Homogeneous Wireless Sensor Network Programming using MuFFIN

Rui Pires, Francisco Martins and Dulce Domingos
Universidade de Lisboa, Faculdade Ciências & Lasige, Lisbon, Portugal

Keywords: Sensor Networks, Middleware, Virtual Machines, Remote Sensor Reprogramming.

Abstract: Web services have been used as an homogeneous interface between client applications and sensor networks. The MuFFIN middleware makes possible for a client application to remotely (re)program sensor networks. However, this (re)programming is dependent on the characteristics of the hardware or of the programming languages provided by manufacturers. To generalize this feature, we propose a middleware extension to include the execution of code on behalf of sensor devices in case they are not (re)programmable. As a proof of concept, we use the MuFFIN middleware and the Callas sensor programming language together with its virtual machine. Additionally, we extend the MuFFIN with a component that supports the communication between two wireless sensor networks. This way, messages can flow from one network to another one without the intervention of the client application, reducing the number of messages exchanged between sensor networks and client applications.

1 INTRODUCTION

Sensor networks consist of a set of interconnected devices that can gather information about the environment that surrounds them. Due to its sensing capabilities, these networks are widely used in various areas such as security, defense, traffic management, and process automation.

Given the wide heterogeneity of sensors available in the market, Web services have been used to provide a uniform interface to interact with sensors. These Web services may be available in the sensors themselves or through middleware systems (Zeng et al., 2011). MuFFIN is an example of this kind of middleware, which also provides a Web service for remote sensor (re)programming, when this functionality is supported either by the sensor's hardware or via a software layer added to the sensor devices (Valente, 2011).

In order to make available the reprogramming capability for all kinds of sensor devices, we propose to extend the middleware system to include the ability to execute code in place of the sensors that are not reprogrammable. Additionally, to address the communication between sensor networks, we developed a new middleware component that handles the routing and the conversion of messages. This way, it is possible to reduce the number of exchanged messages between sensor networks and high-level applications, decentralizing the decision-making process to the middleware or to sensor networks (Haller et al., 2008).

The paper is organised as follows. Section 2 describes a use case scenario and we present related work in section 3. Section 4 describes the approach for extending MuFFIN, whereas Section 5 reports on the solution we propose to establish communication between sensor networks at the middleware system. Finally, sections 6 and 7 discuss an assessment we made to the middleware and draw our conclusions, respectively.

2 MOTIVATION SCENARIO

We present a use case scenario that is used along the paper: a temperature and humidity control system of an historical documents archive. To this end, there exists a sensor network that monitors both the temperature and the humidity of the rooms and another network of actuators that controls the air conditioning equipment. The monitoring application defines the behavior of the sensors in order to be notified when the temperature values are outside the range of 16^{o}C to 20^{o}C or the humidity is outside the range of 45% to 60%. Due to an infiltration, there is the need to change the behavior of the sensors in order to moni-

Pires R., Martins F. and Domingos D..
Homogeneous Wireless Sensor Network Programming using MuFFIN.
DOI: 10.5220/0004730501270132
In *Proceedings of the 3rd International Conference on Sensor Networks* (SENSORNETS-2014), pages 127-132
ISBN: 978-989-758-001-7

tor the humidity in regular time intervals and to detect in advance if the dehumidification system is powerful enough to handle this exceptional situation. For that, sensors need to be reprogrammed. With the extension of the MuFFIN middleware proposed in this paper, this reprogramming can be performed regardless of the capabilities of the sensors.

3 RELATED WORK

We group the related literature on middleware technologies and on virtual machines. On what concerns middleware, the Sensor Web Enablement standards (SWE) from the Open Geospatial Consortium (OGC) define a set of interfaces and metadata aimed at hiding the details of communication and the heterogeneity of sensors. The Sensor Event Service standard (SES) manages subscriptions and filters; the Sensor Observation Service standard (SOS) provides a standardized way to access data from observations of the devices; the Observation and Mesurement standard (O&M) provides a schematic representation of the measurements and observations received from sensors; and SensorML specifies models and XML schemas to describe sensor networks.[1]

Existing middleware systems provide sensor information through web services and interact with sensor networks through gateways, which encapsulate sensor specificities (Valente, 2011; Kansal et al., 2007; Bröring et al., 2011). In particular, the Middleware Framework For the Internet of Things (MuFFIN) (Valente, 2011; Valente and Martins, 2011) follows the SWE standard, including SOS for making available sensor data to applications, the O&M for storing the data from sensor networks, and the Web Service Notification (WS-N) (Huang and Gannon, 2006) for notifying applications via *web* services. MuFFIN is implemented on top of FuseESB[2] and its modules follow the publish/subscribe paradigm. Figure 1 depicts the MuFFIN architecture.

