Domain-Specific Languages for Wireless Sensor Networks

Daniel A. Sadilek

Humboldt-Universität zu Berlin
Institute for Computer Science
Rudower Chaussee 25
12489 Berlin, Germany

sadilek@informatik.hu-berlin.de

Abstract: Programming wireless sensor networks is difficult. Using low-level languages like C or Assembler requires detailed knowledge about the hardware and its limitations. The behavior of a sensor node has to be described in terms of memory addresses, registers, and very basic data manipulation operations. Domain-specific languages help to ease the programming. They raise the programming abstraction and, for instance, allow to describe a sensor node's behavior in terms of current sensor values and sensor value change events. By abstracting from low-level details, a domain-specific language may even enable domain-experts to describe the behavior of a wireless sensor network. The goal of my thesis is to develop and evaluate an approach for defining domain-specific languages for wireless sensor networks and for simulating, compiling, and executing programs formulated in these domain-specific languages.

1 Introduction and motivation

Wireless sensor networks (WSNs) consist of resource-constrained microcontroller devices, called *WSN nodes*, that communicate wirelessly and are battery-powered. They have sensors for, e.g., temperature, humidity, and acceleration and can detect environmental phenomena in a distributed fashion.

Software development for WSNs, if done with low-level languages like C or Assembler, is very difficult. It requires detailed knowledge of the very limited hardware; testing and debugging programs on the sensor nodes is very expensive and difficult. To overcome these problems, multiple programming languages have been proposed that are tailored specifically to the requirements of WSNs (nesC [GLvB+03], for instance).

These programming languages are mainly targeting at computer scientists. But WSNs are embedded in an environment and usually computer scientists are not experts for this environment and do not know what the sensor network should do exactly. Consider, for example, a WSN that is deployed across a city and monitors ground acceleration to detect earthquakes. A seismologist knows how an earthquake propagates and what kind of signal processing of the acceleration measurements has to be performed to detect earthquakes.

But a typical seismologist does not know how to describe this knowledge with one of the general-purpose programming languages for WSNs. Therefore, he usually explains his knowledge to a programmer who creates the software.

If the seismologist—or *domain expert*, in general—had a programming language to describe his knowledge, there wouldn't be the need to explain all the domain knowledge to the computer scientist. Such a language is a *domain-specific language* (DSL). It is not tailored to programming WSNs in general but to a specific application area. A DSL provides the domain expert with terms and notations that match his cognitive space and intuition. A seismologist with background in signal processing, for example, could use a stream-oriented DSL that allows him to formulate an earthquake detection algorithm in terms of streams that come out of stream sources, go through filters and then into sinks. For this DSL, a graphical notation showing the stream flow will probably be appropriate.

In the following two sections, I state the dissertation problem and my approach. In Section 4, I present the results of my work to date. I discuss future work in Section 5.

2 Problem statement

The goal of my thesis is to develop and evaluate an approach for defining domain-specific languages for wireless sensor networks and for simulating, compiling, and executing programs formulated in these domain-specific languages.

This goal can be divided into two sub-problems: (1) the prototyping and simulation of DSLs and (2) the execution of these languages on resource-constrained WSN nodes.

An approach that solves both sub-problems has to provide[1]

- a way to define the *abstract syntax* of a DSL,
- a way to define a *purpose-built concrete syntax* of a DSL,
- a way to define the *execution semantics* of a DSL,
- the possibility to *simulate* DSL programs in the development environment, and
- the possibility to *execute* DSL programs on the WSN nodes.

Besides these functional requirements, it should be *cheap to define and use* a DSL. DSLs have a narrow application domain when compared to general-purpose programming languages. If the definition of a DSL costs more than what can be saved by using it, developing the DSL does not pay off. Making DSLs cheap to define and use means to allow using them not only in widespread application domains like database querying (SQL) but also for small projects and very narrow application domains like earthquake early warning. Thus, DSLs aren't necessarily implemented by some big software vendor but by developers that use the DSL to raise the abstraction and structure the program code of a project they are working on.

[1] due to space given without justification

3 Related work and my approach

Metamodeling frameworks are a common tool to define DSLs. A metamodel defines the abstract syntax of a DSL. On this basis, a purpose-built concrete syntax can be defined and a corresponding graphical editor for a DSL can be created [GMF07, LBM+01]. Furthermore, the execution semantics of a DSL can be defined and DSL programs can be simulated [MFJ05, CESW04]. Finally, code can be generated from DSL programs [Tol04, LBM+01].

Normally, an interpreted description of the execution semantics is used to simulate DSL programs in the development environment and some form of transformation or code generation is used to make DSL programs executable on the target platform. In the existing approaches, this leads to redundancy if both simulation and execution on the target platform are necessary. My approach avoids this redundancy and uses just one description of the execution semantics for both purposes.

For this, I extend *EMOF* with classes for specifying the execution semantics of a DSL, similar to the approaches of Kermeta [MFJ05] or the Mosaic Framework [CESW04]. In contrast to these approaches, I extend EMOF with operations that can be implemented in a *lambda-calculus based language* to describe the execution semantics of a DSL. The description of a DSL's execution semantics can then be used in two ways. First, it can be interpreted in the development environment and can be bound to a *discrete-event simulation kernel*, which allows for simulating a WSN consisting of multiple nodes. Second, it can be compiled for the target platform.

