96-27

WWW-AQ:
WORLD WIDE WEB INTERFACE FOR THE AQ LEARNING SYSTEM

User's and Programmer's Guide

Seok Won Lee
swlee@aic.gmu.edu
Machine Learning and Inference Laboratory
George Mason University
Fairfax, VA 22030-4444

MLI 96-11
P96-27

December 1996

WWW-AQ: WORLD WIDE WEB INTERFACE FOR THE AQ LEARNING SYSTEM

User's and Programmer's Guide

ABSTRACT

Current research on graphical user interfaces is highly focused on developing WWW application-based Web interfaces using JAVA/CGI programming. In this report, we present the WWW-AQ, which is an World Wide Web Interface for the AQ Learning System by integrating a various kinds of intelligent and useful program modules. WWW-AQ provides not only a simple user-oriented interface, but also an *intelligent mail agent* by which the internet users can experiment with the AQ Learning System through World Wide Web. WWW-AQ also provides the user the most efficient ways of preparing a learning data set through an *auto-formatting agent* and three kinds of robust rule testing methods through an *EDC (Experimental Design Component) agent*. By having such integrated agents on the Web, this study has demonstrated its powerful portability and accessibility which are not limited to specific platforms, and shows the potential for research in areas such as building learning agents in distributed environment.

KEY WORDS: WWW User Interface, CGI Programming, Intelligent Agent, Learning System Evaluation Methods

ACKNOWLEDGEMENTS

The author thanks Dr. Ryszard S. Michalski for useful comments and criticism. The author greatly appreciate the support of a Doctoral Fellowship from the School of Information Technology and Engineering at George Mason University. This research was conducted in the Machine Learning and Inference Laboratory at George Mason University. The Laboratory's research activities are supported in part by the National Science Foundation under grants IRI-9020266 and DMI-9496192, in part by the Defense Advanced Research Projects Agency under grant F49620-95-1-0462 administered by the Air Force Office of Scientific Research, and in part by the Office of Naval Research under grant N00014-91-J-1351.

TABLE OF CONTENTS

1 INTRODUCTION

Current research on graphical user interfaces is highly focused on developing a WWW application-based Web interface using JAVA/CGI programming. WWW-AQ is develped to provide the AQ learning system to Internet users with various kinds of integrated intelligent agents. Dramatic growth in the number of World Wide Web (WWW) users indicate the future importance of user-oriented graphical interfaces. As long as users can access the Internet, they can use their data to run the AQ learning system and obtain results immediately. It is very convenient for users to set parameters by just clicking the menu buttons and having the *integrated agents* finish the tedious work (e.g. data set preparation, testing data set etc.) as compared with the classical, time consuming, hand work under the interactive UNIX programming environment. (This typically requires the manual entry of several tables for making up input files.)

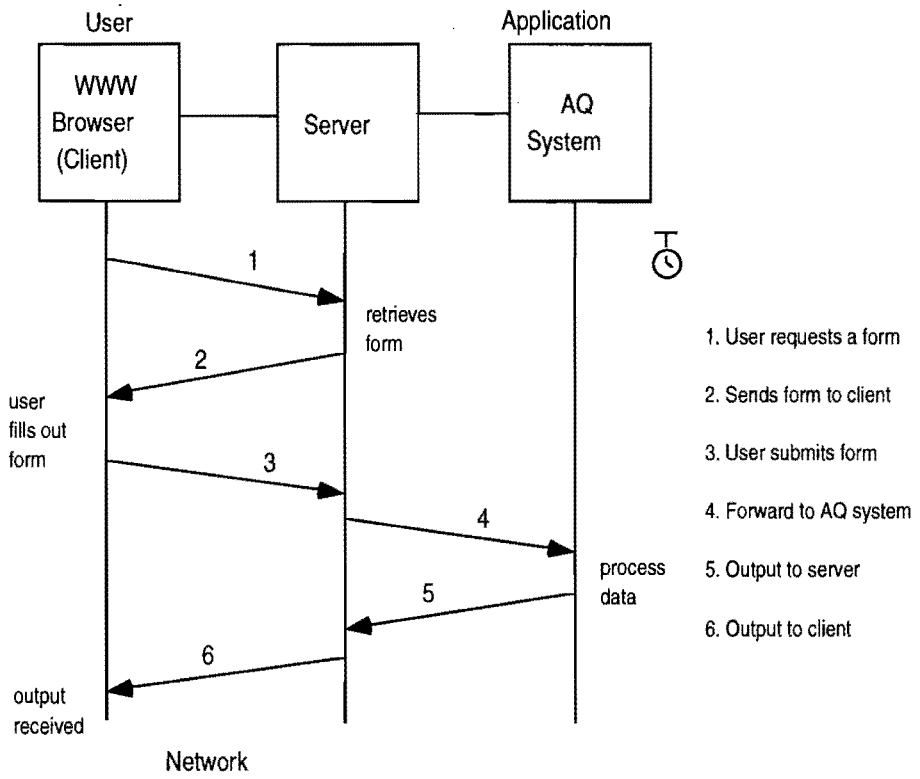

Figure 1. General structure of WWW-AQ

Figure 1 shows the general structure of WWW-AQ. Users can access the WWW-AQ home page through http://www.site.gmu.edu/~swlee/aq-index.html. The home page is temporarily

connected through the developer's home page. It will be remaintained under the Machine Learning and Inference Laboratory's home page (*http://www.mli.gmu.edu*) after careful