As for virtual machines, Squawk (Simon and Cifuentes, 2005) is a Java Virtual Machine (JVM) designed to run on devices without an operating system installed, having been tested on SunSPOT devices. This virtual machine has the abilities to run multiple applications simultaneously and to migrate applications to other devices as long as they run the same virtual machine. Maté (Levis and Culler, 2002) is a communication system based on a virtual machine

[1]Open Geospatial Consortium — http://www.opengeospatial.org/

[2]Fuse ESB — http://fusesource.com/products/enterprise-servicemix/

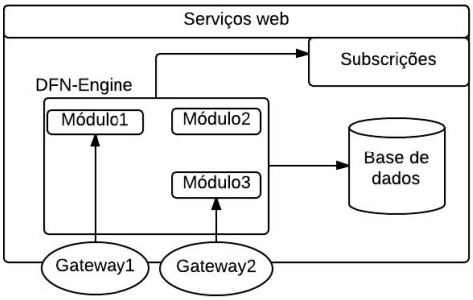

Figure 1: MuFFIN architecture.

for sensor networks, breaking complex programs into small programs in order to reduce energy costs associated with the transmission of programs. This virtual machine runs on TinyOS and only requires 1kb of RAM and 16KB of memory for instructions.

At last, the Callas Virtual Machine (CVM) runs applications developed in the Callas programming language (Martins et al., 2010), abstracting the hardware and the software installed on the devices. This virtual machine is available for Arduino and SunSPOT devices. The Callas programming language is strongly typed, avoiding certain kind of runtime errors and enabling the reprogramming of sensors remotely. This allows the remote updates to the system without the need to physically access the devices. We have implemented a proof of concept for our proposal in MuFFIN, running the Callas Virtual Machine.

4 MuFFIN WITH THE CALLAS VIRTUAL MACHINE

This section presentes an extension to MuFFIN that makes it possible to reprogram sensor devices homogeneously. We start by describing how MuFFIN executes code on behalf of real sensor devices and then present how it simulates the communication between simulated sensors.

MuFFIN allows for remote reprogramming of sensor networks via web services, in case sensor devices support this functionality. In order to make such a functionality available for non-reprogrammable networks, we have integrated the CVM into MuFFIN (MuFFIN-CVM component) to be able to run code on behalf of sensor devices that are not reprogrammable. Figure 2 presents the architecture of the MuFFIN-CVM component.

When the reprogramming web service is invoked, MuFFIN sends the code to the sensor network via its gateway. If the network is not reprogrammable, the gateway sends the code to the MuFFIN-CVM com-

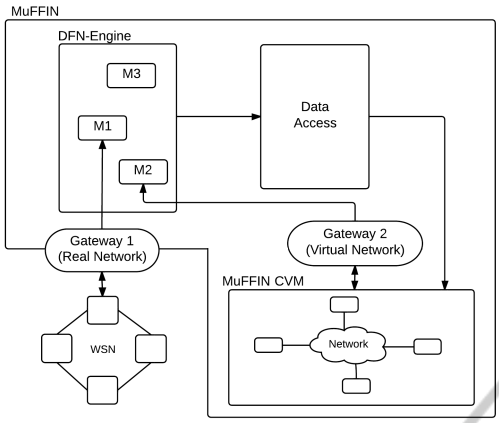

Figure 2: MuFFIN extended with the CVM module.

ponent, which executes it. Additionally, it is necessary to inform the MuFFIN-CVM about the number of sensors that constitute the network. To do this, we changed the installation process of the gateway to include a SensorML file with the description of the network to run virtually. This information is sent to the MuFFIN-CVM together with the code.

Instead of creating an instance of the CVM to run each virtual sensor, MuFFIN executes a single CVM instance and creates an object per virtual sensor to hold its execution state. Once the MuFFIN-CVM executes code on behalf of real sensor devices, it is necessary to simulate system calls, such as the observations real sensors make and the receiving and sending of messages to the network. To obtain the observation values, each CVM state instance has a unique identifier that refers to a single real sensor device. Then, with this key we fetch the corresponding observations from the database, which is stored when the actual readings from the real sensor devices are sent to the middleware.