Knowledge how to compile lambda-calculus based languages to efficiently executable code is available—for instance, for the *Scheme programming language*. My idea is to leverage this knowledge by compiling DSL programs to a form directly executable on WSN nodes in two steps:

(1) A compiler translates the DSLs metamodel, its execution semantics and the DSL program to Scheme. For this, it will be necessary to represent the object oriented structure of the DSL in Scheme. Fortunately, knowledge how to efficiently and flexibly represent object oriented programs in lambda-calculus based languages is also available in form of the *Common Lisp Object System*. The metamodel classes will be compiled to classes of a Scheme object system (that reflects EMOF's capabilities). The execution semantics description is compiled to procedures in the object system and the DSL program itself is compiled to initialization code that instantiates the compiled classes.

(2) The resulting Scheme program could be compiled with a standard Scheme compiler. However, deployment of large binary programs costs much of the node's scarce energy and WSN nodes typically don't have memory protection or preemptive multitasking. Therefore, it will be necessary to develop a special compiler that is geared towards WSN nodes and compiles DSL programs to *bytecode*, which is small and which can be executed safely in a *virtual machine* on the WSN node. Applying existing optimization techniques for Scheme compilation can help to minimize the size of the bytecode further.

4 Results so far

First, I analyzed the integration of WSNs into *disaster management* information systems and identified the need to provide experts of the domain disaster management with DSLs [STW06].

Based on Eclipse EMF as metamodeling framework, I implemented a first version of my approach [Sad07c, Sad07b] in which EMF and Scheme were not yet integrated as tightly as presented in this paper: the DSL program in the Scheme representation was not object-oriented and the first compilation step not a generic one. However, this first prototype successfully demonstrated the possibility to combine EMF and Scheme. As an example usage, I defined a *stream-oriented DSL* for the description of earthquake-detection algorithms. The prototype already contains a discrete-event simulation kernel that allows to simulate the stream-oriented DSL in the development environment. I used an existing Scheme to C compiler to make the stream-oriented DSL executable on a Wifi-Router (that had significantly more resources than the WSN nodes I am actually targeting at).

For the compilation for WSN nodes, I developed a mathematical model for optimizing the energy usage of WSN nodes having a virtual machine that can be extended with native code [Sad07a]. The model formalizes the energy trade-off between slow but small bytecode and fast but big native code. It can be used to decide where the boundary between bytecode and native code should lie and thus allows energy-aware compilation.

5 Future work

Currently, I am working on a specification of the virtual machine as an Abstract State Machine. The main task here is to define its instruction set. It should be efficiently executable and should allow for concise bytecode. The specification will serve as an unambiguous documentation of the virtual machine, as a reference for a native implementation, and as a basis for testing the compiler for the second compilation step.

Another task is to implement this compiler. Here the following questions arise: Is a generic compiler actually possible? Are extensions of the Scheme programming language necessary? Are extension points for the compiler necessary to support different DSLs? If yes, how do they look like?

Later, I will implement the virtual machine. Here, one problem will be providing automatic memory management for the WSN nodes because the instruction set will probably rely on automatic memory management.

Finally, I will evaluate my approach. On the one hand, I will measure—for a simple application—the runtime overhead that is introduced by my approach compared to a native C implementation in terms of processing time and energy consumption. On the other hand, more DSLs will be necessary to evaluate the broader applicability of my approach. Up to now, I use earthquake detection as a scenario and a stream-oriented DSL as an example DSL. Other scenarios like a chemical plant in which chemicals have to be monitored or

fire monitoring in a forrest give rise to other example DSLs.

A future task, which may be beyond my dissertation, will be to extend the compiler so that it can generate native code for selected code parts and to extend the simulation with energy estimations. This will be the basis for implementing energy-aware compilation.

Literatur

[CESW04] T. Clark, A. Evans, P. Sammut und J. Willans. *Applied metamodelling: A foundation for language driven development*. `www.xactium.com`, 2004.

[GLvB+03] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer und David Culler. The nesC language: A holistic approach to networked embedded systems. In *PLDI '03*, Seiten 1–11, New York, NY, USA, 2003. ACM Press.

[GMF07] The Eclipse Foundation, `http://www.eclipse.org/gmf/`. *Eclipse Graphical Modeling Framework (GMF)*, May 2007.

[LBM+01] Ákos Lédeczi, Árpád Bakay, Miklós Maróti, Péter Völgyesi, Greg Nordstrom, Jonathan Sprinkle und Gábor Karsai. Composing Domain-Specific Design Environments. *Computer*, 34(11):44–51, 2001.

[MFJ05] Pierre-Alain Muller, Franck Fleurey und Jean-Marc Jézéquel. Weaving Executability into Object-Oriented Meta-Languages. In *MoDELS '05*, Seiten 264–278, Berlin, Germany, October 2005. Springer Verlag.

[Sad07a] Daniel A. Sadilek. Energy-aware compilation for Wireless Sensor Networks. In *MidSens '07: Proceedings of the International Workshop on Middleware for Sensor Networks*, Newport Beach, USA, Nov 2007. ACM Press.

[Sad07b] Daniel A. Sadilek. Prototyping and Simulating Domain-Specific Languages for Wireless Sensor Networks. Informatikbericht 217, Humboldt-Universität zu Berlin, Berlin, Germany, Nov 2007.

[Sad07c] Daniel A. Sadilek. Prototyping Domain-Specific Languages for Wireless Sensor Networks. In *ATEM 07: 4th International Workshop on Software Language Engineering*, October 2007.

[STW06] Daniel Sadilek, Falko Theisselmann und Guido Wachsmuth. Challenges for Model-Driven Development of Self-Organising Disaster Management Information Systems. In *IRTGW'06: Proceedings of the International Research Training Groups Workshop, Dagstuhl, Germany*, Jgg. 3, Seiten 24–26, 2006.

[Tol04] Juha-Pekka Tolvanen. MetaEdit+: domain-specific modeling for full code generation demonstrated [GPCE]. In *OOPSLA '04*, Seiten 39–40, New York, NY, USA, 2004. ACM.