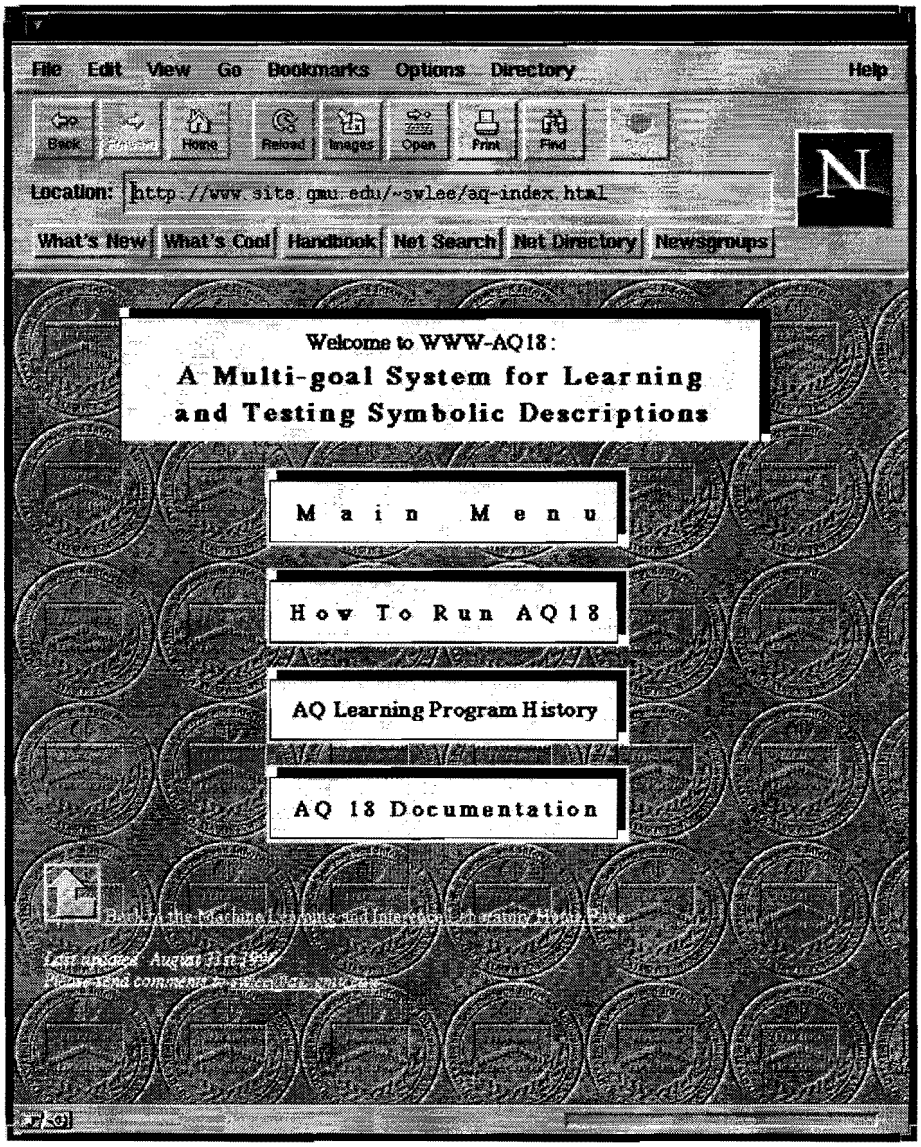

Figure 2. WWW-AQ Home Page

experimentation. Users request a *form* through the http server (by use of its URL) and fill out the form. At this time users remotely set the parameters and the data that they want to experiment with the AQ learning system. Users submit the form to the server and the server passes it to the AQ learning system. AQ accepts the information the user filled out and processes the data through the *integrated intelligent agent modules*. Finally the AQ learning system forwards the various results (e.g. rules learned, learning error graphs etc.) to the server and the server displays the results on the WWW browser. Those results include some

text and graphical files. As we can see from Figure 2, The WWW-AQ home page is consisted of four parts and the "Main Menu" is the core part that includes the *forms*. The "How to run

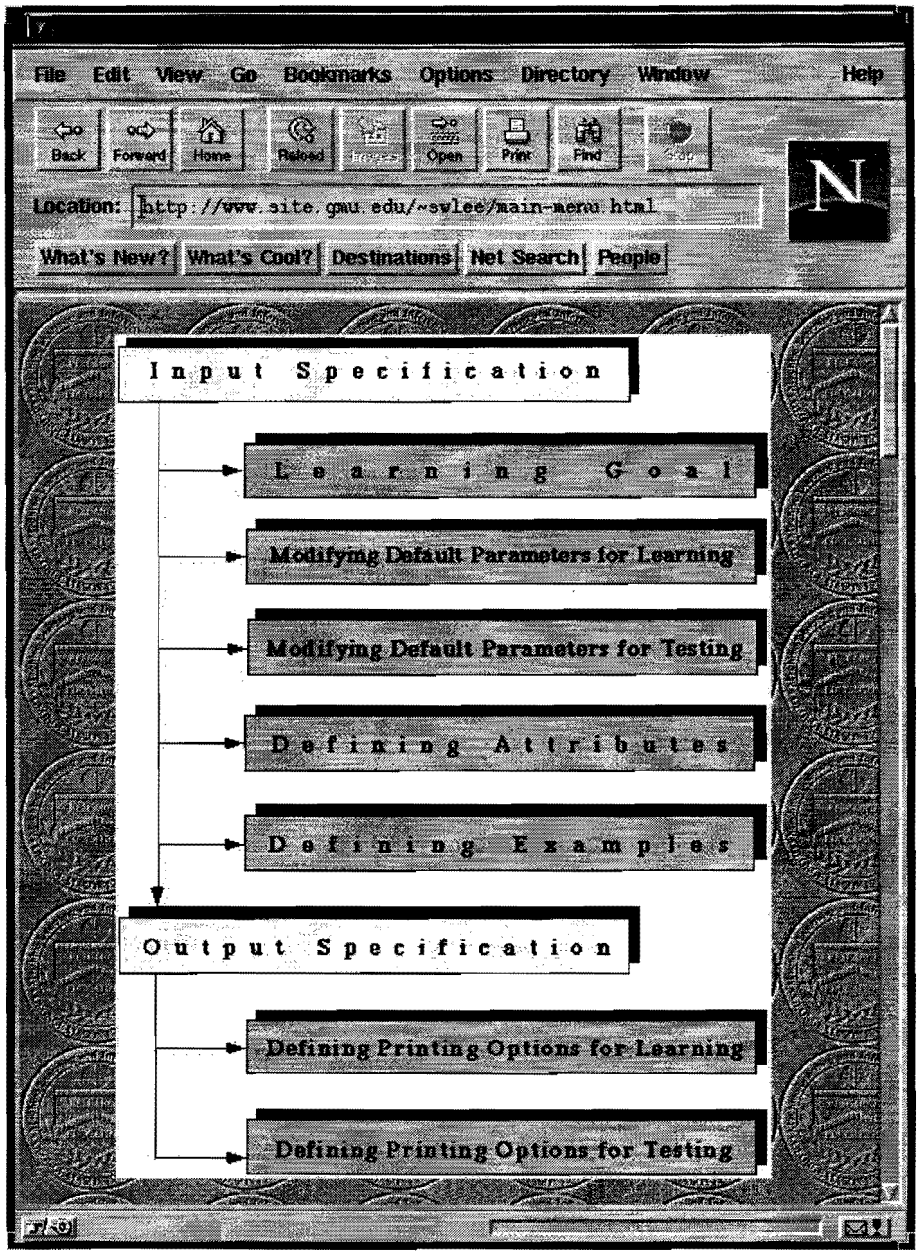

Figure 3. Specifying Learning Methods and Parameters

AQ 18" provides some instructions to the users when using WWW-AQ. It would be very helpful for users to know the general structure of the system before they start. "AQ Learning Program History" illustrates the long history of AQ family systems. "AQ 18 Documentation" contains the full manuals of the AQ18 Learning System. In fact, the users will likely need to read the first part of the manuals to be acquainted with the meanings of the parameters and

the tables that they need to set through the various input forms. After clicking the "Main Menu", Figure 3 shows up. The users who are already familiar with the Machine Learning Program would notice that these parameter setting mechanisms are well organized and

