Erlang meets WSNs: a functional approach to WSN programming

Alessandro Sivieri
Dipartimento di Elettronica e Informazione
Politecnico di Milano, Italy
sivieri@elet.polimi.it

***Abstract*—The scarce diffusion of Wireless Sensor Network systems is due to several factors, among them the complexity in designing, programming and testing applications for this technology. Too often too specialized solutions have been proposed for specific scenarios, impeding code reuse and supporting poorly engineered development methodologies (i.e. "code and fix"). In this work we claim that Erlang, a concurrency-oriented functional language, could greatly reduce the complexity of designing this kind of applications, while at the same favouring sound engineering practices and simplifying testing and simulation of applications. There are challenges in effectively using this language on WSN systems, but in our opinion the effort will be paid off.**

***Index Terms*—Wireless Sensor Networks; Erlang; programming languages; application frameworks;**

I. Introduction and motivation

The last decade saw a growing hype around the various, related, concepts of *Pervasive Systems*, *Internet of Things* (IoT), and *Wireless Sensor Networks* (WSNs). The recent advances in microelectronics and communication technology, transformed the vision of a world populated by millions of devices capable of sensing, actuating, and communicating, from science fiction to reality. On the other hand, there are still relatively few examples where this technology has been applied with success outside the research laboratories.

Most of the researchers operating in the area agree that what limits the diffusion of pervasive systems in general, and WSNs in particular, is the complexity in designing, programming, testing, and deploying real scale applications based on such technology. Developers in this area must face several challenges: not only a sensors network is a distributed system, but it also runs on a large number of heterogeneous devices, each with different technical characteristics and capabilities, communicating with wireless technologies, often characterized by low bandwidth, short communication range and unreliable links.

Indeed, the community has proposed a whole landscape of possible solutions [1], spanning a great number of different programming paradigms and architectures; all these approaches can be classified at least under two main areas: *micro-programming* and *macro-programming*. The former leaves to the developer the responsibility of decomposing the problem in the distributed system and implementing each single node; this often requires the knowledge of all the aspects of the system, from the hardware characteristics of the nodes to the routing capabilities of the communication system. On the other hand, the latter approach tried to ease the development by seeing the whole network as a single entity to be implemented, while the underlying framework will take care of decomposition and communication. This forces applications to loose generality and often allows reusability of existing solutions only in few similar scenarios.

As if this were not enough, the development process adopted by the WSN community rarely applies the best practices proposed by Software Engineering, while it often relies on a "code and fix" approach [2], which in turn relies on the low-level primitives offered by the operating system and on the developers' skills. This blocks code reuse and creates difficulties in testing applications or verifying functionalities in more formal ways.

II. An old language for a new problem

What we need is a programming paradigm that could merge the best of the two paradigms presented in the previous section, and at the same time allow for a better engineering of WSN applications, and *we claim that Erlang could play this role*.

Erlang [3] is a functional language originally developed by Ericsson in the '80s to be used in embedded telecommunication systems; overtime, it has grown into a complete platform for developing a wide range of applications, and now contains many libraries offering different functionalities. It is an interpreted language: each source is compiled in bytecode and this is executed at run-time by a virtual machine.

First of all, Erlang combines a functional core with dynamic typing and the use of pattern-matching as the main mechanisms to access data and guide the computation, supporting a form of declarative programming that allows programmers to focus on *what* a computation is supposed to do instead of *how* to achieve it. This results in extremely compact code that is easy to develop and maintain, and is highly suitable for programs that have to deal with collecting data, filtering it, and sending it around.

At the same time, while being a high-level language, Erlang does not forget it was designed with embedded systems in mind. As a result, it includes binaries (sequences of bytes) among the natively supported data types, together with bitwise operators, and also the ability to pattern-match on bitstreams (bit sequences), allowing high-level description of packet manipulation functions.

978-1-4244-9528-3/12/$26.00 ©2012 IEEE

In addition, Erlang enriches its functional, declarative core, with ad-hoc language constructs to support concurrent and distributed programming, following an actor-like concurrency model [4], which allows for easily and naturally organizing computations in sets of light processes, handled directly by a run-time that handles message passing between processes and even interpreter instances located on different nodes of the network transparently with respect to the programmer.

