Closing the Sensing-Reasoning-Actuating Loop in Resource-constrained WSANs through Distributed Symbolic Processing

S. Gaglio, G. Lo Re, G. Martorella, D. Peri, S.D. Vassallo

In Proceedings of the 20th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'15)

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

Closing the Sensing-Reasoning-Actuating Loop in Resource-constrained WSANs through Distributed Symbolic Processing

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
Salvatore Davide Vassallo

Abstract

Many issues in creating complex applications for pervasive environments are primarily due to the effort required to integrate perception, reasoning and actuating tasks in an efficient and homogeneous way, especially when the underlying infrastructure consists of wirelessly networked embedded devices. To mitigate the complexity of the actual implementation, satisfactory programming paradigms supporting the integration and coordination among heterogeneous devices are required. In this paper we show how a distributed symbolic processing approach that is particularly suited for resource constrained devices, such as the nodes of a Wireless Sensor and Actuator Network (WSAN), may be apt to the purpose. We also discuss a case study in which sensors and actuators, without any centralized control, act on the environment according to the thermal preferences that are continuously learned and monitored.

1 Introduction

The opportunity of easy deploying sensor and actuator devices permits to advance the ordinary notion of the environment as a place where people just live and act on. Endowing the environment with advanced skills - e.g. decision making and actuation - has attracted considerable interest in several application fields ranging from comfort improvement in office rooms [1], homes [2], to energy consumption saving [3] and industrial scenarios [4]. Each of these application areas share the same challenges due to a heterogeneous physical infrastructure in which devices differ in terms of available resources and computational capabilities, and in terms of the internal representation of the domain data.

In this paper we draft a methodology to close the sensing-reasoning-actuating loop avoiding to rely on any third entity and without decoupling required sophisticated skills from the sensing and actuating infrastructure. In our approach, applications perform distributed processing and symbolic computations on resource-constrained sensor nodes. Although applications can be coded in a very high-level way, they are executed with minimal abstraction just above the hardware, leading to compact and effective implementations. The remainder of the paper is organized as follows. In section 2 we briefly describe the distinctive features of our approach and development methodology, abstracting the computational schemes used to implement cooperative behaviors. To show a complete example that incorporates sensing, decision making and actuation, in Section 3 we describe an application to control autonomously the heating, ventilation and air conditioning system (HVAC) in an office room according to the user thermal preferences. Experimental results are shown in Section 4, while Section 5 concludes the paper.

2 A Distributed Symbolic Processing Approach

The biggest challenge, in creating complex applications for pervasive environments, is the difficulty of developing sensing, actuation and reasoning processes over a heterogeneous infrastructure. Most standards lack support for actuators, especially in the industrial sector, where, next to a Wireless Sensor Network (WSN), another wired infrastructure for actuators is often present [5]. Applications are often carried out in a centralized fashion, and designed according to layered architectural models [6] in which the system intelligence progressively decreases from top toward low levels [1].

Differently from other approaches, in which intelligence and complexity proceeds from the bottom upwards, we propose a horizontal architecture in which sensing and actuation, as well as decision making, are locally implemented at the same level of abstraction. This approach is embodied in our platform for symbolic distributed computing that targets resource constrained devices, also exploiting intetractivity in order to speed up the development of distributed applications [7]. The environment relies on a stack-based language interactive interpreter and compiler implementing the Forth methodology according to which applications are coded as sequences of symbols, the so-called *words*. To introduce dynamic computation into the network [8] as well as cooperative schemes, our platform provides the abstraction to permit the transmission of executable code among nodes [9]. This mechanism relies on the words `tell:` and `:tell` to create and send messages composed of 802.15.4-2003 frames. The executable code is enclosed among these two words as such: `tell:` <code> `:tell`, or `[tell:]` <code> `[:tell]` when the construct is inside a word definition. The message payload is the sequence of words that must be interpreted by the remote destination node. In order to support a broad range of applications, we have also shown how simple words can be defined to sense and act on the environment accordingly [7]. Our symbolic programs can be made quite compact, and therefore also reasoning, which may be a complex task, can be fitted in reduced memory. Indeed, symbolic abstractions –e.g. fuzzy logic [10]– can be incorporated in the platform when necessary.

As an example, Listing 1 shows the interaction model that can be adopted to exchange and average time series of temperature samples in order to obtain an averaged temperature trend.