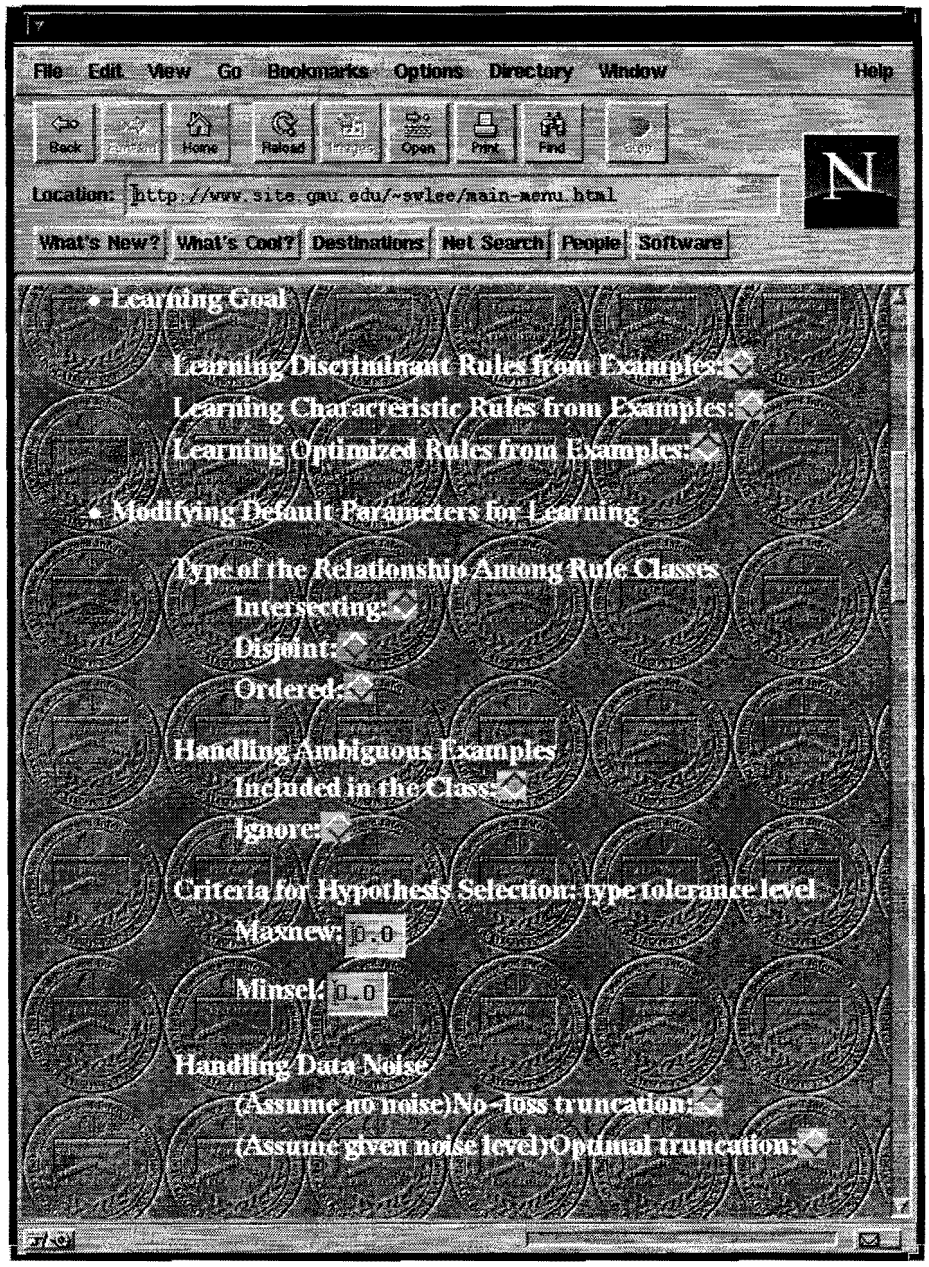

Figure 4. Specifying Learning Parameters: screen 1

clearly defined in an easy to understand format. Under the "Input Specification" title, there are five subfields to be filled. They are "Learning Goals", "Modifying the Default Parameters for Learning", "Modifying the Default Parameters for Testing", "Defining Attributes", and "Defining Examples". "Learning Goals" should be selected by chosing one of the three

methods in Figure 4. Options for Specifying Learning Parameters are shown in Figures 5-8. "Modifying the Default Parameters for Learning" and "Modifying the Default Parameters for

