By: Nada Attar

1. Technical Implementation

1.1. Problems and Alternative Solutions:

Although we initially planned to develop our software using database and tables to store them, we eventually had to switch to the idea of parsing the schedule from the website for every bus line that the user requests. We found that it is important to parse the schedule any way even if we have database tables in case the information of the bus schedule changed in the website, then the system always has the latest update as the website. The solution was to keep the design and the idea of the Bus Schedule software, and make small changes to it in order to fit the new technical alternative solution.

1.2. Implementation Script

Language:

1. Ruby:
   Libraries:
   - 'cgi': to write cgi files
   - 'rubygems': is a package manager for the Ruby
   - 'builder': translates codes into valid XML.
   - 'hpricot': is a fast and delightful HTML parser
   - 'open-uri': t is possible for this library to open http/https/ftp URL as usual a file.

Files:

1. .cgi files that have all the vxml tags
2. .rb file that has the functions for parsing the information and data for the website

Programs:

1. EditPlus: to write the cgi files and rb file.
2. Firebug: detects the HTML specific tags' names.

Web/Networking:

1. Be vocal café: We test the system using Be vocal website http://cafe.bevocal.com/
2. MBTA: to parse the schedules form the website in the form of HTML. http://www.mbta.com
3. SSH: to log in to our Linux accounts and compile ruby files.
4. Our Tufts accounts: to generate ruby files to vxml.
5. Mozilla/Internet Explorer: to navigate the website and to look for the information.

More of Technical Issues:

- The number of buses would be in a static grammar.
- Parsing buses schedule(times AM/PM, inbound, outbound, weekdays, Saturday and Sunday)
- Put the stops in a grammar and the times in array and return them from ruby file to the cgi files. .
- We might use variables and arrays.

1.3. Technical Description:

Ruby File:

The system uses one ruby file that has three functions to parse the grammar and the three next stops from the MBTA website.
All the three functions need to open the specific website using 'hpricot' and 'open-uri'.
The idea is challenging because there are five parts need to be sit-up to make the parsing works correctly. Figure 1

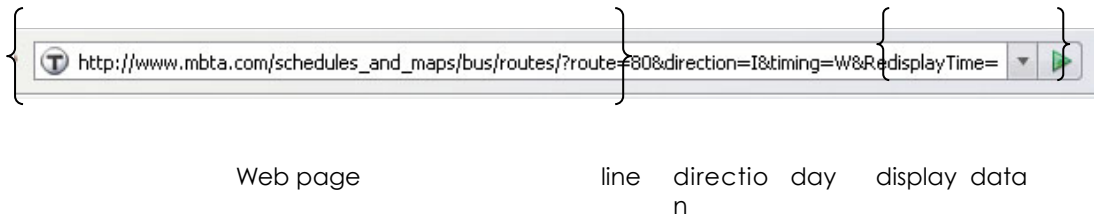
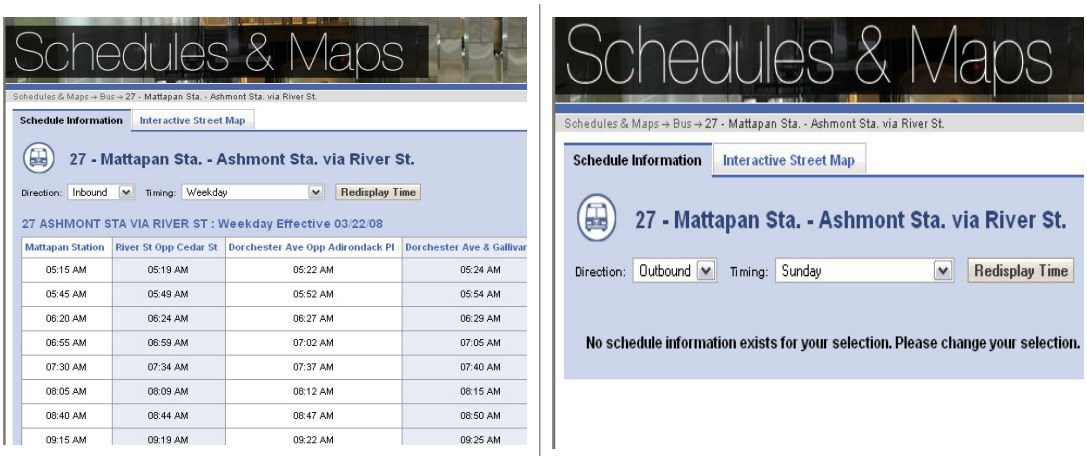


