Programming Abstractions with Debugging Support for Resource-Constrained Devices

Alexander Bernauer
Department of Computer Science
ETH Zrich
bernauer@inf.ethz.ch

Abstract

Abstractions are crucial in order to manage complex systems. In pervasive computing, though, common programming abstractions tend to be too expensive for the employed resource-constrained devices. In recent years, the wireless sensor network community has proposed several solutions to this problem. However, little has been done to also support debugging on the level of the abstraction. Instead, a developer is forced to understand the lower-level details in order to find and correct defects. This clearly hampers the development of applications. We aim at advancing the state of the art in programming of resource-constrained devices by introducing debugging support for programming abstractions.

Keywords

Programming Abstractions, Debugging, Resource-Constrained Devices

1 Introduction

Pervasive computing aims at enriching modern life with IT services which form a smart infrastructure embedded in everyday environments. With increasingly cheaper, smaller, more efficient and in general more capable hardware, more and more pervasive applications are becoming possible. Examples include healthcare platforms for diabetes management [9] and measurement platforms for human contact networks and epidemiology research [15].

The unobtrusive integration of pervasive applications into the real world often requires tiny, battery-powered devices to cooperate in a distributed fashion over wireless links. Due to these requirements, pervasive applications tend to be complex and thus difficult to develop, deploy and maintain. In order to make the complexity manageable, suitable programming abstractions which help writing and maintaining code for the devices are required. Unfortunately, programming abstractions offered by high-level programming languages tend to require more CPU and memory resources than a pervasive computing device typically offers. As a consequence, pervasive applications are often implemented in the C programming language which offers only few and rather low-level programming abstractions. Thus, most pervasive applications are notoriously hard to write, deploy and maintain.

The need for programming abstractions for resource-constrained devices has since long been recognized by the research community, in particular in the field of *wireless sensor network (WSN)*. WSNs are a pervasive computing technology which supports monitoring of physical phenomena over a potentially large spatial region. Because the peculiarities of pervasive computing which drive the demand for programming abstractions are especially predominant in WSNs, a number of solutions have been proposed in the past few years [19].

Little has been done, though, to also support debugging on the level of the abstraction [19]. As a consequence, software developers often have to understand lower-level details such as the syntax and semantics of generated code in order to trace program errors. As hiding lower-level details is one of the main goals of abstractions, we consider programming abstractions without debugging support as incomplete.

We thus want to investigate programming abstractions with debugging support for resource-constrained devices. In particular, we focus on WSN because many programming abstractions without debugging support already exist in this field. We furthermore expect our results to be also applicable to other fields of pervasive computing.

2 State of the Art

WSN programming abstractions can be classified by two major orthogonal dimensions: *node-centric* vs. *distributed* and *imperative* vs. *declarative* [19]. Node-centric abstractions focus on programming single nodes which might or might not form a distributed system. In contrast, abstractions for distributed applications account for the distributed nature of the application and support the programming of groups of nodes such as spatial or logical neighborhoods.

Irrespective of the first dimension, with imperative abstractions *"the intended application processing is expressed through statements that explicitly indicate how to change the program state"* [19]. In contrast, for declarative abstractions *"the application goal is described without specifying how it is accomplished"* [19]. Out of the four possible classes of

Doctoral Colloquium, SenSys 2010, November 2nd 2010, Zürich, Switzerland.

This work has been partially supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322, and by CONET, the Cooperating Objects Network of Excellence, funded by the European Commission under FP7 with contract number FP7-2007-2-224053.

programming abstractions only three are relevant in practice and we discuss them in the following.

Protothreads [6] and TinyVT [23] are node-centric, imperative programming abstractions. They offer compiler-based thread abstractions by taking a thread-based program as input and generating a semantically equivalent event-based program for the targeted event-based operating system. While this combines the comfort of thread-based programming with the efficiency of events, both of them have no debugging support. Thus, a developer is forced to step through the generated event-based code in order to identify defects in the thread-based program.

A second class of programming abstractions is imperative and distributed programming abstractions which are commonly known as macro-programming in the WSN community. While the existing abstractions like Kairos [12] and Regiment [20] are powerful, their practical adoption seems to suffer from the lack of debugging support. In this case, besides the knowledge of lower-level details, a software developer additionally needs special tools such as distributed debuggers [25] because a single macro-program controls the behavior of many distributed devices. Compared to node-centric abstractions, the lack of debugging support on the level of the abstraction is therefore even more severe. To the best of our knowledge, the only macro-programming language with debugging support is MacroLab [13] which is a dialect of MATLAB and can be debugged with the Macrodebugger [24], a post-mortem debugger specifically crafted for MacroLab.

