Red and Green Makes Gray?
The Yale computer scientist David Gelernter, in his book *Machine Beauty: Elegance and the Heart of Technology*, persuasively argues that much of the development of superior technology relies not so much on what the technology does, but how it does what it does. Specifically, Gelernter asks, does the technology work in an elegant manner and is it simple, yet powerful?

Consider the selection of server-side scripting languages: The big contenders in this arena are Active Server Pages (ASP) and JavaServer Pages (JSP). Hypertext Preprocessor (PHP) is also starting to edge in on the market. While each is powerful in its own way, neither of the environments is particularly aesthetic by Gelernter’s definition. Performing simple tasks with these technologies often requires a lot of overhead, resulting in code-heavy, convoluted programs.

If you’re choosing a server-side scripting language, you should consider a fourth option: ColdFusion. Because it acts more like a markup language than a programming language, ColdFusion is an elegant solution for embedding code in HTML files. Allaire, the company that also makes the popular HomeSite HTML editor, has a whole suite of ColdFusion technologies consisting of an application server, an integrated development environment, and of course, the markup language. It is this ColdFusion Markup Language (CFML) that competes directly with ASP and JSP.

**Comparative Languages**

For a better example of CFML’s simplicity, power, and elegance, consider querying a database. Listing 1 shows code that runs a SQL statement against a local data source, retrieves a recordset, and displays the resulting data. Literally, that’s all you have to do—connect to the source, send the SQL command, and display the results. Not bad for nine lines of script!

Now take a look at what you would have to do with ASP and JSP. Listing 2 shows an ASP script that does the same task, and Listing 3 shows the equivalent JSP script. In both cases, notice the wordy and convoluted syntax and compare it to the CFML script in Listing 1.

**A Web Agent**

Querying a data source is one limited example. CFML also works well for a number of other tasks. Listing 4 represents a small ColdFusion application that serves as a Web agent. It takes a Web page that you specify and sends it through the AltaVista Babelfish language translation service. The result is the same Web page, translated into the language of your choice. Furthermore, the script alters each hyperlink in the document so that selecting any one also returns a translated page. In this way, once you’ve translated a page into Portuguese, all linked HTML documents from that point are automatically translated into Portuguese.

To run this code you’ll need to have the ColdFusion server software installed and connected to your Web server. The ColdFusion server acts as the interpreter for all your ColdFusion markup. This process is similar to enabling your server to handle either ASP or JSP.

The key thing to note about this code is its simplicity. Line 1 of the code in Listing 4 simply sets a variable using the `<cfset>` tag. (All tags in ColdFusion begin with `cf`.) This variable holds the URL for the page that the script should initially retrieve; in this case it’s a page on the Johns Hopkins University Web site.

Line 3 dynamically creates a base URL from the hostname of the current server. You could just hard-code the domain name into the script, but generating it dynamically saves you time if you later move the application to another server.

Line 7 is the key to the application. Here the `<cfhttp>` tag requests retrieval of a particular document from the Web. The ColdFusion server receives the document. It’s crucial to realize that the page isn’t automatically returned to the browser. Instead, the ColdFusion server has the opportunity to filter (or otherwise manipulate) the page’s contents. This becomes important later in the
application. Notice also the clean syntax of the \(<\text{cfhttp}>\) tag. Just pass it a URL, tell it to either GET a document or POST to a document, and decide whether to fully resolve any embedded URLs in the document. Like many tags in CFML, this tag lets you do extremely powerful things, yet it’s extraordinarily simple to use.

Lines 9 through 30 place an HTML form at the top of whatever page resides at the startingURL variable from Line 1. You should set startingURL as the first page you want translated. The form lets users select source and target languages, like “English to German,” for example. When the user clicks on the “Translate” button a JavaScript function transfers the client to the Babelfish service, with a request to translate the document and send it to the ColdFusion server, perhaps for further processing.

**JavaScript Output**

Notice also that Lines 12 through 14 use ColdFusion to output JavaScript. That is, the script sets the value of a JavaScript variable with the value of a ColdFusion variable. In this case, the JavaScript urlL variable, which will be the location to which the client navigates, contains the fully qualified URL of the current document. The script also appends two other URL variables to the end of this string: thisURL, which contains the value of the startingURL; and lp, which is a variable Babelfish requires to identify the translation language.

After the script outputs this HTML form to the user’s browser, it then outputs the contents of the \(#\text{cfhttp:FILECONTENT}\) variable. (All variables in ColdFusion are surrounded by pound signs.) This variable contains all the HTML code for the Web page previously retrieved by the \(<\text{cfhttp}>\) tag. This occurs in lines 31 through 33 of the script. In line 39, you’ll find another \(<\text{cfhttp}>\) call that essentially passes the starting page URL to the Babelfish service.

To recap: At this point the user should see a Web page with a language selector at the top in his or her browser. Once the user chooses a language and clicks on the Translate button, the JavaScript function sends the appropriate commands to the Babelfish service for processing. The Babelfish engine processes the page, translates it into the selected language, and then returns it to the ColdFusion server where further processing can occur.

Once the Babelfish service returns the translated page, lines 41 and 42 prepare
the page to be sent to the user. Replacement functions modify each hyperlink on the page so that instead of pointing to untranslated pages, they refer to the Babelfish service.

Note, however, that in addition to these simple replace functions, you can do much more processing on the results of a <cfhttp> call. ColdFusion supports a wide range of regular expressions that could radically reformat the content of this page before returning it to the user's browser. Such regular expressions could, for example, extract specific data from a site, like stock and weather information, and return it on a newly formatted page.

Notice the power and functionality built into a mere 47 lines of script. Think of other programming languages that you know—could you perform the same task with such economy of expression?

Beauty and Brawn
It's true that other Web application languages are far richer than CFML. There are many programming tasks that can be accomplished, for example, in Java with its Java Foundation Classes (the Swing library) that simply can't be accomplished with CFML. The Swing library provides much more UI functionality than the HTML, CSS, and DHTML that ColdFusion relies upon for its client-side user interface. But many applications don't require the sophistication these richer functions provide.

In short, you should choose your language based on the task at hand. In many cases, as in those requiring standard network protocols and services and back-end database interactions, the extent of your programming activities can be best handled with ColdFusion technologies. CFML helps you achieve your programming goals efficiently and rapidly, and according to David Gelernter, that places the language in the category of superior technologies.>

Mark is a Web developer at Johns Hopkins University and Nine Web, LLC. You can contact him at mcyzyk@jhu.edu.