Dolittle: an object-oriented language for K12 education

Susumu Kanemune, Yasushi Kuno

Hitotsubashi University and Tsukuba University, Japan

Hitotsubashi University Computer Center, 2-1 Naka, Kunitachi City, Tokyo, Japan

kanemune@cc.hit-u.ac.jp, kuno@gssm.otsuka.tsukuba.ac.jp

Abstract

We introduce Dolittle, an object-oriented (O-O) programming language suitable for K12 education. One of the authors developed a LOGO interpreter named Logob in 1990 and it was used in many schools in Japan. However, today's software is largely dependent on object-orientation, and modern software elements such as GUI components or animated graphics, which we consider mandatory to attract children's interests, are difficult to use without O-O. Therefore we have developed Dolittle in year 2000. Dolittle adopts prototype-based O-O (as in Self or JavaScript), so children can use O-O without learning complex constructs like class or inheritance. To ease children's learning, Dolittle programs can be expressed in multiple localized languages, e.g. Japanese, Korean, English, and so on. In this paper, we describe outline of the language, along with our experiences in elementary, junior-high and high school classrooms. Finally, we introduce two applications of Dolittle that we are currently working on: distributed programming in a junior high school and collaboration programming between some countries.

Keywords

Programming, object-oriented, Dolittle

1. Introduction

Programming experience 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[1][2], and have been conducting experiments by teaching Dolittle in classes. As a result, we are confident that Dolittle can be used in classes and students can learn various aspects of computers through their programming experience.

One of the authors developed a LOGO interpreter named Logob[3] in 1990, which have been used in many schools in Japan. However, LOGO does not have object-orientation[4][5] as its core component. Modern software makes extensive use of O-O, and widely used software concepts such as GUI elements (buttons, menus, ...) or animated graphics cannot be handled easily without O-O. We think it prerequisite for educational programming languages to properly incorporate these concepts, because without them children cannot realize connection between what they are constructing and “real” software they are daily using on their computers. This was our motivation for Dolittle, new object-oriented educational programming language.

The biggest problem was how to introduce concepts of O-O to children, which are said to be difficult even for professional programmers. After some thoughts, we have adopted the idea of LOGO[6][7] turtle graphics for drawing operation, with a few extensions namely: (1) every turtle is a self-contained object, and (2) every figure drawn is likewise. These "visible" objects were of great help to children grab the concept of object as a self-contained unit. We also used (3) the idea of cloning objects from prototype-based O-O languages such as Self[8] or JavaScript[9]. Prototype-based O-O does not need class constructs, which are cumbersome and difficult for children. We designed and implemented Dolittle in the year 2000. Dolittle is interpreter-based and does not require verbose declarations or lengthy compilation processes. Dolittle is written in Java[10], so that it runs on various computer environments used in schools.

There are several Logo-based educational environments, such as MicroWorlds[11] or Imagine[12]. Especially, the latter extends Logo with flexible class-based O-O facility. Strong points of those environments are that they form a kind of authoring system for educational materials; users can place GUI parts and other multimedia materials at will with mouse operations. On the other hand, what we have done in Dolittle is to provide image- and GUI- facilities in library objects, which can be called arbitrary from ordinary program code with minimal complexity. We have chosen this approach because we wanted to teach students that simple (O-O) languages can be very flexible and powerful with appropriate support objects; everything can be performed by means of program code.

In this paper, we first describe the outline of Dolittle. Then we report on the experiences of using Dolittle in classes. Finally, we introduce three applications of Dolittle that we are currently working on: robot control, distributed programming, and international communication.

2. A Programming Language Dolittle

When we designed Dolittle, we paid close attention to making it a simple language in order to use it in school lessons. Figure 1 shows a sample Dolittle program that is written in English.

```
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].
```

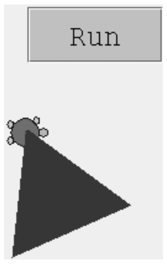


Figure 1. A sample Dolittle program and its execution

Dolittle programs call on objects using "!" symbol. The 1st statement sends a "create" message to the prototype object "turtle" to let it clone itself, and then assigns the created turtle object to the variable called "kameta." The period (".") indicates the end of the statement. As soon as a clone of the turtle object is created, its icon (by default turtle-like image) appears on the screen.

The square brackets ("[...]") represent a block. In the 2nd statement, "[...]! 3 repeat" sends a message "Execute yourself three times" to the block.