Two well-known declarative and distributed programming abstractions are Cougar [26] and TinyDB [18]. They model a WSN as a relational database and support SQL-like queries to retrieve information about the monitored physical phenomena. While this is what users presumably want, if an error occurs somewhere in the distributed system, the user has to resort to other means like passive inspection of the network traffic [21] in order to find the cause of the problem.

Since WSN applications are rather diverse, suitable programming abstractions will have to be specifically crafted for a given application domain. Such programming abstractions are usually implemented by so-called *domain-specific languages (DSL)* [10]. In software engineering there is an ongoing trend towards DSLs in general and DSLs for building DSLs in particular [4, 5, 8, 14, 16]. With the support of these so-called *language workbenches* [11] it is rather easy to quickly introduce DSLs and to implement the corresponding compiler. Again, though, the issue of debugging on the level of the abstraction has not been solved yet.

3 Goals of the Thesis

The goal of the thesis is to introduce debugging support of programming abstractions for resource-constrained devices. To this end, we will focus our investigations on two major topics.

First, as a concrete example we want to introduce a debuggable, compiler-based thread abstraction for event-based operating systems. The expected outcome is that from a developer's perspective there is not much difference from using our compiler to using a thread library. But still, the achieved efficiency is as close as possible to the efficiency of hand-written event-based code. Furthermore, source-level debugging of the thread-based program is possible.

Compared to thread libraries there are theoretical limits of what features a compiler-based approach can offer [3]. These limitations stem from the fact that a compiler can only process decidable problems. In order to stay within these theoretical bounds, we can neither commonly support calling blocking operations by function pointers nor calling blocking functions from recursive functions. Furthermore, dynamically starting new threads is not possible. However, as it is advisable to statically ensure that the scarce resources of a given device suffice the demands of a given program, such dynamic features are rarely used for embedded systems. Thus, we don't consider these limitations to be severe, especially considering that we expect to be able to get close to the efficiency of the event-based paradigm.

Additionally, if the compiler records which thread-based code was translated into which part of the event-based program, it becomes possible to perform source level debugging in the thread-based code. In principle, this technique is very similar to what C tool chains use in order to support source-level debugging of C programs. Thus, we expect to be able to reuse and build upon a large number of existing work such as the DWARF Debugging Standard [7].

For the second part of the thesis, we want to investigate in more general terms how programming abstractions can be designed and built in a way such that debugging support is included. Therefore we want to identify standard debugging techniques for the various types of programming abstractions encountered in the WSN field. By extending existing language workbenches we enable the design and implementation of various types of DSLs with the respective debugging support integrated. The expected outcome is that software developers will be able to easily create programming abstractions like TinyDB or Kairos with integrated debugging support.

For node-centric and imperative DLSs, simple debugging information emitted by the compiler is known to suffice. For distributed DLSs, though, existing debugging tools such as distributed debuggers [25] or passive distributed assertions [22] must be incorporated into the tool chain such that information about an error which is reported from the debugging tool points to the error's cause in the program. In the case of declarative programming abstractions, debugging is even harder because there is no one-to-one mapping between the program and the generated code. Thus, more detailed information is necessary in order to be able to map from the generated code back to the program.

4 Progress to Date

In [2] we presented to the WSN community a first draft of a compilation scheme for compiler-based threads. We then elaborated on the draft and found a complete compilation scheme and a proof of correctness of the transformation. In order to evaluate the efficiency of the generated code we manually applied the compilation scheme to a representative WSN application and measured and compared the code size, the memory consumption and the amount of CPU cycles of

both the generated code and a hand-written variant [3].

This experiment showed that performing the transformation on the level of the C program results in too expensive code. The reason for this is that the smallest callable unit in C is a function which is rather expensive. Instead, we decided to perform the transformation on the level of LLVM [17] bytecode, which can be roughly considered as a portable and typed assembly language. Currently we are working on re-evaluating the performed case study using LLVM.

For the second part of the thesis we discussed the questions concerning debugging support for programming abstractions in [1]. From that we know that little is known in the pervasive computing community about how to support debugging for programming abstractions. So, we plan to start a number of case studies with programming abstractions from different classes in order to identify typical programming tasks and suitable programming abstractions for them. Furthermore, we have recently started a survey of existing language workbenches in order to choose the most promising ones for the demands of the WSN field and for the possibility to add debugging support.

5 References

[1] A. Bernauer and K. Römer. Meta-Debugging Pervasive Computers. The Workshop on Programming Methods for Mobile and Pervasive Systems, 2010.

[2] A. Bernauer, K. Römer, and S. Santini. Threads for the Programmer, Events for the Machine. In *Adjunct Proc. of the 7th European Conference on Wireless Sensor Networks (EWSN 2010)*, Coimbra, Portugal, 2010.