Listing 1: Words defined on sensor nodes to implement a distributed computation scheme for the average temperature trend.

```
1  : start-averaging
2     proportional-timer
3     on-timer ['] send-samples once fire ;
4
5  : sample&average
6     samples available? if
7        start-averaging
8     else
9        temperature sample
10    then ;
```

The word `sample&average` is responsible of the distributed process running on the sensor nodes. It starts by checking whether all the `samples` are available (line 6). If not, the nodes `sample` the `temperature` (line 9). Otherwise, the collaborative process to compute the array of average samples is triggered. When a node executes `start-averaging`, in order to avoid frame collision, it waits for a time that is proportional to its unique ID (its MAC address). An interrupt is generated when the timer expires, triggering the execution of the word `send-samples`. The syntactic construct `on-timer...fire` sets the timeout value, and the word that is to be executed when the timer expires (line 3). The word `send-samples` broadcasts the content of `samples` and the code to make each node average the samples it has received together with those previously acquired. A few important and standard built-in words are used in the code: `:` (colon) switches the interpreter to the compiling mode thus syntactically beginning a new definition, `;` ends the compilation, `[']` leaves on the stack the "execution token" of the subsequent word which, alike to a function pointer in C derived languages, can be later used for "vectored execution". The words `@` and `!` are used respectively to fetch a value from the address on top of the stack, and to store a value to the address atop the stack.

3 Case Study: A Pervasive Application to Control the HVAC System

To provide a complete example comprising sensing, decision making and actuation, we show a WSAN application to control autonomously the HVAC system in an office room according to the thermal preferences of occupants. The

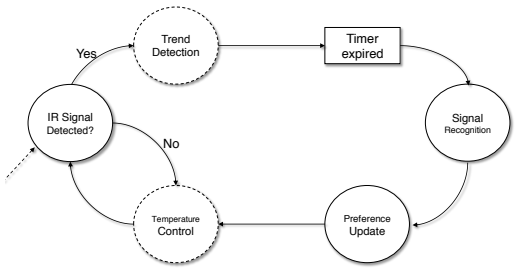

Figure 1: An overview of the whole application. Events are represented by boxes, circles represent distributed application phases while dashed circles indicate application phases characterized by cooperative schemes. The whole application logic is distributed among the IR receiver, the IR emitter and the sensing units.

network actuates the HVAC exclusively by infrared (IR) commands mimicking the remote control provided to the users. At the beginning, the network does not have any *a priori* knowledge about the structure and the meaning of the IR commands. A distributed event detection process is instead used to associate events to remote control IR codes, while updating the user thermal preferences.

An overview of the application is shown in Figure 1. The control logic is distributed across multiple entities: an IR receiver, an IR emitter and some heterogeneous cooperating devices that perform in-network tasks –e.g. samples aggregation– as if they were a single entity. Precisely, once an IR signal is detected, the IR receiver node stores the current temperature and broadcasts the code to undertake the assessment of the average room temperature trend, as described in Section 2. The actuator node asks the network for the trend. If an event is detected, the actuator asks the IR receiver for the last received signal, and associates it to the correct trend. Then it updates the preferred thermal conditions of the occupants and enters the temperature control phase, invoking the distributed computation scheme to make all the nodes compute a shared average room temperature value. Decision making about the actuation is undertaken on the basis of the current average temperature and the current user preferences. If any IR signal is detected, the system re-enters the temperature control phase, otherwise user preferences are updated.

Listing 2: Words implementing signal detection on the IR receiver node.

```
1  : ir.isr
2          signal max-length? if
3          temperature!
4          bcst trigger-sampling
5          else
6          signal add-pulse
7          then ;
8
9  : +ir
10         INT7 any-change ['] ir.isr
11         attach-interrupt
12         1micro resolution
13         timer0 start-timer ;
14
15 : -ir
16         INT7 detach-interrupt
17         timer0 stop-timer   ;
```

As showed in Listing 2, once a signal pulse is captured by the IR receiver, an interrupt on the INT7 pin is generated and the interrupt handler is executed. Despite Listing 2 deals with low level details concerning the sensor interface and the microcontroller timers, definitions are rather high level and descriptive enough to be on par with the pseudocode often found on technical documents.

Changes in terms of temperature trend must be detected to understand the input IR signal. The basic idea is that, under natural conditions, the room temperature can be considered uniformly distributed, especially whether the sensors are not directly exposed to sunlight, e.g. close to the window. Although the instant temperature detected by a sensor may differ from that of a closer sensor, in absence of external sources, the temperature trend detected is the same on all nodes and varies slowly. Differently, when artificial cooling or heating is on, the temperature variation does not follow a natural trend. Exploiting this idea, the nodes closer to the HVAC are able to detect temperature