Furthermore, Erlang provides an extensive, standard library, called the Open Telecom Platform (OTP), which offers predefined modules for process linking and monitoring, which can be used for supervising processes and developing fault-tolerant applications, and hot-swap functionalities, which support long-running applications, whose code can be changed on-the-fly without interrupting the execution of the program.

In our opinion, these characteristics perfectly fit WSN applications:

- functional languages can easily manipulate data collected by a sensor network
- declarative code is more readable
- being high-level, it eases reusability of components
- its distributed and communication facilities adapt to the intrinsic characteristics of WSN systems
- the platform already offers some tools for testing applications, and some third-party applications allow formal verification of Erlang programs [5]
- usage of a virtual machine helps solving the heterogeneity of hardware platforms used in WSNs.

III. Challenges

While originally designed to run on embedded platforms, Erlang grew during its evolution to include a large number of libraries and a complex run-time infrastructure that helps programmers in hiding distribution as much as possible and in guaranteeing fault tolerance with minimal effort. While nice and welcome in many domains, most of these advanced features find limited application in WSNs, increasing the resources consumed by the standard Erlang run-time system.

Accordingly, we developed a custom version of the Erlang interpreter that includes only the core and basic functionalities of the standard interpreter, removing most of the advanced features. This reduced the requirements of the platform both in terms of storage and memory. While the footprint is still quite high compared to the typical hardware characteristics of WSNs, it is acceptable for the pervasive, embedded scenarios we consider, like monitoring building for maximizing energy efficiency or helping elderly people in performing their daily activities in properly equipped houses; as of this moment, our primary objective is anyway to show the advantages of our platform in terms of expressiveness and ease of use, and our first prototype pursues this goal.

We also need to adapt the existing communication facilities of the language to a different scenario: indeed, the standard Erlang platform assumes the availability of a full fledged TCP/IP stack, with routing and transport layers offering good reliability in sending messages among nodes, independently from their physical location. This is a strong assumption for most WSN applications, which we decided to relax in WSN-Erlang. The new inter-node communication functions only assume the availability of a link-layer protocol and build upon it. In a wireless network this means that a message may reach its destination only if the two nodes (the sender and the receiver) are in range. Also the naming mechanism for processes located on different nodes has to change, because inter-node communication is now based on addresses and registered names instead of other kinds of identifiers. Internally, each node can use the standard Erlang functionalities in terms of communication and process handling.

IV. Current and future works

We are in the process of developing the first prototype of our framework and platform, which will offer a (relatively) small footprint Erlang interpreter, which could be used on embedded devices; an application library introducing the functionalities required by WSN programs, as described previously; a simulator allowing developers to test their code on completely or partially simulated networks. We are performing also some tests using common WSN algorithms for data propagation and collection, comparing Erlang implementations with others developed using some of the most used WSN platforms (i.e. *TinyOS* and *Contiki*).

The current prototype will be deployed on different platforms with different hardware capabilities, to test the impact of the run-time system on the overall device performances; this will be only a first step to demonstrate the advantages of the Erlang programming model when applied to WSN applications. In the future we plan to continue our work on our run-time platform to further reduce its requirements, fully rewriting the Erlang interpreter from scratch, if required.

Acknowledgments

This work was partially supported by the European Commission, Programme IDEAS-ERC, Project 227977-SMScom. The author would like to thank his advisor, Prof. Gianpaolo Cugola, for his suggestions and guidance throughout this whole work.

References

[1] L. Mottola and G. P. Picco, "Programming wireless sensor networks: Fundamental concepts and state of the art," *ACM Comput. Surv.*, vol. 43, pp. 19:1–19:51, 2011.

[2] G. P. Picco, "Software engineering and wireless sensor networks: happy marriage or consensual divorce?" in *Proceedings of the FSE/SDP workshop on Future of software engineering research*, 2010.

[3] J. Armstrong, *Programming Erlang: Software for a Concurrent World*. Pragmatic Bookshelf, 2007.

[4] C. Hewitt, P. Bishop, and R. Steiger, "A universal modular actor formalism for artificial intelligence," in *Proceedings of the 3rd international joint conference on Artificial intelligence*, 1973.

[5] Q. Guo, J. Derrick, C. B. Earle, and L.-A. Fredlund, "Model-checking erlang: a comparison between etomcrl2 and mcerlang," in *Proceedings of the 5th international academic and industrial conference on Testing - practice and research techniques*, 2010.