In G. Kadoda (Ed). Proc. PPIG 13 Pages 105-111

# The Science of Web-Programming

Roger Stone
*Department of Computer Science*
*Loughborough University*
*R.G.Stone@lboro.ac.uk*

Keywords: POP-I.C. *web* POP-II.B. *coding* POP-III.B. *scripting languages* POP-IV.A. *functional*

## Abstract

Although the advent of the World-Wide Web solved the problem of sharing static information between heterogeneous computers and networks, more recently the expectation is that web-sites will offer dynamic information. This increasing dynamic behaviour of web-sites relies on programming. As this is the newest application area for programming it might have been hoped that it would demonstrate the state of the art in programming languages and program design. The truth is that the web has been built on a lowest common denominator approach. This approach together with the reliance on FAQs and web-page tutorials has pervaded programming for the web. However a closer inspection of the process of building a typical dynamic web page reveals an opportunity to exploit some higher level ideas which offer the promise of simpler initial construction and easier maintenance of web-programs.

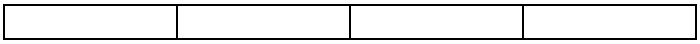
## Introduction

Has the accumulated wealth of Computer Science knowledge informed the design of web-based programming languages and program design? Is this newest application area a place where the latest techniques of program design are used to produce good programs written in clean, well designed programming languages? It seems not. In order to build a web-site a programmer will have access to a WYSIWYG HTML editor and the creation of static prototype pages for the site follows quickly and painlessly. The prototype pages are then converted to give access to dynamic database information and this is where problems arise. The pages no longer contain just HTML but they have scripting elements. In fact there may well be client-side scripting written in one language and server-side scripting in another. The final access to the database is written in a query language so that makes a total of four 'languages' intermingled in a single document and the WYSIWYG editor, if still being used, is merely being used as a text-editor. Furthermore the sequence of prototyping static pages in HTML and then converting them to dynamic pages by adding scripting seems to lead to a particularly low-level coding style. Some ideas are presented from the world of functional programming that could help put back the science into web-programming. By taking advantage of an under-used facility in a scripting language the ideas can be applied immediately without waiting for a revolution in web-programming languages. Finally some reflections are offered on the likely uptake of these ideas in a web-based community.

## Stages in the Construction of a Dynamic Web Page

For the purposes of discussion we will consider the production of a typical web page that contains some tabular data surrounded by some static information. The tabular data is eventually to be provided live from a database. However the ready availability of a WYSIWYG HTML editor causes the page to be prototyped with static place holders. So a page containing a tabular display is very soon constructed which may look something like this (using book data as an example):

| | | | |
|---|---|---|---|
| $ISBN_1$ | $Author_1$ | $Title_1$ | $Publisher_1$ |
| $ISBN_2$ | $Author_2$ | $Title_2$ | $Publisher_2$ |
| ... | ... | ... | ... |



Now comes the harder part of supplying the data ‘live’. The web-programmer investigates the HTML code written by the WYSIWYG editor and sees something like this:

```
<TABLE>
    <TR>
        <TD>ISBN1</TD>
        <TD>Author1</TD>
        <TD>Title1</TD>
        <TD>Publisher1</TD>
    </TR>
    <TR>
        …
    </TR>
        …
</TABLE>
```

The procedural programmer can easily see how to convert this from static to dynamic HTML by introducing scripting into the page and using nested iteration. The outer iteration can be responsible for the ‘row’ level components and the inner iteration can be responsible for the cells within the row (the table data). The programmer has already decided on a program of the form:

```
print “<TABLE>”
for (r to number_of_rows){
      print “<TR>”
      for (c to number_of_columns) {
            print “<TD>”  library[r][c] “</TD>”
      }
      print “</TR>”
}
print “</TABLE>”
```

The programmer can populate the array called library with sample data and test that this script works. The next stage is to provide real data from the database. By consulting FAQ or web page tutorials the programmer discovers that the server-side scripting language offers a query language interface to the database which allows the retrieval of the data required. Furthermore it supports access to successive rows of the retrieved data via a function call. So the program becomes

```
query = “… written in SQL …”
result = select_from_database_according_to( query )
print “<TABLE>”
for (r to number_of_rows){
      print “<TR>”
      this_row = get_next_row( result )
      for (c to number_of_columns) {
            print “<TD>”  this_row [c] “</TD>”
      }
      print “</TR>”
}
print “</TABLE>”
```

The sequence beginning with a static prototype and gradually adding scripting to produce the final dynamic web page has resulted in a very low-level program containing intermingled elements of 3 languages (HTML, server-side scripting and a database query language). There is nothing conceptually wrong with the original design sequence. It is good programming practice to concentrate on one area of the problem at a time. It is only the expression of the result in the muddle which constitutes the final program that is embarrassing. A critical appraisal suggests that there should be a higher-level view, one which does not require the intermingling of so many languages.

## A Functional Approach

The functional approach would identify that there is a string of textual data to be printed which comes from adding markup to some tabular data which in turn comes from a database, in other words we are looking for a functional program made up of the composition of three functions which could be written

```
print • markup_table  • select_from_database
```

using the symbol "•" for function composition or

```
print(markup_table(select_from_database(…)))
```

with the traditional f(p,q) notation. We now investigate how to achieve the markup of the table in a functional style.

