S-DSL: Domain Specific Language for the Programming of Sensor Devices of Environmental Variables

Claudio Omar Biale, Gladis Marleni Sequeira

School of Exact, Chemical and Natural Sciences
Misiones National University
claudio_biale@yahoo.com.ar, gladis.sequeira@gmail.com

Abstract. The aim of this paper is to present S-DSL, an external domain-specific language for programming sensor nodes, which seeks to facilitate the programming of devices using easily understood language that allows the developer to focus on the states which can pass a node and the actions to be developed in each state.
Keywords: sensing devices, domain specific languages, monitoring systems.

1 Introduction

The programming of sensor devices is difficult due to the use of programming languages like C or NesC [1], it is also necessary to know the device hardware and its limitations.

We believe that programming languages of sensor networks have to be simple and easy to use for people with little or no programming experience, avoiding exposing low-level issues such as resource management or communication protocols to users.

With S-DSL, an external domain-specific language for programming sensor nodes, we seek to abstract from low-level issues to the developer, allowing them to focus on the possible states of the sensor node and the actions to be performed by the node according to its current state.

A domain-specific language [2, 3] is a programming language of limited expressiveness focused on a particular domain; we can classify it according to how it is implemented in:

- Internal: it uses the infrastructure of an existing programming language for building domain-specific semantics.
- External: it has its own syntax and a parser is required to process them, their development is similar to the implementation of a new language with its own syntax and semantics.

The advantages of using a domain-specific language to include [3, 4]:

- They are more concise and easier to maintain.
- Allow solutions which are expressed in the language and the level of abstraction of the domain in which the language is focused.

- The experts of such domain can help validate and develop the language.

Among the disadvantages we can mention [4]:

- The cost of design and implementation of the language.
- The cost of teaching a new language to users.
- The potential loss of efficiency of the generated code when compared with the code written by an expert programmer.
- Know how to refine the domain in which the language will be focused.

2 Implementation and Hardware Support

The compiler S-DSL was implemented in Python language, using the PLY library for lexical, syntactic and semantic analysis.

Programs written in S-DSL will be interpreted at compile time to generate a C language program which will then be compiled as a result obtained by mspgcc binary code for the platform that supports the sensor nodes.

Everything that has been indicated in the previous paragraph can be seen in Figure 1.

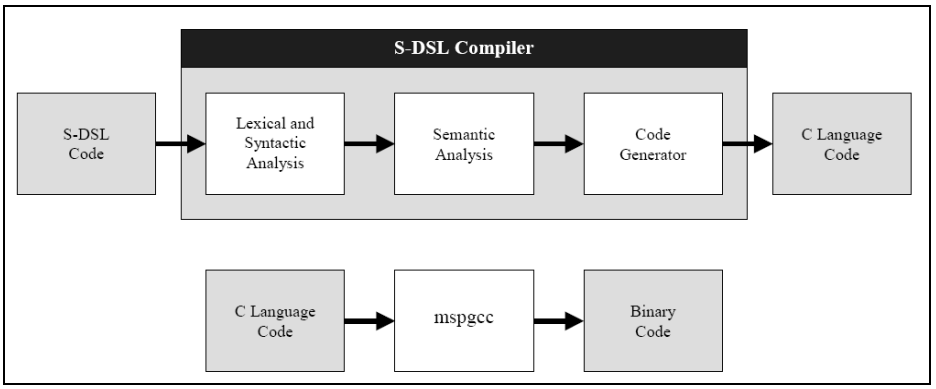

Figure 1. Compilation Process

The resulting code should be tested in a sensor network with star topology, in which each sensor node is based on an MSP430[1] microcontroller and also has:

- A A110LR09A[2] module for wireless communication,
- a digital temperature and humidity MaxDetect RHT03[3] sensor and
- a photo-cell GL5528[4] for light detection.

[1] http://www.ti.com/lit/ug/slau144i/slau144i.pdf
[2] http://www.anaren.com/sites/default/files/user-manuals/A1101R09x_Users_Manual.pdf
[3] http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Weather/RHT03.pdf
[4] http://mdfly.com/Download/Sensor/PD0001.pdf

The code generated in C language by the compiler must include the necessary instructions for obtaining sensor data and it will use the LarsRF library functions[5] for wireless communication.

3 Description of the Language

The language has the format presented in Figure 2; by using it you can define multiple states and a single initial state.

```
{STATE name_state:
EVERY time
SELECT {variable [, variable] ...}
[SENDIF send_condition]
[CHANGEIF change_condition GOTO new_state];
} ...

START IN initial_state;
```

Figure 2. Language Format

The initial state of the program is specified by the `START IN` instruction; it must go after the definition of the states.

Each state is defined by the `STATE` instruction accompanied by a name that uniquely identifies it within the program, in the specification of each state the clauses `EVERY` and `SELECT` are compulsory and the clauses `SENDIF` and `CHANGEIF` are optional.

By using these clauses you determine how often (counting the seconds) to do the reading of node sensors, which variables are to be obtained, the conditions to be met in order to send the data to the central node and what conditions trigger a state change. The `CHANGEIF` clause is only not specified when there is a single state.

The types of variables to be obtained are represented in the language by:

- `TEM` (Temperature)
- `HUM` (Humidity) and
- `LUM` (Luminosity).

Within the specification of the conditions for sending data or change of state you can access to the value obtained in the previous cycle or the last sent value of each variable by prefixing the adjectives `OLD.` and `SND.` to the name of the variable.