Dolittle interprets identifiers after "!" (such as "forward") as message selectors (operation names). When numeric literals (such as "100") or parenthesised expressions (such 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 statement. This is called a cascade sending of messages. The statement inside the block first sends message "forward" with the argument "100" to "kameta", and then message "rightturn" with the argument "120" to the returned object (same as "kameta" in this case). As the block is executed three times, the 2nd statement draws a regular triangle on the screen.

The lines drawn by "kameta" are 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.

The 3rd 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 predefined variable named "red" stores the color object, which represents red color (variables for other major colors are also predefined). Thus, the 3rd statement creates the triangular figure object, change its color to red, and then stores the figure to the variable "tri."

A "timer" is an object, which executes code pieces (blocks) at a specified interval for a specified period. The 4th statement creates a timer object "clock" and sets the execution period and the interval.

The 5th statement creates a button object, labelled as "Run", and assigns it to the variable "rBtn". When a button object is created, its pushbutton-like image appears on the screen.

The 6th statement assigns a block to the "click" attribute of the "rBtn" object. Dolittle defines a method (object's named action) 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, in which the triangle is rotated 36 degrees once a second, for the duration of 10 seconds.

3. Experimental Lessons

3.1. Experimental Lessons at High School

To evaluate Dolittle, we conducted small experimental lessons at senior high school of University of Tsukuba in Tokyo in 2000. Three first grade (16 years old) students 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 duration of one hour. Table 1 shows the curriculum of the lessons.

Table 1. Curriculum of Experimental Lessons at High School

Lesson	Contents
1	Explanation of messages and objects
2	Execute programs repeatedly at regular intervals using timers
3	Define methods

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

- The mechanism of monitor displays: 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: "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 became aware of the existence of programs in a computer (an operating system).

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 are controlled by 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 the existence of pixels and operating systems.

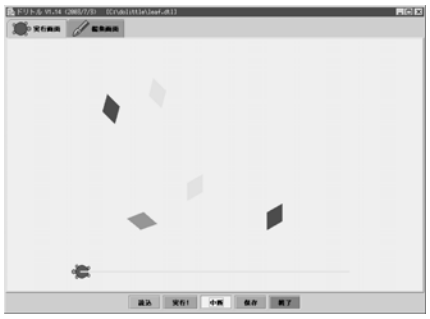


Figure 2. A piece of work from a high school student

3.2. Lessons at Junior High School

We conducted larger experimental lessons at a junior high school in 2001 to check if Dolittle is suitable for school lessons. The experimental lessons were held at Kamata Junior High School in Mie-prefecture, as a part of the technical training course. All of the 132 students in the second grade (14 years old) attended the lessons. A programming lesson was a new experience for all of them. Figure 1 shows a scene of the lesson. Table 2 shows the curriculum of the lessons.

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

Lesson	Contents
1-2	Turtle Graphics
3-5	Generate and Manipulate Figure Objects
6-8	Animation with Timer
9-11	Use of GUI Components (Button)

Table 2. Curriculum of experimental lessons

Figure 1 shows the results of the questionnaire surveys. We conducted the surveys at the end of every lesson. The students evaluated "enjoyment", "achievement", and "difficulty" of the lessons.

We analyzed the results of the surveys as follows:

- The lessons were conducted effectively to the end. Less than 10% of the students answered, "I could not complete the exercise of the day."

- The students felt "Hard Fun!" As the lessons progressed, more students felt "difficulty", but more students felt "enjoyment". The lecturer said, "They were filled with pleasure in achievement."

In lessons 9 to 11, the students designed drawing software, which draws lines and figures using buttons, as an exercise to summarize what they had learned. Figure 5 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 could create useful drawing software using what they had learned in the class.

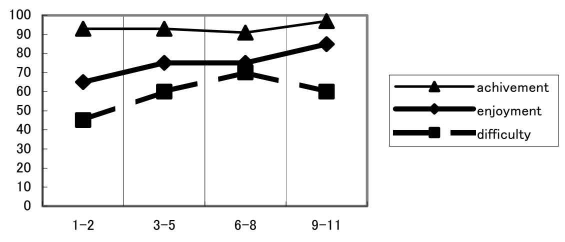


Figure 4. Questionnaire survey results through lessons

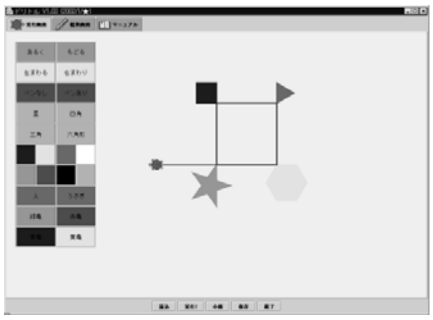


Figure 5. A student's work (drawing software)

Many students have mastered the usage of GUI components (especially buttons) and have shown strong interest in creating programs of “pushing of buttons activates a certain action” style. The lecturer observed, “By using GUI components in programs, such as the drawing software, the students could link the programming they had learned to the software used in the real world.”