variations that deviate by the natural trend, interpreting them as caused by user input. For simplicity, our system recognizes just three events that can be associated to three commands: “temp-up”, “temp-down” , “turn-off”. The first one, is associated with an increasing averaged temperature trend, the second one indicates a descending trend, while the latter refers to an increasing or decreasing natural trend that varies very slowly. Thus, when the nodes cooperatively detect an increasing trend, the last acquired signal is recognized as a command to switch the HVAC to heating mode. Alternatively, a decreasing average trend is associated with a cooling mode command. If any significant changes are detected because the time window is too small to observe natural variations in temperature, then the signal is associated with an off command. Unfortunately, it is prohibitive to find a generally valid formula to determine exactly such a time interval. Many external factors, in fact, must be taken into account, e.g. the size of the room, the number of air conditioners, the room light exposure, and the number of occupants. The strategical placement of some sensors close to the HVAC system is thus useful as these nodes are the first to notice temperature variations, in a ways shorter term than those deployed further away. How short this time must be, varies from case to case and it is for this reason that supporting prototyping in a real scenario is crucial. Code exchange allows to interact with the deployed network to adjust significant parameters depending on the features of the actual installation. For instance, to tell the nodes to acquires samples every 5 minutes for 25 minutes in realtime, the bridge node may be made execute:

```
bcst tell:  5 minutes sampling-interval !  25 minutes sampling-timeout !  :tell
```

to set the sampling interval and the sampling timeout on the already deployed nodes. User interactions with the HVAC system through the remote control, implicitly indicate undesired thermal conditions and are considered as a corrective and adaptive mechanism of the preferences currently stored in the system. Although the temperature range that identifies the user thermal comfort can be provided manually, such a strategy makes the system static and prone to errors if the supplied range no longer satisfies the user preferences. The adoption of an offline training algorithm to store the interactions with the system and to extract useful information such as the minimum and maximum temperature setpoints is prohibitive due to the device constraints. Therefore the actuator node only stores two temperatures, the maximum and the minimum setpoints. Whenever the actuator associates the heating command to the event, it asks the IR receiver for the temperature at the time of user intervention. Then, the latter is used to compute the average with the minimum setpoint value and to update it. The temperature values within the minimum and maximum setpoint range represent the user thermal preferred conditions.

Listing 3: Code to associate a signal to the trend of the averaged samples and to update the user thermal preferences accordingly.

```
1 signal avg-samples trend associate
2 preferences revise
```

Listing 3 shows the code to associate IR codes to temperature trends and to adjust the temperature setpoint taking into account feedbacks from users. The code is quite self-explanatory. It `associate` the last received IR `signal` with the `trend` computed on the averaged samples `avg-samples`. Afterwards, user thermal `preferences` –i.e. the maximum and minimum setpoints– are `revised` and corrected. To close the loop, the system must be capable to autonomously anticipate the actions of the room occupants. Regularly, the actuator invokes the cooperative process to make all nodes calculate the instant room temperature, spatially averaged. As shown in Listing 4, the actuator can then request the temperature to any `neighbor` node (line 13).

Listing 4: Words defined on the actuator node to control the HVAC system.

```
1  : actuate
2     colder? if
3        HVAC-temp-up
4     else
5     warmer? if
6        HVAC-temp-down
7     else
8        HVAC-turn-off
9     then then IR-send ;
10
11 : hvac-action
12    IR-receiver disable
13    neighbor ask room-temperature
```

```
14    actuate
15    IR-receiver enable ;
```

A rule system runs on the node equipped with the IR emitter: if the current average temperature exceeds the maximum temperature setpoint (line 5) the actuator sends the command to decrease the temperature (line 6) –i.e. the last one that it has stored and that has caused temperature decrease. Otherwise, if the room average temperature is lower than the minimum temperature setpoint (line 2), then the node sends the IR command to increase the temperature (line 6). Before performing such actions, the IR emitter node sends to the IR receiver node the code to disable the interrupt routine handling the IR signal reception (line 12) and enables it again after actuating on the environment (line 15). This strategy prevents the system to recognize signals it already holds and permits to undertake the trend detection, signal recognition and preference update phases, as illustrated in Figure 1, only as a result of user interventions.

4 Experimental Results

We carried out our experiments in an indoor environment with four Memsic Iris nodes running our platform. Iris motes are equipped with an Atmega1281 RISC microcontroller running at about 8 MHz, a 128 KB Program Flash memory, a 8 KB RAM, a 4 KB EEPROM, and a 512 KB Measurement Flash. Two of these deployed motes, are equipped with the MTS310 sensor board, and they can monitor room temperature acting as sensing devices. The actuator node is fitted with an IR emitter and is able to change the environmental conditions by acting on the HVAC system as if it were its IR remote control. In addition, another node is equipped with an IR receiver and conveniently installed under the air conditioning system so as to detect user interventions by intercepting signals from the IR remote control.

