Terra: Flexibility and safety in WSNs

Adriano Branco
Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro
abranco@inf.puc-rio.br

Abstract

My research focuses the integration of a program language with a specialized component set to minimize problems encountered when programming wireless sensor networks applications. The scarcity of resources along with the event-oriented nature of applications and the need for coordination among large numbers of nodes are the main sources of programming difficulties in this area. To investigate the suitability of my approach, I built Terra, a system that implements a programming environment based on a reactive high-level language with execution safety and using a virtual machine that integrates customized components. I describe an initial evaluation of Terra's flexibility in adjusting the abstraction border of run-time components and the script complexity.

Keywords

WSN-Wireless Sensor Network, Virtual Machine, Safe Execution, Reactive Program Model, DSL support

1 Introduction

Programming a wireless sensor network (WSN) remains a challenge. WSNs are typically composed by computing devices (*motes*) that communicate via radio and rely on batteries for energy. Although a whole range of microcontrollers can be used in this setting, it is very common, due to cost restrictions and scale of usage, to employ units with very limited memory and computing resources. This scarcity of resources, along with the event-oriented nature of applications and the need for coordination among large numbers of nodes, make programming applications a difficult and error-prone task [1, 8, 11].

It is also often the case that the user must reprogram sensor network nodes after they are in place. This is hard to do physically, because in most cases it is difficult to recover the motes from the position in which they are installed. The obvious solution is to do the updates through radio messages; however, transferring complete binaries over radio can lead to high energy consumption, and is thus undesirable.

On the other hand, because of their restricted resources and deployment characteristics, a given sensor network is normally used for a single category of application, such as environment monitor or plant control, even if the application itself evolves over time. Taking this into consideration, Levis proposed [9] that the task of programming and reprogramming WSNs can be simplified by using the combination of a domain-specific language and a virtual machine (ASVM-Application Specific Virtual Machine). In his work with the virtual machine Maté, Levis experimented with three different domain-specific languages, each of them translated to code interpretable by the virtual machine using specific compilers. Balani et al., based on ASVM, proposed DVM [3]. DVM allows dynamic loading of system modules as well as of high-level scripts

However, developing a separate compiler for each new application niche is a very costly proposal. Besides, even considering different niches, WSN applications tend to follow restricted patterns of behavior and interaction, such as creating and maintaining groups of nodes to collect data and make local decisions; or forwarding the collected data to a sink node. Thus, I present an alternative approach, in which common programming patterns are designed and implemented separately as libraries of components. These components may be combined as needed, creating customized virtual machines. The interfaces of the included components will then provide the abstractions that are convenient for the application script. The scripting language is the result of combining a base programming language with the abstractions offered by the chosen components. I can say I'm proposing the use of internal domain-specific languages, which are existing languages used in particular ways, as opposed to that of external domain-specific languages (as proposed in ASVM), which have a custom syntax and need full-blown parsers [7].

I formulated the following as research question for my thesis: *To what extent can a programming environment based on a reactive high-level language with execution safety and using a virtual machine that integrates customized components minimize problems encountered in building WSN applications, provide appropriate abstraction levels, similar to those attainable by DSLs, thus minimizing errors typical of*

SenSys'13, November 11–15, 2013, Rome, Italy.

distributed systems and event-driven programming?

To investigate this idea I built Terra, a flexible system that targets both domain experts and system experts. The domain expert benefits from a high level programming environment where the system expert may easily integrate new operations as needed. The system uses virtual machine concept to facilitate the low energy program script remote distribution.

Section 2 present the current Terra System characteristics. Section 3 present my early proposition for the evaluation. section 4 has the final considerations. The related work is commented during the text.

