Project Proposal

Mirage

A Graphical and Parallel Sketching Language

Columbia University
COMS W4115 Programming Languages and Translators
Fall 2006

Prof. Stephen Edwards

Team Members:

Abhilash I	**ai2160@columbia.edu**
Ming Liao	**ml2288@columbia.edu**
Nalini Vasudevan	**nv2144@columbia.edu**
Peili Zhang	**pz2128@columbia.edu**

I. Overview

Motivation:

The main objective for our project is to develop an efficient graphical language that not only does assist the learning process for the beginners, but also emphasizes on the parallel processing and makes the compiler to be able to trace each of the parallel blocks independently.

In addition to build a compiler that makes programming efficient for certain applications specific fields, we will develop a compiler that can also support parallelism as well as provide a debugging feature for programmers.

Introduction:

The **Mirage** programming language is designed to be a graphic-specific, functional, and translated programming language, with the target language being Java. The graphic output of the language produced from the *Java.swing* package since it provides sufficient utilities to generate any 2D diagrams.

Similar to Logo programming language, it is easy to read, write and learn. It is used mainly for educational purpose, especially for teaching and inspiring programming beginners. It can also enable more advanced programmers to recursively create fantastic “Fractal like” graphics. Not only can it create "mirage graphics", but also it has useful facilities for handling image files and I/O. Moreover, Mirage can be used to teach most basic computer science concepts.

Mirage can do simple actions in motional way. For example, the motion of move forward 50 pixels or turn certain angle clockwise can be shown on the screen. From the small building blocks, programmers are able to build more complex geometries (e.g. rectangular, oval, triangles) and graphics (e.g. fractal). In addition, the idea of Mirage graphics is useful for example in Lindenmayer system for generating fractals.

II. Language Features

First of all, we decide that PEN and ERASER are two pre-defined objects, the former is used to draw geometries and graphics and the later is used to erase certain previous drawings.

Primitive Data Types:

There is only one basic numerical data type in Mirage, namely VAR, which is the type of integer. It is used for specifying the moving distances and turning angles of PEN, as well as for doing more advanced calculations to enable programmers to create complex geometries or graphs.

Key words

All the key words are written in uppercase in the purpose of helping programmers to easily differentiate with non-key words. For example, “PENUP”, “PENDOWN”, “RETURN”, “BREAK”, and “PARALLEL”, etc… …

Comparative and Mathematical Operators:

Operators are necessary for condition checking and calculations. Mirage have all the essential comparative and mathematical operators, for examples, = =, ! =, >=, =<, +, -, *, /, <, >, =, ( ), &&, etc…

Control Statements:

To control program’s flow and carry out certain algorithms, we will implement and support the following control statements:

- WHILE loop: for example, we can use a WHILE loop to draw certain geometries repeatedly.
- FOR loop: for example, we can use a FOR loop to repeat a line certain times.
- IF /ELSE condition: for example, depends on different condition, we can use IF/ELSE to draw different geometries.

Other Features:

- Each procedure or function encloses a group of statements by ‘{‘ and ‘}’. Each statement is separated by a ‘;’.
- Comments start with ‘/*’ and ends with ‘*/’.
- Motion of PEN include “FORWARD”, “BACKWARD”, “CLOCK”, “ACLOCK”, etc;
- PEN State include PENUP, PENDOWN, PENERASE;
- PEN COLOR can be any color, for example, one can use color code <R, G, B> to specify certain color.
- **Parallelism**: one can start two or more threads to create different graphics simultaneously. The parallel threads will follow an asynchronous rendezvous. In other words, the main program forks into two or more and joins simultaneously.
- One can specify the maximum time that the main program can wait before the children converge.

Syntax:

The syntax of Mirage resembles Logo but with some modifications. For example, some basic syntax will be as follows:

```
VAR variable_name;     /* all VAR are Integer type variable:
                       It is a local variable if defined in a procedure
                       or a global variable if passed to any procedure */
```

```
PENUP; or PENDOWN;      /* indicate the state of PEN */

COLOR <255, 0, 0>       /* use color code to specify the color of PEN */

THICKNESS t;      /* specify the thickness of PEN where t range in [1, 10] */

SPEED sp;         /* specify the speed of PEN where sp range in [1, 10] */


FORWARD 40; or BACKWARD 30;   /* specify the moving direction and distance of
                              PEN */

CLOCK 20; or ACLOCK 5;        /* specify turning angles of PEN */
```

Other key words includes:
RETURN, BREAK, PARALLEL, IMAGE, OPEN, READ, WRITE, CLOSE, RESET, etc…

```
WHILE(condition){

      /* statement */;
      ... ...

}

FOR(i=0; i<10; i++){

      /* statement */;
      ... ...
}

IF(condition){
      /* statement */;
      ... ...
}
ELSE IF(condition){
      /* statement */;
      ... ...
}
ELSE{
      /* statement */;
      ... ...
}

REPEAT 4 { /* statements */;  .... ... } /* statements */... ... ;

procedure_name(a, b){

      /* statement */;
      VAR c = a + b;
      ... ...
}

MAIN(){
      /* statements… */;
      VAR a, b, c;
      a = 6;
      b = 7;
      procedure1 (a, b);
      c = 4;
      ... ...

}

/* overall structure in a program might look like: */

procedure1()
{
      ... ...
}

procedure2()
{
      ... ...
}

/* … statements … */;

MAIN()
{
      ... ...

      {
            /* statements */
            ... ...
      }

      PARALLEL

      {
            /* statements */
            ... ...
      }

      ... ...

}
```

III. Design Goals

Mirage*: A simple, intuitive, efficient, flexible, robust, high-performance, interpreted, architecture neutral, portable language.*

Simplicity and Intuitive:

Mirage uses simple, consistent and intuitive syntax that are easy for beginners to learn and understand, as well as allows experienced users to read and write quickly. Code definitions in Mirage are structured and blocked, which increases the readability of the language.