3.3. Lessons at Company

Programming in Dolittle is also applicable to adult education. Figure 6 shows a scene of the lecture at a company. This company adopted Dolittle as the first language for newcomers. By learning Dolittle, learners could easily understand such concepts as: objects, variables, methods and so on. After learning Dolittle, they learned Java more effectively than the other classes that used only Java.

Figure 6. A scene of the lecture

4. Objects in the Real World

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

Dolittle provides some objects to control external devices. The sample program in this section controls a robotic car designed for schools.

Figure 7 shows a robotic car. The board has a one-chip PIC microprocessor. 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. As in Figure 7, a program written in Dolittle was transferred to the robotic car using infrared ray.

Below is a sample program, which controls a robotic car. It generates a “serialport” object “robot” and downloads a small program that controls the robotic car. (The robot's methods such as “forwarduntilcollision” send out corresponding numeric operation codes through the serial line.)

```
robot = serialport ! create.
robot:script = [
  ! startrobot switchstart
  10 back 15 rightforward 15 leftback

  forwarduntilcollision
  10 back 15 leftforward 15 rightback
  endrobot ].
robot ! "com1" opensesame.
robot ! script run closesesame.
```

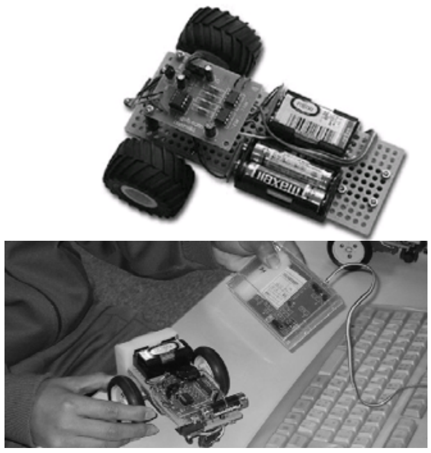


Figure 7. A robotic car, sample program and transferring programs using an infrared ray

Controlling robots was appropriate topic for lower grade education since students can touch it and see how it behaves. Figure 8 shows a scene of a lesson conducted during the comprehensive studies in the sixth grade (12 years old) of Kanezawa elementary school in Chiba-prefecture. 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.

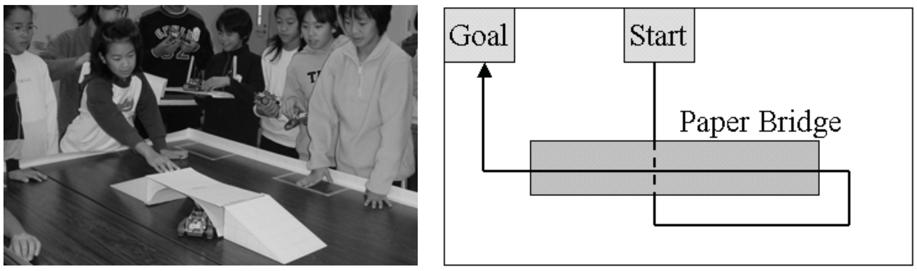


Figure 8. Robotic car contest in an Elementary School

Figure 9 (left) shows a scene of a lesson conducted in the technical training course at Nishimashizu junior high school in Shizuoka-prefecture. Students in the second grade (14 years old) programmed the robotic cars, which negotiated around obstacles in a simple maze to reach the goal.

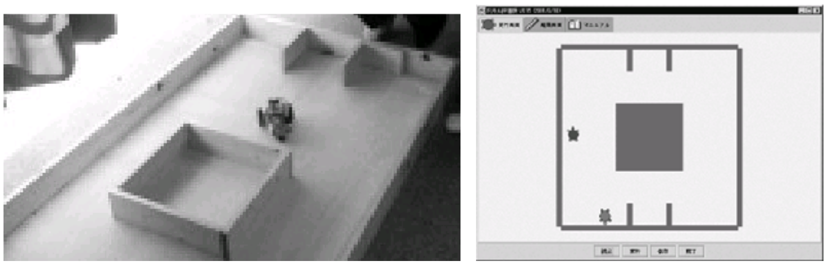


Figure 9. Maze and its simulation program in a Junior High School

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 some other objects. Figure 9 (right) shows a sample execution of the simulation.