2 The Terra System

As an introductory example, the Terra program in Figure 1 repeatedly reads the light (lines 4-5) and temperature (lines 7-9) sensors and tests (line 10) whether either reading has risen above a predefined limit. If this happens, the program turns on a led (line 11) to indicate the anomalous condition. Then the execution blocks waiting for 1 minute (line 13) before repeating the loop again. Section 2.2 presents the language and a program example using a high-level abstraction component.

```
 1: var ushort tValue,pValue;
 2: loop do
 3:   par/and do
 4:     emit REQ_PHOTO();
 5:     pValue=await PHOTO;
 6:   with
 7:     emit REQ_TEMP();
 8:     tValue=await TEMP;
 9:   end
10:   if pValue > 200 or tValue > 300 then
11:     emit LED0(ON);
12:   end
13:   await 1min;
14:   emit LED0(OFF);
15: end
```

Figure 1. Simple program example

As said, the Terra system implements a virtual machine with customized components. For the scripting language, I chose to implement an interpreted version of the CÉU programming language [14]. CÉU is a programming language developed at PUC-Rio which offers a number of safety guarantees. CÉU's reactive programming model and its approach for split-phase support is very well suited to the typical event driven model from WSN applications. Typical WSN hardware keeps the CPU in sleeping mode until it receives an interruption. In that case, these interruptions work as system events. Also, the CÉU verifications in compile and execution time are valuables guarantees when programming in remote distributed environment.

The custom operations are built as embedded components of the virtual machine runtime and are accessed as CÉU extended commands. In general, these operations represent specific abstractions used by the domain expert to build new applications. In this way Terra gives support to create new internal Domain Specific Languages (DSL) where we don't need to produce a new compiler for each new domain [7].

Also Terra gives the flexibility to balance the abstraction level of the runtime with script complexity. Some projects may require, based on their maturity level, to leave some operations at script level instead of to have it as runtime component. In this way, the application can evolve through the remote configuration facilities.

2.1 Currently implemented components

Terra provides a library of TinyOS components that implement typical programming patterns. When building a VM for a given application area, the programmer can choose whether or not to include each Terra component in the configuration, setting different boundaries for the provided abstractions [9].

In order to determine the set of components that would be important to implement for a basic library, I considered, on the one hand, proposals for facilitating programming in WSNs with restricted resources including macroprogramming proposals [12, 13, 8, 1, 10, 6, 2] and, on the other, some typical applications of this platform. As a result of this work [5], I organized the identified functionalities in four areas:

1. group management — support for group creation and other control operations;
2. communication — support for radio communication among sensor nodes;
3. aggregation — support for information collection and synthesis inside a group;
4. local operations — support for accessing sensors and actuators.

2.2 Program environment

Terra uses the CÉU programming language [14] for scripting. CÉU follows a synchronous execution model [4] which enforces a disciplined step-by-step execution that enables race-free concurrency. In Terra, CÉU scripts are translated to virtual-machine code, and CÉU's synchronous and static nature allows the translator to verify that reactions to the environment are deterministic and execute within bounded memory and CPU time. Scripts interact with the environment by emitting and reacting to events that are, respectively, captured and emitted by components in the installed virtual machine. Thus, the only part of scripts that escapes static analysis are calls to components provided by Terra's VMs, which are encapsulated in modules and have been extensively tested beforehand. In this way, Terra allows for remote update of programs while providing a number of execution guarantees.

The basic language flow control structures are *if-then-else* and *loop*, but CÉU has a special control structure based on *par-with* command. A *par-with* structure has two or more program blocks separated by the *with* declaration. Each program block may have its execution blocked without interfering with the others. The *par* declaration may include a */and* or */or* complements. The */and* defines that the *par* structure ends when all blocks are ended. The */or* defines that when any block ends, the *par* full structure ends and cancels the execution of the others blocks.

The VM interacts with the external world through `emit` and `await` CÉU commands. The `emit` command calls a custom built-in function and the `await` command blocks waiting

for an external event triggered by custom built-in operations. Also the `await <time>` command blocks for specific timer interval. A specific VM customization defines a specific set of events and thus a specific high-level language.

Figure 1, in the beginning of this section, presents a small program example that uses the three flow control structures.

