Dolittle — Experiences in Teaching Programming at K12 Schools

Susumu Kanemune[†], Takako Nakatani[‡], Rie Mitarai[*], Shingo Fukui[†], Yasushi Kuno[†]
[†]Graduate School of Business Sciences, University of Tsukuba
[‡]S-Lagoon Co., Ltd., [*]Armat Corporation
kanemune@logob.com, tina@s-lagoon.co.jp, rie@armat.com, {fukui,kuno}@gssm.otsuka.tsukuba.ac.jp

Abstract

The Japanese government has been promoting IT education, including programming, at elementary and secondary (K12) schools since 2002. We have developed Dolittle, an object-oriented programming language suitable for K12 education, and evaluated it through the teaching opportunities available in classrooms. This paper describes the outline and the future outlook of Dolittle, and reports on the examples of how it is used in classes.

1. Introduction

This paper describes a programming language called "Dolittle" which we have developed for programming education, and reports on the teaching experiences using Dolittle.

The Japanese government has been promoting IT education at elementary and secondary schools since 2002. Learning programming in an experiential way as a part of IT education allows students to get a better understanding of software, which is an essential part of computers. In the past, however, there was no object-oriented language suitable to elementary and secondary education. We have developed such language, called Dolittle, and have been conducting experiments by teaching Dolittle in classes. As a result of the experiments, we have confirmed that Dolittle can be used in classes and students can learn various aspects of computers through the programming experience.

There are several related works such as Squeak[4], which is based on Smalltalk[3], its educational language SqueakToys[5]. Smalltalk is an object-oriented language but it is not easy for children. SqueakToys is a programming environment for children but it is not a text language. Therefore we have designed a text language that even children are able to understand. We use the idea of LOGO[6]'s turtle graphics for drawing operations. We also use the idea of cloning objects from prototype-based object oriented languages such as Self[8].

In this paper, we at first describe the outline of Dolittle. Then we report on the examples of using Dolittle in the lessons at lower and upper secondary schools. Finally, we introduce two applications of Dolittle that we are currently working on: robot control and distributed programming.

2. A Programming Language Dolittle

When we designed Dolittle[1], we paid close attention to making it a simple language in order to use it in school lessons, but we also paid attention to making it a useful language to allow students to develop practical programs. The design policy of Dolittle is listed as follows:

- Simple syntax with Japanese words
  The syntax of the language should be simple and easy to learn. Programs should be written with Japanese words and symbols. The ease of the language allows students to use the language for learning instead of spending a long time to learn the language itself.
- Incremental programming
  The language should accept a single line program. The program can be extended by adding new lines at the end of it. Students need not write class/function/variable definitions which inevitably increases the difficulty of learning programming.
- Text-based programming
  The language should use text-based representation because it allows concise and precise representation of programs with maximum freedom.
- Object-oriented
  The language should support object-oriented programming. Students manipulate objects on the screen by sending instructions to it. The figure drawn with turtle graphics also becomes an object. Dolittle adopts prototype-based object-orientation. It generates new objects by cloning existing objects and deals with the concept of inheritance as a link between the objects.
- Open expandability
  The language should provide a way to control external devices and to communicate over networks. The controllability of external devices enables students to expand the virtual world of the screen into the real world. The communication over networks allows students to expand the closed individual learning process into group-wide or world-wide learning processes.

Figure 1 shows a sample Dolittle program that is written in both Japanese and English. The rest of this section explains the syntax of Dolittle base on the sample program.

```
カメ太＝タートル！作る。
「カメ太！100 歩く 120 右回り」！3 繰り返す。
三角形＝カメ太！図形にする（赤）塗る。
時計＝タイマー！作る。
実行ボタン＝ボタン！"実行" 作る。
実行ボタン：動作＝「時計！「三角形！36 右回り」実行」。
```

```
kameta=turtle!create.
[kameta!100 forward 120 rightturn]!3 repeat.
tri=kameta!makefigure (red) paint.
clock=timer!create 1 period 10 duration.
rBtn=button!"Run" create.
rBtn:click=[clock![tri!36 rightturn]execute].
```

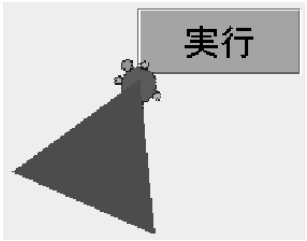

Figure 1: A sample Dolittle program and its execution

```
kameta=turtle!create.
```

Dolittle programs call on objects using "!." This sample statement sends a "create" message to the prototype object "turtle" to let it clone itself, then assigns the created turtle object to the variable called "kameta." The period (".") indicates the end of the statement.

