Poster Abstract: SEAL: An Easy-to-use Sensor Node Application Development System

Atis Elsts
Faculty of Computer Science
University of Latvia, and
Institute of Electronics and Computer Science
Email: atis.elsts@lu.lv

Janis Judvaitis
Faculty of Computer Science
University of Latvia
Email: janis.judvaitis@lais.lv

Leo Selavo
Faculty of Computer Science
University of Latvia, and
Institute of Electronics and Computer Science
Email: leo.selavo@lu.lv

Abstract—Majority of potential wireless sensor network (WSN) users have no formal education in computer science. To make WSN application development more feasible and attractive to this target audience, we have designed and implemented SEAL, a domain-specific language for WSN application description. Our idea is based on the observation that even though the low-level details of WSN are swarming with complexities, a broad class of WSN applications are conceptually simple – they can be described either as sense-and-send or event detection. SEAL decouples the low-level networking and hardware details present in sensor networks from application logic, thus reducing the complexity visible to the user. We suggest that our approach leads to reduced source code sizes and faster development times, without sacrificing user freedom and energy-efficiency.

I. INTRODUCTION

The WSN technology was envisioned as a tool for a broad range of purposes and target audiences. The fulfillment of this original WSN promise is hindered by the complexities inherent in WSN programming and maintenance, which can make these tasks forbidding for people without expertise in computer science (CS). To ameliorate these difficulties, we propose our solution: Sensor Application Development Language (SEAL), a language and programming environment targeted towards domain experts and novice programmers. SEAL allows the user to avoid thinking about networking and hardware details and focus on high-level application logic instead. SEAL was evaluated on agriculture scientists and on CS students with no WSN programming experience. In future, SEAL is going to be used in a WSN project for precision agriculture [1].

II. THE LANGUAGE IN BRIEF

SEAL is a domain-specific language for WSN application description. Our initial idea was to make SEAL purely declarative. Compared to the imperative approach, this has the benefit of conceptual simplicity, because one has to specify only what a program should do, not how exactly it should do that. On the second thought, we decided to include in SEAL syntax for specifying state-machine semantics as well, because of the difficulties presented by describing some common tasks (such as event detection with hysteresis) in a declarative language.

SEAL syntax features three kinds of descriptive statements (for sensors, actuators and system outputs), conditional statements, and syntax for describing state variables.

Fig. 1: SEAL graphical user interface

SEAL compiler works by translating SEAL code to C code, which is then compiled natively for a specific architecture.

SEAL is integrated in MansOS [2] and was experimentally evaluated using this WSN OS. Nevertheless, we believe it could be easily adapted for TinyOS, Contiki, or any other WSN OS that provides support for components used by SEAL (such as LED, software timers, serial port, radio, external flash memory, ADC channels etc.). Since SEAL is used to specify only user-level logic of the application, it relies on the OS for details such as scheduling algorithms or multi-hop networking.

SEAL has natural ties with MansOS run-time reprogramming system, which allows partial reprogramming of the motes. If a SEAL script is modified, only user part of the binary code must be reprogrammed, leaving majority of the code intact.

When designing SEAL, we paid attention to principles identified by usability experts [3] and recommendations for designers of novice programming systems [4]. For example, the language features “match between system and the real world” (every sensor present on a mote has a corresponding name in SEAL), “recognition rather than recall” (most of SEAL code is plain English), “aesthetic and minimalist design” (compared to relative richness of C or NesC).

To make developing WSN applications feasible even for users with no programming experience, we developed a GUI for SEAL (Figure 1), written in Python.

III. EVALUATION

As the SEAL source code examples given below show, when compared to native programming in TinyOS or MansOS (Table I), they tend have much smaller code size, thus making software prototyping faster.
The C code generated by SEAL compiler uses event-based control flow, and consequently works well together with duty-cycling. The sample applications presented here all have small duty cycles: 0.037% for blink, 0.627% for sense-and-send, 0.701% for event detection (evaluated on Tmote Sky at 4MHz).

Listing 1 Blink
use RedLed, period 1000ms;

Listing 2 Sense-and-send
read Light, period 2s;
read Humidity, period 6s;
output Radio;

Listing 3 Event detection with hysteresis
state temperatureCritical false;
when Sensors.Temperature > 50C:
  set temperatureCritical true;
when Sensors.Temperature < 40C:
  set temperatureCritical false;
when temperatureCritical: // blink red LED
  use RedLed, period 100ms;

SEAL was evaluated in tests on university students and domain scientists.

Firstly, four CS students from University of Latvia with no WSN experience were asked to write an advanced sense-and-send program in SEAL. The results (Figure 2) were compared with an earlier test, where different groups of students were asked to program the same application in MansOS and TinyOS. Three of the students produced a solution that was either completely correct or had only small syntactical defects. Despite their previous C programming experience, the students produced worse results when using plain C, and much worse results when asked to program in NesC. SEAL was also evaluated as both easier to learn and to use in expert level.

TABLE I: Lines of Code Comparision

<table>
<thead>
<tr>
<th>Program</th>
<th>NesC (TinyOS)</th>
<th>C (MansOS)</th>
<th>SEAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blink</td>
<td>26</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Sense-and-send</td>
<td>93</td>
<td>38</td>
<td>3</td>
</tr>
<tr>
<td>Event detection</td>
<td>42</td>
<td>22</td>
<td>7</td>
</tr>
</tbody>
</table>

Secondly, four agriculture scientists from Latvia State Institute of Fruit-Growing were asked to program the same solution using SEAL GUI. After initial help with installation and setup, all were able to complete this assignment on their own or using only discussions with colleagues. Such as result would be unthinkable with C, where they would have to battle with issues such as C pointer semantics and memory alignment.

IV. RELATED WORK

WSN query systems such as TinyDB [5] have the closest relation to our work. However, we believe their SQL-like syntax is not as intuitive for novice users, and, unlike SEAL, they are not agnostic to network protocols. Consequently, these systems are less portable and much harder to implement. Declarative WSN programming systems have been proposed before [6]. However, they were not aimed towards novice users and their Prolog-like syntax is intimidating to all but seasoned CS professionals. As for WSN macroprogramming systems such as Regiment [7] – we consider them as a complementary, not a conflicting solution.

V. CONCLUSION

SEAL has the potential to make WSN application development more accessible to non CS-majors, to make application prototyping faster, while keeping the benefits offered by low duty-cycling and partial reprogramming. It has been evaluated favorably on the target audience.

ACKNOWLEDGMENTS

This work was supported by European Social Fund grant Nr. 2009/0219/1DP/1.1.1.2.0/APIA/VIAA/020. We would like to thank all test subjects and Girts Strazdins from Institute of Electronics and Computer Science for helping to conduct the experimental evaluation.

REFERENCES