

Client-Side Profile Language and Interpreter

Ying Xing, Nesime Tatbul, Olga Karpenko, and Jonathan Cohen

May 12, 2000

1 Introduction

Designing a profile language which can be used to continuously query a web site's data and the changes in that data is one of the main objectives of this project. This language should enable better query capability than server-side profiles where users fill in forms to specify the topics in which they are interested in and then a server-side program returns the related pages to the user. However, since our language will be used by a client-side query system, there is the disadvantage that the web site to be queried is autonomous and client-side program can not ask web server to do anything further than downloading the web pages by HTTP requests. Therefore, there is a need to somehow model the web site's data at the client-side and then send the HTTP requests accordingly or to target at XML data rather than HTML data so that the data provides its own description.

2 Data Model

Our initial plan was to work on a web site which presents its data in XML and use an XML-QL-like query language for our profile language. We searched through the web and could not come up with a web site whose data is written in XML and whose content is subject to frequent changes. We first attempted to take a web site and convert its content to XML either automatically or semi-automatically. However, this idea did not come out to be very practical. Finally, we decided to find another representation for the web site's data and came up with an object-oriented-like data model. We chose `cnn.com` as the web site to apply our system on.

We model a web site as a tree, and a collection of web sites as a forest. Each node in the tree corresponds to a page in the web site. To specify a particular page, the user is required to specify a path from the root node to the desired node. The user may also specify an entire subtree by specifying the path to the root of the subtree followed by '*'. The structure of the tree is specified in a site map file, which must be generated by hand for a given web site. The examples of site maps we used for weather and sports sections of CNN are given in Figures 1 and 2.

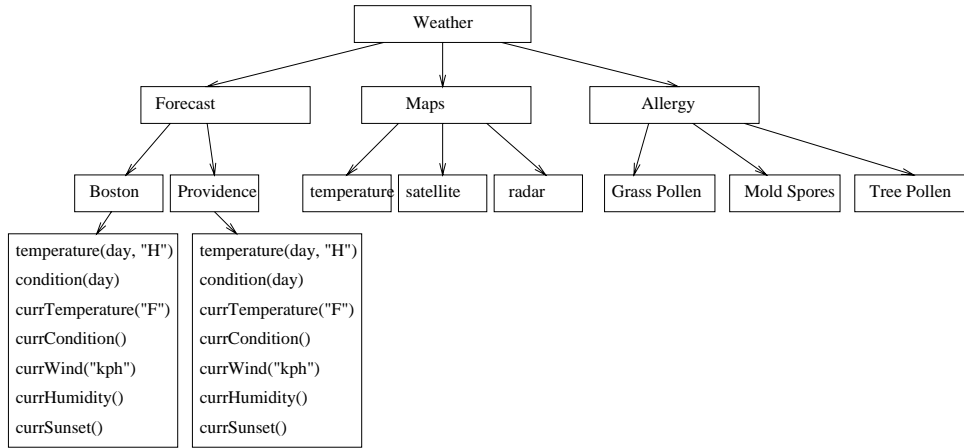


Figure 1: Example model of the weather section of cnn.com

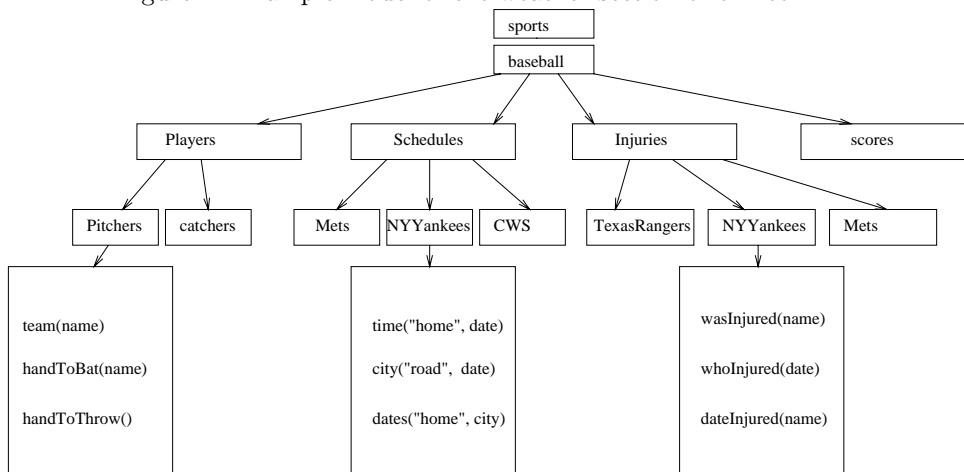


Figure 2: Example model of the sports section of cnn.com

Certain “functions” may be applied to specific html pages obtained from the web site. For example, the function “contains(HTML, text)” returns whether the HTML page contains the given text. Other functions may be specific to a particular web page, such as “currTemperature” which takes an HTML weather page from cnn.com and returns the integer value of the current temperature. Associated with each node in our web site model is a list of functions that may be applied to that node. While this method is somewhat ad hoc (since for each web page that we want to obtains values from, we must write a new function), it appears to be the only alternative in lieu of an XML data source.