Execution of the above statement draws a turtle object on the screen. The turtle object is an object which implements turtle graphics. The turtle move around on the screen with the lines which show its tracks.

```
[kameta!100 forward 120 rightturn]!3
repeat.
```

The square brackets ("[...]") represent a block. In this sample statement, "[...]!3 repeat" sends a message "Execute yourself three times" to the block.

Dolittle interprets identifiers after "!" (such as "forward") as message selectors. When numeric literals (such as "100") or parenthesized expressions (shuch as "(x)", "(x + 1)") are placed between "!" and the message selector, they become arguments to the message. When an object processes a message, it normally returns the object itself. The returned object receives the next message within the block. This is called a cascade sending of messages. This sample statement sends "rightturn" to the result of "forward" (the object itself).

Execution of the above statement draws a regular triangle on the screen.

```
tri=kameta!makefigure (red) paint.
```

The lines drawn by "kameta" is a part of "kameta" itself (imagine that the tail of the turtle is lengthened.) If you send "makefigure" to "kameta", it separates the drawn lines from itself and returns the figure as a new figure object.

This sample statement sends "paint" message to the created figure object. When you are going to refer identifiers (variables) within an argument list, you should enclose them with brackets ("(...)"). The sample statement assigns the generated figure object to the variable "tri."

Execution of the above statement paints the triangle red.

```
clock=timer!create 1 period 10
duration.
```

A "timer" is an object which executes code pieces (blocks) at a specified interval for a specified period. This sample statement creates a timer object "clock" and sets the execution period and the interval.

```
rBtn=button!"Run" create.
```

This statement creates a button object called "rBtn." Execution of this statement displays a GUI part of a button shape.

```
rBtn:click=[clock![tri!36 rightturn]
execute].
```

This statement assigns a block to the "click" attribute of the "rBtn" object. Dolittle defines a method by assigning a block to an attribute. The "click" is the method to be executed when the button is clicked.

When you click the button, the "execute" message is sent to the "clock", and the "clock" repeats the execution of the block passed as an argument at the specified interval for the specified period. As a result, the screen shows an animation that the triangle rotates 36 degrees at a time.

We designed and implemented Dolittle in the year 2000. Dolittle is written in Java so that it runs on various computer environments most often used in schools.

3. Experimental Lessons

3.1. Experimental Lessons at High School

To evaluate Dolittle, we conducted small experimental lessons at a senior high school.

Three first-year students of the High School at Otsuka, University of Tsukuba (an attached high school of University of Tsukuba) attended the lessons. One of them had some experience in Visual Basic, the other two students had no programming experience. One of the beginners knew nothing about the concepts of programming and software.

Three lessons were held after school hours every two weeks. Each lesson had a duration of one hour. One of the authors took charge as the lecturer. Table 1 shows the curriculum of the lessons.

In the first half of each lesson, we explained a sample program (about 10 lines) and the students entered the code to confirm the behavior of the program. Then the students did a programming exercise in the second half.

The two programming beginners discovered the following principles of computers through the programming experience:

Table 1: Curriculum of Experimental Lessons at High School

Lesson	Contents
1	Explanation of messages and objects Execute programs interactively using turtle graphics
2	Execute programs repeatedly at regular intervals using timers Manipulate multiple objects
3	Define methods Inheritance of objects

