Diffusing events through JMS on the Sun SPOT platform: a practical experience report*

Roberto Baldoni, Roberto Beraldi, Michele Dominici, Leonardo Querzoni
Dipartimento di Informatica e Sistemistica, Sapienza Universitá di Roma
[baldoni,beraldi,querzoni]@dis.uniroma1.it

ABSTRACT

In the last few years there has been a growing interest in small, low-power hardware platforms that integrate sensing, processing and wireless communication capabilities that can be adopted to quickly deploy powerful wireless sensor networks. Applications for WSNs often apply the event-based interaction paradigm for communication among participants. In this paper we report our experiences with the testing of JORAM, a well known JMS implementation, on top of the Sun SPOT wireless sensor platform.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

1. INTRODUCTION

Wireless sensor networks (WSNs) are today starting to become widely adopted for a number of different settings and applications: from industry settings for controlling machine behaviour to security, from environmental control to vehicular applications. This success is mainly due to the availability of numerous platforms that embed in a small package sensing/acting capabilities, powerful microprocessors and complete radio stacks, all at reasonable prices. The behaviour of nodes constituting a WSN clearly depends on the specific application; however is it common to consider applications where the nodes must periodically sense the surrounding environment using their on-board transducers and report sensed values, or alarms caused by abrupt changes in these values, to a central host that will act accordingly. This event-based interaction between the nodes and the central host is typical of publish/subscribe applications. In this paper we report our practical experiences developed while testing JORAM, an open source Java Messaging System (JMS) implementation, on the Sun's sensor platform (SPOT). This work was mainly motivated by activities running within the Smart Homes 4 All (SM4All) [1] project. Goal of the project is the design and implementation of an innovative middleware platform for inter-working of smart embedded services in domotic environments. As the middleware should be broadly deployable, keeping the overall costs of the solution down is an objective of the project. From this point of view, both the JMS standard and the Sun SPOT WSN platform are in principle good candidates for an integration inside the future SM4All architecture. Scope of this work was thus to explore the possibilities of the integration between these two technologies and their current limits.

*This work was partially supported by the SM4All European Project (FP7-224332).

DEBS'09, July 6-9, Nashville, TN, USA.

2. BACKGROUND

Wireless sensor networks are systems constituted by a set of independent devices whose main purpose is to sense data from the surrounding environment and send it to a central host deputed to its elaboration and storage. Each device has some computing capabilities, can send and receive messages using a wireless network adapter but has a limited amount of available energy to perform these operations. The transfer of data from sensors to the central host is usually performed periodically or based on the occurrence on some specific events (e.g. sensed temperature surpassing a predefined threshold). This latter model of data transmission within the WSN perfectly fits the publish/subscribe communication paradigm. Currently, however, WSNs does not employ standard publish/subscribe middleware platforms, like JMS or DDS, but rather prefer to resort to ad-hoc solutions [2, 3], mainly for reasons related to energy consumption.

The Sun SPOT project - Sun SPOTs (Small Programmable Object Technology)[7] are battery-powered sensors developed at Sun Microsystems Laboratories. In its basic form each SPOT includes a CPU, onboard memory and a wireless network adapter. Sun SPOTs run an *ad-hoc* small-footprint Java virtual machine, called Squawk, that can host multiple applications concurrently, requires no underlying operating system and offers a fully capable Java ME (JME) environment [5] that supports CLDC 1.1 and MIDP 1.0. Stackable boards can be used to expand the abilities of a SPOT and include application-specific sensors and actuators such as accelerometers, light detectors, temperature sensors, LEDs, push buttons and general I/O pins. A special SPOT unit, called *base station*, can be used to bridge connections between computers in a LAN (or the Internet) and a network of SPOTs. In order to build energy efficient applications Sun SPOTs have power conservation firmware that enables three modes of operation: (i) *Run* mode with all subsys-

tems running, (ii) *Idle* mode with CPU and radio switched off and (iii) *Deep-sleep* mode with all subsystems shut down except for the standby voltage regulator, power-control and the RAM memory. The capacity of the built-in battery is 720 milliampere-hours, so in *Run* mode the battery can support about 7 hours of operation. This can be extended by having the processor sleep and turning off the radio when it is not in use. In deep sleep mode the battery is expected to last for more than 900 days.

