YETI: A TinyOS Plug-in for Eclipse

Nicolas Burri
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland
burri@tik.ee.ethz.ch

Roland Schuler
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland
rschuler@student.ethz.ch

Roger Wattenhofer
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland
wattenhofer@tik.ee.ethz.ch

ABSTRACT

Wireless ad hoc and sensor networks are constantly gaining importance due to their wide range of possible applications. The employed sensor hardware and software is becoming more complex and projects realized by means of sensor networks are increasingly ambitious. Despite these rapid advantages tools for sensor network development are still very primitive: A generic text editor and a command line shell are the common tools used by the majority of sensor network developers.

In this paper we present YETI, a TinyOS plug-in for Eclipse. Besides the basic functionality known from other development environments YETI features a set of specially optimized tools for TinyOS development.

Categories and Subject Descriptors

D.2.3 [**Coding Tools and Techniques**]: Program editors

Keywords

Sensor network development, TinyOS-1.x, Eclipse plug-in

1. INTRODUCTION

Since different fields of applications for sensor networks oftentimes require specific hardware, a lot of work has been spent on building highly efficient nodes. Various node families have been developed with different design goals and target applications in mind. A perfect node, ideal for all tasks cannot be designed. However, today's platforms offer sophisticated solutions for most requirements, be it a large set of preinstalled sensors [11], simple extensibility [2, 13], a long transmission range [13], multiple on-board radio devices [1], or an integrated USB interface [8]. Also TinyOS [4], the de facto standard operating system for sensor networks, is under constant development. An active community of developers improves the system's performance and extends its functionality.

Despite the large user base and the advances in hardware and software, development tools for TinyOS are still rare and often severely limited in terms of functionality. Hence, TinyOS developers are forced to use generic text editors for writing their applications. A command line shell is required to compile code and flash nodes with the resulting binaries. The lack of convenience functions such as real time spell checking, code completion, or even a correct syntax highlighting makes the development of TinyOS applications an unnecessary cumbersome task.

In this paper we present YETI[1], an Eclipse plug-in providing support for TinyOS development from within the Eclipse framework [9]. YETI provides all important features known from development environments for other programming languages and is designed to be of use for both, unexperienced and professional sensor network developers.

The remainder of this paper is organized as follows: In Section 2 requirements of TinyOS developers are discussed. Section 3 introduces YETI and presents its most important features. In Section 4 the underlying technical aspects of the system are discussed. The subsequent Section compares our tool to related work. Finally, Section 6 concludes the paper and gives an outlook on future work.

2. DEVELOPMENT REQUIREMENTS

From our own experience and numerous discussions with other TinyOS and sensor network developers we found that the requirements of TinyOS newcomers and experienced developers vary. Newcomers who have little experience in writing sensor network applications need help on fundamental aspects of TinyOS development. For one, getting used to the design philosophy of the operating system is not easy. Especially its completely modular application design and the unique way of combining modules by means of a so called *wiring* require some time to get used to. The well written tutorial on the TinyOS homepage helps to overcome this steep learning curve since all important features of TinyOS and its programming language *nesC* [3] are discussed. Still, it is not unusual for the first contact with TinyOS to be discouraging. The installation of the system and the necessary toolchains often leads to problems. If the provided installer fails, repairing a new TinyOS installation requires a sound knowledge of the system which unexperienced developers have not yet achieved.

Another problem for new TinyOS developers is the vast amount of files included in the sources of the system. TinyOS features numerous modules solving many common tasks. Alas, it is often difficult to find the correct files providing the

[1] **Y**ETI is an **E**clipse based **T**inyOS **I**DE

REALWSN'06, June 19, 2006, Uppsala, Sweden.

required functionality and thus newcomers show a tendency to ignore them. Experienced developers are more aware of these sources. Yet, in spite of their in depth knowledge of the system they spend a significant amount of their development time browsing through various TinyOS directories looking for adequate implementations of the functions they need.

For more ambitious projects also aspects such as rapid prototyping, cross platform development, and support for backup and version control systems are of greater importance. Furthermore, the possibility of having several parallel installations of the TinyOS source tree is critical for many developers. On the one hand a snapshot installation provides a stable development environment while on the other hand a Concurrent Version System (CVS) checkout of the operating system allows using bleeding edge technology which has not yet made it into the stable release.

