The javaTEX project & web2java*

Timothy Murphy
<tim@maths.tcd.ie>
School of Mathematics, Trinity College Dublin

29 July 1996

Abstract

The general aim of the javaTEX project is to examine the relation — if any — between Java and TEX, and in particular (as a first step) to translate the standard TEX programs (`tangle`, `tex`, `mf`, etc) into Java, using `web2java` — a simple modification of the standard UnixTeX utility `web2c`.

Contents

1 Project details

All the material in the project is in the public domain, and can be retrieved by anonymous FTP from `ftp://ftp.maths.tcd.ie/pub/TeX/javaTeX`. (It is hoped later to submit the programs to the CTANs.)

Any suggestions and contributions are very welcome.

A mailing list has been set up. To join this, email `majordomo@maths.tcd.ie` with the message (not heading)

```
subscribe javatex
```

The material forming the project is also available through the mailing-list. You can obtain this file, for example, by emailing `majordomo@maths.tcd.ie` with the message

```
get javatex javaTeX.tex
```

*This document — possibly extended and updated — can be retrieved from `ftp://ftp.maths.tcd.ie/pub/TeX/javaTeX/javaTeX.tex`

To obtain a list of available files, send the message

```
index javatex
```

For information on the working of `majordomo` send the message

```
help
```

to `majordomo@maths.tcd.ie`.

2 Why Java?

Java is a dialect of C; and 95% of the code produced by `web2java` is indistinguishable from C. At the same time, Java is much safer than C; for example, run-time checking of array-bounds catches many errors that would be missed by C.

Java is object-oriented, which is of course a Good Thing. Although in fact we have made very little use of this — apart from hiving off a few functions (I suppose we should say "methods"!) like `reset` and `rewrite` into `TeXlib.class`. But it would be nice to have a generic DVI driver which sent "messages" to an abstract `printer`, which could be "instantiated" as a PostScript printer, a screen viewer, etc.

Java has a simple graphics interface, although again we have made no use of this to date. It might prove *too* simple for the purposes of TEX.

Java also has a simple network interface, which could prove useful in integrating TEX into WWW (the World-Wide Web).

A theoretical advantage of Java is that it compiles into device-independent "bytecode" format, so that, for example, `tangle.class` compiled on a Sun should run under Windows 95.

There remains the intriguing possibility of DVI "Applets", as a way of putting TEX on the Web.

Against all these advantages there is at present one overwhelming disadvantage — Java is extremely slow. For example, it takes about 40 times as long to `tangle tex.web` in Java as it does in C. Doubtless this ratio can be somewhat reduced by better coding; but it seems improbable that it could be improved more than four-fold.

Our hope must be that a true Java compiler will soon be available, converting Java into machine code. Since Java is a standard programming language there seems no reason why such a compiler should not emerge; we would exect to see one before the end of the year.

3 How `web2java` works

`Web2java` — like `web2c` — is a post-processor to `tangle`. To create `foo.java` from `foo.web` and `foo.ch` one first runs `tangle`:

```
tangle foo.web foo.ch
```

This creates the Pascal (or pseudo-Pascal) file `tangle.p`.

(If you like driving in the slow lane, you could run the Java tangle instead:

```
java javaTeX.tangle foo.web foo.ch
```

Class files are supposedly machine independent, so `tangle.class` from the javaTeX distribution should run on any system. Note that this file, like all javaTeX programs, is defined to be in the `javaTeX` package, and so must be placed in a subdirectory called `javaTeX` relative to the `CLASSPATH`.)

This file is then passed through `web2java` to create `foo.java`:

$$\texttt{foo.web} + \texttt{foo.ch} \xrightarrow{\texttt{tangle}} \texttt{foo.p} \xrightarrow{\texttt{web2java}} \texttt{foo.java}.$$

Actually, this is a slight oversimplification. The file `common.defines` is prepended to `foo.p` *before* passing through `web2java`. And the resulting file is passed through `javafix.perl` *after* `web2java`:

$$\texttt{common.defines} + \texttt{foo.p} \xrightarrow{\texttt{web2java}} \mid \xrightarrow{\texttt{javafix.perl}} \texttt{foo.java}.$$

All this is completely analagous to `web2c`, except that we have replaced the C program `fixwrites.c` by the Perl script `javafix.perl`.