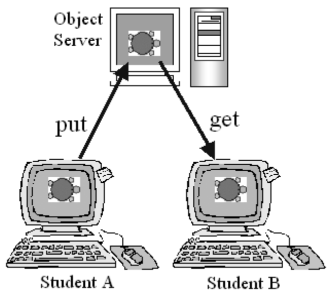
5. Local and International Communication

We are exploring further extensions of Dolittle, so that programming in Dolittle can support wider form of students' experiences. Here we present two cases: (1) object exchange in the class, and (2) program exchange between countries.

5.1. Object Exchange and Sharing in Class

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 extends this approach to the local area network (LAN). Figure 10 shows how one can clone objects across machines. The object server on the network manages objects. Object Server and Dolittle clients communicate with Java Remote Method Invocation (RMI) protocol.

```
kameta=turtle!create.
server!"sv1" connect.
server!"t1" (kameta) put.
```

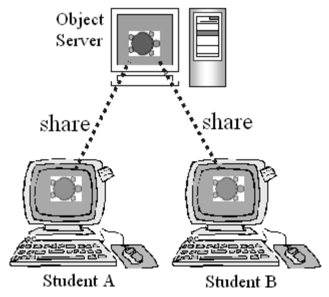


```
server!"sv1" connect.
turtle1=server!"t1" get.
turtle1!10 forward.
```

Figure 10. Registering, cloning 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.

```
kameta=turtle!create.
server!"sv1" connect.
server!"t1" (kameta) put.
turtle1=server!"t1" share.
```



```
server!"sv1" connect.
turtle1=server!"t1" share.
turtle1!10 forward.
```

Figure 11. Sharing objects

Using the "share" method in place of "get", multiple Dolittles can share single object inside the server. For example, when a turtle object is shared, "move" method call from one machine cause movement of the turtles in all participating machines.

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

We conducted an experimental lesson in the third grade (15 years old) in junior high school in 2004. Figure 12 shows a scene of a lesson conducted in the technical training course at Aojima junior high school in Shizuoka-prefecture. At first students programmed the ping-pong game standalone version shown in Figure 13. In this program, a ball, which is a turtle object, moves on a screen of single machine.

Figure 12. A scene of a lesson at junior high school

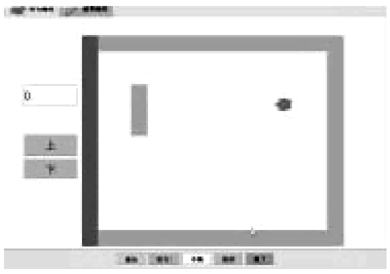


Figure 13. A ping-pong game (standalone version

Then, students are paired and each pair cooperatively developed the network version of the ping-pong game, as shown in Figure 14. In these programs, ball positions (x and y coordinates) and ball directions (signs of x and y) are stored in the Object Server. One of these programs reads ball directions and moves ball position on the server periodically. Both of these programs read ball positions from the server and put the ball on the Dolittle screen. When a ball crashes into a wall (or paddle), the wall sends the collision message to the ball. Then the ball changes own direction on the server.