The tabular data is thought of as being in a two dimensional array. Each element of the array needs to be marked up as a table cell with <TD>…</TD>. Each row needs to be marked up with <TR>…</TR> and the whole table needs to be marked up with <TABLE>…</TABLE>. Using "." to represent string concatenation we can create the following simple string functions:

```
tag_cell( c ) = "<TD>" . c . "</TD>"
tag_row( r ) = "<TR>" . r . "</TR>"
tag_table( t ) = "<TABLE>" . t . "</TABLE>"
```

The next step requires the use of a higher-order function which is capable of applying a function supplied to it as one parameter to each member of an aggregate supplied as another parameter. The function is usually called map or apply_to_all. If we have a row of cells in a vector row such that cell c is accessed by row[c] then we can markup the row by writing

```
markup_row( row ) = tag_row( map( tag_cell, row ))
```

We can extend this one stage further if we have a table t of cells such that row r is accessed by t[r] and cell c in row r by t[r][c]

```
markup_table( t) = tag_table( map( markup_row, t ))
```

This program looks most impressive (most compact) in the notation of the fp language introduced by Backus [Backus 1978] where the symbol "&" is used for map. [Note that in fp, map yields a list of values, but for our present purposes we imagine it to produce a string containing all the values concatenated together]

```
markup_table = tag_table • &( tag_row • & tag_cell)
```

In this notation the three 'tag' functions are written in this way

```
tag_cell = [%"<TD>", id, %"</TD>"]
```

[Note that in fp the notation %c represents a constant function and id the identity function. The display [f,g,h] produces a list resulting from applying the three functions f, g and h to the same data. Once again we imagine it as producing a string containing the same three result values concatenated together].

So we now have the desired functional program

```
print • markup_table • select_from_database
```

where

```
markup_table = tag_table • &( tag_row • & tag_cell)
```

and

```
tag_cell = [%“<TD>”, id, %“</TD>”]
tag_row = [%“<TR>”, id, %“</TR>”]
tag_table = [%“<TABLE>”, id, %“</TABLE>”]
```

This is what we wanted as it clearly separates the concepts and works directly with the aggregate concepts of table and row. The question remains as to whether it is possible to code in this style using current web-programming languages.

## Functional Programming in PHP

Up to now a generic scripting language has been used. There are several actual server-side scripting languages that can be used. Microsoft’s solution is to script in Visual Basic and use Active Server Pages (ASP) technology. Another more recent possibility is Java Server Pages (JSP). A relative newcomer that has been very successful recently especially among the Linux community is PHP [http://uk.php.net]. PHP originally stood for Personal Home Page but now PHP is regarded as a Hyper-Text Processor. It borrows shamelessly from C and Perl and contains functions which allow it to link to virtually any database as well as handling XML, PNG graphics, PDF, etc..

Investigation of the function library of PHP reveals the function array_walk. It is used in this way

```
array_walk( f, v)
```

meaning that the function f is applied to every value of the vector v. Unfortunately the function array_walk only returns an integer indicating success or failure. It does not return the results of applying the function to the data. However by exploiting the ability of PHP to define functions to have call-by-reference parameters it is possible to define a function map using the function array_walk which does return the desired value. This allows a program to be written in the functional style in PHP.

One of the known advantages of the fp style of functional programming is the relative ease by which programs expressed in that notation can be transformed. This is because the ‘parameters’ of the functions have been abstracted away leaving the program written entirely in terms of function symbols and function combinators. A program such as &f•&g can be transformed to &(f•g) from first principles and libraries of transformation rules built up. A similar transformation can be exploited in our web program context if required to remove the need to fully build the data selected from the database as a two-dimensional array before applying the markup.

## Work Remaining

Although this work looks very promising this demonstration only shows a bland markup into HTML. It would be more realistic to add to the table a row containing column headers perhaps underlined or marked up in bold. It may be required for reasons of readability to alternate the colours of the rows, to adjust the thickness of the table border, etc. Ideally the solution would be along the lines of

$$\texttt{print} \bullet \ldots \bullet \texttt{adjust}_2 \bullet \texttt{adjust}_1 \bullet \texttt{markup\_table} \bullet \texttt{select\_from\_database}$$

where a sequence of adjustments are made to the bland markup before printing. However because the value being passed between the functions in the composition is currently a single string, the adjust functions would have to parse the string in order to add their contribution to markup. It seems that a composite object is required which has separate data and markup fields. In fact the markup field should be an object which has tag and attribute fields and the attribute field should be an associative array. The composite object should be preserved until the final print step where a string is required.

## Concluding Remarks

Taking the optimistic view that this approach can be demonstrated to work well in practise, there remains the issue of how (and indeed whether) it could be effectively communicated to the web-programming community. The community expects technologies to be free (e.g. HTML, XML, JavaScript, PHP) only paying for software to improve productivity (e.g. DreamWeaver). The community relies heavily on web-based documentation, FAQs, news groups and electronic tutorials. The impression is that first news of how to do something in a new technology seems to travel well. A follow up suggesting that it could be done better another way is less exciting and may not be so eagerly spread. In other words, “It may already be too late!”.

## References

PHP - http://uk.php.net

Backus, J. (1978) Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. *Communications of the Association for Computing Machinery*, 21,613-641.