Efficiency and Flexibility:

Mirage can be implemented in a very efficient way with minimal system resources utilized. Since it is a simple graphic/geometry domain-specific language, it has a small set of commands and operators. Programs written in Mirage can be compiled very quickly. On the other hand, Mirage can also provide users flexibility for designing any kind of geometries or graphs. Moreover, Mirage employs flow control mechanisms such as IF/ELSE conditional checking and WHILE and FOR loops.

Robust and High-performance

Not only does Mirage enable programmers to create arbitrary geometries or fantastic graphics, it can also be used to help programmers to simulate drawing in a Polar Coordinate System. Provided ability to perform calculation and function of parallelism, a programmer can create complicated graphics in parallel mode. Errors in the Mirage code are detected at compile time and guarantee the compiled Mirage code to be accurate.

Portability:

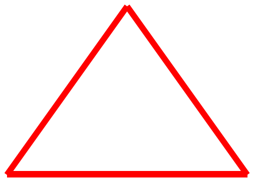
Since we use ANTLR as the syntax recognizer, compiler will accept Mirage code as an input and generate the corresponding Java code. This source code operates on Java Virtual Machine, thus all Mirage code can be interpreted and executed on any machine that have a Java Runtime Environment. Users are able to create any geometry or graphics in the same way on a Windows platform as on a Unix platform. Thus, Mirage is a platform independent and portable language.

IV. Sample Programs

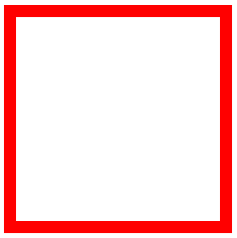
Three sample Mirage programs being traced shown below, which will draw a trianle, a square, and an arbitary shape using parallelism. Althought Mirage code looks very simple, but one can group sets of instructions, employ recursive algorithms, parallel processes to create more complex graphics. Key words are usually to be written in upper case , while user-defined variables and function names are written in lower case. (Note: the codes in italics run in parallel.)

```
/*Program to draw a triangle*/
MAIN()
{

      PENDOWN;               /*Ready to draw*/
```

```
        COLOR <255, 0, 0>;      /* set color to red */
        THICKNESS 2;            /* Set pen thickness to 2 */
        SPEED 5;                /* set pen speed to 5 */

        CLOCK 30;               /* Turn the pen by 30 degree */
        FORWARD 40;             /* Draw a line by 40 pixel */

        CLOCK 120;              /* Turn the pen by 120 degree */
        FORWARD 40;             /* Draw a line by 40 pixel */

        CLOCK 120;              /* Turn the pen by 120 degree */
        FORWARD 40;             /* Draw a line by 40 pixel */


        PENUP;                  /* stop to draw */

}
```



```
/*Program to draw a square*/

MAIN ()
{
        VAR i;          /* declare the variable i */

        PENDOWN;        /* ready to draw */

        COLOR <255, 0, 0>;      /* Set color of the pen to red */
        THICKNESS 5;            /* Set pen thickness to 5 */
        SPEED 3;                /* set pen speed to 3 */

        FOR (i=1; i<=4; i++)    /* Repeat 4 times */
        {
                FORWARD 40;     /* Draw a line by 40 pixel */
                CLOCK 90;       /* Turn the pen by 90 degree */
        }

        PENUP;                  /* stop to draw */
}
```



```
/* Program to draw an arbitrary shape */
MAIN()
{
      COLOR <255, 0, 0>;  /* Set color of the pen to red */
      THICKNESS 3;        /* Set pen thickness to 3 */
      SPEED 3;            /* set pen speed to 3 */

      PENDOWN;            /* ready to draw */
      FORWARD 40;         /* Draw a line by 40 pixel */

      /* First parallel thread */
      {

            ACLOCK 45;    /* Turn the pen to left by 45 degree  */
            FORWARD 20;   /* Draw a line by 20 pixel */
            PENUP;        /* stop to draw */
      }

      PARALLEL    /* parallelism */

      /* Second parallel thread */
      {
            CLOCK 45;    /* Turn the pen to right by 45 degree  */
            FORWARD 20;  /* Draw a line by 20 pixel */
            PENUP;       /* stop to draw */
      }

      PARALLEL    /* parallelism */

      /* Third parallel thread */

      {
            CLOCK 90;    /* Turn the pen to right by 90 degree  */
            FORWARD 30;  /* Draw a line by 20 pixel */
            PENUP;       /* stop to draw */

      } 1000;           /* Main function wait for only 1000 milliseconds
                        before three threads converge */

      FORWARD 30;        /* Draw a line by 30 pixel */

      PENUP;             /* stop to draw */

}
```