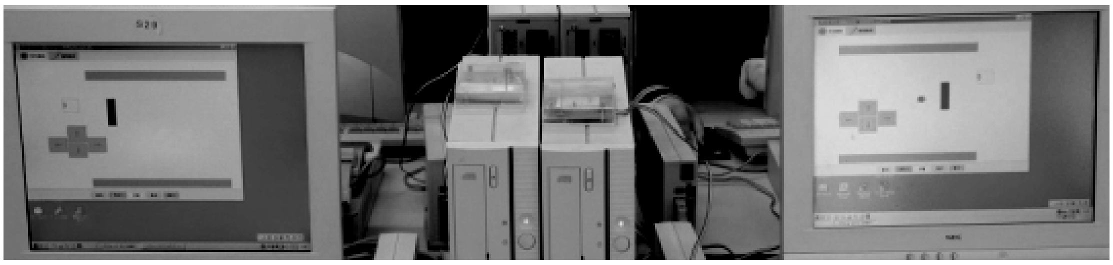


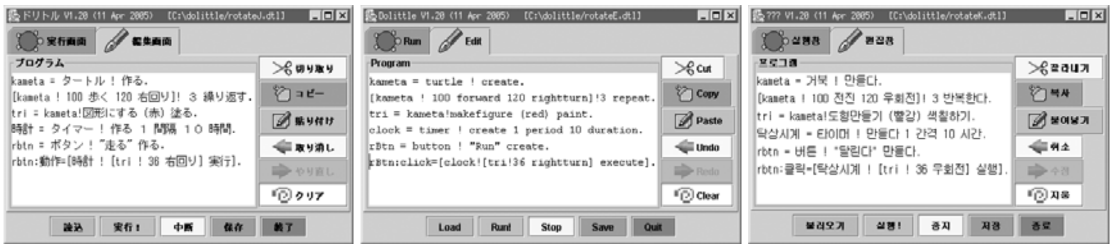
Figure 14. A ping-pong game (network version)

Through the experience of network programming, students could learn principles of network services (such as E-mail, WWW, cellular phone, etc) and roles of network servers.

Imagine[12] also provides network facilities, but their communication model is based on remote procedure call and passive data exchange, i.e. controls and data are separated. We have designed Dolittle's network facility to become inherently object-based; any object (with its own method) can be transferred between machines. Moreover, our object sharing can provide very abstract view of networked objects, so that students can easily experience powerful nature of networks.

5.2. International Program Exchange

Dolittle language dose not have any reserved word by itself; all of the predefined names designate standard objects (e.g. “Turtle”, “red”) or methods (e.g. “forward”, “execute”) through system dictionary. By changing this dictionary, we can easily convert Dolittle to other languages. Currently, Dolittle system for Japanese, English and Korean are distributed as shown in Figure 15. Moreover, simple word-by-word substitution can transfer a Dolittle program from one language to the other, leading to international program exchange.



```
kameta = タートル ! 作る.
[kameta ! 100 歩く 120 右回り]! 3 繰り返す.
tri = kameta!図形にする (赤) 塗る.
時計 = タイマー ! 作る 1 間隔 10 時間.
rbtn = ボタン ! "走る" 作る.
rbtn:動作=[時計 ! [tri ! 36 右回り] 実行].
```

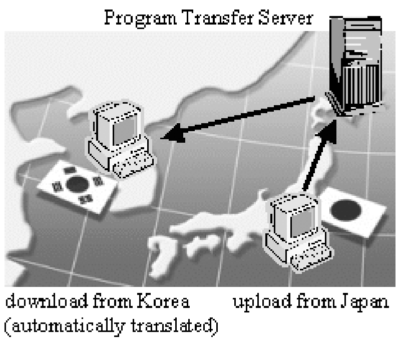
```
kameta = turtle ! create.
[kameta ! 100 forward 120 rightturn]!3 repeat.
tri = kameta!makefigure (red) paint.
clock = timer ! create 1 period 10 duration.
rBtn = button ! "Run" create.
rBtn:click=[clock![tri!36 rightturn] execute].
```

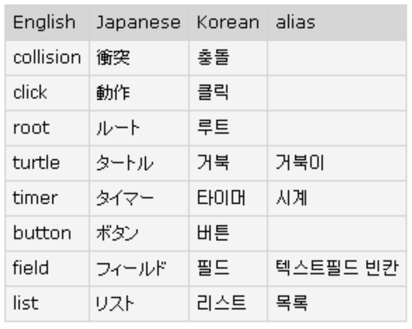
```
kameta = 거북 ! 만들다.
[kameta ! 100 전진 120 우회전]! 3 반복한다.
tri = kameta!도형만들기 (빨강) 색칠하기.
탁상시계 = 타이머 ! 만들다 1 간격 10 시간.
rbtn = 버튼 ! "달린다" 만들다.
rbtn:클릭=[탁상시계 ! [tri ! 36 우회전] 실행].
```

Figure 15. Screens from the Japanese, English and Korean version

To facilitate the exchange, we are developing Program Transfer Server. User can upload program from Dolittle and can download with language translation. Figure 16 shows a structure of Program Transfer Server and Translation Table. Program Transfer Server and Dolittle clients communicate with XML-RPC protocol.

We used simple token-by-token translation (e.g. no inflection support), but as we wanted to teach difference between programming languages and natural languages, we think this design appropriate. In reality, Korean researchers registered multiple inflected forms of single Korean word as corresponding to single Japanese token, for the convenience of their students.





