NDS Research Group - Networking and Distributed Systems
University of Palermo, ITALY
http://www.dicgim.unipa.it/networks

High-level Programming and Symbolic Reasoning on IoT Resource Constrained Devices

Salvatore Gaglio
salvatore.gaglio@unipa.it

Giuseppe Lo Re
giuseppe.lore@unipa.it

Gloria Martorella
gloria.martorella@unipa.it

Daniele Peri
daniele.peri@unipa.it

Abstract

While the vision of Internet of Things (IoT) is rather inspiring, its practical implementation remains challenging. Conventional programming approaches prove unsuitable to provide IoT resource constrained devices with the distributed processing capabilities required to implement intelligent, autonomic, and self-organizing behaviors. In our previous work, we had already proposed an alternative programming methodology for such systems that is characterized by high-level programming and symbolic expressions evaluation, and developed a lightweight middleware to support it. Our approach allows for interactive programming of deployed nodes, and it is based on the simple but effective paradigm of executable code exchange among nodes. In this paper, we show how our methodology can be used to provide IoT resource constrained devices with reasoning abilities by implementing a Fuzzy Logic symbolic extension on deployed nodes at runtime.

1 Introduction

According to the Internet of Thing (IoT) vision [1], all kinds of devices, although computationally limited, might be used to interact with people or to manage information concerning the individuals themselves [2]. Besides reactive responses on input changes, the whole network may exhibit more advanced behaviors resulting from reasoning processes carried out on the individual nodes or emerging from local interactions. However, nodes' constraints leave the system designers many challenges to face, especially when distributed applications are considered [3]. Conventional programming methodologies often prove inappropriate on resource constrained IoT devices, especially when knowledge must be treated with a high level representation or changes of the application goals may be required after the network has been deployed [4]. Moreover, the implementation of intelligent mechanisms, as well as symbolic reasoning, through rigid layered architectures, reveals impracticable on resource constrained devices such as those commonly used in Wireless Sensor Networks (WSNs). Often this issue is faced through the adoption of an intelligent centralized system that uses WSNs as static sensory tools [5]. Indeed, integration of WSN devices in the IoT seems quite natural and desirable, provided that the aforementioned issues be addressed. In our previous work [6, 7], we introduced an alternative programming methodology, along with a lightweight middleware, based on high-level programming and executable code exchange among WSN nodes. The contribution of this paper consists in the extension of the methodology to include symbolic reasoning even on IoT resource constrained devices. The remainder of the paper is organized as follows. In Section 2 we describe the key concepts of our methodology and the symbolic model we adopted. In Section 3, we extend the symbolic approach characterizing our programming environment with Fuzzy Logic, and in Section 4 we show an application to make the nodes reason about their position with respect to thermal zones of the deployment area. Finally, Section 5 reports our conclusions.

2 Key Concepts of the Development Methodology

Mainstream praxis to program embedded devices consists in cross-compilation of specialized application code together with a general purpose operating system. The resulting object code is then uploaded to the on-board permanent storage. Instead, our methodology is based on high-level executable code exchange between nodes. This mechanism, while abstracted, is implemented at a very low level avoiding the burden of a complex and thick software layer between the hardware and the application code. Indeed, a Forth environment runs on the hardware providing the core functionalities of an operating system, including a command line interface (CLI). This also allows for interactive development, which is a peculiar feature of our methodology that can be used even to reprogram deployed nodes. This way, nodes can be made expand their capabilities by exchanging pieces of code among themselves in realtime. The CLI is accessible through either a microcontroller's UART or the on-board radio [6]. The Forth environment is inherently provided with an interpreter and a compiler. Both can be easily extended by defining new *words* stored in the *dictionary*. Being Forth a stack-based language, words use the stack for parameters passing. A command, or an entire program, is thus just a sequence words. The description of a task in natural language and its implementation can be thus made very similar. Our programming environment is composed of some nodes wirelessly deployed and a wired node that behaves as a bridge to send user inputs to the network. In previous work [7], we introduced the syntactic construct that implements executable code exchange among nodes:

`tell:` ⟨*code*⟩ `:tell`

in which ⟨*code*⟩ is a sequence of words, sent as character strings, to be remotely interpreted by the receiver node. The address of the destination node is left on the top of the stack. A numeric as well as a string value, can be taken at runtime from the top of the stack and inserted in the outgoing packet when special markers, such as ∼ for numbers and ∼`s` for strings are encountered.

