Cocoon

Ramzi Abdoch
Language Guru

Neha Aggrawal
System Integrator

Lawrence Candes
Project Manager

Clementine Barbet
System Architecture

Swikriti Jain
Testing
Who here takes Selfies?
Do you also use Instagram?

- Filters, how do you use them?
- There’s only like 15, boring, right?
- Define our own filters
Buzzwords + Usability

Current

Readable

Functional

Efficient

Interactive
Cocoon v.s. Java

- Significant reduction in code length
- Much more readable
- Inspired by simple Python + Java Syntax, so easy to learn

<table>
<thead>
<tr>
<th>Java</th>
<th>Cocoon</th>
</tr>
</thead>
</table>
| try {
  BufferedImage master = ImageIO.read(new File("/path/to/file.bmp"));
  BufferedImage gray = ImageIO.read(new File("/path/to/file.bmp"));
  for (int x = 0; x < gray.getWidth(); x++) {
    for (int y = 0; y < gray.getHeight(); y++) {
      Color color = new Color(gray.getRGB(x, y));
      int red = color.getRed();
      int green = color.getGreen();
      int blue = color.getBlue();
      red = green = blue = (int)(red * 0.299 + green * 0.587 + blue * 0.114);
      color = new Color(red, green, blue);
      int rgb = color.getRGB();
      gray.setRGB(x, y, rgb);
    }
  }
} catch (IOException ex) { } | main() {
  img f;
  f = batch[i]
  greyscale(f);
  sysout(f); |
System Architecture

Cocoon Program → Lexer

JSON File → Preprocessing

Input Images → Output Java Executable

Output Images → ImageMagick

Lexer → Parser

Parser → AST

AST → Symbol Table

Symbol Table → Target Code Generation

Target Code Generation → Java Compiler

Java Compiler → Output Java Executable

Output Java Executable → Target Code Generation
Design Issue: Specification of Batch

Option 1: Using directory structure

PROS:
- Straightforward for the user

CONS
- More complex implementation
- Environment dependent
- Requires to define some ordering within directory files which may bring done usability

Option 2: Using a description file

PROS:
- Easier implementation
- Environment independent

CONS
- Slightly intimidating for the layman
- Simpler implementation
- Requires additional parsing
Design Issue: Specification of Batch

Option 1: Using directory structure

PROS:
- Straightforward for the user

CONS:
- More complex implementation
- Environment dependent
- Requires to define some ordering within a directory which may bring done usability

Option 2: Using a description file

PROS:
- Easier implementation
- Environment independent

CONS:
- Slightly intimidating for the layman
- Simpler implementation
- Requires additional parsing
Configuration JSON File

```json
{
    'batch1': { 'folder_label': 'D:\spring_break_pics'},
    'roses': { 'image_label1': './output/test.jpg',
                'image_label2': './output/test2.jpg'},
    'batch3': { 'image_label3': 'D:\Columbia_pics\11_20_111.jpg'}
}
```
System Architecture

- **Lexer**
- **Parser**
- **AST**
- **JSON File**
- **Preprocessing**
- **Symbol Table**
- **Target Code Generation**
- **Java Compiler**
- **ImageMagick**

**Input Images**
- **Output Java Executable**
- **Output Images**

**Cocoon Program**

**Output Images**
Symbol Table

Key Value mapping used to perform:
1. Type checking
2. Type Compatibility of Operators
3. Handles declaration in a given scope.
4. Performs necessary checks for existence of declared identifiers in the scope.
Data Types + Functions

DATA TYPES:
- **Image** - *URL hidden in symbol table*
- **Batch** - Collection of Images, JSON-encoded, defined in preprocessing step

FUNCTIONS:
- **Sysout** - Output the filtered image in the output folder.
- **Filter** - standard library functions
- **Pipe** - sequential collection of filters / other pipes
public class CocoonRunner{
    private static ArrayList<String> batch1;
    private static ArrayList<String> roses;
    private static ArrayList<String> batch3;
    static{
        batch1=new ArrayList<String>();
        roses=new ArrayList<String>();
        batch3=new ArrayList<String>();
        roses.add("./output/test.jpg");
        roses.add("./output/test2.jpg");
        batch3.add("D:/Columbia_pics/11_20_111.jpg");
    }
    public static void main(String args[]){
        try{
            Image m;
            Image m1;
            Image m2;
            m=new Image(roses.get(0));
            m1=m.solarize();
            m2 = grayscale(m);
            sysout(m1);
            sysout(m2);
        }catch(MagickException e){
            e.printStackTrace();
        }
    }
}
Results
How to Compile and Run

- `make compileall`
  Compiles AST, Parser, ST, Preprocessor, and CG

- `make run`
  Runs `Parser.java` on `<input_program>.silk` and generates `CocoonRunner.java`

- `make runCocoon`
  links libraries and runs Cocoon program
Testing

- Unit Testing
  Testing of individual component at every step by each team member.

- Integration Testing
  We wanted to build an automated test suite to perform regression testing however due to the delay in completion of the project we could not implement this, we did manual tests with various input programs to check if all modules were functioning well.
Demo

Let's start with a HelloWorld program:
This is a basic program in Cocoon to convert a color image to grayscale.
Demo 2

This program applies the greyscale, newspaper, edge and solarize filters on the image m.
Demo 3

This demo shows an application of a pipe which is a bunch of filters to an image.
Lessons Learned

- Meet often (even if its just for a coffee :) )
- A good makefile goes a long way
- Code early, code often!
- Breaking down the project into smaller modules helps a lot!
- Watching your compiler run and the generated code run is a very satisfying experience!
Questions?

Ceci n’est pas une pipe.