

Language Engineering in The Context of A Popular, Inexpensive Robot Platform

Li Xu

Department of Computer Science
University of Massachusetts Lowell
Lowell, MA 01854, USA
xu@cs.uml.edu

ABSTRACT

Language engineering – the theory and practice of building language processors and compilers, has long been recognized as important subject in Computer Science curricula. However, due to lack of suitable target systems, educators face significant challenges to teach language engineering classes effectively.

Leveraging the emerging inexpensive robot devices, this paper presents a new approach of using robots as system context to teach language engineering topics. We designed the Chirp-Scribbler Language, which targets the popular Scribbler robot; combined together, they provide an engaging and feature-rich platform to teach a wide range of topics in language engineering.

This paper describes the Chirp-Scribbler Language, its integration with the target robot, and the teaching practice of using them to teach language translation basics in an undergraduate programming course.

Categories and Subject Descriptors

K.3.2 [Computer and Information Science Education];
D.3.4 [Programming Languages]: *Compilers*

General Terms

Language Engineering, Compiler, Microcontroller, Robotics, Embedded Systems

Keywords

Language Translation, Compiler Construction, Educational Robotics

1. INTRODUCTION

Language engineering – the theory and practice of building language processors and compilers, has long been recognized as important subject in Computer Science curricula. Knowledge of language engineering and compilers is

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGCSE'08, March 12–15, 2008, Portland, Oregon, USA.

Copyright 2008 ACM 978-1-59593-947-0/08/0003 ...\$5.00.

key to understand modern programming languages and systems. However, due to limited resources and time constraints, many institutions can only teach language engineering with a selective number of concepts and students work on exercises to understand the concepts, but rarely do they have the opportunity to explore these concepts and their interactions in the context of realistic target systems. We have used the conventional approach to teach language engineering, but the students are disengaged and are unclear of the relevancy between the taught concepts and real-world language applications.

Leveraging the emerging inexpensive robot devices, this paper presents a new approach of using robots as system context to teach language engineering topics. We designed the Chirp-Scribbler Language, which targets the popular Scribbler robot; combined together, they provide an engaging and feature-rich platform to teach a wide range of topics in language engineering.

This paper describes the Chirp-Scribbler Language, its integration with the target robot, and our teaching practice of using them in an undergraduate programming course. This paper makes the following contributions:

1. We examine the current teaching practice of language engineering and identify the lack of suitable realistic target systems as the biggest limitation factor;
2. We present the new robot-based approach of using inexpensive robot devices as system context, and designed the Chirp-Scribbler Language as teaching tool;
3. We present teaching practice of using Chirp-Scribbler platform and demonstrate its effectiveness.

The rest of the paper is organized as follows: Section 2 describes the motivation of using robots as the target system. Section 3 gives the background of robots in CS education and the Scribbler robot. Section 4 describes the target robot system, the Chirp-Scribbler Language, and their use as teaching tool for language engineering. Section 5 describes the teaching practice of using Chirp-Scribbler in an undergraduate programming course to teach language translation basics. Section 6 looks at future work. We conclude in Section 7.

2. MOTIVATION

The importance and benefits of language engineering – the theory and practice of building language processors and compilers have long been recognized in Computer Science curricula. All past ACM Computing Curricula (CC '68, '78,