Figure 1: how the three functions divide the link of the web into several variables to parse the webpage

a. **check(line, dir, dayOfWeek):** this function returns two values, "true" if there if a bus schedule for the user's input 'Day' and "false" if there is no schedule for the user requests. This function is useful because some days (ex: Saturday or Sunday) do not have bus line in those days. See figure 2



Check function will return "true"

Check function will return "false"

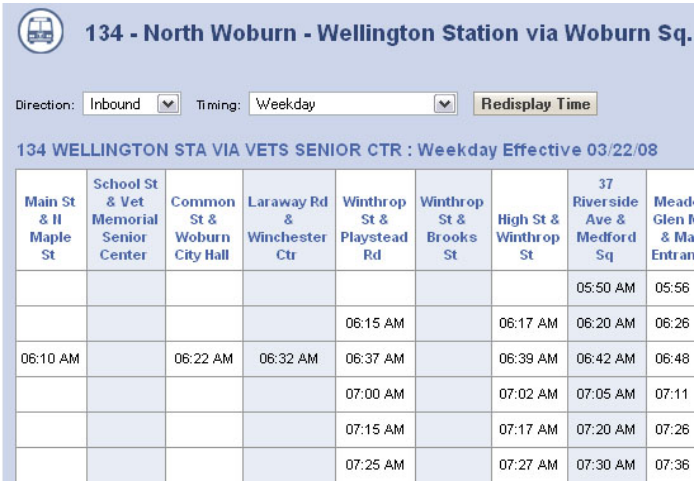
Figure 2: the function check recognize the empty schedule and return the appropriate value

b. **busStops(line, dir):** it returns the grammar for specific bus line. The grammar is the first line in the schedule, which contains all the stop names. In addition, this function converts the unknown or unrecognizable words before it sends them to the cgi file. For example, it convert '&' to 'and' and 'St' to 'Street' so, this help the user to speak naturally without any concern of the limitation in coding part.

c. **return_three_stops(line, direction, dayOfWeek, whichStop, whatTime, am_pm):** this function has the most logical parts. It uses the five variables to get the correct webpage. First, the function searches for the name of the stop. When it finds it, then it will take the entire column under this stop. After that, it puts the times in this column in array and converts the times to type of 24 hours instead of 12/12 hours. When the function puts all the time in array, it ignores all the nil values (see Figure 3) so the user will never get a nil time. In addition, this function handles the times in a very clever way. It can check if the time that the user wants falls between 'am' and 'pm' time (point of changing). Finally, after it reaches to the next three stops that the user wants, the three stop times are stored in a temporary array to return it to the cgi file.

Moreover, before returning the temporary array, it checks if the array is nil or empty (this can happen when the time the user requests is near the end of the array). If there are no times it be returned, there will be a sentence in this array to be played as prompt "*Oh, I am sorry, there is no stop time for*

the time you've requested" or any thing that the cgi file can recognize it to knows that there is no times for this request.



134 - North Woburn - Wellington Station via Woburn Sq.