3. FEATURES

The goal of YETI is to provide an efficient development tool for experienced users and a convenient, easy to use environment for newcomers. Consequently, all requirements mentioned in Section 2 have to be considered. Also aspects such as Look-and-Feel are of importance if a large number of users is to work with the tool. We therefore decided to build YETI on top of the widely used Eclipse framework [9]. Eclipse provides a powerful plug-in mechanism allowing nearly unlimited extensions and enhancements of its inbuilt functionality. Due to this ease of extensibility Eclipse has become first choice for many developers and plug-ins supporting various programming languages such as C(++), Fortran, or Cobol have been written. Furthermore, Eclipse is designed to allow easy incorporation of existing features in a new plug-in. Consequently, YETI benefits from various existing Eclipse components such as the basic editor, the persistency system, or the CVS client. Also updating the plug-in is possible using Eclipse's update mechanism.

YETI consists of two Eclipse plug-ins: The *System* plug-in containing the functionality of the development environment and the the *TinyOS Environment Wrapper* plug-in providing access to a TinyOS installation. In the subsequent sections we will discuss these two plug-ins in more detail.

3.1 System Plug-in

The System plug-in provides the actual programming environment and tools for TinyOS development in Eclipse. As can be seen in Figure 1, once YETI is installed a new custom *TinyOS perspective* becomes available. This perspective is optimized for the task of writing sensor network programs and hosts helpful features for all stages of development.

Project Creation

New projects are created using the *TinyOS project wizard*. In a short dialog the user can name the new project, choose one of the available TinyOS installations (also see Section 3.2), and define a default make target. This target specifies which sensor node platform to use as a default if no other arguments are specified. YETI does not offer a hard coded list of targets but queries the TinyOS make system for supported devices. This guarantees that for each installation of the TinyOS system all supported node platforms are available to the user.

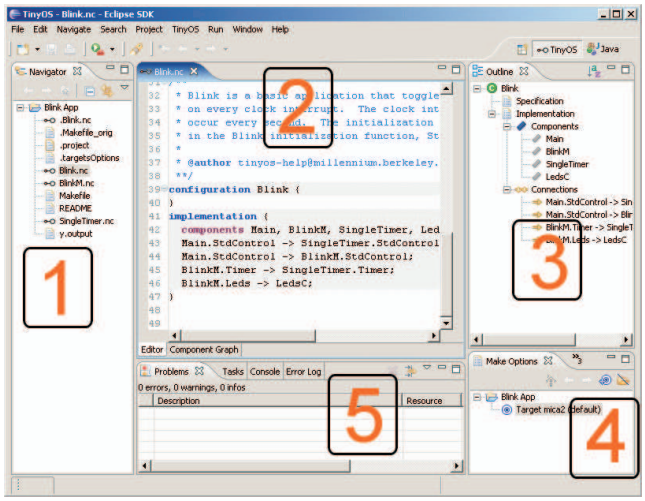

Figure 1: Screenshot of YETI. 1: Navigator listing all files of the project. 2: Main window showing the editor with the current file or the application graph. 3: Outline of the open file showing its structure. 4: Make option window containing predefined make targets. 5: Multi purpose panel hosting various features such as problem view, console and TinyOS search.

File Editing

For the development of applications YETI provides a customized editor supporting the nesC programming language. It features a correct syntax highlighting and incorporates various commodity functions known from other development environments. Its most important feature is definitively the real time spell checker. Syntactic and semantic errors are detected within a fraction of a second and are marked with a red X at the beginning of the line. Furthermore, an error message is generated in Eclipse's Problems log containing clickable links pointing to the corresponding location in the source code. For the most common problems such as missing semicolons the error messages also offer a suggestion on how to fix them (also see Section 4.2). Moreover, the editor contains a code completion function which can be used to create stubs for methods which have to be implemented in a file providing or using a specific interface.

Outline

For a better overview of the application YETI provides an *Outline* of the open file. This outline lists all *components*, *interfaces*, *modules* and *configurations* which are the building blocks of every TinyOS applications. With a simple click it is possible to open the declaration of an interface or to jump to a specific function within the open source file. Due to the lack of an explicit package structure within TinyOS multiple implementations of the same interface may be available. This is mostly the case if hardware specific features are used. For example, accessing hardware timers works differently on different processors and thus the *Timer* interface used in nearly all TinyOS applications has many custom implementations for the various sensor node platforms. YETI uses the currently chosen target platform to decide which of the various files to open. Consequently, the user always

sees the implementation which will be used to compile the application.

TinyOS Specific Search

A TinyOS specific search function allows browsing through all available interfaces and to scan for modules implementing them. For this purpose the structure of the appropriate source files is parsed and evaluated. Several special search modes are available. *Interfaces*, *modules*, and *configurations*[2] can be listed, filtered and accessed from the search frame.