During the execution of each virtual sensor, the data it produces (the result of internal sensor computation) is sent via the gateway of the virtual network like any other real network does. This allows MuFFIN to treat all networks homogeneously. This way, for instance, features for processing information (such as data filters or topic subscriptions) are also available for virtual networks.

For simulating the sending and receiving of messages we use a circular buffer, wherein each message contains (1) the information sent by sensors (the payload); (2) an identifier of the sensor emitting the message that ensures that the message is not delivered to itself; (3) a message identifier that prevents each sensor from reading repeated messages; and (4) the target port the message is addressed to, in order to model various communication channels. The use of a circular buffer also simulates message loss, since if there is a large number of messages in the "network", a message in the buffer can be overwritten before being read.

5 COMMUNICATION BETWEEN SENSOR NETWORKS: THE THINGSGATEWAY - COMMUNICATION COMPONENT

The motivation scenario presented in Section 1 mentions two sensor network: one to monitor the temperature and humidity of the rooms, and another that actuates on an equipment that has the ability to control these variables. Considering that these networks cannot communicate directly with each other (e.g., they use different communication equipments), their coordination would have to be performed at the application level. In this section we present a MuFFIN extension that supports the communication between sensor networks. The main challenge of this new feature is the conversion of sensor networks protocols.

5.1 Message Protocol Converters

In order to support communication between networks at the middleware level, we developed a system for converting messages from one protocol to another, in case network protocols do not match. The conversion takes the message received from the source network gateway, respecting the protocol of the source network, and directs the message though a sequence of converters that cast it to match the protocol of the destination network. Then, the final step routes the message to the destination network gateway to be delivered to the devices of that network. Notice that the conversion process may generate new messages to, or prevent some messages from reaching, the destination network in order to follow its protocol.

To solve the problem of protocol conversion between networks, we initially considered two approaches: (1) to use a generic protocol as an intermediary between source and destination protocols, (2) use of specific protocol converters for each pair of networks. The first approach poses the problem of finding such a generic protocol, while the second requires a converter for each pair of distinct protocols. Thus we choose to extend MuFFIN to allow the installation and composition of converters in such a way that converters can be used in adapting different protocols and, if desired, can be used to convert

```
<?xml version="1.0" encoding="UTF-8" ?>
<deploy>
 <description>Example of a converter</description>
 <instanceSource id="1"/>
 <instanceDest id="2"/>
 <converters>
  <converter serviceId="3"/>
  <converter serviceId="1"/>
 </converters>
 <externalService>
    http://www.lasige.fc.ul.pt/externService
 </externalService>
 </deploy>
```

Figure 3: XML example of the association between networks.

to a generic protocol as well. This way, we benefit from the advantages of both approaches, while avoiding their limitations. For examples, we can create a converter from network A to network C by composing converters from network A to network B and from network B to the network C. These elementary converters are coded as Java classes that implement a common interface.

5.2 Interconnecting Converters

The association between converters and gateways is defined by an XML file. This file includes the source and destination gateway identifiers and the converters to be used. Additionally, we can set a Uniform Resource Identifier (URI) of an external entity, whose explanation is delayed until section 5.3. The middleware creates an instance of each converter specified in the XML file, which are interconnected in the sequence described by the file. Figure 3 presents an example of an XML file that associates gateways 1 and 2 to converters 3 and 1.

The communication between the source gateway, converters and the destination gateways is established using the ActiveMQ tool, that follows a publish/subscribe paradigm. Thus, converters subscribe the information sent by the source network and, after processing each message, publish them so that another converter will pick it. At the end of the conversion path, the destination gateway subscribes the last converter and gets the message ready to be sent to its network.

Figure 4 illustrates the motivation scenario presented in section 1, in which it is necessary to establish communication between the sensor network (represented by *Gateway origin 1*) and the actuators network (represented by *Gateway destination 2*). For such, converters C1 and C3 are installed and instantiated in MuFFIN, which can adapt the protocols used by the networks. Each message sensor network send

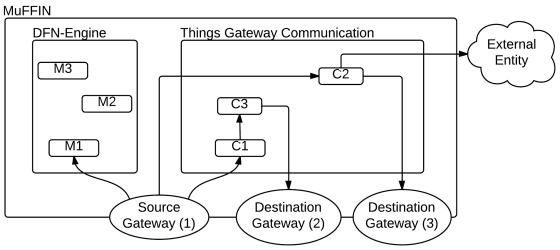

Figure 4: ThingsGateway-Communication component.

arises at *Gateway origin 1* and then follows through convertes *C1* and *C3*, reaching *Destination gateway 2*—the actuator network.