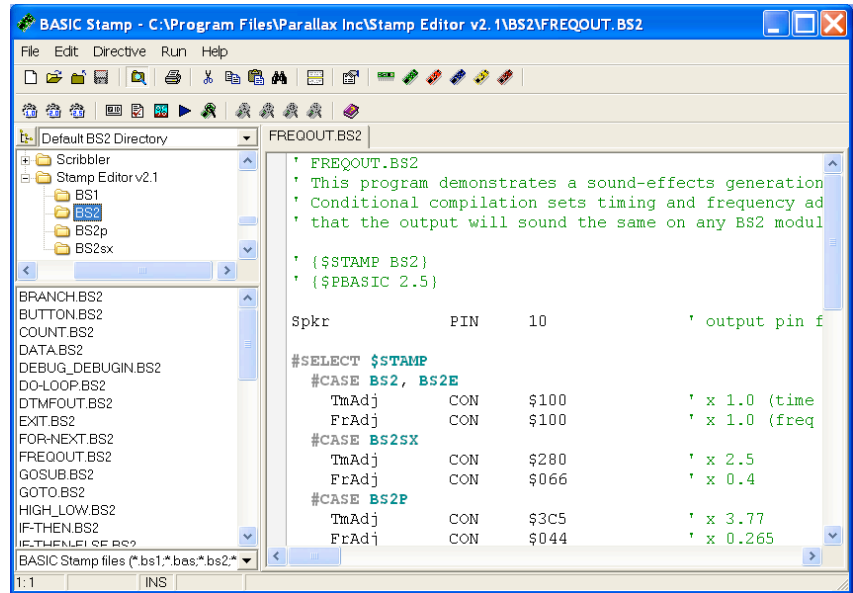


Figure 1: Parallax Scribbler robot and BASIC Stamp Editor

'91) listed compiler and language implementation as main curriculum topics [1, 2, 3]; the latest Computing Curricula 2001 (CC'2001) includes the Language Translation Systems and Compiler Construction unit as a single upper-level elective course (Unit PL8) and the Introduction to Language Translation unit (Unit PL3) as required core subject [7].

Such courses teach the concepts, techniques and tools of programming language translation and processing. Through these courses, students can gain in-depth knowledge on language implementation, deepen understanding of language features as related to the underlying system, and develop a comprehensive view of software and hardware interactions in modern systems [13, 26, 8]. This knowledge is crucial to tackle problems in both application and system areas [6]. The techniques learned from these courses are also broadly applicable to other domains and enable students to develop language-based solutions which are often more flexible and powerful [8, 4, 9].

Although language engineering brings tremendous benefits and value to the CS curriculum, however, to realize such benefits and achieve objectives of these courses, as well as to fit with the mandate of CC'2001, compiler and language educators need to address the following significant challenges:

1. **Curriculum and time constraint.** In CC'2001, language engineering topics are taught either as one-semester compiler elective or part of a language introduction course, and need to cover a broad scope of subjects, including lexing, parsing, semantics analysis, storage management, code generation and optimization. Such a tight schedule with a wide range of topics leads to significant pressure on how to present the topics coherently [28].
2. **Topics selection.** Language engineering and compiler design have become a rich and sophisticated subject of study in its own right [4, 8]. Instructors must carefully choose a selective set of key concepts and bal-

ance the depth and breadth of topics.

3. **Resource constraint.** Compilers are increasingly used as an integrated component in a larger system context rather than as a standalone artifact [13, 6]. However, in typical academic environment, due to resource constraints, it is often difficult to provide realistic systems as the target of student compilers.
4. **Teaching projects.** In order to give students hands-on experience in language engineering, instructors must design useful teaching projects to demonstrate course topics. Due to the above time and resource constraints, this is no easy task.

Due to the above limitations, language engineering courses often focus on limited front-end topics such as lexing and parsing, and are taught without real systems as the context; as results, important system-related topics, such as back-end design, system integration, etc, are de-emphasized or totally ignored. In the courses, students implement contrived languages and use simulators, rather than use real language and real system as the target; teaching projects are designed to understand the selected concepts, but rarely do students have the opportunity to explore those concepts and their interactions in the context of real systems.

We have used this conventional approach to teach language engineering classes at UMass Lowell, but our students were disengaged and were unclear of the relevancy between the taught concepts and real-world language applications. Our difficulties are reported by other educators in similar courses [14, 9, 28].

In examining the current practice, we identify the lack of system context as the biggest limitation to teaching language engineering effectively. To address the above challenges and difficulties, we have developed a new approach of using inexpensive educational robots as the system context to teach compilers and language engineering topics. Such robots provide an affordable, yet highly engaging, and feature-rich tar-

get platform which can be used to teach both introductory and advanced language and compiler subjects. We present our methodology and experience in this paper.

3. BACKGROUND

3.1 Robots in CS Education

Robots have long been used to teach related Artificial Intelligence and Robotics. The emerging inexpensive robot devices greatly lower the barrier for educators to adopt them as valuable teaching tools. Products such as LEGO Mindstorms RCX and NXT robot kits [21], Parallax Scribbler robot [25], and MIT Handyboard and Handy Cricket [23] have been widely used to teach AI and Robotics [20, 5, 19, 17, 16], and more recently general Computer Science courses [11, 10, 27, 12, 18, 15].

We believe these inexpensive educational robots can play an equally valuable role in compiler and language engineering courses and provide the long sought “missing” system context. Their affordable price makes it possible for large-scale use in an undergraduate course. More importantly, the new generation robots also provide considerable processing power and resources capable of sophisticated applications; yet their simple structure lend themselves well to pedagogical use in language and system courses. In our class, we use the Parallax Scribbler robots (under \$100 per unit) to teach both introductory and advanced topics in compiler design.