- There are rules in the execution of programs
  The sequential execution, which executes instructions from top to bottom, is one of the most basic behaviors of computers. One of the students said “Programs are executed from the top!” This means the student has realized a basic principle that programs are executed basically from top to bottom.
- The mechanism of monitor displays
  Bitmap displays used for computers represent characters and graphics as a set of dots (pixels). One of the students asked a question, “Why does the turtle need 100 steps to walk only 5 centimeters? Does it move along the 100 dots on the screen? ” This student noticed that computer monitors handle length as a set of dots (pixels) instead of a unit of length such as centimeters.
- The existence of operating systems
  Computers use operating systems to control user interfaces and peripheral devices. We used Windows for these lessons. “If we need a program to manipulate this turtle, do we need another program running in the computer to move this mouse cursor? If the programs display not only the mouse cursor but all these windows, there must be huge programs in this tiny notebook computer!” This student was aware of the existence of programs (an operating system) to move a mouse cursor which was analogous to the existence of the program to move the turtle. The student also analogized that a computer consists of vast amounts of software to control many windows and application programs in addition to a mouse cursor.

Figure 2 shows a piece of work from a student. It is an animation program of falling leaves. The behavior of the falling leaves is embedded into them as a method. The leaves with same color inherit the method of the first leaf, which means that the student understood the inheritance of objects.

Computers operate with software but that mechanism is not visible to us. In the experimental lessons, the students learned through the active experience of programming. This enabled the students to realize the various aspects of computers such as the principle of sequential execution, and the existence of pixels and operating systems.

3.2. Lessons at Junior High School

We conducted large experimental lessons at a junior high school in 2001 to check if Dolittle is suitable for school lessons.

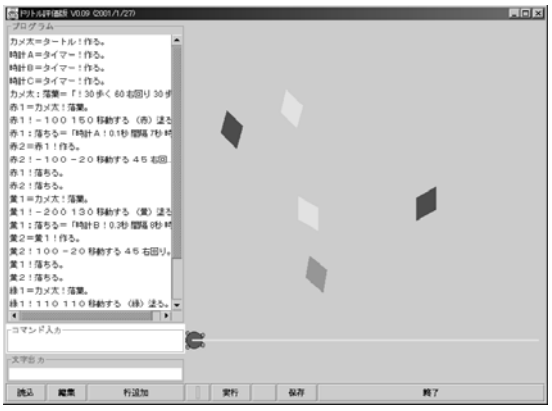

Figure 2: A piece of work from a high school student

The experimental lessons were held at Kamata Junior High School in Matsuzaka-city, Mie-prefecture, as a part of the technical training course. All of the 132 second year students attended the lessons. A programming lesson was a new experience for all of them. A teacher of the junior high school took charge as the lecturer. We had provided the lecturer with the documents and the workshops in advance and supported him with e-mail when needed. We also created the questionnaire and the exam questions to evaluate the students’ comprehension.

In the lessons, the lecturer explained at first the key points of the day on the blackboard, then the students did some exercises. Although the time length of one lesson in junior high schools was fifty minutes, the actual length of each lesson was only forty minutes since the startup of computers and the filling-in and collecting of the questionnaires took five minutes each. Figure 3 shows a scene of a lesson.

Figure 3: A scene of a lesson at junior high school

The students received documents at the beginning of each lesson which formed the text of the day. This text contained the explanatory examples, the description of new syntax, and the exercises.

Table 2 shows the curriculum of the lessons. The titles shown in the contents column are the actual titles of the texts used in the lessons.

Figure 4 to Figure 6 shows the results of the questionnaire surveys. We conducted the surveys at the end of every lesson and the students evaluated “enjoyableness”, “achievement”, and “difficulty” of the lessons in four scales.

Table 2: Curriculum of experimental lessons

Term	Lesson	Contents
2		<Turtle Graphics>
	1	Let's Try
	2	Triangles and Squares
		<Generate and Manipulate Figure Objects>
	3	Stars, Circles, and Paint
	4	Move Figures
	5	Draw Variety of Figures
3		<Animation with Timer>
	6	Timer
	7	Set Timer in Your Program
	8	Work on Your Own Program
		<Use of GUI Parts (Button)>
	9	Button
	10	Button
	11	Work on Your Own Program

We analyzed the results of the surveys as follows:

- The lessons were conducted comfortably
  The survey results of "achievement" shows that less than 10% of the students answered "I could not complete the exercise of the day (achievement=1)." This result indicates that the lessons were conducted effectively to the end without undue stress.