The communication established between gateways is richer than just a point to point communication, i.e., there can be a communication from one source gateway to more than one destination gateways, even in the presence of different destination protocols. Figure 4 illustrates the scenario just described (one to many communication) in which *Source gateway 1* sends information to *Destination gateway 2* and to *Destination gateway 3*.

5.3 External Entity

There is also a mechanism that allows the use of external entities, so that when it is not possible to convert the data received from a network using the installed converters, the middleware can use an external service that can perform the conversion (e.g., converting a Java object to a Callas module). This conversion requires a JVM (to interpret the Java bytecode) and the backend of the Callas compiler (to synthesise the Callas module), which are definitely not to be part of a middleware system. This entity is specified by an URI upon the instantiation of an external converter. The user needs to guarantee that the external entity will perform the conversion according to what is required.

6 EVALUATION

The middleware is installed in the Fuse ESB application server. We use several services from Fuse ESB, in particular, ActiveMQ is used in the communication between MuFFIN components and gateways. MuFFIN is deployed on a virtual machine inside Virtual Box that virtualizes an Intel(R) Core(TM) Duo E8400 @ 3.00GHz (1 CORE) CPU, 1000 Mb of RAM, using the Linux Ubuntu 12.04 Server. This section focuses on the performance test results of the new components, thus not taking into account the time messages take from the a sensor network to its gateway

Table 1: Results for the performed tests.

Num. sensors	10	50	100	200	500	1000	1500	2000
Boot time								
Average (ms)	11.08	33.22	64	130.73	351.38	795.66	1167.1	1625.01
Std. dev.	1.107	1.685	2.471	3.703	19.972	44.405	64.757	67.251
Boot and processing times								
Average (ms)	20.69	46.22	78.69	160.56	450.96	955.37	1448.15	2025.14
Std. dev.	3.78	3.196	4.417	5.761	13.685	24.534	37.038	58.736

and vice versa.

Our tests to the CVM-muffin component comprise the time to boot a virtual sensor network, accounting for the period of time that mediates from the arrival of the Callas code to the beginning of code execution by some device on the virtual network, and the time to boot the network plus the reading of a sensed value (stored in the database). In both cases, to determine how the number of devices influence the processing time, we simulated sensor networks with different dimensions (10, 50, 100, 200, 500, 1000, 1500, and 2000 nodes).

To evaluate the ThingsGateway-Communication component we performed two types of tests: (1) networks use the same language/protocol, without needing message conversion and (2) networks need to convert messages from a string to an integer value. Since we only want to evaluate new MuFFIN extensions, we did not perform tests with external conversion entities.

Figure 5 shows the performance of the MuFFIN-CVM component, considering the simulation of the startup of a network with several nodes and the startup of a network with several nodes where each sensor also performs an observation, i.e., a read operation to the database. For each type of network, we repeated the tests 100 times. The figure shows that the boot time and start-up of networks up to 200 nodes are almost identical. However, for networks with more than 500 nodes, processing begins to take longer, but in both cases the growth is linear. Table 1 has their average values and standard deviations.

Figure 6 shows the results of measuring the time taken for routing messages from one gateway to another. We performed 100 measurements that resulted in an average value of 11.8 ms (with a standard deviation of 1.13) for messages sent without adaptation (left bar). When using an adapter for adjusting messages, we get an average value of 12.2 ms (with a standard deviation of 1.17) (right bar). The obtained values show that, for this particular case, the cost of adapting a message is minimal (3.27%) when compared to the routing cost.

