Main topics of the week:
- Domain Specific Languages

Case Study: Mawl – A Web Authoring Language

So far in this course we have seen several different programming language paradigms, e.g., imperative, object-oriented, and functional languages. Another aspect of programming languages is the design of small languages to solve particular types of problems. Such languages are called application oriented languages or domain specific languages. Rather than being general purpose programming languages, the domain specific language is intended to make easier the expression of solutions to problems in a specific area. Examples of such problem domains are string/pattern manipulation (awk is a domain specific language for this domain) and database queries (SQL). Domain specific languages may be small and special purpose, but they use the same concepts we have seen for general purpose programming languages.

Mawl is an example of a domain specific language for which I wrote a compiler when I worked for Bell Labs. It is an interesting example of a special purpose language that forces one to think about many of the concepts we have been discussing. Mawl stands for the “Mother of All Web Languages”, and was the result of research into programming languages. The intent of Mawl was to address the difficulty of programming services for the web.

One can view the web as just a way of accessing static pages, but the introduction of forms and CGI scripts to web servers and browsers opened up the possibility of implementing interactive services over the web. That is, rather than just fetching static pages, the browser can send information to the web server which in turn can determine pages to send to the browser with possibly requests for more information. A standard example is the shopping cart used for making purchases over the web. The shopper views their interaction as a sequence of actions and reactions, accumulating items in a shopping cart. However, the web is basically a stateless entity, which means that each fetch of a page is an independent event. Forms allow information to be submitted to the server, but there is really no inherent relation between the display of one page and the display of the next.

The statelessness poses a difficulty for the implementer of a web service. Some convention must adopted to logically connect the sequence of pages. For example, clicking on a button to add an item to the shopping basket must somehow communicate to the server which item and which basket. So the server must maintain some state information between the actions of the user. Obviously, all of this can be done, and one way is to use CGI scripts or something like them on the server side which are essentially programs to correspond to each action the user might take. These separate programs must access some shared data.

When one really looks at this model, the problem is that the environment for programming a service requires a bunch of separate programs for each step in some service logic. From the service author’s point of view, it would be nicer to view the service as a single program with coherent logic. This is the programming problem Mawl attempts to solve.
The goal of Mawl is to allow what appears to be a single logical program to describe a web service. The interaction with the user should appear as just that – input from and output to a user – rather than independent fetches of pages or scripts that somehow must know they are connected. The Mawl program should also have the ability to maintain variables, which represent the state information between page fetches. So the central feature of the Mawl language is the specification of interaction with the user. This means output to the user, i.e., delivering a web page to the user. And input from the user, i.e., retrieving the values submitted on a page. Taking its cue from the HTML form construct, Mawl allows objects to be declared of type \texttt{form}. Form objects have a method \texttt{put}, which sends a page to the user. In the Mawl program, the return value of the put method call is the data submitted by the user. The job of Mawl is to hide all of the details that make this happen. Moreover, the Mawl program should be able to have normal variables declared, whose value is preserved over the form put calls. And in addition, it is reasonable to want variables whose values are preserved between instances of executing the service, i.e., between separate sessions of the service.

The following example shows a simple Mawl program that displays a page in which the user enters their name. The service responds with greeting the user by name and also indicating a count of how many times the service has been invoked. This service can be tried out at \url{http://www.cs.uoregon.edu/~datkins/mawl-bin/simple}. The source code for the service looks like:

```java
static int cnt_visitors = 0;
session Greet {
    auto form {} -> { id } getName;
    auto string id = getName.put({}).id;

    auto form { int count, id, int x, string y} -> {} Greet;
    Greet.put({++cnt_visitors, id, 0, ""});
}
```

Notice that \texttt{getName} is declared as a form object and that a form object is like a function specification in that it specifies data given to the form when it is sent to the user, and data retrieved from the form when the user submits it. As the name suggests, \texttt{getName} just prompts the user for their name, it has no input, but produces the output of a string named \texttt{id} in the form’s output data structure. Similarly, there is a second form to use the name in a greeting and also give the count of visitors. Some other dummy data (an integer \texttt{x} and a string \texttt{y}) is also given for test purposes.

A design intent of Mawl is to separate the design of presentation (i.e., the web page layout seen by the user) from the service programming logic. To this end, Mawl allows form objects in the service logic to be bound to a template HTML like file. This template is like HTML, but has a few extensions to permit the use of the data passed by the mawl service, as well as a few other extensions. To allow service code to be tested separately from the presentation, Mawl does not require the template files, but will just produce a simple page with the form data and input fields for the submitted data. This allows testing of the service logic without regard to presentation design layout. For this simple service, you can see what it looks like without the templates at \url{http://www.cs.uoregon.edu/~datkins/mawl-bin/simple-nohtml}. You can also look at the templates and Mawl source code at \url{http://www.cs.uoregon.edu/~datkins/mawlsrc/simple/}

You can probably see from this example that Mawl is a C/C++/Java like imperative language. It introduces a form type and a record type (using braces) like C structures. Also notice the keywords \texttt{auto} and \texttt{static} on variables. For Mawl, \texttt{auto} means that the
value persists just for the session (the sequence implemented by the service logic), but
not between separate sessions. Static means persistent over separate sessions. And
implicit in all of this is that Mawl arranges for the state of the session to be preserved
between the delivery of web pages and the submission of form data from those pages.

The implementation of Mawl services is as a single CGI program. This means that the
Mawl compiler must arrange for a way of saving the state of the session between
invocations of form put’s. This state must contain the values of automatic variables as
well as some type of program counter and saving of the stack. The static variables must
also be dealt with, and since they are persistent across sessions, their value must be able
to be seen at any time. Thus static variables are stored immediately upon assignment, and
are fetched from this permanent storage whenever they are needed in an evaluation.

Some other language issues that arose in the Mawl prototyping were resource
contention (since static variables are effectively shared), synchronization (to allow
programmer management of resource contention), and how much to include in the Mawl
language itself. To address this last issue, Mawl permits the inclusion of C/C++ source
code in its compilation, and has a language interface for calling these “external” C/C++
functions. For efficiency, Mawl itself implements the basic C expression syntax. As well
as producing more efficient programs, this also gives a familiar environment for C/C++
programmers.

A less trivial example of a Mawl service is an implementation of a simple number
guessing game which can be found at
http://www.cs.uoregon.edu/~datkins/mawlsrc/guess/

A more complex example of a service that implements a version of the Mastermind
sequence guessing game can be found at
http://www.cs.uoregon.edu/~datkins/mawlsrc/mastermind/