This feature is especially helpful since TinyOS uses a complex set of rules to decide which modules to include when compiling an application. YETI's search function follows all valid paths, including custom imports made by the user, to find files matching the entered search queries. This ensures that all valid files are found but no sources incompatible to the current make target are shown.

Compiling and Flashing

For compiling applications and flashing sensor nodes with the resulting binaries YETI relies on the TinyOS make system. However, users are no longer required to type in cryptic command line calls but a simple wizard helps setting up make options and stores them for later reuse. YETI automatically identifies all available target platforms for a given TinyOS installation and also examines further valid parameters such as possible extension boards. The identified options are displayed in a dialog and the user can create even complex make calls by means of simple point and click operations.

Feedback on the results of a call to the make system are printed to Eclipse's built in console. This user interface is not only easier to utilize but it also prevents the generation of invalid make calls. Furthermore, YETI allows batch execution of the make system simplifying the tedious process of reprogramming large numbers of nodes.

Application Graph

Another feature of the System plug-in is the *Application Graph*. This tool produces a graphical representation of the currently developed application and can be used to plot the relation between its modules. Exploiting the hierarchical structure of TinyOS modules the user can decide on the graph's level of abstraction by expanding or collapsing some of the elements . If required it is possible to expand the graph to show *all* modules forming the current application including the ones of the operating system. However, as can be seen in Figure 2 even simple programs such as the *Blink* demo application lead to complex graphs if fully expanded. Therefore, in most cases it is advisable to keep a certain level of abstraction to view the structure of a program.

3.2 TinyOS Environment Wrapper

The only task of the TinyOS Environment Wrapper is to provide the System plug-in with a well-defined access to a TinyOS installation. This separation of development environment and TinyOS system has several advantages. First, it allows having several independent installations of TinyOS for different target platforms. This is desirable since in many cases the tool chain necessary to compile applications for one sensor node platform interferes with the tools for another one. This problem is one of the main reason why it is so tedious to test newly written applications on various nodes. With YETI it is a matter of one click to change between the different available TinyOS environments and to test the application on all available node types.

Another advantage of this separation is that hardware producers may provide their own TinyOS Environment wrappers. With such individually optimized TinyOS installations it can be ensured that the application developers work on a correctly configured environment. For the software developers this approach has the advantage that a newly installed environment will not interfere with or even destroy existing TinyOS installations as it is sometimes the case without YETI.

The only drawback of this approach is the increased hard disk space necessary to install several independent TinyOS environments. However, with the ever-growing hard disk sizes it should not be a problem to have several installations with a total size of one to two gigabytes.

Currently, YETI provides three different TinyOS Environment Wrappers[3]. The first one contains a full installation of the current TinyOS 1.1.15 release. It is the best choice for most developers as it provides a stable environment with support for various sensor node platforms. The second wrapper provides an installation optimized for the TinyNode 584 platform by Shockfish SA. Finally, an "empty" skeleton wrapper is available which allows to connect YETI to existing TinyOS installations. This is the only wrapper requiring manual configuration since the user needs to enter the path to some few important TinyOS directories.

4. CODE ANALYSIS

Most features such as the "Code Outline" or the spell checker require a syntactic and semantic understanding of the application which can only be achieved by scanning and parsing the source code. These operations need to be executed nearly in real time since users are unwilling to wait for several seconds before a new input is validated. At the same time the results have to be correct or the development environment produces false alerts, making it mostly useless to the developer.

4.1 Scanner and Parser

The analysis of source code is traditionally split in three phases: lexical, syntactic, and semantic analysis. In the phase of the lexical analysis the nesC source files are tokenized. A token is defined as a sequence of logically connected items building atomic structures of the programming language. This includes keywords such as "module" or "implementation" and also strings, numbers, and type names. Tokens are created by comparing the source code to predefined patterns. The tool executing this lexical analysis is called *Scanner* or *Lexer*.

The goal of the syntactic analysis is to group the individual tokens and to validate their correctness according to a given grammar of a programming language. As a result of this analysis a syntax tree is built on which the semantic analysis is executed. In this last step unreasonable code

[2]In TinyOS *configurations* are used to combine several modules to an application or a subprogram.

[3]Environment wrappers are platform dependent since the compilers and tools used to build TinyOS applications are also platform dependent. All currently available wrappers require to be installed on a Microsoft Windows system.