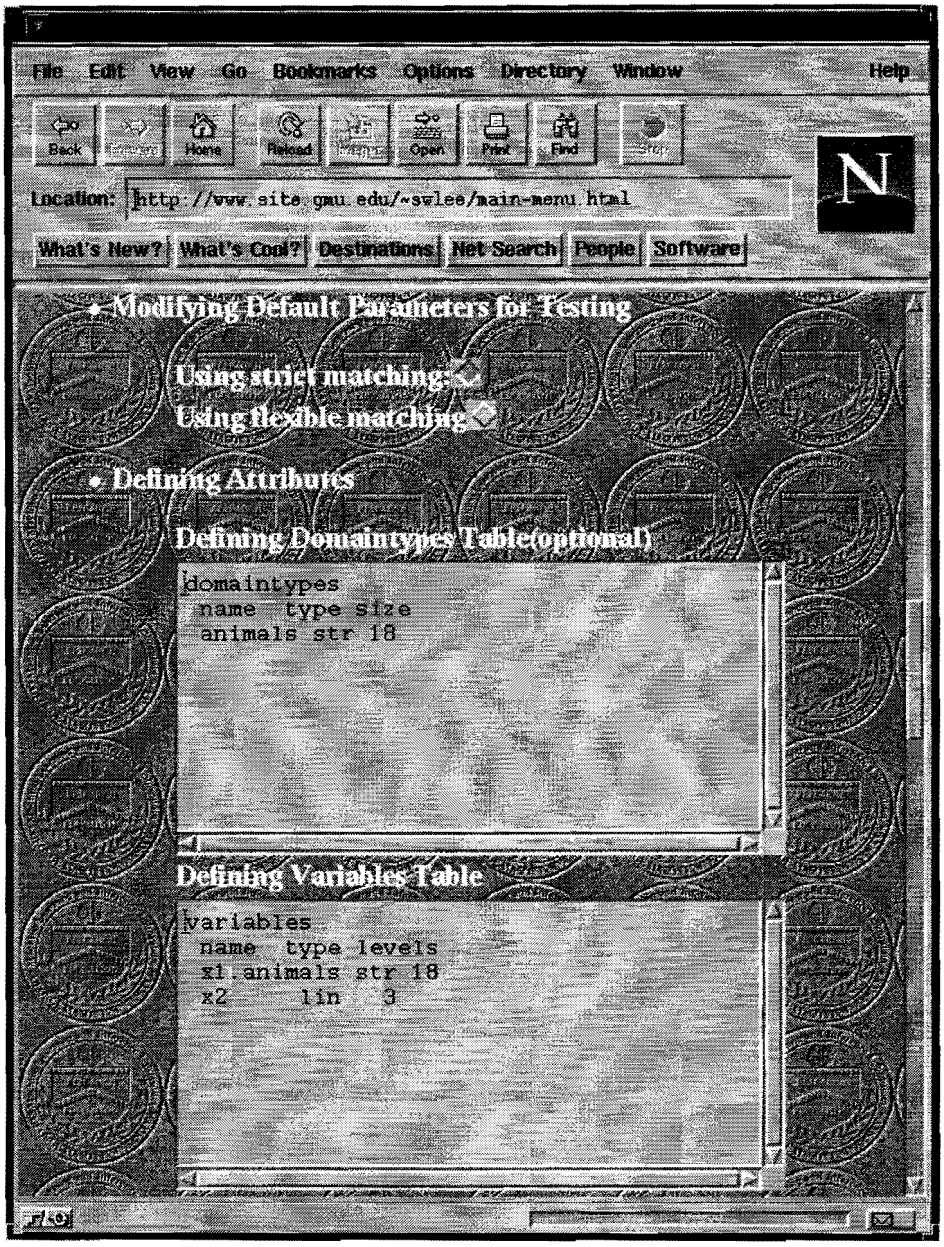

Figure 5. Specifying Learning Parameters: screen 2

Testing" provide various options for the user's different kinds of experiment objectives. In case parameters are not specified, the AQ system works with the available default values which was already set automatically. "Defining Attributes" and "Defining Examples" initially contain a set of sample examples. If the users have a small data set which can easily

be typed in by hand, this form has no problem. However, what if the users have huge data set that can not be entered by hand? This problem can be handled by choosing an alternative

Figure 6. Specifying Learning Parameters: screen 3

option from the main menu. Users may need to pick either one of these options before they start. This alternative version requires users to send their data files by email in advance. In other words, users are submitting the data files first and then selecting the parameters and

other options. *The mail filtering agent* will automatically forward the data set to the other related program modules for the AQ Learning System. These techniques are fully explained in section 3. After reading the documentation of the AQ learning system, readers might

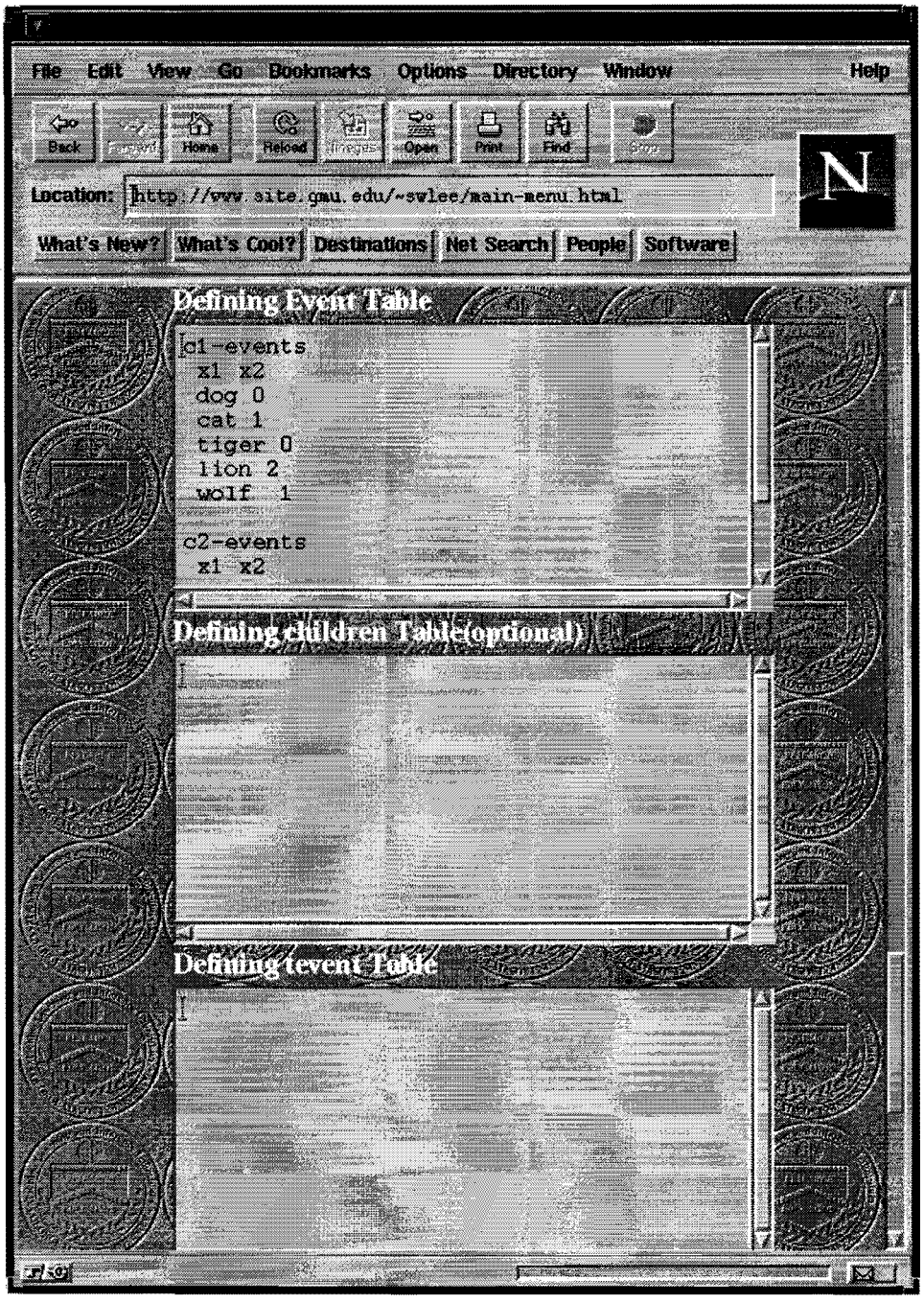

Figure 7. Specifying Learning Parameters: screen 4

notice that the AQ learning system also can support various kinds of hierarchical data structures (e.g. structure table, children table, etc.) which are very helpful for constructing