[3] A. Bernauer, K. Römer, S. Santini, and J. Ma. Threads2Events: An Automatic Code Generation Approach. In *Proc. of the 6th Workshop on Hot Topics in Embedded Networked Sensors*, Killarney, Ireland, 2010.

[4] R. Carrara. Actifsource Code Generator for Eclipse. http://www.actifsource.com, 2010.

[5] S. Dmitriev. Language Oriented Programming: The Next Programming Paradigm. *JetBrains OnBoard Online Magazine*, 2004.

[6] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali. Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems. In *Proc. of the 4th Conference on Embedded Networked Sensor Systems*, pages 29–42, 2006.

[7] DWARF Standards Committee. *The DWARF Debugging Standard, Version 4*, 2010.

[8] S. Efftinge and M. Voelter. oAW xText: A Framework for Textual DSLs. In *Workshop on Modeling Symposium at Eclipse Summit*, 2006.

[9] L. N. et al. Jog Falls: A Pervasive Healthcare Platform for Diabetes Management. In *Proc. of the 8th Conference on Pervasive Computing*, pages 94–111, 2010.

[10] M. Fowler. Domain Specific Languages. http://martinfowler.com/dslwip/.

[11] M. Fowler. Language Workbenches: The Killer-App for Domain Specific Languages? http://martinfowler.com/articles/languageWorkbench.html, 2005.

[12] R. Gummadi, N. Kothari, R. Govindan, and T. Millstein. Kairos: A Macro-Programming System for Wireless Sensor Networks. In *Proc. of the 20th Symposium on Operating Systems Principles*, 2005.

[13] T. Hnat, T. Sookoor, P. Hooimeijer, W. Weimer, and K. Whitehouse. Macrolab: A Vector-based Macroprogramming Framework for Cyber-Physical Systems. In *Proc. of the 6th Conference on Embedded Networked Sensor Systems*, pages 225–238, 2008.

[14] L. Kats and E. Visser. The Spoofax Language Workbench. Rules for Declarative Specification of Languages and IDEs. In *Proc. of the 25th Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2010.

[15] M. Kazandjieva, J. W. Lee, M. Salathe, M. Feldman, J. Jones, and P. Levis. Measuring a Human Contact Network for Epidemiology Research. In *Proc. of the 6th Workshop on Hot Topics in Embedded Networked Sensors*, 2010.

[16] S. Kelly and J. Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society, 2008.

[17] C. Lattner and V. Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. *Code Generation and Optimization, IEEE/ACM International Symposium on*, 0:75, 2004.

[18] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. The Design of an Acquisitional Query Processor for Sensor Networks. In *Proc. of the Conference on Management of Data*, pages 491–502, 2003.

[19] L. Motolla and G. Picco. Programming Wireless Sensor Networks: Fundamental Concepts and State of the Art. *ACM Computing Surveys*, to appear.

[20] R. Newton, G. Morrisett, and M. Welsh. The Regiment Macroprogramming System. In *Proc. of the 6th Conference on Information Processing in Sensor Networks*, pages 489–498, 2007.

[21] M. Ringwald and K. Römer. SNIF: A Comprehensive Tool for Passive Inspection of Sensor Networks. *6. GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze"*, pages 59–62, 2007.

[22] K. Römer and J. Ma. PDA: Passive Distributed Assertions for Sensor Networks. In *Proc. of the Conference on Information Processing in Sensor Networks*, pages 337–348, 2009.

[23] J. Sallai, M. Maróti, and A. Lédeczi. A Concurrency Abstraction for Reliable Sensor Network Applications. In *Proc. of the 12th Conference on Reliable Systems on Unreliable Networked Platforms*, pages 143–160, 2007.

[24] T. Sookoor, T. Hnat, P. Hooimeijer, W. Weimer, and K. Whitehouse. Macrodebugging: Global Views of Distributed Program Execution. In *Proc. of the 7th Conference on Embedded Networked Sensor Systems*, pages 141–155, 2009.

[25] J. Yang, M. Soffa, L. Selavo, and K. Whitehouse. Clairvoyant: A Comprehensive Source-Level Debugger for Wireless Sonsor Networks. In *Proc. of the 5th Conference on Embedded Networked Sensor Systems*, pages 189–203, 2007.

[26] Y. Yao and J. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. *ACM SIGMOD Record*, 31(3):9–18, 2002.

6 Biography

Alexander Bernauer graduated in Computer Science from the University of Ulm, Germany in 2006. Subsequently he was working as a C++ software engineer developing tracking and navigation products at a start-up company in Heerbrugg, Switzerland. Since 2009 he has been a Ph.D. candidate under the supervision of Prof. Friedemann Mattern at the Institute of Pervasive Computing, ETH Zürich, Switzerland. His expected date of dissertation is February 2013.