4 The `web2java` program

The filter `web2java` is created by the programs `flex` and `bison` (or `lex` and `yacc`) from the files `web2java.l` and `web2java.y`. This is completely analagous to `web2c`.

The file `web2java.l` is identical to `web2c.l`, with the addition of 5 new tokens: `try`, `catch`, `finally`, `new` and `cast`.

The syntax description in `web2java.y` has rather more changes, compared with `web2c.y`.

On the plus side, since Java has no pointers all the pointer-related material has been deleted. There is no attempt to determine if a function argument is "formal var" or not; and no need therefore to re-name functions with such arguments.

Classes have been introduced in a rather crude way, by allowing

```
VARIABLE = VARIABLE '.' VARIABLE
```

Thus in `Math.abs(...)` `Math` has been defined as a variable (`@define var Math;`).

On the other hand, in

```
web_file: Data_Input_Stream;
```

the class `DataInputStream` is defined as a type (`@define type DataInputStream;`).

No class (so far) has appeared in both rôles, so no conflict has arisen. (An earlier version, with `class` and `object` tokens, proved too complicated.)

5 Implementation

For the most part Web to Java is if anything simpler than Web to C. One apparent difficulty is the lack of a pre-processor in Java, since `web2c` leaves a good deal of work to `cpp`. This means that more must be done in the change file, which is probably a Good Thing.

The 3 main issues which arise are:

- The absence of `goto`s in Java;
- The lack of `typedef`s in Java; and
- Input/Output.

These are discussed in the following 3 subsections.

5.1 Removing `goto`'s

Java has no `goto`; Perhaps in compensation, it allows `break` and `continue` statements to carry a *label*, as eg in `break lab21` or `continue lab3`. The corresponding labels must appear at the beginning of the loop in question. (A `break` label can also be attached to a `switch` statement, but we make no use of that.)

We implement this as follows: `goto <num>` or `goto +<num>` are converted to `break lab`, as eg

$$\texttt{goto 6} \mapsto \texttt{break lab6}, \quad \texttt{goto +17} \mapsto \texttt{break lab17}.$$

On the other hand `goto -<num>` is converted to `continue` , as eg

$$\texttt{goto -7} \mapsto \texttt{continue lab7}.$$

Finally, `goto 0` is converted to `break`:

$$\texttt{goto 0} \mapsto \texttt{break}.$$

Labels of the form ±`<num>:` are commented out:

$$\texttt{-20:} \mapsto \texttt{/* lab20 */}.$$

As we shall see, this allows old labels to be left.

In practice, new labels are needed in virtually all cases. Here is a simple example

```
while n<>2 do begin
  ... goto done ...
end;
done: ...
```

Here `done` is defined at the beginning of the web file:

```
@d done=30 {go here to exit a loop}
```

We alter this (in the change file, of course) to

```
@d Done=30 {go here to exit a loop}
@d done==+Done
```

and we modify the loop above to

```
Done: while n<>2 do begin
  ... goto done ...
end;
done: ...
```

This translates in the Pascal file to

```
30: while n<>2 do begin
  ... goto +30 ...
end;
-30: ...
```

and this in turn translates into the Java code

```
lab30: while n<>2 do begin
  ... break lab30 ...
end;
/* lab30: */ ...
```

It will be seen that in this case only one line needed to be changed, by the addition of the label `Done:`.

More often, 2 lines need to be changed. The following is a typical case.

```
reswitch: case ch of
  'x': ... goto reswitch ...
endcases;
```

At the beginning of the web file we read

```
@d reswitch=21 {go here to start a case statement again}
```

We change this to

```
@d Reswitch=21 {go here to start a case statement again}
@d reswitch==-Reswitch
@d break==goto 0
```

Now we alter the switch statement to

```
Reswitch: loop begin case ch of
  'x': ... goto reswitch ...
endcases; break end;
```

In the Pascal code this becomes

```
31: loop begin case ch of
  'x': ... goto -31 ...
endcases; goto 0 end;
```

which `web2java` converts to

```
lab31: while (true) { switch(ch) {
  case 'x': ... continue lab31 ...
} break; }
```

In practice all `goto` statements in the 'classic' web files can be dealt with in this way, introducing an appropriate loop if necessary. One can imagine cases where this would be very difficult, if not impossible. Fortunately these do not arise in practice. In fact virtually all `goto` statements can be eliminated as above, by following a very small number of practical rules.

