2013.

[7] Californium (Cf) CoAP framework in Java, http://people.inf.ethz.ch/mkovatsc/californium.php. 2013.

[8] T. R. G. Green and M. Petre, "Usability Analysis of Visual Programming Environments : A ' Cognitive Dimensions ' Framework," J. Vis. Lang. Comput., vol. 7, pp. 131–174, 1996.

[9] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: an Acquisitional Query Processing System for Sensor Networks," ACM Trans. Database Syst., vol. 30, no. 1, pp. 122–173.

[10] Y. Yao and J. Gehrke, "The Cougar Approach to In-Network Query Processing in Sensor Networks," SIGMOD Rec., vol. 31, no. 3, pp. 9–18.

[11] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A. L. Murphy, and G. Pietro Picco, "Mobile data collection in sensor networks: The TinyLime middleware," Pervasive Mob. Comput, vol. 1, no. 4 (December 2005), pp. 446–469, 2005.

[12] L. Mottola and G. Pietro Picco, "Programming wireless sensor networks with logical neighborhoods," in Proceedings of the first international conference on Integrated internet ad hoc and sensor networks (InterSense '06), 2006.

[13] R. Newton, G. Morrisett, and M. Welsh, "The Regiment Macroprogramming System," in International Conference on Information Processing in Sensor Networks (IPSN'07), 2007.

[14] L. Mottola and G. Pietro Picco, "Programming Wireless Sensor Networks : Fundamental Concepts and State of the Art," ACM Comput. Surv., vol. 43, no. 3, pp. 19:1–19:51, 2011.

[15] E. Cheong, E. A. Lee, and Y. Zhao, "Viptos: a graphical development and simulation environment for TinyOS-based wireless sensor networks," in SenSys '05, 2005.

[16] J. B. Lim, B. Jang, S. Yoon, M. L. Sichitiu, and A. G. Dean, "RaPTEX: Rapid prototyping tool for embedded communication systems," ACM Trans. Sens. Networks, vol. 7.

[17] W. P. McCartney and N. Sridhar, "TOSDev: a rapid development environment for TinyOS," in SenSys '06, 2006, pp. 387–388.

[18] N. Burri, R. Flury, S. Nellen, B. Sigg, P. Sommer, and R. Wattenhofer, "YETI: an Eclipse plug-in for TinyOS 2.1," in SenSys '09, 2009, pp. 295–296.

[19] D. Guinard, V. Trifa, F. Mattern, and E. Wilde, "From the Internet of Things to the Web of Things : Resource Oriented Architecture and Best Practices," in Architecting the Internet of Things, 2011, pp. 97–129.

[20] The LabVIEW Wireless Sensor Network Module - Under the Hood, http://www.ni.com/white-paper/9006/en/. 2013.

[21] ClickScript, http://clickscript.ch/ide/cs/util/tutorial/tutorialEN.html. 2013.

[22] M. Kovatsch, M. Lanter, and S. Duquennoy, "Actinium: A RESTful runtime container for scriptable Internet of Things applications," in 3rd IEEE International Conference on the Internet of Things (loT' 12), 2012, pp. 135–142.

[23] L. Mainetti, V. Mighali, L. Patrono, P. Rametta, and S. L. Oliva, "A novel architecture enabling the visual implementation of Web of Things applications," in 21st International Conference on Software, Telecommunications and Computer Networks (SoftCOM), 2013, pp. 1–7.

[24] F. Casati, F. Daniel, G. Dantchev, J. Eriksson, N. Finne, S. Karnouskos, P. Moreno, L. Mottola, F. J. Oppermann, G. Pietro Picco, A. Quartulli, K. Römer, P. Spieß, S. Tranquillini, and T. Voigt, "Demo Abstract: From Business Process Specifications to Sensor Network Deployments," in EWSN 12, Poster and Demo Proceedings, 12AD, pp. 25–26.

[25] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The Scratch Programming Language and Environment," Trans. Comput. Educ., vol. 10.

[26] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: Programming for All," Commun. ACM, vol. 52, no. 11, pp. 60–67, Nov. 2009.

[27] R. V. Roque., "OpenBlocks : an extendable framework for graphical block programming systems," Massachusetts Institute of Technology, 2007.

[28] M. T. Hansen and B. Kusy, "TinyInventor : A Holistic Approach to Sensor Network Application Development," in Extending the Internet to Low power and Lossy Networks. IPSN, 2011.

[29] R. Pausch and C. Kelleher, "Lowering the Barriers to Programming: A Survey of Programming Environments and Languages for Novice Programmers," 2003.

[30] C. Kelleher, "Alice: Using 3D Gaming Technology to Draw Students into Computer Science," in Game Design and Technology Workshop and Conference, 2006, pp. 16–20 (Invited paper).

[31] C. Kelleher, "Motivating Programming : using storytelling to make computer programming attractive to middle school girls School," 2006.