3 Distributed Processing and Symbolic Reasoning

In our programming environment, purely reactive behaviors can be easily implemented on the remote nodes by sending them the sequence of words to be executed if certain conditions are met. Let us consider the following command given through the CLI of the bridge node:

```
bcst tell: close-to-window? [if] red led on [then] :tell ;
```

This command broadcasts (`bcst` is the reserved address for the purpose) the code between the `tell:` and `:tell` words. Once received, each node executes the word `close-to-window?` to evaluate if it is close to the window and, if so turns the red LED on. The word `close-to-window?`, already in the dictionary, performs temperature and luminosity measurements and checks if both sensory readings are above a predefined threshold. As it can be noticed, the code is quite understandable, although all the words operate just above the hardware level by setting ports or enabling the ADCs to read temperature and light exposure. This code, as well as those in the rest of the paper, has been used on Iris Mote nodes equipped with the MTS400 sensor board to acquire data about temperature and light exposure. For the sake of showing how it is possible to incorporate in our middleware new abstractions to support intelligent applications here we introduce a Fuzzy Logic extension. Fuzzy Logic has the peculiarity to be appropriate to implement approximate reasoning in several contexts as well as for machine learning purposes [8]. We adopted a classic Forth Fuzzy Logic implementation [9] that we enriched with the possibility to exchange definitions and evaluation among nodes. Moving on with the above example, in place of two crisp variables, the fuzzy variables `temp` and `lightexp` can be easily defined on deployed nodes, using the word `fvar` to define the related membership functions (Figure 1) placed between `tell:` and `:tell`. A node can be made measure light exposure, and fuzzify it with the code:

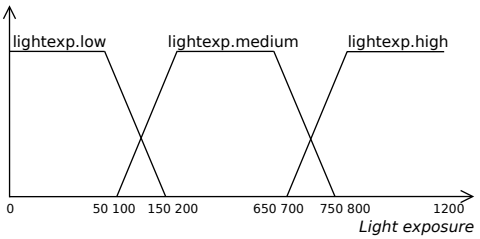
```
lightexp measure apply
```

while the code:

```
lightexp.low @
```

pushes onto the stack the truth value by using the built-in word `@` (*fetch*). Rather than through a thresholding process, a device can establish if it is close to the window through the evaluation of fuzzy rules in the form:

```
temp.high @ lightexp.high @ & => close-to-window
```




```
0 1200 fvar lightexp
0 0 100 150 member lightexp.low
100 200 650 750 member lightexp.medium
700 800 1200 1200 member lightexp.high
```

Figure 1: Fuzzy set associated with the fuzzy variable `lightexp`. On the right side, the code to define the fuzzy variable `lightexp` and its membership functions. The definition domain, corresponding to the raw readings values interval [0,1200], is given before the word `fvar`, while the word `member` defines each of the three trapezoidal membership functions by using four control points (bottom-left, top-left, top-right, and bottom-right).

where `temp.high` and `lightexp.high` are membership functions of the fuzzy input variable `temp` and `lightexp` respectively, and `close-to-window` is one of the linguistic labels associated to the output variable. Similarly to the case of the thresholding process, if both the temperature and the light exposure levels are high a node can infer to be under sunlight, and thus close to the window.

4 Inferring Nodes' Distribution according to Thermal Zones

Let us suppose we intend to make the deployed nodes able to discover their distribution with respect to thermal zones of an environment lighted by some windows exposed to direct sunlight, and lamps. Each node assesses in turn the thermal zone it belongs to, and makes the others aware of this information. We defined the syntactic construct `classification` to make the nodes able to classify according to an arbitrary number of fuzzy variables. With the previously defined input variables `temp` and `lightexp` the code:

```
temp lightexp 2 classification thermal-zone
```

creates the new word `thermal-zone`, which is bound to the two fuzzy variables `temp` and `lightexp`. The new word `thermal-zone` measures the temperature and luminosity, fuzzifies the crisp inputs and evaluates the rules by storing the firing strength for each rule, indicating the degree to which the rule matches the inputs. The rule generation process considers all the possible combinations of all the membership functions, -i.e. in this case, the set of all ordered pairs `(a,b)` where `a` and `b` are linguistic terms associated respectively with `temp` and `lightexp`. When handling few variables, this does not cause excessive memory occupation. It offers instead the advantage of considering a fine-grained classification based on all the n-tuples, that in this case, are all valid. However, optimization methods for the reduction of a large scale rule base may be required in real-time fuzzy systems [10, 11, 12]. When needed, the table is traversed to compute the membership grade of the output by aggregating all rules. The rule with the maximum strength is taken as the output membership class. This way, each node is able to classify itself into one of the thermal zones. To support more sophisticated behaviors, it is possible to exploit the mechanism of code exchange among nodes to trigger the process of neighbor discovery in order to keep track of their classification into thermal-zones. For this purpose, it is necessary to define the table `nodes-distribution` to contain the number of nodes for each thermal zone. For instance, each device can transmit once, after waiting (word `on-timer`) for a time that is function of its unique ID. When its time is elapsed, the word `classification-spread` is executed, the node classifies itself into a thermal zone and then broadcasts the class it belongs to, together with the code to make the others update the whole distribution. The Forth code required for the entire process is the following:

```
: local-update nodes-distribution update ;
: spread dup local-update bcst [tell:] ~ local-update [:tell] ;
: classification-spread thermal-zone spread ;
 on-timer ' classification-spread
```

in which the word `spread` creates a message with the code to make the other devices update locally the `nodes-distribution`. At the end of the `update` process, each node holds the current nodes distribution in terms of thermal zones, as such:

```
Class 1 2 3 4 5 6 7 8 9
    # 5 1 0 0 0 0 0 1 1
```