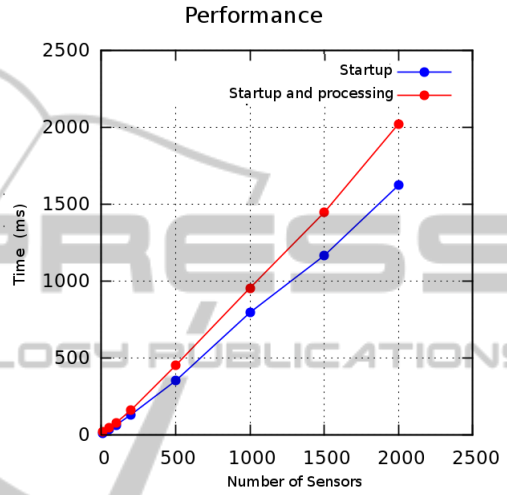

Figure 5: Sensor startup performance.

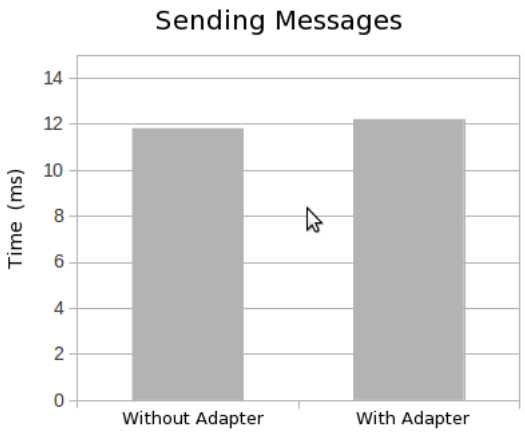

Figure 6: Message routing time.

7 CONCLUSIONS

Providing the sensor remote reprogramming functionality depends on the hardware characteristics or on the programming languages. In this paper, we present an extension to the MuFFIN middleware for generalizing this feature, making it available through web services. To meet this goal, we incorporated the Callas virtual

machine into MuFFIN. Additionally, we also developed a middleware component that supports the communication between different sensor networks, avoiding messages to be routed to client applications.

With these two middleware extensions, MuFFIN can support the decomposition and distribution of business processes through sensor networks. Indeed, by converting to Callas code the business logic sensor networks will execute, we can use MuFFIN to send it to sensors. When sensors do not have the CVM, MuFFIN executes it on their behalf. If business processes foresee direct communication between sensor networks, MuFFIN also ensures that communication, in case they are not directly connected.

As future work, we plan to adapt MuFFIN to run on devices with limited computing capabilities, such as the Raspberry Pi, so it can be used in on-board vehicle technology, providing an interface to intelligent transportation system.

ACKNOWLEDGEMENTS

This project is supported by portuguese Foundation for Science and Technology (FCT) through the Macaw project (PTDC/EIA-EIA/115730/2009) and the LaSIGE multi-year funding programs (UI 408 - 2011-2013, ref PEst-OE/EEI/UI0408/2011)

REFERENCES

Bröring, A., Echterhoff, J., Jirka, S., Simonis, I., Everding, T., Stasch, C., Liang, S., and Lemmens, R. (2011). New generation sensor web enablement. *Sensors*, 11(3):2652–2699.

Haller, S., Karnouskos, S., and Schroth, C. (2008). The Internet of Things in an Enterprise Context. In *Proceedings of FIS 2008*, volume 5468 of *LNCS*, pages 14–28. Springer.

Huang, Y. and Gannon, D. (2006). A comparative study of web services-based event notification specifications. In *ICPP Workshops*, pages 7–14. IEEE Computer Society.

Kansal, A., Nath, S., Liu, J., and Zhao, F. (2007). SenseWeb: an infrastructure for shared sensing. *IEEE MultiMedia*, 14(4):8–13.

Levis, P. and Culler, D. (2002). Maté: a tiny virtual machine for sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(5):85–95.

Martins, F., Lopes, L., and Barros, J. (2010). Towards the Safe Programming of Wireless Sensor Networks. In *Proceedings of PLACES'09*, volume 17 of *EPTCS*.

Simon, D. and Cifuentes, C. (2005). The squawk virtual machine: Java on the bare metal. In *Proceedings of OOPSLA'05*. ACM Press.

Valente, B. and Martins, F. (2011). A middleware framework for the internet of things. In *Proceedings of AFIN 2011*, volume 57, page 139 to 144.

Valente, B. A. L. (2011). Um middleware para a Internet das coisas. Master's thesis, Universidade de Lisboa, Faculdade de Ciências.

Zeng, D., Guo, S., and Cheng, Z. (2011). The web of things: A survey (invited paper). *JCM*, 6(6):424–438.