- Enjoyableness increases as lessons carry forward
  The survey results of "enjoyableness" shows that the ratio of the students answered "enjoyable (enjoyableness = 3 or 4)" increases from 20% to 40% as lessons carry forward. This is significant from the result of the statistical sign test between first half and second half of the lessons. On the other hand, the ratio of students answered "unenjoyable (enjoyableness = 1)" is always less than 10% and goes down to 0% at the end.

- Filled with pleasure on achievement
  As lessons progressed, more students felt that the lessons were more "difficult" but more students felt that the lessons were "enjoyable." The correlation between "achievement" and "enjoyableness" in the last lesson is 0.55, which indicates that the achievement of the exercises is linked to enjoyableness. The lecturer, Mr. Idosaka, commented that "Many of the students were delighted in themselves in being able to overcome the difficulties." We have succeeded in providing the lessons with moderate difficulty and accomplishment for junior high school students.

These lessons were conducted as a part of the technical training course and were included in the range of two regular examinations. We created the ground plan of exam questions and completed them with the lecturer. Each question contains check points to deeply analyze how much the students understood the programming concepts. Then we examined their answers to determine if they understood the concepts as we had intended.

The result of examinations indicated that more than 85% of the students understood the basic programming concepts

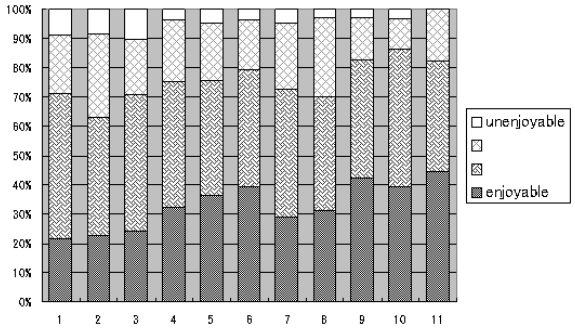

Figure 4: Survey results (enjoyableness)

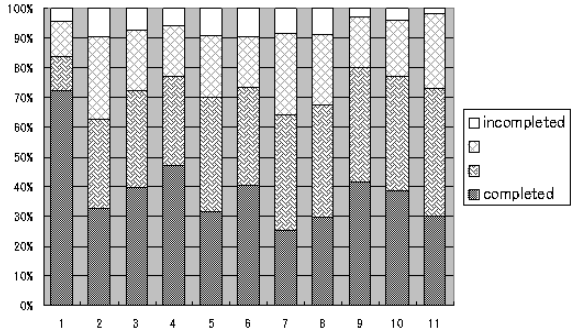

Figure 5: Survey results (achievement)

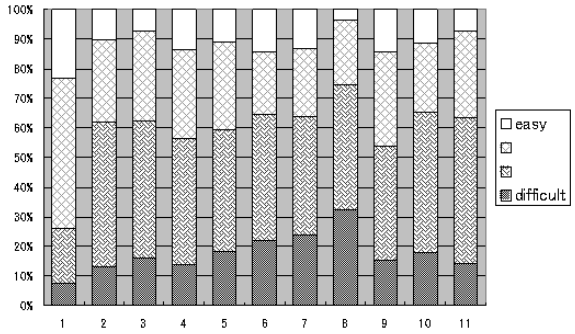

Figure 6: Survey results (difficulty)

such as the control structure (ex. iterations) and the detection of syntax errors.

In lessons 9 to 11, the students created a drawing software, which draws lines and figures using buttons, as an exercise to summarize what they had learned. Figure 7 shows an example of one student's work. This program places the buttons on the left-hand side of the screen and defines a method for each button. Most of the students created functional drawing software using what they had learned in the class.

Table 3 shows the functionalities and the objects used in various students' work including the drawing software. More than 85% of students used a variety of objects including buttons and timers.