3.2 Scribbler, BASIC Stamp and Tools

Scribbler robot is fully assembled and can be used immediately. It features a BASIC Stamp 2 (BS2) microcontroller with an array of sensors, two DC motor wheels and a top Pen Port to insert marker pen, so the robot can “scribble”. Scribbler can be programmed through PBASIC commands using the Stamp Editor – GUI programming tools developed by Parallax [24]. The Scribbler robot and the Stamp Editor are shown in Figure 1.

4. LANGUAGE ENGINEERING WITH SCRIBBLER

Scribbler provides an inexpensive, yet feature-rich, and ready-to-use, target system to teach language engineering topics.

4.1 System Features

Scribbler includes a Parallax BASIC Stamp 2 (BS2) microcontroller. The BS2 controller is based on a Microchip PIC processor and runs PBASIC interpreter: Scribbler PBASIC commands are compiled into byte codes (“tokens”) and executed by the interpreter. BS2 microcontrollers have been used to build a wide range of embedded systems, including industrial applications such as environment monitoring and automotive control [22].

The BS2 controller has 2K bytes EEPROM to store PBASIC byte codes and user data, and 32 bytes (6 bytes reserved) RAM which can be used as general-purpose registers.

The BS2 controller has 16 I/O pins which control Scribbler sensors, motors, LED lights and on-board speaker. The system spec is listed in Table 1.

Components	Spec
Processor	8-bit Microchip PIC16C57c
Speed	20 MHz
RAM	32 Bytes
EEPROM	2K Bytes
I/O Pins	16
PBASIC Commands	42
Light Sensors	3
Obstacle Sensors	3 (2 emitters, 1 detector)
Line Sensors	2
Stall Sensor	1
DC Motors	2
LED Lights	3
Speaker	1
Serial Port	1

Table 1: Scribbler System Spec

```

const int do = 391; // note Do's frequency
const int re = 494; // note Re
const int mi = 523; // note Mi
int tune_time;

bit stalled;
pin(11) speaker;

void startTune() {
    tune_time = 200; // 200ms
    System.sound(speaker, tune_time, do);
    System.sound(speaker, tune_time, re);
    System.sound(speaker, tune_time, mi);
}

void checkState() {
    Scribbler.senseStall(stalled);
    if (stalled) {
        Scribbler.setLED(1, 1, 1);
        Scribbler.moveBackward(5, 5, 1000);
        Scribbler.moveLeft(5, 5, 1000);
    } else {
        Scribbler.setLED(0, 0, 0);
    }
}

void main() {
    startTune();
    loop {
        Scribbler.moveForward(5, 5);
        checkState();
    }
}

```

Figure 2: Chirp-Scribbler Sample Program

4.2 The Chirp-Scribbler Language

To teach language engineering with Scribbler, drawing inspiration from the Chirp language [29, 30], we have designed the Chirp-Scribbler Language for programming the Scribbler robot.

Chirp-Scribbler has syntax similar to C. It defines data types and control constructs to target the BS2 architecture. A sample Chirp-Scribbler program is shown in Figure 2. It consists of variable definitions and three functions (including `main` function): the robot starts by playing music notes on the speaker pin (Pin 11), then moves forward; it checks the stall sensor to see if it hits obstacles, and if so, it turns on all three LEDs, backs away and turns left.

Chirp-Scribbler's data types matches those of BS2: it has `int` (16-bit) and `byte`, also the `nib` (4-bit), `bit` and special I/O pin (1-bit) types, as BS2 RAM variables are bit and nibble addressable and BS2 allows direct pin access. It also has `rom_int` and `rom_byte` types to access EEPROM data.

Chirp-Scribbler supports expressions on variables and constants using common arithmetic, logic and comparison operators. Due to lack of stack on BS2, all Chirp-Scribbler variables are global, and functions do not have local variables or return value. Function body consists of statements, including assignment, if-else statement, conditional and unconditional loop, for-loop, function call and return statements. Functions with `System` and `Scribbler` prefixes are special system functions for I/O control and robot operation.

4.3 Chirp-Scribbler As Teaching Tool

Chirp-Scribbler can be used to teach a wide range of topics in language engineering:

- It can be used to teach traditional front-end design such as lexers and parsers. The small size of the Chirp-Scribbler language makes it easy to build lexers and parsers, and fits well with the tight schedule of language courses; it also has common language constructs to demonstrate key lexing and parsing techniques.
- It provides a realistic target to teach back-end design. Instructors can develop teaching projects to generate PBASIC commands which can then run on Scribbler. The simple and easy to understand BS2 command set greatly reduces the learning curve to target Chirp-Scribbler; students can experiment with code generation by testing their output directly using the BASIC Stamp Editor and Scribbler robot.
- It serves as concrete examples for advanced topics, including compiler optimization, language design, and system integration. For example, BS2's small RAM register set can be used as real example for register allocation; the `System` and `Scribbler` functions expose interactions of systems and language design; by building their Chirp-Scribbler translator, students gain first-hand experience on compiler tool integration.