good knowledge representation spaces. Users may define the attributes with hierarchical structures. In the AQ learning system documentation [Wnek 1996], there are some examples that describe these hierarchical concepts in detail. The formats of training examples (event

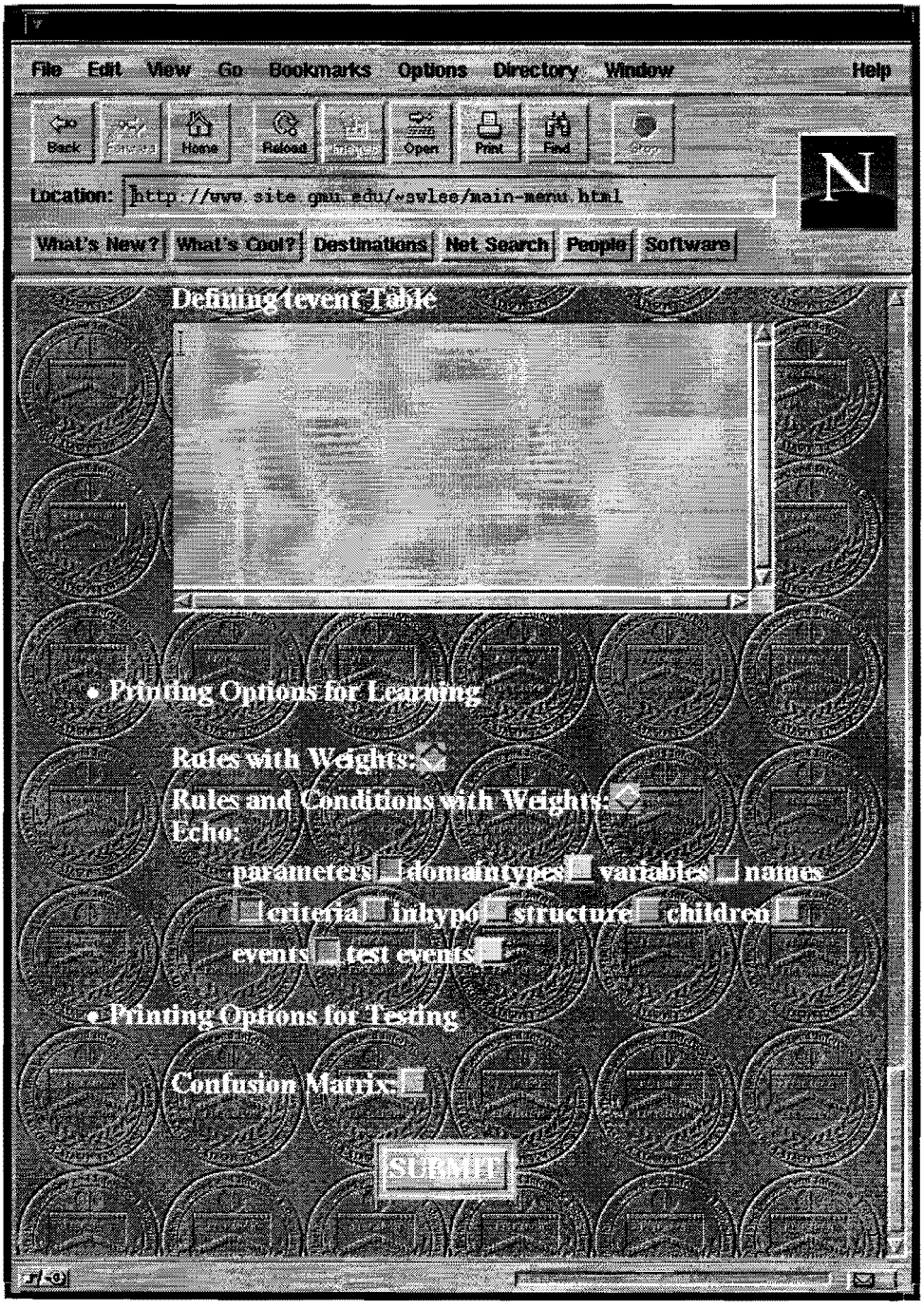

Figure 8. Specifying Learning Parameters: screen 5

tables) and the testing examples (tevent tables) are different from their equivalents in other learning programs (e.g. C4.5). However, If the users are using emailed data instead of using interactively hand-typed data, the emailed data set will be automatically transformed into the AQ input format by the *auto-formatting agent* called "aqformat", which is described in Section 5. Under the "Output Specification", there are two subfields called "Defining printing options for Learning" and "Defining printing options for Testing". Both control the display of reports obtained from the AQ learning system. In Figure 8, the users submit the required information to run the AQ learning system. The system processes this information with an integrated intelligent program module agent, described in the following sections. WWW-AQ also includes another program module agent called "EDC" to provide three kinds of learning system test methods (e.g. Hold out, K-fold cross validation, and Leave-one-out). EDC will be described in Section 4.

2 CGI PROGRAMMING

The Common Gateway Interface(CGI) emerged as the first way to present dynamically generated information on the World Wide Web. CGI allows the computer to generate Web pages instantly at the user's request rather than being written by someone in advance. CGI turns the Web from a simple collection of static hypermedia documents into a whole new interactive medium, in which users can ask questions and run applications. WWW-AQ is one of its typical applications. One of the most prominent uses of CGI is in processing *forms*. Forms are a subset of HTML that allow the user to supply information. The forms interface makes Web browsing an interactive process for the user and the provider. Figures 4 - 8 are examples. Web gateways are programs or scripts used to access information that is not directly readable by the client.