Through the lessons, we have confirmed that:

- the programming lessons have been conducted effectively in the classroom education for the entire grade (4 classes of 132 students),

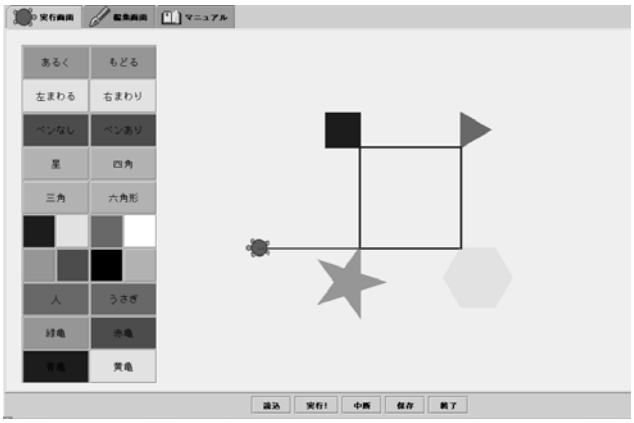

Figure 7: One student's work (drawing software)

Table 3: Analysis of students' work

Fuction	Usage rate
Turtle	100%
Figure object	97%
Iteration	96%
Button	93%
Timer	85%

- it is possible to conduct IT education using object-oriented language in junior high schools. The concept of object-orientation didn't cause any problems with learning Dolittle programming.
- many students have mastered the usage of GUI parts (especially buttons) and have shown strong interest in creating programs of a "pushing buttons activates a certain action" style. The lecturer, Mr. Idosaka, observed that "By using GUI parts in programs, such as the drawing software, the students could link the programming they had learned to the software used in the real world."

4. Objects in Real World

Robots are real world objects. Through experience gained in programming robots, students can link on screen programming in the virtual world to the real world.

In this section, we describe the control of external devices using Dolittle, then we introduce robotic cars, an application of Dolittle, and report on the lessons we conducted at an elementary school and a junior high school using the robotic cars.

Dolittle provides the objects to control external devices. The sample program in this section is to control a robotic car. The robotic car we use here was designed by an engineer, Mr. Masami Okada. He has published the robotic car's specifications on his web site[2].

Figure 8 shows an example of the robotic cars. The board is loaded with a one-chip microcomputer PIC which stores up to 39 steps of instructions. The sensor switch on the front edge of the car detects collisions with walls. It has two motors to control two wheels so that the left and right wheels rotate backwards and forwards independently each other. A program written in Dolittle was transferred to the robotic car using infrared ray (The robotic car can also be manipulated with a TV remote controller. Students can play with it at home without a computer). In Figure 9, a student is transferring the program to the robotic car.

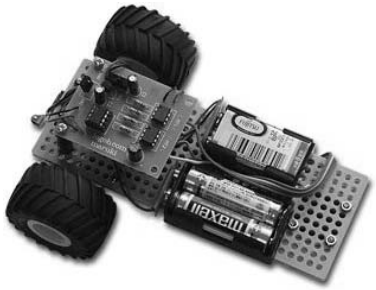

Figure 8: A robotic car

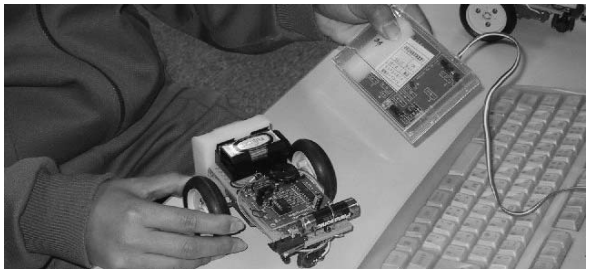
Figure 9: Transferring programs using an infrared ray

Figure 10 shows a sample program to control a robotic car. It generates a "serialport" object "robot" and invokes the methods to control the robotic car. The robotic car behaves as follows:

- starts executing the program when the switch is turned on,
- moves forward until it hits something,
- when it hits something, steps back and turns to the left,
- moves forward again until it hits something,
- when it hits something, steps back and turns to the right,

Note that the "!" at the beginning of the method block has no object specified in front of it. This means that the message is sent to the object that has the method defined in it (in this case, "robot").

```
robot=serialport!create.
robot:script=[
  !startrobot
  switchstart
  forwarduntilcollision
  10 back 15 rightforward 15 leftback
  forwarduntilcollision
  10 back 15 leftforward 15 rightback
  endrobot].
robot!"com1" opensesame.
robot!script.
robot!run.
robot!closesesame.
```
Figure 10: A sample control program

4.1. Lessons at Elementary School

A robot is suited even for lower grade education since students can touch it and see how it behaves.