In summary, Chirp-Scribbler provides a rich and versatile environment for language engineering courses.

5. TEACHING EXPERIENCE

We have been using Chirp-Scribbler to teach language engineering components in an intermediate undergraduate

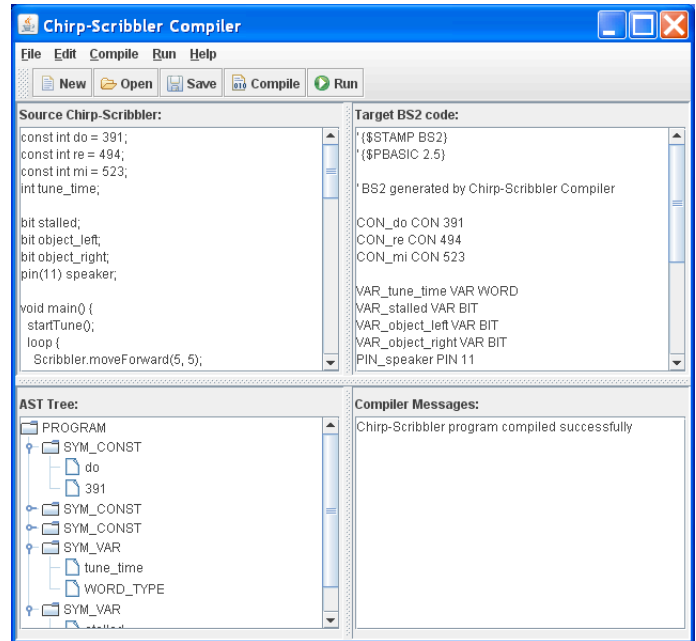


Figure 3: Chirp-Scribbler Compiler

programming course at UMass Lowell. The class includes introduction to language translation as listed in Unit PL3 in CC'2001.

In comparison with our previous practice, using Chirp-Scribbler enables us to teach similar topics in a more hands-on, system-oriented approach. Specifically,

1. It provides students an easy to understand and fully accessible target system. In the class, each student is given a Scribbler robot for use during the semester. Students learn Scribbler and the PBASIC commands quickly by experimenting with the Stamp Editor tool, which itself is a real-world example of production compiler.
2. The small size of Chirp-Scribbler allows us to develop lexer and parser projects with much manageable complexity. The grammar consists less than 30 nonterminal symbols and around 70 production rules, yet it realizes majority BS2 features, including all sensor and motor control of Scribbler robot.
3. Previously, the code generation subject is only mentioned briefly due to lack of suitable target; using Scribbler, such back-end topics can now be explored in much greater depth. For example, the `System` functions are directly mapped to PBASIC command: `System.sound(pin, time, freq)` is translated to `FREQOUT` in PBASIC. Students can learn such target code generation through an iterative exploratory process: first targeting individual construct; then testing the code immediately on Stamp Editor and Scribbler; after it works, moving on to larger constructs. Students feedback indicated such method makes the otherwise difficult code-generation topic much more accessible.
4. Chirp-Scribbler illustrates language design issues in

the context of real systems. The **Scribbler** functions are in fact system libraries implemented through compiler translation [13]. Such usage requires a real system context to demonstrate.

- Finally, Scribbler conveys the impressionable modern whole-system perspective which compiler and language tools belong to. In the class, students integrate their compilers into robot development suite by modeling the Stamp Editor tool. Students build GUI interface with program editor and message panes similar to Stamp Editor. The student Chirp-Scribbler Compiler is shown in Figure 3.

Our preliminary results indicate Chirp-Scribbler has been an effective teaching tool for language engineering topics. In addition, students interest levels are high, and students love to explore their code on Scribbler. We are currently evaluating our findings and will publish the results in a followup report.

6. FUTURE WORK

Chirp-Scribbler opens up many exciting possibilities in language engineering and other Computer Science system courses. As future work, we are interested in adapting it to support more courses and evaluate its effectiveness.

We are also looking at targeting other popular educational robot platforms, such as LEGO NXT. This will make the Chirp-Scribbler approach applicable by educators on other platforms.