Figure 2 shows the data collected in the office when the HVAC system is off. The temperature measurements are cooperatively averaged after being acquired by the sensors. The nodes sample and average the temperature every 10 minutes. As it can be noticed, the temperature follows a natural trend with slow transitions and tends to increase during the warmest hours of the day. Figure 3 shows the pattern observed in the room temperature trend when the system is running.

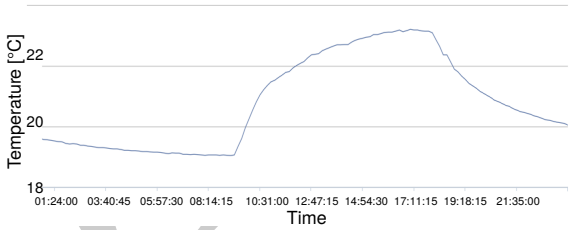

Figure 2: Data acquired in the real scenario when the HVAC system is off. The temperature follows a natural trend with any significant peak during the day.

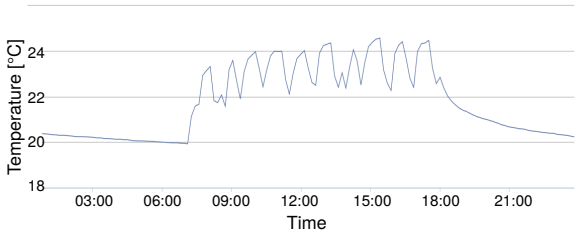

Figure 3: Data acquired in the real scenario when the system is working from 7:00 am to 18:00 pm. The temperature fluctuates around 22 Celsius degrees, which is the temperature preferred by the user. The data are sampled every ten minutes.

5 Conclusions

Current research trends highlight the need of alternative programming environments to simplify the development of a broad range of applications on wirelessly connected embedded devices. In this paper we showed how a distributed symbolic processing approach can lead to WSAN applications that are able to share and process symbolic knowledge about the environment even on resource-constrained nodes. We also discussed a case study including sensing, decision making and actuating tasks in a real scenario with a WSAN that is able to learn user preferences and act on the environment controlling the HVAC system of an office room by mimicking the HVAC IR remote control. IR signals are not known *a priori* to the network but are learned and associated to actions of users by a distributed event detection process running distributely on the network nodes, without resorting to a powerful centralized entity. Further investigations will enable the autonomic calculation of those configuration parameters that depend on the specific environment in which the system is installed. The instantiation of other distributed computing schemes will be also investigated, as well as the inclusion of self-diagnosis capabilities, at both node and system level.

References

[1] A. De Paola, S. Gaglio, G. Lo Re, and M. Ortolani. Sensor9k: A Testbed for Designing and Experimenting with WSN-based Ambient Intelligence Applications. *Pervasive and Mobile Computing*, 8(3):448 – 466, 2012.

[2] M. Usman, V. Muthukkumarasamy, Xin-Wen Wu, and S. Khanum. Wireless Smart Home Sensor Networks: Mobile Agent Based Anomaly Detection. In *Ubiquitous Intelligence Computing and 9th International Conference on Autonomic Trusted Computing (UIC/ATC), 2012 9th International Conference on*, pages 322–329, Sept 2012.

[3] A. De Paola, M. Ortolani, G. Lo Re, G. Anastasi, and S.K. Das. Intelligent Management Systems for Energy Efficiency in Buildings: A Survey. *ACM Computing Surveys*, 47(1):38, 2014.

[4] J. Neuzil, O. Kreibich, and R. Smid. A Distributed Fault Detection System Based on IWSN for Machine Condition Monitoring. *Industrial Informatics, IEEE Transactions on*, 10(2):1118–1123, May 2014.

[5] J. Akerberg, M. Gidlund, and M. Bjorkman. Future research challenges in wireless sensor and actuator networks targeting industrial automation. In *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*, pages 410–415, July 2011.

[6] C. Gomez-Otero, R. Martinez, and J. Caffarel. ClimApp: A Novel Approach of an Intelligent HVAC Control System. In *Information Systems and Technologies (CISTI), 2012 7th Iberian Conference on*, pages 1–6, June 2012.

[7] 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.

[8] Philip Levis, David Gay, and David Culler. Active Sensor Networks. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 343–356, Berkeley, CA, USA, 2005. USENIX Association.

[9] 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).

[10] Salvatore Gaglio, Giuseppe Lo Re, Gloria Martorella, and Daniele Peri. High-level Programming and Symbolic Reasoning on IoT Resource Constrained Devices. In *Accepted at The First International Conference on Cognitive Internet of Things (COIOTE 2014)*, October 2014.