Figure 11 shows a scene of a lesson conducted during the hours for comprehensive studies in the sixth grade. Mr. Kazuhiro Satoh, a teacher of Kanazawa Elementary School in Chiba-city, Chiba-prefecture, conducted the lessons. In the lessons, groups of students created paper bridges and programmed their robotic cars to pass under and then go across the bridge to reach the goal. A group consisted of three or four students and each member assumed a position of either a director, who managed the group, designer(s) who designed and created the paper bridge, or programmer(s) who controlled the robotic car.

The following comments are from Mr. Satoh:

- It is difficult to control the robotic cars along the course. Some students recorded data to figure out how long they should activate the motors to rotate the wheels for a certain amount of time or to move the robotic car for a certain distance. Students actively discussed and shared information to resolve problems. The lessons were proceeded by setting goals (to cover the whole distance of the course) and by making hypotheses.
- Dolittle allows students to write a program, execute it, check the behavior, and then go back to the programming. This interactive process remarkably facilitates their programming. Their thought streams have never been broken throughout the lessons.

Figure 11: Robotic car contest

4.2. Lessons at Junior High School

Students deepen their understanding of objects through experience in controlling robots in the real world.

Figure 12 shows a scene of a lesson conducted in the technical training course at a junior high school. Mr. Shuji Kurebayashi, a teacher of Nishi-mashizu Junior High School in Fujieda-city, Shizuoka-prefecture, conducted the lessons. Students programmed the robotic cars which negotiated around obstacles in a simple maze to reach the goal.

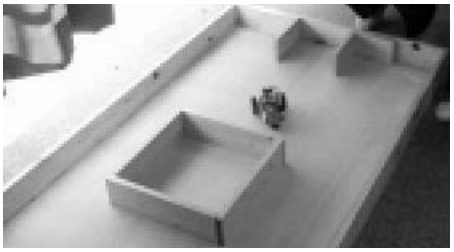

Figure 12: Maze

Then students learned how to simulate the behavior of robots on the screen. We use the “collision” method of the turtle to detect collisions with obstacles. If students define this method in the turtle object, it is executed when the turtle hits the other objects. Figure 13 shows a sample execution of the simulation.

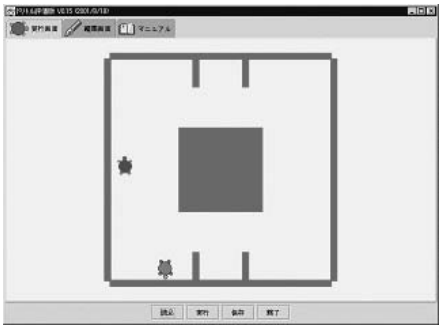

Figure 13: Sample execution of robot simulation

The turtle object, which represents a robotic car, contains the instructions (methods) to perform specific actions when it collides with the other objects. These methods enable students to define the behavior of the turtle when it hits the other turtles or the walls. The turtle object moves by receiving instructions that are sent repeatedly in a certain interval from the timer object.

The following comments are from Mr. Kurebayashi:

- Dolittle effectively facilitates the students’ programming. LOGO or BASIC language, which they previously used, controlled the entire behavior of the robots as a single program. Simulations of multiple robots with these languages require complex programs, which are not suitable for school lessons.
- In Dolittle, a specific action is defined as a method in the object, and the action can be executed repeatedly at a regular interval using a timer. These features of Dolittle facilitate the simulations on the screen.
- Through the programming of robots, the students have learned that physical phenomena affect the behavior of objects in the real world. Even if they use the same program, the robot behaves differently because of other factors such as friction with the ground surface and the attrition of the battery. The students have learned through the comparison between the simulations on the screen and the actual behavior in the real world.

5. The World of Distributed Sharing

We evolve the learning process of programming by extending the Dolittle objects to be network capable.