7. CONCLUSIONS

Leveraging the emerging inexpensive robot devices, this paper presents a new approach of using robots as system context to teach language engineering topics. Targeting the popular Scribbler robot, we designed the Chirp-Scribbler Language; combined together, they provide an engaging and feature-rich platform to teach a wide range of topics in language engineering. The preliminary results demonstrate that the Chirp-Scribbler Language and the target robot system can help students learn language engineering basics quickly, and provide useful context to explore more advanced topics.

8. ACKNOWLEDGMENTS

Fred Martin has been instrumental in our approach of using educational robots to explore compiler and language engineering projects. This work has been supported by grants from the UMass President's Office and UMass Lowell. We would like to thank Mark Schlesinger and Marvin Stick for their help and support.

9. REFERENCES

- ACM Curriculum Committee on Computer Science. Curriculum '68: Recommendations for the undergraduate program in computer science. *Communications of the ACM*, pages 151–197, Mar. 1968.
- ACM Curriculum Committee on Computer Science. Curriculum '78. *Communications of the ACM*, pages 147–166, Mar. 1979.
- ACM Curriculum Committee on Computer Science. Computing Curricula 1991. *Communications of the ACM*, pages 68–84, June 1991.
- A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.
- D. Blank, L. Meeden, and D. Kumar. Python robotics: an environment for exploring robotics beyond legos. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pages 317–321, 2003.
- R. E. Bryant and D. R. O'Hallaron. *Computer Systems: A Programmer's Perspective*. Prentice Hall, 2003.
- CC2001 Task Force. Computing Curricula 2001, Computer Science Volume. <http://www.sigcse.org/cc2001/>, Dec. 2001.
- K. D. Cooper and L. Torczon. *Engineering a Compiler*. Morgan Kaufmann, 2003.
- S. Debray. Making compiler design relevant for students who will (most likely) never design a compiler. In *SIGCSE '02: Proceedings of 2002 SIGCSE technical symposium on Computer science education*, pages 341–345, 2002.
- B. Fagin and L. Merkle. Measuring the effectiveness of robots in teaching computer science. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pages 307–311, 2003.
- B. S. Fagin and L. Merkle. Quantitative analysis of the effects of robots on introductory computer science education. *Journal on Educational Resources in Computing*, 2(4):2, 2002.
- M. Goldweber, C. Congdon, B. Fagin, D. Hwang, and F. Klassner. The use of robots in the undergraduate curriculum: experience reports. In *SIGCSE '01: Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*, pages 404–405, 2001.
- J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach, 3rd Edition*. Morgan Kaufmann, 2002.
- T. R. Henry. Teaching compiler construction using a domain specific language. In *SIGCSE '05: Proceedings of 2005 SIGCSE technical symposium on Computer science education*, pages 7–11, 2005.
- Institute for Personal Robots in Education. <http://www.roboteducation.org/>.
- F. Klassner. A case study of lego mindstorms' suitability for artificial intelligence and robotics courses at the college level. In *SIGCSE '02: Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, pages 8–12, 2002.
- F. Klassner. Enhancing lisp instruction with rclisp and robotics. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 214–218, 2004.
- F. Klassner and C. Continanza. Mindstorms without robotics: an alternative to simulations in systems courses. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 175–179, 2007.
- A. N. Kumar. Three years of using robots in an artificial intelligence course: lessons learned. *Journal on Educational Resources in Computing*, 4(3):2, 2004.
- D. Kumar and L. Meeden. A robot laboratory for teaching artificial intelligence. In *SIGCSE '98: Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, pages 341–344, 1998.
- LEGO Group. Lego Mindstorms Robots web page. <http://mindstorms.lego.com/>.
- A. Lindsay. *What's a Microcontroller*. Parallax, Inc.
- F. Martin, B. Mikhak, and B. Silverman. Metacrick: A designer's kit for making computational devices. *IBM Systems Journal*, 39(3 & 4), 2000.
- Parallax, Inc. *BASIC Stamp Reference Manual, version 2.1*.
- Parallax, Inc. Scribbler Robot web page. <http://www.scribblerrobot.com/>.
- D. A. Patterson and J. L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann, 1997.
- E. Sklar, S. Parsons, and P. Stone. Using robocup in university-level computer science education. *Journal on Educational Resources in Computing*, 4(2):4, 2004.
- W. M. Waite. The compiler course in today's curriculum: three strategies. In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 87–91, 2006.
- L. Xu and F. Martin. The chirp language specification. Technical Report TR-2005-003, Dept. of Computer Science, UMass Lowell.
- L. Xu and F. G. Martin. Chirp on crickets: teaching compilers using an embedded robot controller. In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 82–86, 2006.