2.1 UNCGI VERSION 1.7

This UNCGI is a frontend for processing queries and forms from the Web on UNIX systems. It can be downloaded from *http://www.hyperion.com/~koreth/uncgi.html*. To process a form without this program, one would have to either write or find the application routines to translate the values of the form's fields from "URL encoding" to whatever program required. UNCGI decodes all the form fields and puts them into environment variables for easy perusal by a shell script, a C program, a Perl script, etc. and then executes the specified program.

2.2 USING UNCGI FROM C

UNCGI can be called as a library function from a C program. as follows:

1) Compile uncgi.c
   gcc -DNO_MAIN -c uncgi.c // This command will create a "uncgi.o" file.
2) Call uncgi() function from within a C program.
3) Use environment variables to read the form results.
4) Link the program with uncgi.o
   gcc -o aqinterface.cgi aqinterface.c uncgi.o
   // This command will create an executible file called "aqinterface.cgi".
5) Install in the http server's cgi-bin directory.
6) Write a form to call the program.

Please refer to the source codes in the appendix for more information about CGI programming techniques used here. Manuals also can be downloaded from *http://www.hyperion.com/~koreth/uncgi-c.html.*

3 MAIL FILTERING AGENT

In order to use data files created offline, users should send the data set to the WWW-AQ server by email. *The mail filtering agent* will screen all incoming mails and will filter the data set. We employed the ELM filtering system to implement this agent.

3.1 ELM MAIL FILTERING SYSTEM

The ELM filtering system allows users to define a set of rules by which all incoming mail is screened, and a subsequent set of actions to perform based on whether the conditions are met or not. This filtering system also has the ability to mail a summary of what actions it performed on the incoming mail as often as is desired. The language for writing filter rules is quite simple. The fundamental structure is:

if (condition) then action

where condition is constructed by an arbitrary number of individual conditions of the form "<field> <relation> <value>" (an optional type of rule is of the form "always action" but this should only be used as the last rule in the ruleset for obvious reasons). The field value can be subject, sender, from, to, lines. For the field lines, the relation can be any of the standard relationships('>', '<', '>=', '<=', '!=' and '='). Contains is equivalent to the relation '=' or

the relationship maybe skipped entirely. The value is any quoted string that is to be matched against, or a number if line is the field being considered. In order to use this mail filtering technique, two files have to be created "$HOME/.forward" and "$HOME/.elm/filter-rules". The contents of the "$HOME/.forward" file are as follows:

```
"|/usr/local/bin/filter"
```

$HOME/.elm/filter-rules contains the following rules and this rule is only for WWW-AQ.

```
# $HOME/.elm/filter-rules
# rule 1
  subject = "aq.data" ? savecopy "$HOME/tmp"
```

Meaning: All messages with a subject that contains the string "*aq.data*" should be saved in the folder $HOME/tmp and also dropped into the account mailbox.

If the remote users want to use their data set they should send them by email to the server. The subject of the email must be "aq.data" so that the mail filtering system can filter them out and save them to a location known to the AQ learning system. The *auto-formatting agent* called "AQFORMAT" will take the data set and transform its format to an appropriate format that the AQ learning system can accept. AQFORMAT, which performs the data transformation process, includes some C programs and shell scripts. The data set that users are sending to the WWW-AQ server should have the following format:

```
<attribute1>  <attribute2>  <attribute3>  <attribute4> ...<attribute m>
 <value11>     <value12>     <value13>     <value14> ...<value 1m>
                             .....
 <value n1>    <value n2>    <value n3>    <value n4>...<value nm>
```

This kind of data format is very popular and easy to maintain; it is used by many database management systems.

4 AQ EVALUATION PROGRAM MODULE (EDC)

One of the most important characteristics of a learning system is its predictive accuracy that is, its ability to correctly classify previously unseen cases. The need for a standardized testing procedure has led to the development of several cross-validation methodologies that evaluate a system's predictive accuracy in terms of its produced error rates. Furthermore, it is often useful to monitor a system's incremental improvement through a multistage cross-validation procedure. EDC (Experimental Design Component) is a program module that integrates these methods with the AQ learning system and provides a complete testbed for experimentation.

WWW-AQ uses this *EDC agent* to test the data set with various data testing methods and provides learning error graphs for each methods applied. Currently the method is set to Hold Out with 3 stages, and 70% training examples. Please refer to [Doulamis 1996] for more information.

5 AQFORMAT PROGRAM MODULE

AQFORMAT, which represents an *auto-formatting agent*, reads the emailed data set and transforms it into a data format that the AQ learning system can process. AQFORMAT accepts the following data format.

<attribute1>	<attribute2>	<attribute3>	<attribute4> ...<attribute m>
<value11>	<value12>	<value13>	<value14> ...<value 1m>
		.....	
<value n1>	<value n2>	<value n3>	<value n4>...<value nm>

AQFORMAT takes 4 inputs:

1) filestem name. It reads input from filestem.data in the currrent directory. filestem.data must be available in the same directory.
2) attribute index/name. Attribute index is used at this time. We can specify the last column (attribute) as the class variable by specifying '0' if you do not know the exact index.
3) %training. This is the percent of data from each class that will be allocated to training events.
4) %testing. This is the percent of data from each class that will be allocated to testing events. %training + %testing must equal 100.

Example: > aqformat aqsamp 0 60 40
Meaning: aqformat program is running with the last column as the class attribute, 60% of the data is used for training and 40% is used for testing. Data is read from the file aqsamp.data

Example:> aqformat aqsamp 1 100 0
Meaning: aqformat program is running with the first column as the class attribute, all of the data is used for training and no examples are provided for testing. Data is read from aqsamp.data.

Output is put in <filestem>.* files:
1) <filestem>.domain.aq - names tables, variables table in aqformat
2) <filestem>.train - events tables
3) <filestem>.test - tevents tables

If <filestem>.train, <filestem>.test or <filestem>.domain.aq are present in the directory when aqformat is run, they will be overwritten when aqformat is run.