JMS and the JORAM platform - The Java Message Service (JMS) [6] is a standard for the implementation of message-oriented middleware on the Java platform. The JMS API is built upon a *topic-based* event structure. Publishers and subscribers are anonymous and can dynamically publish and subscribe to various topics. Different levels of reliability and QoS requirements can be defined for each topic. JORAM (Java Open Reliable Asynchronous Messaging) [4] is a 100% Java open source implementation of the JMS 1.1 specification. The JORAM platform is constituted by two distinct softwares: the JORAM *server* that manages the JMS abstractions and the JORAM *client* that is bound to the application wanting to leverage JMS based communications. JORAM clients communicate with server instances usually relying on TCP/IP. One of the most interesting characteristics of JORAM is the availability of a lightweight client named kJORAM targeted at JME compliant devices. Applications running on JME compliant devices can thus use kJORAM to cooperate with other JMS applications[1]. In this case, interactions between kJORAM clients and JORAM servers take place using SOAP/XML over a HTTP connection. Compatibility with the JORAM server is guaranteed by the adoption of a server-side SOAP proxy service whose main purpose is to translate XML messages produced by the kJORAM client in a format that can be directly treated by the JORAM server and by the standard J2SE JORAM client.

3. EVALUATION

The testing platform adopted for evaluating the porting of kJORAM on the Sun SPOT platform was constituted by a single Sun SPOT equipped with ligt and temperature sensors, that was running an application whose only purpose was to publish events. A SPOT base station was used to receive events published by the sensor and forward them to the JORAM server running on a PC. The publication of a JMS message from the publisher SPOT followed these steps: (1) the remote SPOT asks the base station for a HTTP connection to the application server hosted on the PC; (2) the base station forwards the HTTP connection request to the application server and acts as a gateway between the two entities; (3) as soon as the connection with the server is established, the SPOT sends a SOAP request for publishing a JMS message; (4) the SOAP proxy server running in the application server forwards the request to the JORAM server that is in charge of maintaining the topic where the message has been published; (5) if the JORAM server correctly receives the publish request, it immediately sends back an acknowledgement toward the SPOT. The simplicity of this testbed was a key point to correctly profile, during the testing phase, the behaviour of the kJORAM client and more generally, the behaviour of the SPOTs. During the tests we concentrated our attention on the evaluation of two fundamental metrics: *Memory consumption* used by the JMS client, and *Battery drain*, i.e. the amount of energy used to communicate through the JMS client. The aforementioned metrics have been evaluated in a suite of different tests aimed at investigating how different aspects of the JMS client impact on memory consumption and, more interestingly, on battery drain. The following tests were conducted:
Test 1 - the SPOT publishes 100 JMS messages containing dummy content with a frequency of one message every 6 seconds (on average); no data is read from the on-board transducers;
Test 2 - the SPOT executes exactly the same code used in test 1 but does not send any message; no data is read from the on-board transducers;
Test 3 - the SPOT publishes 100 JMS messages containing the current temperature value with a frequency of one message every 6 seconds (on average). The temperature value is obtained by one of the on-board transducers;
Test 4 - the SPOT checks every 10 minutes (for a total of 8 hours) the light amount in the surrounding environment; if the variation with respect to the previously read value exceeds a predefined threshold the application publishes a JMS message containing the new value; light intensity is obtained by one of the on-board transducers; the SPOT is put in deep-sleep mode after every read cycle and awaken 10 minutes later;
Test 5 - the SPOT executes exactly the same code used in test 4 but does not send any message;

The test results showed a stable and predictable behaviour of the system characterized by a large battery consumption. The main contribution to this consumption is due to periodic radio transmissions used to deliver JMS events to the base station. As a consequence, our conclusions are that in this current form these two combined technologies are not easily adoptable in the considered domotic environment due to the excessively short lifespan of the sensors battery. However, a more careful design of the communication middleware could in principle reduce the number of transmission and consequently improve the energy consumption figures by better leveraging the duty-cycling characteristics of the sensor platform.

[1]Note that currently only a subset of the JMS API is available to developers through the kJORAM lightweight client.

4. REFERENCES

[1] The Smart Homes 4 All (SM4All) project. STREP project founded by the EC. `http://www.sm4all-project.eu/`.
[2] P. Levis and D. E. Culler. Maté: a tiny virtual machine for sensor networks. In *ASPLOS*, pages 85–95, 2002.
[3] T. Liu and M. Martonosi. Impala: a middleware system for managing autonomic, parallel sensor systems. In *PPOPP*, pages 107–118. ACM, 2003.
[4] ObjectWeb. Java Open Reliable Asynchronous Messaging (JORAM). `http://joram.objectweb.org/`.
[5] Sun Microsystems Inc. Java Micro Edition. `http://java.sun.com/javame/index.jsp`.
[6] Sun Microsystems Inc. The Java Message Service (JMS). `http://java.sun.com/products/jms/`.
[7] Sun Microsystems Laboratories. Project Sun SPOT. `http://www.sunspotworld.com/`.