Here is a fairly complicated case, from `dvitype.ch`. Recall that the material in `dvitype.web` between `@x` and `@y` is replaced by the material between `@y` and `@z`.

```
@x
@<Start translation of command |o| and |goto| the appropriate label to
  finish the job@>;
fin_set: @<Finish a command that either sets or puts a character, then
    |goto move_right| or |done|@>;
fin_rule: @<Finish a command that either sets or puts a rule, then
    |goto move_right| or |done|@>;
move_right: @<Finish a command that sets |h:=h+q|, then |goto done|@>;
show_state: @<Show the values of |ss|, |h|, |v|, |w|, |x|, |y|, |z|,
  |hh|, and |vv|; then |goto done|@>;
done: if showing then print_ln(' ');
@y
Done: loop begin
show_state: loop begin
move_right: loop begin
fin_rule: loop begin
fin_set: loop begin
@<Start translation of command |o| and |goto| the appropriate label to
  finish the job@>;
break end; @<Finish a command that either sets or puts a character, then
    |goto move_right| or |done|@>;
break end; @<Finish a command that either sets or puts a rule, then
    |goto move_right| or |done|@>;
break end; @<Finish a command that sets |h:=h+q|, then |goto done|@>;
break end; @<Show the values of |ss|, |h|, |v|, |w|, |x|, |y|, |z|,
  |hh|, and |vv|; then |goto done|@>;
break end; if showing then print_ln(' ');
@z
```

5.2 Type definitions

There are no `typedef`s in Java. In theory one could replace `typedef`s by class definitions, but that would add considerable complication to the code. Instead we simply change them to substitutions (as though in C changing `typedef`s to `#define`'s).

So for example we make the change

```
@x
@<Types...@>=
@!ASCII_code=0..255;
@y
@d ASCII_code==0..255
@z
```

Later `web2java` will replace this range `0..255` by an appropriate type (currently `int`). This entails some changes in `web2java.y`, to allow ranges for procedure and function parameters, as eg in

```
procedure p(x:0..255);
```

Presently all ranges are replaced by `int`, since Java is rather strict about type conversion, and requires casting where C does not.

5.3 Input/Output

On the whole, Java I/O is closer to Pascal syntax than is C. Thus

```
write_ln(term_out, 'value is ', v);
```

in Pascal becomes

```
System.out.println("value is " + v);
```

in Java.

However, we follow `web2c` in leaving this translation to the post-processor — in our case the script `javafix.perl`.

This Perl script looks for 'words' starting with `jT`, and deals only with these. Thus the translation above is implemented through the definitions

```
@d term_out == System.out
@d write_ln == jT_print_ln
```

The only unusual feature of Java I/O is that most I/O statements must be contained in a `try` statement, which must be followed by a `catch` statement to catch any I/O 'errors'. However, this is perfectly straightforward, as may be illustrated by an I/O function from `dvitype.ch`:

```
function signed_pair:integer; {returns the next two bytes, signed}
var a,@!b:eight_bits;
begin a:=0; b:=0;
try begin a:=dvi_file.readByte; b:=dvi_file.readUnsignedByte; end;
catch (ex: IOException) EOF_dvi_file:=true;
if EOF_dvi_file then signed_pair:=0
else begin cur_loc:=cur_loc+2; signed_pair:=a*256+b; end;
end;
```

For simplicity, `reset` and `rewrite` are defined in the 'TeX library class' `TeXlib.java`. Thus

```
reset(web_file);
```

is replaced in `tangle.ch` by

```
web_file:=TeXlib.reset(web_name);
```

6 Conclusions

Writing a Java change file is a straightforward exercise, following the lines suggested above.

Whether it is time well-spent is another matter! As we have said, the resulting programs are painfully slow; and the viability of the exercise must depend on the development of true Java compilers.