Figure 2 presents a program example that uses the Grouping abstraction component. Lines 1 and 2 create a new group definition. In this case the parameters represent: group identifier (the two first integers 1,1); maximum range (3 hops); group activated status (TRUE); group election procedure (OFF); and initial leader node (id 0 - don't used here). The if/then/else structure (lines 7-20) divides the execution in two blocks. The first block (lines 8-14) runs only on the node with ID 2 and the second block (lines 16-19) runs on all other nodes. The loop (lines 10-14) sends a incremental counter to neighbour nodes defined in the *gr1* group and waits 1 second. All nodes up to 3 hops from node 2 will receive this counter and write the received value on its on-board LEDS (lines 16-19).

```
 1: var group gr1;
 2: grNew(&gr1,1,1,3,TRUE,OFF,0);
 3:
 4: var ubyte x;
 5: var msg counterMsg;
 6:
 7: if NODE_ID == 2 then
 8:   counterMsg.value1 = 0;
 9:   emit LEDS(0x7);
10:   loop do
11:     counterMsg.value1 = counterMsg.value1 + 1;
12:     emit SEND_GR(&gr1,counterMsg);
13:     await 1s;
14:   end
15: else
16:   loop do
17:     counterMsg = await REC_GR;
18:     emit LEDS(counterMsg.value1);
19:   end
20: end
```

Figure 2. Terra program using Grouping abstraction component

2.3 Preliminary evaluation

I already evaluated the virtual machine costs related to the use of CPU and energy consumption as compared with direct execution over TinyOS. Currently Terra is implemented in MicaZ and TelosB version. The test scenario uses a intensive I/O bound application with the maximum sensor reading loop. In Terra, CPU was active 12.8% of the time, while in nesC only 5.3%. Although the execution of interpreted code is more expensive than that of the native, nesC, code, this difference is too small in an I/O-bound, low-frequency, pattern.

3 Proposed evaluation

The main idea is to evaluate Terra in two points. One point will focus Terra ability as DSL support. The other point will focus on Terra flexibility to adjust the abstraction border of component abstraction level and the script complexity. The different abstraction levels from the second point automatically will give different DSL flavours to be used in the first point, in that way it is possible to have the same program examples to address both evaluation points.

As a basis for this evaluation I will use the Terra version containing the generic component set defined in section 2.1. The functionality of these components ranges from local mote, low-level, operations to high-level abstraction like group operations. This makes it possible to build either small script applications using pre-built high-level component or implementing similar high-level functionality with a more complex script program based on low level components. Also, as these components are generic, it is possible to build specific component encapsulations, creating DSL flavours for each target domain.

For these experiments I need to select some different application domain and the respective applications definitions (functional specifications). Possible application domains I am considering are forest monitoring, building automation and urban parking. These applications need to be reviewed in more details to guarantee more significant test cases. I will design at least two versions for each application, with different abstraction borders. At this point it could be necessary to build new components to reach a specific DSL flavour.

As early ideas for basic metrics I am considering: code size, memory usage, remote reconfiguration cost, responsiveness, CPU usage, and power consumption. For example, it will be possible to evaluate the cpu usage and power consumption overhead for different abstraction levels. Also I am planning to define a code complexity value where I may summarize how many times the programmer faces resource concurrency control and in how many of these situations the Terra safeties prevents the possible errors. My goal is to compare the behavior of each configuration and the respective cost and benefits. The evaluation results will help understand how much the proposed program environment may simplify and minimize programming errors in building WSN application.

For DSL support evaluation I intend to compare the expressiveness of two or three Terra DSLs to corresponding specific-domain program environment, for example TinyDB for database like domains.

4 Conclusions

This document presents Terra, a system that implements a programming environment based on a reactive high-level language with execution safety and using a virtual machine that integrates customized components. I also discussed how I am planning to evaluate it against typical programming problems when building WSN distributed applications. I defined a preliminary set of metrics to measure the costs and benefits for different program versions that stress Terra in two point. The flexibility of adjusting the abstraction border and the ability to support DSL flavors. The main research contribution will be the understanding of the potential of the internal DSL I am proposing in the context of WSN programming. Another contribution is the usage of CÉU language extended with custom components to obtain a WSN domain specific language with safety guarantees.