English	Japanese	Korean	alias
collision	衝突	충돌	
click	動作	클릭	
root	ルート	루트	
turtle	タートル	거북	거북이
timer	タイマー	타이머	시계
button	ボタン	버튼	
field	フィールド	필드	텍스트필드 빈칸
list	リスト	리스트	목록

Figure 16. Program Transfer Server and Translation Table

Using Program Transfer Server, we communicated with some students at the Korea University. Figure 17 shows a sample program, which we exchanged. From Japan, we uploaded the music program, and from Korea, the students downloaded it and executed it to play music. (For this explanation, it was also translated in English)

```
かえる=メロディ！作る。
かえる！『ドレミファミレド～ミファソラソファミ～』追加。
かえる！『ド・ド・ド・ド・ドドレレミミファファミ・レ・ド～』追加。
ピアノ1 =楽器！『ピアノ』作る。
ピアノ1！(かえる) 設定。
ピアノ2 =楽器！『オルガン』作る。
ピアノ2！(メロディ！8 無音(かえる) 追加) 設定。
楽器！演奏。
```

```
frog=악보!만들다.
frog!"도 레 미 파 미 레 도 ㅁ 미 파 솔 라 솔 파 미 ~"add.
frog!"도.도.도.도.도 도 레 레 미 미 파 파 미.레.도 ~"add.
피아노1 =악기!"piano"만들다.
피아노1!(frog) set.
피아노2 =악기!"organ"만들다.
피아노2!(악보!8 무음(frog) add) set.
악기!연주.
```

```
frog = melody ! create.
frog ! "do re mi fa mi re do - mi fa sol ra sol fa mi -" add.
frog ! "do.do.do.do.do do re re mi mi fa fa mi.re.do -" add.
piano1 = instrument ! "piano" create.
piano1 ! (frog) set.
piano2 = instrument ! "organ" create.
piano2 ! (melody! 8 silent (frog) add) set.
instrument ! play.
```

Figure 17. Music programs in Japanese and translated in Korean and English

We are currently improving upon the Program Transfer Server and intend to conduct experimental lessons at K12 schools.

6. Conclusion

Object-orientation is widely adopted in current software development, and we think it mandatory for today's educational programming languages. With such languages only, children can realize principles and functions of current state-of-the art computers and software. Dolittle was carefully designed to archive the goal stated above, and is simple but also powerful that children can easily use GUIs and animated graphics in their programs. In our experience, Dolittle was successfully taught in elementary, junior-high and high schools classes and could also be used for robot control programming classes, or for in-class/international program exchange.

7. Acknowledgments

We are grateful to the following teachers and schools/companies for their cooperation on the experimental lessons: High School at Otsuka, University of Tsukuba, Mr. Idosaka at Kamata Junior High School, Mr. Junn Ohta at Ricoh Co., Ltd., Mr. Kazuhiro Sato at Kanazawa Elementary School, Mr. Shuji Kurebayashi at Nishimashizu Junior High School, Mr. Hirofumi Nishigaya at Aojima Junior High School and Prof. Won-Gyu Lee at Korea University.

This project was partially funded by the Information-technology Promotion Agency, the Nissan Science Foundation, and the Kozuki Foundation for Sports and Education.

8. References

[1] *Dolittle Programming Language*, http://kanemune.cc.hit-u.ac.jp/dolittle/

[2] Susumu Kanemune, Takako Nakatani, Rie Mitarai, Shingo Fukui, and Yasushi Kuno (2004). *Dolittle - Experiences in Teaching Programming at K12 Schools*. The Second International Conference on Creating, Connecting and Collaborating through Computing, IEEE, 177-184

[3] *Logo Information Room*, http://kanemune.cc.hit-u.ac.jp/logo/

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

[5] Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace and Alan Kay (1997), *Back to the future: the story of Squeak, a practical Smalltalk written in itself*, ACM SIGPLAN

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

[7] Brian Harvey (1985), *Computer Science Logo Style*, Vol. 1-3, The MIT Press

[8] David Ungar and Randall B. Smith (1987), *Self: The Power of Simplicity*, OOPSLA'87, 227-242

[9] European Computer Manufacturers Association (1999), *ECMAScript Language Specification*, http://www.ecma.ch/ecma1/stand/ecma-262.htm

[10] James Gosling, Bill Joy and Guy Steele (1996), *The Java Language Specification*, Addison-Wesley

[11] LCSI, http://www.microworlds.com/

[12] L'ubomir Salanci (2001), Networking in Logo, EuroLogo 2001