Direction: Inbound Timing: Weekday Redisplay Time

134 WELLINGTON STA VIA VETS SENIOR CTR : Weekday Effective 03/22/08

Main St & N Maple St	School St & Vet Memorial Senior Center	Common St & Woburn City Hall	Laraway Rd & Winchester Ctr	Winthrop St & Playstead Rd	Winthrop St & Brooks St	High St & Winthrop St	37 Riverside Ave & Medford Sq	Mead Glen & Ma Entran
							05:50 AM	05:56
				06:15 AM		06:17 AM	06:20 AM	06:26
06:10 AM		06:22 AM	06:32 AM	06:37 AM		06:39 AM	06:42 AM	06:48
				07:00 AM		07:02 AM	07:05 AM	07:11
				07:15 AM		07:17 AM	07:20 AM	07:26
				07:25 AM		07:27 AM	07:30 AM	07:36

Figure 3: Example of nil values that the function successfully handles them for each specific column

CGI Files:

1.4. Pseudo Code:

101

"say bus line or bus stop"

Grammar
(all bus lines and all stops)
/Grammar

If say bus line
Assign variable line to what they said
Goto 201

If say bus stop
Assign variable stop to what they said
Goto 401

201

"say the name of stop or list them"

Grammar
Populate with stops from line
Select stops
From line

Add 'list' to grammar
/grammar

If said stop name
Assign variable stop to what they said
Goto 501

If said 'list'
Goto 301

301
"the stops are…"
Select stops
From line
"
Grammar
Select stops
From line
/grammar

Assign variable stop to what they said
Goto 501

401

***Not sure what to do here yet**

501
"inbound or outbound"

Grammar
(inbound, outbound)
/grammar

If inbound
Set variable In=1, Out=0
Else
Set variable In=0, Out=1

Goto 601

601

If In=1

Select regular inbound
From stop
Where value in reg inbound>= current time

Return first 3 in prompt

If out=1

Select regular outbound
From stop
Where value in reg outbound>= current time

Return first 3 in prompt

"if it doesn't work say schedule"

Grammar
(schedule, yes/done)
/grammar

If schedule
Goto 701

Else
Goto 1201

701

"do you want todays schedule?"

Grammar
(yes, no)
/grammar

If yes
Set day to current day
Goto 901

If no
Goto 801

801

"what day do you want?"
Grammar
(days of week)

/grammar

If weekday
Set variable day to weekday

If Saturday
Set variable day to Saturday

If Sunday
Set variable day to Sunday

Goto 901

901
"what time?"

Set variable time to time they say

If say am/pm
Goto1101

Else
Goto 1001

1001
"am or pm"

Append value to time
Goto 1101

1101

If day=weekday and in=1
    Return
        Select regular inbound
        From stop
        Where value of RI >=time

If day=weekday and out=1
    Return
        Select regular outbound
        From stop
        Where value of RO >=time

If day=sat and in=1

Return
Select sat inbound
From stop
Where value of SatI >=time

If day=sat and out=1
Return
Select sat outbound
From stop
Where value of SatO >=time

If day=Sun and in=1
Return
Select Sun inbound
From stop
Where value of SunI >=time

If day=Sun and out=1
Return
Select Sun outbound
From stop
Where value of SunO >=time

"another bus or goodbye"

If another
Goto 101

Else
Goto 1201

1.5. Observations:

The MBTA website has very clear schedules that make any user understands them. We try to observe this simplicity by making the system as clear as the website. MBTA Bus Schedule system has user-friendly interface. We took into considerations the design of other Voice applications and we implemented the functionally part according to the simple and clear design. For example: the website has two simple dropdown lists, one for the direction (Inbound/ Outbound) and the other one to choose the date (Weekday/ Saturday/Sunday). We implemented the functions to deal with those

few variables without any needing to ask the user any other information that is not in the website (we use also the bus line, time, bus stop that are existed also in the website).

1.6. Obstacles and Limitation:

Because of time constraints and switching form database to parsing data from the MBTA website, we could not expand the system to handle more cases for buses lines or give more futures such as fares and direction to each bus stop to pleased our target users. Although the system works correctly and handles the most tricky cases for many bus lines, sometimes it gives un accurate times for few specific bus lines. We have worked hardly to fix this bug but then we think that the problem might happen because of the HTML of these specific schedules.

1.7. Future Implementations:

The System could be improved to handle any request from the user whiter the user says the bus line first or jump to the name of bus stop. Now the system only handles the case when the user starts by the name of bus line.