6 DRAWING ERROR GRAPHS WITH MATLAB

In order to draw learning error graphs with the results obtained from the *EDC agent*, the MATLAB package is used. The results from the EDC agent are saved in a file read by MATLAB, which generates three error graphs (overall, omission, commission). The following script called "aqescript.m" is input to the MATLAB package and graphs are saved at /tmp/aqe-test.gif. Figure 10 shows the error graphs generated by the MATLAB.
Example: matlab < aqescript.m

```
load /tmp/aqeresult.dat
stage = aqeresult(:,1)
overall = aqeresult(:,2)
commission = aqeresult(:,3)
omission = aqeresult(:,4)
plot(stage, overall, 'y-', stage, commission, 'r-', stage, omission,'b-')
xlabel('Stages')
ylabel('Error Rates')
title(' Evaluation of AQ')
title(' Over_all Error (yellow), Commission Error (red), Omission Error (blue)')
print -dgif8 /tmp/aqe-test.gif
quit
```

7 EXPERIMENT WITH WWW-AQ

For experiment, we have used industrial application data set called "wind bracings". Figure 9 and 10 are the main result screens obtained from WWW-AQ. In Figure 9, the parameters already specified by the users are displayed with the results from the AQ learning system, so that the users may inspect the results given the defined parameters. Figure 10 shows the learning error graphs generated from the EDC agent by use of MATLAB. Red, blue, and yellow curves indicate commission, omission, and overall errors, respectively. The error rates

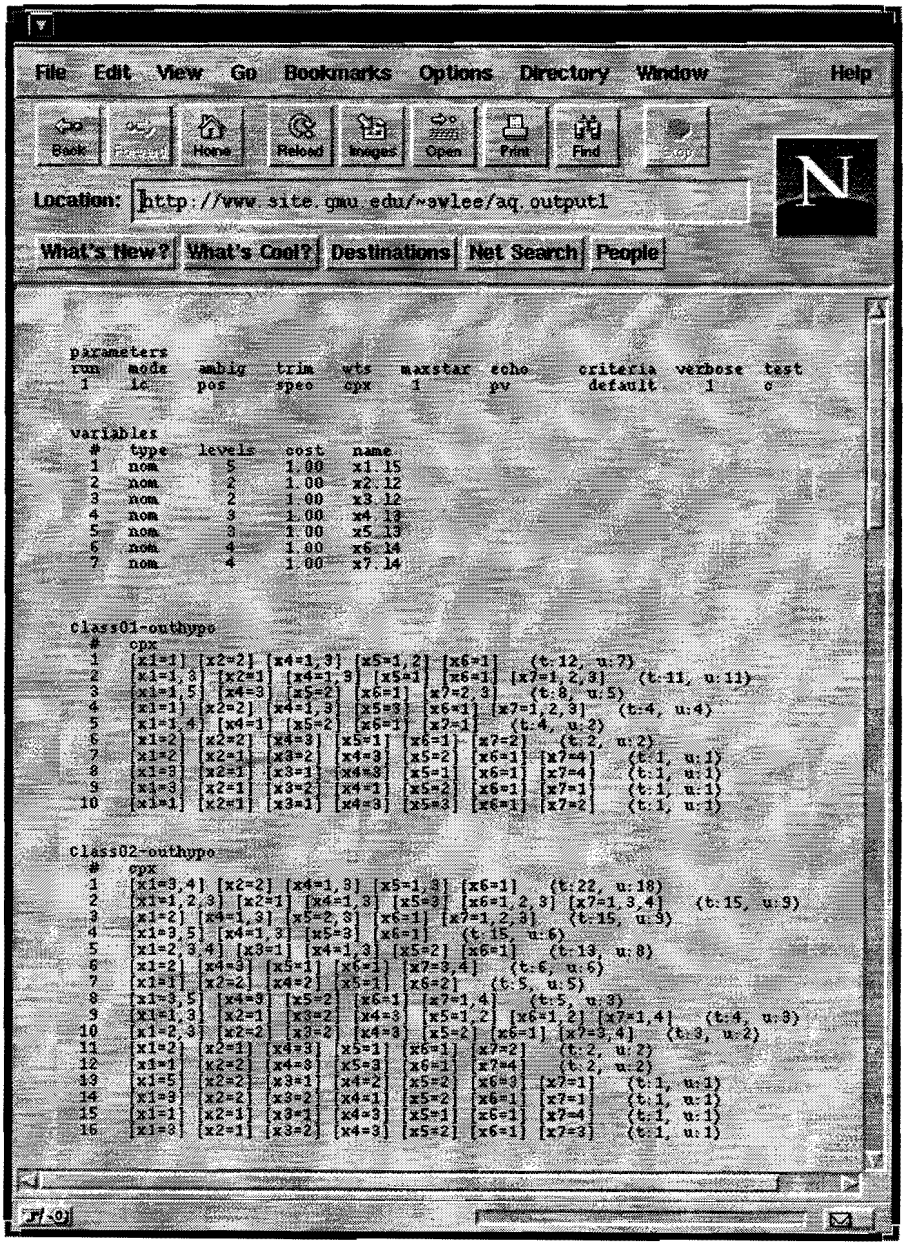

Figure 9. Output Screen from AQ Learning System

are automatically calculated after the specified cross validation testing is performed. Please refer to [Weiss et. al 1991] for more information about these testing method algorithms. The graphs in Figure 10 are generated based on the following numbers obtained from the EDC agent. The number of stages, the testing method, and the other parameters can be set by the users in future versions.

3-stage Holdout results with 70% training

Stage	Overall Error %	Error of commission %	Error of omission %
1	32	10.7143	22.2222
2	15	0	11.185
3	6	0	7.69453