[5] https://github.com/mobilars/LarsRF-mspgcc

4 Practical Example

In Example 1 shows an S-DSL programme, in which it is defined the states *e_uno* and *e_dos*, specifying *e_uno* as initial state.

```
STATE e_uno:
    EVERY 4
    SELECT
        tem, hum
    SENDIF
        tem >= OLD.tem * 2 OR
        hum > SND.hum
    CHANGEIF
        tem < 10 GOTO e_dos;

STATE e_dos:
    EVERY 5
    SELECT
        tem, lum
    CHANGEIF
        tem >= 10 GOTO e_uno;

START IN e_uno;
```

Example 1. S-DSL Programme

In the *e_uno* state it is defined that every four seconds it should obtain the values of temperature and humidity for each sensor, sending the data if the temperature value exceeds one hundred percent to the value of temperature obtained in the previous cycle or if the humidity value is greater than the last value of moisture sent, the e*_dos* state change occurs when the temperature drops to less than ten degrees Celsius.

In the *e_dos* state the temperature and luminosity values should be obtained every five seconds and then send those values to the central node, changing to the *e_uno* state when the temperature obtained is greater than or equal to ten degrees Celsius.

5 Related Work

Programming languages at the network level treat the network as a single logical machine. WASP [5] is composed of two segments, the code segment at sensor node level specifies the node's behaviour and the code segment at the network level defines how data is transmitted and grouped. In TinySQL [6], Cougar [7] and IrisNet [8] programmers see the sensor network as a database and use queries to determine the overall system behaviour.

Programming languages at node level consider the network as a set of entities that communicate, where it is necessary to specify the behaviour of each node and its way of communication. SensorBASIC [9] was based on uBASIC interpreter [10] to which they added instructions for the development of sensor network applications. TinyScript [11, 12] is an event-based imperative language similar to BASIC. Mottle [12] is inspired by Scheme and it has syntax similar to C. Mottle and TinyScript are supported by the Maté virtual machine [13]. TinyGALS [14] allows the development of modules formed by components; each component has a set of internal variables, external variables and methods. SNACK [15] is composed of a configuration language, a compiler and a library of components and services; it allows the creation of applications by combining services. NesC and C are languages used to program at node level; NesC is based on events and allows component-oriented application design, C needs specific libraries for managing the communication between network nodes.

6 Conclusions and Future Work

In this paper S-DSL is presented, which is an external domain specific language for programming sensor devices. End users of the sensor network will be able to program the nodes simply by specifying the various states through which the sensor node could pass and the actions to be performed in every state.

As future works it is foreseen:

- Complete the code generation for the MSP 430 platform.
- Implement an integrated development environment for the language.
- Testing on the target platform.

Acknowledgments

We gratefully acknowledge the assistance of María Fabiana Picasso for the translation of this paper.

References

1. Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E., Culler, D.: The nesC Language: A Holistic Approach to Networked Embedded Systems, In: Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation, pp. 1--11, ACM, New York (2003).
2. Fowler, M., Domain Specific Language, http://www.martinfowler.com/bliki/DomainSpecificLanguage.html
3. Hudak, P.: Domain-Specific Languages. In Salus, P. H. (ed.) Handbook of Programming Languages Vol. III: Little Languages and Tools, pp. 39--60. McMillan Technical Publishing, Indianapolis (1998).

4. van Deursen, A., Klint, P., Visser, J.: Domain-Specific Languages: An Annotated Bibliography. In: SIGPLAN Notices 35(6), pp. 26--36, ACM, New York (2000).
5. Bai, L. S., Dick R. P., Dinda P. A.: Archetype-Based Design: Sensor Network Programming for Application Experts, not Just Programming Experts. In: Proceedings of the 2009 International Conference on Information Processing in Sensor Networks, pp. 85--96, IEEE Computer Society, Washington (2009).
6. Madden, S. R., Franklin, M. J., Hellerstein, J. M., Hong, W.: TinyDB: an Acquisitional Query Processing System for Sensor Networks. In: ACM Transactions Database Systems 30(1), pp. 122--173, ACM, New York (2005).
7. Yao, Y., and Gehrke, J.: The Cougar Approach to In-Network Query Processing in Sensor Networks. In: SIGMOD Record 31(3), pp. 9--18, ACM, New York (2002).
8. Gibbons, P. B., Karp, B., Ke, Y., Nath, S., Seshan, S.: IrisNet: An Architecture for a Worldwide Sensor Web. In: IEEE Pervasive Computing 2(4), pp. 22--33, IEEE Educational Activities Department, Piscataway (2003).
9. Miller, J. S., Dinda, P. A., Dick, R. P.: Evaluating a BASIC Approach to Sensor Network Node Programming. In: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, 155--168, ACM, New York (2009).
10. Dunkels, A., uBASIC. http://dunkels.com/adam/ubasic/
11. Levis, P., The TinyScript Language., http://www.eecs.berkeley.edu/~pal/mate-web/files/tinyscript-manual.pdf
12. Levis, P., Gay, D., Culler, D.: Bridging the Gap: Programming Sensor Networks with Application Specific Virtual Machines. Technical report, UC Berkeley (2004).
13. Levis, P., Culler, D.: Maté: A Tiny Virtual Machine for Sensor Networks. In: Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems, 85--95, ACM, New York (2002).
14. Cheong, E., Liebman, J., Liu, J., Zhao, F.: TinyGALS: A Programming Model for Event-Driven Embedded Systems. In: Proceedings of the 2003 ACM symposium on Applied computing, 698--704, ACM, New York (2003).
15. Greenstein, B., Kohler, E., Estrin, D.: A Sensor Network Application Construction Kit (SNACK). In: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, 69--80, ACM, New York (2004).