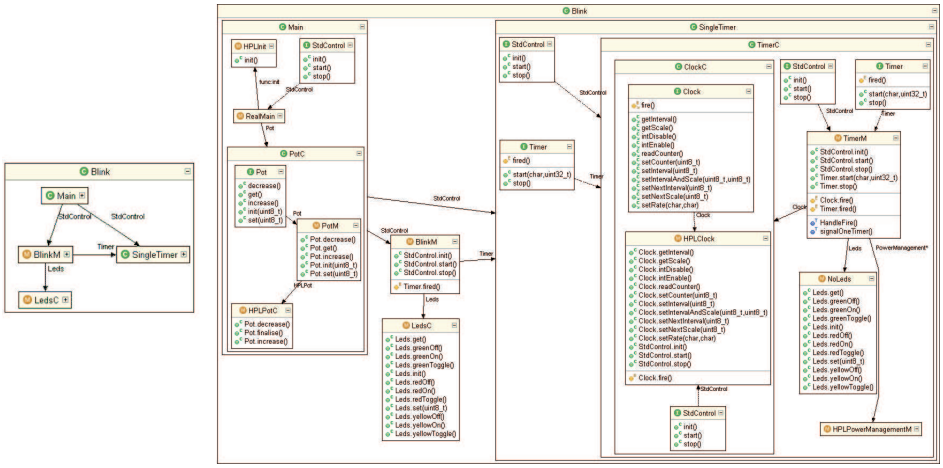

Figure 2: Graph of the Blink application at two different levels of abstraction

which is syntactically correct is identified.

YETI contains a custom scanner and parser which were realized using *JFlex* [6] and *jay* [12], Java implementations of the well known tools *Lex* and *YACC*. Figure 3 shows a schematic representation of the internal interconnections between the parser and the visual tools of the development environment.

For the syntactical analysis a jay specification file was written, based on the nesC language definition found in [3]. With this specification as an input jay was used to create a finite state machine implementing a nesC parser. As can be expected this process was not straight forward. Starting with a YACC specification file for ANSI C the production rules were adapted to model the nesC programming language. Unfortunately, the resulting grammar was highly ambiguous leading to various shift/reduce and reduce/reduce conflicts. These problems had to be solved by major reordering of parser rules.

Another problem arose from jay's limitation to create only standard a LR(1)[4] parser: Due to a conflict between identifiers and typedef-names, C and thus also its derivate nesC are not LR(n) compliant [7]. To avoid this problem extensions to the grammar were necessary. Scoping and obscuring is taken into account in order to produce the correct type of token and to prevent the parser from failing.

Finally, nesC also supports individual name spaces for *configurations* and *interfaces*. Since these constructs are not known in pure ANSI C, the grammar had to be extended to consider these additional name spaces.

4.2 Extending the Parser

Human readable error reports are crucial for any development environment. A simple output saying "Syntax Error" is not really helpful to any developer. What we want is an expressive report about the location and the nature of the problem. Jay already produces quite precise error statements but it is possible to further improve them. YETI provides a powerful mechanism to extend the parser's error messages by feeding it with specially prepared files.

To illustrate the process of adding a new error message

[4]LR indicates that rules are executed from left to right. The number in brackets specifies the number of tokens the parser can look ahead to optimize its decisions;

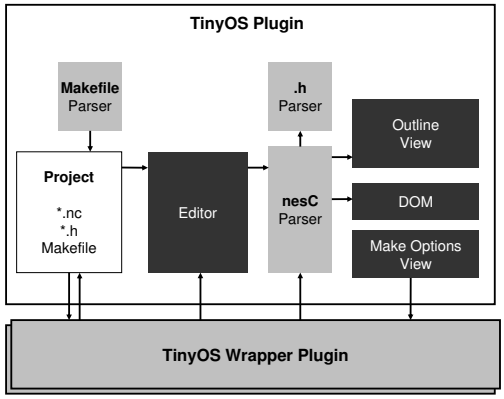

Figure 3: Internal configuration of YETI's components

to the parser Listing 1 shows an adapted configuration file of the *Blink* demo application. This version of the file differs from the original in that on line 1 an error message was added. Furthermore, on line 5 the character '>' was removed.

```
1  :::Wiring symbol '-'unknown, use '<-' or '->'
2  configuration Blink{}
3  implementation {
4   components Main, BlinkM, SingleTimer, LedsC;
5   Main.StdControl - SingleTimer.StdControl;
6   Main.StdControl -> BlinkM.StdControl;
7   BlinkM.Timer -> SingleTimer.Timer;
8   BlinkM.Leds -> LedsC.Leds;
9  }
```

Listing 1: Sample file teaching the parser a new error message if a '>' is missing in the wiring.