3 Profile Language

There were two alternatives for the profile language: We would either design a site-specific language which could be only used on our application web site (cnn.com) or we would design a general purpose profile language which could be used on any web site. We decided to be general and took the second approach. Our profile language can be used to query any web site provided that the web site’s data is modeled according to the web site data model format we developed.

We designed an SQL-like profile language. It has the following grammar:

```
profile          : query_list

query_list      : query
                | query query_list

query           : SELECT variables
                FROM sources
                where_clause
                time_start
                time_end
                frequency

variables       : variable var_list

var_list        :
                | COMMA variables

sources         : source
                | source COMMA sources

source          : PATH variable
                | URL variable

where_clause    : WHERE constraints
```

```

|
time_start      : START time_const
|
time_end        : END time_const
|
frequency       : FREQUENCY INT time_unit
|
time_const      : NOW
| TODAY time_option
| YESTERDAY time_option
| TOMORROW time_option
| MONDAY time_option
| TUESDAY time_option
| WEDNESDAY time_option
| THURSDAY time_option
| FRIDAY time_option
| SATURDAY time_option
| SUNDAY time_option
| DATE time_option
| TIME
time_option     : COMMA TIME
|
time_unit       : SECOND
| MINUTE
| HOUR
| DAY
| WEEK
| MONTH
| YEAR
constraints     : constraint_term
| NOT LEFTPAR constraints RIGHTPAR
| constraints OR constraint_term
constraint_term : constraint_fact
| constraint_term AND constraint_fact
constraint_fact : constraint
| LEFTPAR constraints RIGHTPAR

```

```

constraint      : variable DOT function_call cons_rest

cons_rest :
| operator righthand_side

function_call  : variable LEFTPAR parameters RIGHTPAR
                | variable

parameters    : parameter_list
                |

parameter_list : parameter
                | parameter_list COMMA parameter

parameter     : INTCONST
                | STRCONST

righthand_side : atomic_value
                | complex_value

atomic_value  : INTCONST
                | STRCONST

complex_value : variable DOT function_call

operator      : EQ
                | LESS
                | LESS_EQ
                | GREAT
                | GREAT_EQ
                | NOT_EQ

```

A profile is a collection of SELECT-FROM-WHERE queries. In SELECT clause, user can specify which web pages he wants to be returned to him after the query has been executed. In FROM clause, he can either specify the URL's of the web pages to be queried (if he knows) or the path where the data most likely to be found at. He can not use regular path expressions but can put a "*" at the end of the path to mean that every page reachable from that path. We treat paths like paths in a directory structure of a file system. The answer to the question of how will the user know the correct paths is that we provide him a representation of the data model of the web site and he can write his queries accordingly. We were planning to develop a GUI for interactive query construction but we did not have enough time for that. Currently, the user is expected to look at a text file where he can see the model of the web site to write his queries.

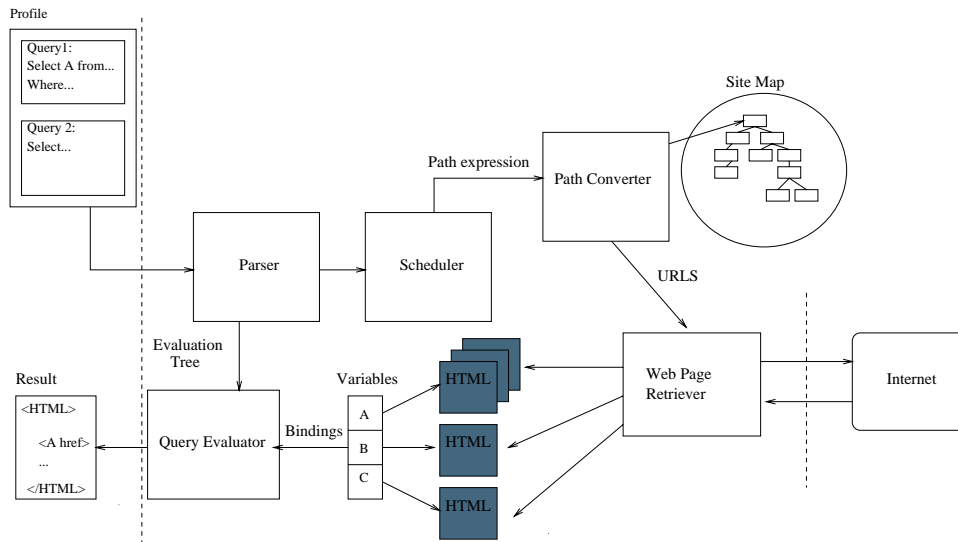


Figure 3: Overall design of our system

The queries may also have optional clauses related to time. The user can specify when to start executing the query, until what time and with what frequency. For example he can say "Execute this query starting now till 5.30 tomorrow every 30 minutes.". The language also allows "execute on update" option but we could not realize this in practice since CNN's web pages are automatically generated by a server side program on every HTTP request and that's why a page always appears to have a new time stamp although its content has not changed.