Five nodes belongs to class 1, one node to class 2 and so on. Each node knows the number of nodes in the network and their position, without any centralized computation. Once some nodes are moved from their position to another, and the process is triggered again, each node is able to detect the new distribution. Moreover, the analysis of the nodes distribution may lead a node to classify itself as an outlier, to trigger self-diagnosis operations, and even to take specific actions, by reasoning about the whole network configuration and its membership thermal zone. The interactivity granted by our approach permits the programmer to communicate with the network through the serial shell of the bridge node. For instance, the programmer can tell the nodes belonging to class 8 to turn their red LED on:

```
bcst tell: thermal-zone 8 class? [if] red led on [then] :tell ;
```

5 Conclusions

In this paper, we showed how distributed symbolic reasoning can be implemented on resource constrained IoT devices by exploiting executable code exchange. Our contribution aims to fill the lack in the absence of programming paradigms enabling a vast adoption of IoT in everyday life. The possibility to exchange executable code makes the system adaptive and autonomous, since each node can evolve on the basis of realtime inputs, in terms of both data and executable code, from other nodes and from the user. We showed how abstractions and symbolic expression evaluation can be efficiently incorporated into a programming model for such networks by exploiting both interpretation and compilation of code. As an example, we described the syntactic constructs that can be defined to make the nodes aware of their position with respect to a subdivision of the environment into thermal zones. Our methodology reveals suitable for implementing more advanced behaviors on IoT devices since symbolic reasoning is performed even on inexpensive and resource constrained microcontrollers.

References

[1] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A Survey . *Computer Networks*, 54(15):2787 – 2805, 2010.

[2] Bin Guo, Daqing Zhang, Zhiwen Yu, Yunji Liang, Zhu Wang, and Xingshe Zhou. From the Internet of Things to Embedded Intelligence. *World Wide Web*, 16(4):399–420, 2013.

[3] Gloria Martorella, Daniele Peri, and Elena Toscano. Hardware and Software Platforms for Distributed Computing on Resource Constrained Devices. In Salvatore Gaglio and Giuseppe Lo Re, editors, *Advances onto the Internet of Things*, volume 260 of *Advances in Intelligent Systems and Computing*, pages 121–133. Springer International Publishing, 2014.

[4] G. Kortuem, F. Kawsar, D. Fitton, and V. Sundramoorthy. Smart Objects as Building Blocks for the Internet of Things. *Internet Computing, IEEE*, 14(1):44–51, Jan 2010.

[5] Alessandra De Paola, Marco Ortolani, Giuseppe Lo Re, Giuseppe Anastasi, and Sajal K. Das. Intelligent Management Systems for Energy Efficiency in Buildings: A Survey. *ACM Comput. Surv.*, 47(1):13:1–13:38, June 2014.

[6] Salvatore Gaglio, Giuseppe Lo Re, Gloria Martorella, and Daniele Peri. A Fast and Interactive Approach to Application Development on Wireless Sensor and Actuator Networks. In *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, pages 1–8, Sept 2014.

[7] Salvatore Gaglio, Giuseppe Lo Re, Gloria Martorella, and Daniele Peri. A Lightweight Middleware Platform for Distributed Computing on Wireless Sensor Networks. *Procedia Computer Science*, 32(0):908 – 913, 2014. The 5th International Conference on Ambient Systems, Networks and Technologies (ANT-2014), the 4th International Conference on Sustainable Energy Information Technology (SEIT-2014).

[8] M Navara and D Peri. Automatic Generation of Fuzzy Rules and its Applications in Medical Diagnosis. In *Proc. 10th Int. Conf. Information Processing and Management of Uncertainty, Perugia, Italy*, volume 1, pages 657–663, 2004.

[9] R. VanNorman. Fuzzy Forth. *Forth Dimensions*, 18:6–13, March 1997.

[10] Alessandra De Paola, Giuseppe Lo Re, and Antonio Pellegrino. A Fuzzy Adaptive Controller for an Ambient Intelligence Scenario. In Salvatore Gaglio and Giuseppe Lo Re, editors, *Advances onto the Internet of Things*, volume 260 of *Advances in Intelligent Systems and Computing*, pages 47–59. Springer International Publishing, 2014.

[11] Yaochu Jin. Fuzzy Modeling of High-dimensional Systems: Complexity Reduction and Interpretability Improvement. *Fuzzy Systems, IEEE Transactions on*, 8(2):212–221, Apr 2000.

[12] Yeung Yam, P. Baranyi, and Chi-Tin Yang. Reduction of Fuzzy Rule Base via Singular Value Decomposition. *Fuzzy Systems, IEEE Transactions on*, 7(2):120–132, Apr 1999.