5 References

[1] A. Awan, S. Jagannathan, and A. Grama. Macroprogramming heterogeneous sensor networks using cosmos. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 159–172, New York, NY, USA, 2007. ACM.

[2] A. Bakshi, V. K. Prasanna, J. Reich, and D. Larner. The abstract task graph: a methodology for architecture-independent programming of networked sensor systems. In *Proceedings of the 2005 Workshop on End-to-End, Sense-and-Respond Systems, Applications and Services*, EESR '05, pages 19–24, Berkeley, CA, USA, 2005. USENIX Association.

[3] R. Balani, C.-C. Han, R. K. Rengaswamy, I. Tsigkogiannis, and M. Srivastava. Multi-level software reconfiguration for sensor networks. In *Proceedings of the 6th ACM & IEEE International Conference on Embedded Software*, EMSOFT '06, pages 112–121, New York, NY, USA, 2006. ACM.

[4] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages 12 years later. *Proceedings of the IEEE*, 91(1):64 – 83, jan 2003.

[5] A. Branco. A WSN programming model with a dynamic reconfiguration support. Master's thesis, PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO - PUC-RIO, april 2011. Text in portuguese.

[6] H. Cervantes, D. Donsez, and L. Touseau. An architecture description language for dynamic sensor-based applications. In *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pages 147–151, Jan. 2008.

[7] M. Fowler and R. Parsons. *Domain-Specific Languages*. Addison-Wesley Professional, 1st edition, September 2010.

[8] N. Kothari, R. Gummadi, T. Millstein, and R. Govindan. Reliable and efficient programming abstractions for wireless sensor networks. *PLDI '07: Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 200–210, 2007.

[9] P. Levis, D. Gay, and D. 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.

[10] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30(1):122–173, 2005.

[11] L. Mottola and G. P. Picco. Programming wireless sensor networks: Fundamental concepts and state of the art. *ACM Comput. Surv.*, 43(3):19:1–19:51, Apr. 2011.

[12] R. Newton, G. Morrisett, and M. Welsh. The Regiment macroprogramming system. In *IPSN '07: Proceedings of the 6th International Conference on Information Processing in Sensor Networks*, pages 489–498, New York, NY, USA, 2007. ACM.

[13] R. Newton and M. Welsh. Region streams: functional macroprogramming for sensor networks. In *DMSN '04: Proceeedings of the 1st International Workshop on Data Management for Sensor Networks*, pages 78–87, New York, NY, USA, 2004. ACM.

[14] F. Sant'Anna, N. Rodriguez, R. Ierusalimschy, O. Landsiedel, and P. Tsigas. Safe system-level concurrency for resource-constrained nodes. In *Proceedings of the 11th International Conference on Embedded Networked Sensor Systems*, SenSys '13. ACM, 2013.

Student biographical sketch

Adriano Branco currently is a PhD candidate of Computer Science Department at PUC-Rio. His research focus on Wireless Sensor Network (WSN) in distributed system area. He also got his master in computer science at PUC-Rio in 2011 working with WSN. He undergraduates in Electronic Engineering at CEFET/RJ in 1992. From the undergraduate course he worked as electronic engineer and system developer at CBPF/CNPq (Brazil) and CERN (Switzerland). At CPBF, in the LAFEX laboratory, he worked on the parallel computer program from Fermilab collaboration group. At CERN he spent two years working in the New Trigger Project for LEP Delphi Experiment. After that he had worked more than 12 years in system integration consulting projects (as developer and project manager) for large companies. Mainly for the Industrial Automation and Telecommunications industries, including an international project in Manila/Philippines.

Research advisor: Noemi Rodriguez – Departamento de Informática – PUC-Rio – Pontifícia Universidade Católica do Rio de Janeiro – noemi@inf.puc-rio.br

Research co-advisor: Silvana Rossetto – Departamento de Ciência da Computação – UFRJ – Universidade Federal do Rio de Janeiro – silvana@dcc.ufrj.br

Dissertation submission expected to end of 2014.