5.1. Distributed Sharing Dolittle

The distributed sharing version of Dolittle should enable students to:

- learn collaboratively
  Students can learn programming by collaborating with other students, in addition to individual learning.
- develop programs which operate collaboratively
  Students can develop programs that communicate with each other, in addition to programs that operate independently.
- experience programs which operate over networks
  Students can experience programs that communicates over networks, in addition to programs that operate only within a stand-alone computer.
- boost international exchange
  Students can boost international exchange over networks using Dolittle programs, in addition to the lessons in classrooms.

In Dolittle, we are able to create a new object by cloning existing objects and store it in variables or arrays. The distributed sharing version of Dolittle follows this approach. Figure 14 (left) shows how to clone objects. The object server on the network manages objects. Dolittle registers a clone of a local object by invoking a "put" method of the "server" object. And it clones the registered object in the server to a local machine by invoking a "get" method of the "server" object.

These features enable students to register and publish their own objects on the server and enables other students to re-use them in their own programs.

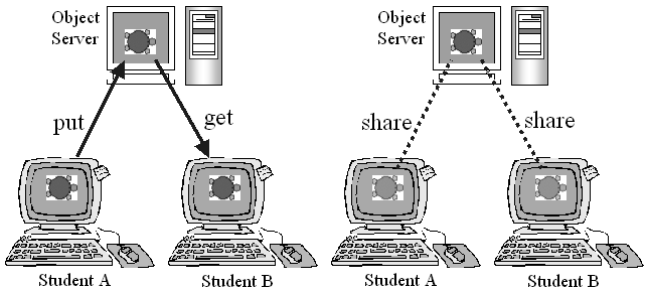

Figure 14: Registering, cloning and sharing objects

Figure 14 (right) shows how to share objects. Dolittle creates a view of an object on the server to the local machine by invoking a "share" method of the "server" object. In the Model-View-Controller (MVC) model, an object on the server is equivalent to Model and a shared local object is equivalent to View and Controller.

Figure 15 shows a sample program to share an object. This program connects to the server "sv1" and shares an object "turtle1" in the server with the name "t1." Then it sends a message to "t1" to move it forward on the screen. The message to "t1" is transferred to the server and executed. If another Dolittle program shares the same "turtle1", this message moves the object on its screen as well.

```
server!"sv1" connect.
t1=server!"turtle1" share.
clock=timer!create.
clock![t1!10 forward]execute.
```

Figure 15: A sample program to share objects

There are some related works such as NetMorph[7] as a distributed environment. NetMorph is a mobile object system of Squeak and its object can move around some machines. On the other hand, Distributed Sharing Dolittle is a object server system and its object can be stored and shared between some clients.

We are currently improving upon the distributed sharing version of Dolittle and intend to conduct experimental lessons at junior high schools in 2004. In this section, we discuss the lessons conducted at a company.

5.2. Lessons at Company

Programming in Dolittle is also applicable to adult education.

We conducted a three day programming lecture using Dolittle as part of company's training course. The title of the lecture was "The principles of computer operation." One of the authors took charge as the lecturer. The students were 15 new employees who intend to be sales representatives or system engineers. Two of the students had used programming for research work, and another two had experienced programming in university. The other 11 students had no experience with programming.

In the two days prior to this lesson, the students had attended a lecture on how to assemble a personal computer kit. Since they already had learned about the hardware-side principles of computers, we focused on teaching the software-side principles through programming experience in Dolittle.

Table 4 shows the curriculum and Figure 16 shows a scene of the lecture. In the morning class of the third day, we explained that the PCs in the classroom were connected to network in a LAN environment, and then the students dealt with the following features of the distributed sharing version of Dolittle:

- Transferring objects: The students created buttons which performed certain actions when clicking them and registered them with the server. Then they cloned the other students' buttons onto the local machines and executed them to see what occurred.
- Sharing objects: All students share a turtle object on the server and manipulated it from their own programs simultaneously.