This file is now processed by a special tool included in YETI and the parser will analyze its content. The first line is stored as an error report but otherwise the parser ignores it completely. After parsing the rest of the file the error on line 5 is detected. The parser now stores the new custom error message in a consistent hashtable using a combination

of its current internal state and the next expected toksen as a key. Next time the parser encounters the same problem it will check the hashtable for a custom error message. If the table contains an entry for the current parser state the stored message is displayed. If there is no custom message known, the default output of jay is shown. The correctness of this procedure was proven in [5].

5. RELATED WORK

To the best of our knowledge there are only two other projects aiming at providing a development environment for TinyOS. Like YETI both of them are realized as Eclipse plug-ins but they differ in various respects.

The first tool is *TinyOS IDE* [14] by Richard Tynan which was the first publicly available TinyOS development environment. TinyOS IDE provides little advantages over using an advanced text editor and a shell. It provides syntax highlighting for nesC files and the option to compile applications from within Eclipse. However, to enable the compile function, it is necessary to have a preinstalled working TinyOS installation. Also the TinyOS specific environment variables need to be defined system wide or the tool cannot find the compiler. Similarly to YETI, TinyOS IDE allows to compose make calls by selecting the various options from a dialog. TinyOS IDE does not generate this dialog automatically but simply loads the information from a handwritten configuration file. The displayed make options are not guaranteed to be reasonable and if a parameter is required which is not available in the default menu the configuration file must be adapted. TinyOS IDE does not provide a spell checker but after building an application compiler errors are made available in Eclipse's error log.

The second tool is called *TinyDT* [10] and is developed at Vanderbilt University. TinyDT also provides a custom perspective within Eclipse and has an inbuilt TinyOS parser. Thus, TinyDT also provides a spell checker, an outline of the open file, and code completion for interface members. The parsers of TinyDT and YETI differ in one important aspect: While our parser is optimized for fast execution at the cost of some imprecision when handling preprocessor statements, the parser of TinyDT is designed to be completely accurate. The drawback of this solution is a slow response time of the system. Even a change of one character in the source code takes several seconds before the file is revalidated and potential errors are detected.

Like TinyOS IDE also TinyDT requires a preinstalled TinyOS environment. The user needs to specify where to find the compilers for the different target platforms and the bash executable. TinyDT does not detect available node platforms but only supports nodes of the mica family, telos, telosb, and the tmote.

6. CONCLUSION AND FUTURE WORK

YETI is currently in a stage of open beta test and we are collecting feedback from more than 1000 developers who have downloaded the tool within the first month after its public announcement. Besides the invaluable bug reports, incoming feature requests help us to decide which additional features are most wanted by the community. We are currently working on an adaptation of the system making it Unix/Linux compatible. Another important open question is how to integrate existing TinyOS simulators and which of them to support. We are also looking into TinyOS 2.x and will try to adapt YETI to support this new operating system. Finally, additional tools for deployment, management, and debugging of sensor networks are under development.

7. DOWNLOAD

YETI is available for download as a full archive or directly from within Eclipse. For more information about YETI and a small video presenting its main features please visit the project homepage at `http://www.dcg.ethz.ch`.

8. REFERENCES

[1] J. Beutel, O. Kasten, F. Mattern, K. Römer, F. Siegemund, and L. Thiele. Prototyping wireless sensor network applications with BTnodes. In *Proc. 1st European Workshop on Sensor Networks (EWSN 2004)*, volume 2920 of *Lecture Notes in Computer Science*, pages 323–338. Springer, Berlin, Jan. 2004.

[2] Crossbow Technology. MICA2 Wireless Measurement System. `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf`.

[3] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. In *Proc. of ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2003.

[4] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[5] C. L. Jeffery. Generating LR syntax error messages from examples. *ACM Trans. Program. Lang. Syst.*, 25(5):631–640, 2003.

[6] G. Klein. JFlex. `http://jflex.de`.

[7] W. M. McKeeman. Resolving Typedefs in a Multipass C Compiler, March 1991.

[8] moteiv. Tmote Sky. `http://www.moteiv.com/products-tmotesky.php`.

[9] Object Rechnology International, Inc. Eclipse Platform Technical Overview. Tech report, IBM, 2001.

[10] J. Sallai, G. Balogh, and S. Dora. TinyDT. `http://www.tinydt.net`.

[11] Scatterweb. ESB Embedded Sensor Board. `http://www.scatterweb.net/research_products/esb.en.html`.

[12] A.-T. Schreiber and B. Kuehl. jay. `http://www.informatik.uni-osnabrueck.de/alumni/bernd/jay`.

[13] Shockfish SA. Tinynode 584. `http://www.tinynode.com`.

[14] R. Tynan. TinyOS IDE. `http://tinyoside.ucd.ie`.