We also allow keyword-based queries. The user can query web pages according to the words they contain or do not contain by using a special function called "contains(text)."

4 System design and operation

To evaluate a profile, we need to schedule it, parse the query into an evaluation tree, compute the bindings for the variables in the query expression, and then evaluate the expression tree with those variable bindings. Figure 3 shows the main components of our system and connections between them.

First we parse the profile and break it up into different queries, each with different scheduling requirements. We schedule each of these queries separately. When a query is ready to be evaluated, we parse the query and store the WHERE constraints into an evaluation tree structure. Then we identify the path/url's to be bound to variables together with the functions to be executed on those variables. Then we look at the site map representation of the web site

to be queried and convert these path expressions into actual url addresses.

Once a path expression is evaluated, we have a list of urls for web pages that need to be bound to a variable's value. To process the contents of these web pages, we need to download the HTML pages from the specific web site. The task of downloading web pages is realized by using the programs provided by Xcite libwww which is published by W3C. Libwww is a client side Web API written in C. It provides a lot of general-purposed functions that can be used directly, such as download web pages or their headers. In this project, we download every page each time we evaluate a query.

It seems that the downloading process is the bottleneck of query evaluation process. It is also the bottleneck of the whole system. We have been thinking of maintaining a local cache for pages we have downloaded. That is we want to store the pages we have downloaded on the local disk, and when we need to download new pages, we check first whether it's in the local database. If so, we will download the header of that page and see whether it has been updated or not. If it has been updated, we will download it again from the web site, otherwise, we can use the file in the local disk directly. This idea is not suitable for our web site of www.cnn.com however. It seemed that they are generating web pages dynamically because the last modified times of their pages are always the same as the downloading time. We also looked at a lot of other web sites, and find that for a lot of commercial web sites, they either generate pages dynamically, or change their pages very frequently. For a lot of web sites, the server doesn't provide the last modified times of their web pages in the header. All this situation makes a simple generic caching technique difficult to be realized. We think if we want to step further in this direction, we can look at the caching technique of those web browsers, such as netscape or IE. In our project for cnn, we don't use any caching technique.

After a page is downloaded, the functions are executed and the WHERE constraint tree is evaluated. The web page bindings that satisfy the constraints are returned to the user as the result.

5 Examples of queries

Below we give some examples of the queries users can ask for weather, entertainment and sports (baseball) sections of cnn-site with our system. As a user can "extract" structured information from html-pages which are the leaves of the site map tree using the pre-defined site-specific functions, write queries which need joins and do keyword search in addition, our system is much more powerful than existing server-side query tools (like myCNN.com).

- `SELECT A`
`FROM /cnn/entertainment/movies A`
`WHERE A.contains("Flintstones") OR A.contains("Love")`
- `SELECT A`
`FROM /cnn/sports/baseball/schedules/* A`

```
/weather/forecast/Boston B  
WHERE A.city("home", "May 11") = "Boston" AND  
B.temperature("day2", "high", "F") > "50"
```

- SELECT A
FROM /cnn/sports/baseball/schedules/TexasRangers A
WHERE A.dates("road", "Detroit") = "May 30" START NOW END
SATURDAY, 12:45 am

6 Conclusion

For this project we designed a generic profile language which can be used to query well-structured web sites and implemented its interpreter. Although the language itself is generic, we need to manually build site trees for the web sites we want to apply it to, and to specify some functions on the leaves of these trees to allow finer-grained information retrieval. Such "half-automatic" extraction of structure and information from HTML-pages is not very elegant of course, but it seems that the only way to allow to do it automatically is to store data on the web in XML.

Our system should be easily extendible to query and integrate data from multiple sites. This is the potential advantage of this client side system versus the sever side service.

Also, our system is a time based continuous querying system. The user can specify the beginning, end and frequency of the querying process. Whenever the user logs to our system, he/she can see the most recently updated result immediately without inputing and executing his/her query each time.

We also found a problem in this simple system that the bottleneck of the downloading process makes the naive approach very inefficient if we want to be continuously querying a large amount of profiles. The dynamic or frequently changing nature of many web sites as well as the diversity of web severs make a general caching technique difficult to be realized to relieve the bottleneck. We think that one way to reduce the number of pages needed to be downloaded, is to extract the common part from multiple profiles. This could be future work for this project.