In the questionnaire survey we conducted after the lecture, some students commented that "I could see the programs in

Table 4: The curriculum of the lecture

Lesson	Contents
1st. day (AM)	Relation between computers and software; Install Java and Dolittle
1st. day (PM)	Your first program
2nd. day (AM)	Repetition using timer
2nd. day (PM)	Method definition, Network
3rd. day (AM)	Work on your own program
3rd. day (PM)	Presentation of own programs

Figure 16: A scene of the lecture

the computer communicating with each other." This result indicates that the students, who had never thought about the mechanism of WWW or mobile phone networks, understood the mechanism of data communication.

The distributed sharing version of Dolittle becomes a tool for international exchange if we extend the network from a LAN in a classroom to the Internet.

We are currently working on:

- Multilingualization
  Dolittle currently supports Japanese, English, and Korean. Students who use different languages are able to exchange their programs and execute them without translation. In future, Dolittle will support Chinese and many other languages.
- Internet server
  Locating object servers on the Internet will enable students to share objects over the Internet.
- Applets
  The applet version of Dolittle will enable students to execute Dolittle programs on Web browsers.

Figure 17 shows a sample screen from the Korean version.

6. Conclusion

Object-oriented languages are suitable languages for beginners in order to learn programming. In this paper, we introduced a programming language called Dolittle which has simple syntax and does not require class/function/variable declarations. Then we examined the practicality of Dolittle through the experimental lessons. In the small experimental lessons for high school students, we confirmed that students understood the principles of software through the programming experience. In the large experimental lessons for junior

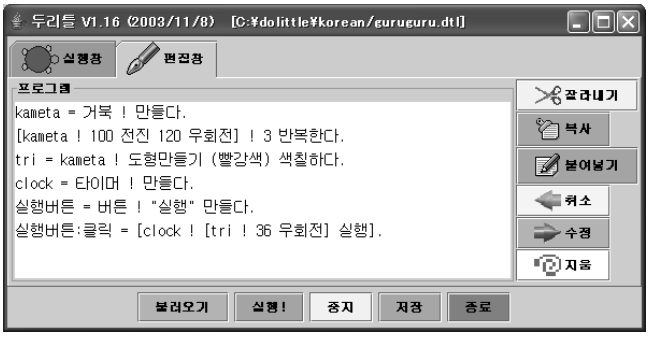

Figure 17: A screen from the Korean version

high school students, we made sure that the programming lesson could be conducted in classrooms using Dolittle. Then we verified that elementary and junior high school students were able to learn programming effectively using robots controlled by programs. Then we introduced the idea of making Dolittle objects network capable and discussed the lessons learned using the distributed sharing version of Dolittle. Finally, we explained our future conceptions of Dolittle. In our future work, we hope to develop more programs using the distributed sharing feature and study the usage of Dolittle in international exchange.

Acknowledgment

We are grateful to the following schools and company for their cooperation on the experimental lessons: High School at Otsuka, University of Tsukuba, Kamata Junior High School, Matsuzaka-city, Mie-prefecture, Nishi-mashizu Junior High School in Fujieda-city, Shizuoka-prefecture, Kanazawa Elementary School, Chiba-city, Chiba-prefecture and Ricoh Co., Ltd.

References

[1] Dolittle programming language. http://www.logob.com/dolittle/.

[2] Measurement and control using Logob and Dolittle. http://www.logob.com/users/seigyo/.

[3] Adele Goldberg and David Robson. *Smalltalk-80: The Language and Its Implementation*. Addison-Wesley, 1983.

[4] Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay. Back to the future: the story of Squeak, a practical Smalltalk written in itself. In *Proceedings of the 1997 ACM SIGPLAN conference on Object-oriented programming systems, languages and applications*, pp. 318–326, 1997.

[5] Alan Kay. Etoys and simstories in Squeak. http://www.squeakland.org/author/etoys.html.

[6] Seymour Papert. *Mindstorms : children, computers, and powerful ideas*. Basic Books, 1980.

[7] Masashi Umezawa, Kazuhiro Abe, Satoshi Nishihara, and Tetsuya Kurihara. NetMorph — an intuitive mobile object system. In *Proceedings of the Conference on Creating, Connecting and Collaborating through Computing (C^5)*, pp. 32–39, 2003.

[8] David Ungar and Randall B. Smith. Self: The power of simplicity. In *OOPSLA'87*, pp. 227–242, 1987.