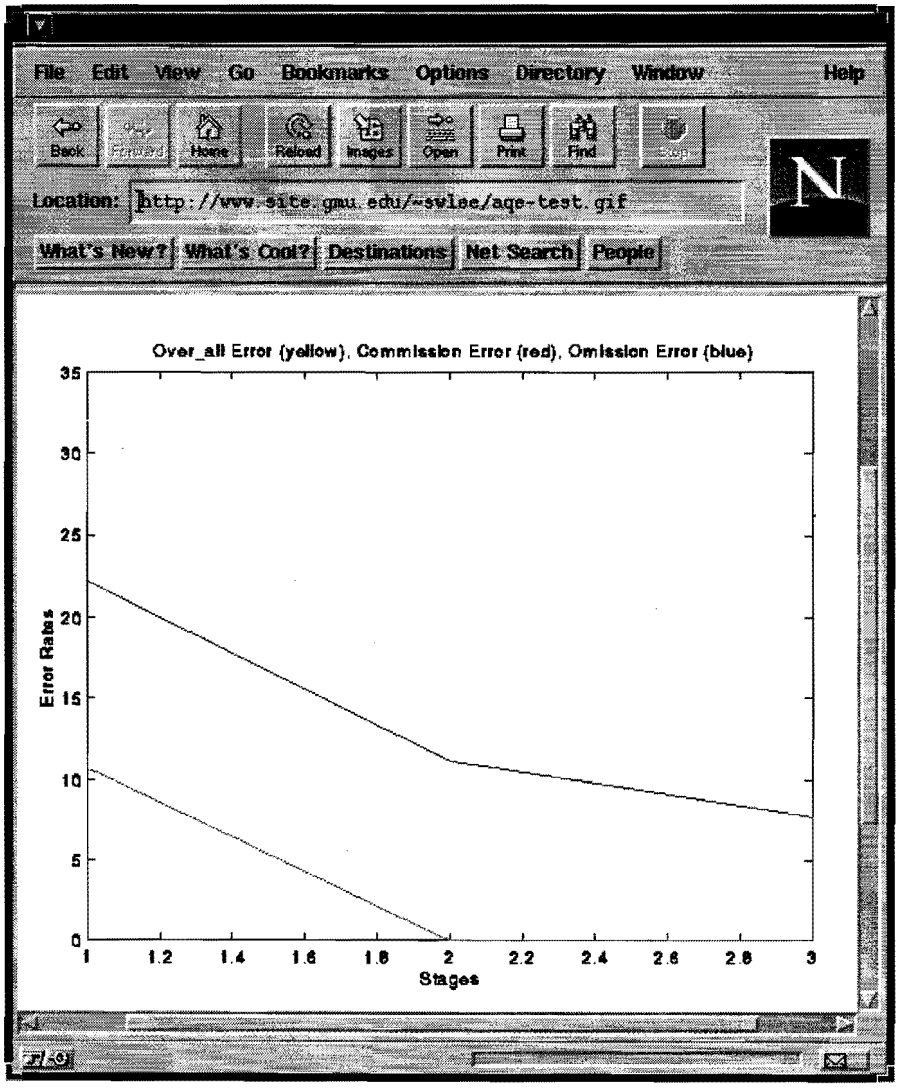

Figure 10. Error Graph from AQ Evaluation Program

7.1 INTEGRATING PROGRAM MODULE AGENTS

The structures of the integrated intelligent agents that work with the WWW-AQ are described in Figure 11. Each agent has been discussed in previous sections. However, it is very helpful to understand the structure of the entire WWW-AQ system by examining the diagram.

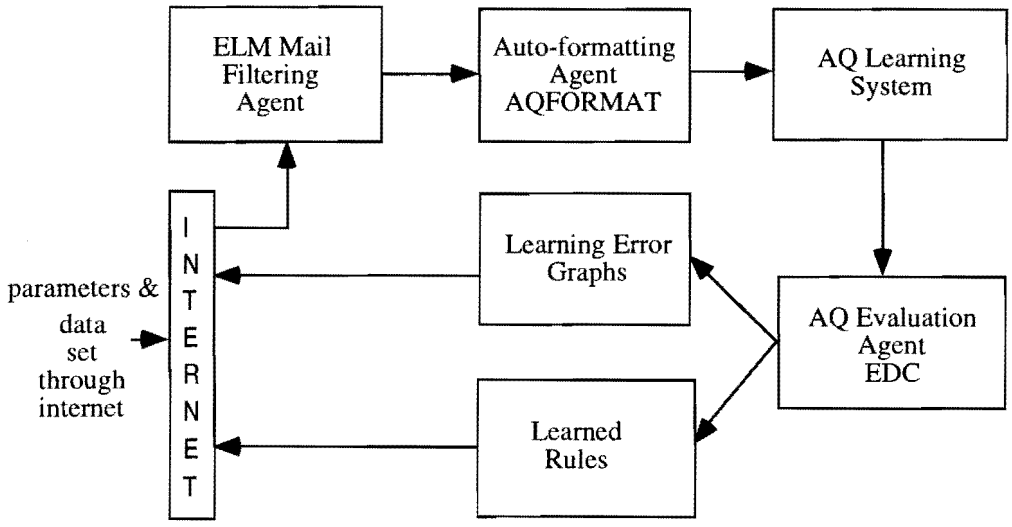

Figure 11. Integrating Intelligent Agents in WWW-AQ

Compared to Figure 1, this is a wider view in terms of interactions between users and program agents through the Internet. Figure 11 also represents the inside structures of the "AQ System" in Figure 1.

7.2 PROGRAMMING ENVIRONMENT

WWW-AQ is currently installed on the SITE (School of Information Technology and Engineering) machine at the George Mason University. In order to install WWW-AQ, a machine must meet the following conditions:

1) SunOS or Solaris Operating System to install the AQ learning System
2) HTTP server
3) Fully installed ELM mail server
4) Fully installed UNCGI program

The users might notice that the WWW-AQ server sometimes does not respond quickly after submitting the main form due to heavy network traffic.

7.3 IMPLEMENTATION ISSUES

There were several implementation difficulties on this work because the HTTP process and the CGIs run as the user *nobody*. The files created by "nobody" are being used as an input to other agents (e.g., EDC: AQ evaluation agent) However, they can not be accessed by the WWW-AQ server because the ownerships are different. Therefore, it was necessary to create the files (that the "nobody" will create) by the WWW-AQ before their contents are loaded and change the protection mode to 777. The "nobody" process will then overwrite the pre-existing files and the ownership of the files are not changed to *nobody*. The author has examined other systems regarding these matters and has determined that these are system dependent issues.

It is necessary to create the following files in advance with WWW-AQ's ownership under /tmp directory.

aq.inp1 aq.out1 aq.output1 aqnew.inp aqe.out (executible of EDC)
aqresult.dat aqescript.m (script for MATLAB) aqe-test.gif
aq.inp aq.out run00.out aq-format.data aq-mail.data
aq.output aqformat (executible) aq-format.domain.aq
aq-format.train aq-format.test

8 FUTURE WORK

There are several features to be added to WWW-AQ in the future. An access counter is one, by which will allow us to know how many users have used the WWW-AQ Learning System. Second, the mail filtering system should keep the information about the data sets that have been filtered out. In addition, size of the file, sender, and the time are useful information for report purposes.

Currently the WWW-AQ system is not been fully tested in a multi-user environment. However, it is necessary to satisfy this requirement in the future. Some implementation difficulties already mentioned should be considered. Finally this work also shows the need for potential research in developing intelligent agents on the Web. Progress on this work will be reported in future reports.

REFERENCES

Doulamis J., "Experimental Design Component: User's and Programmer's Guide" *Reports of the Machine Learning and Inference Laboratory*, George Mason University 1996 To apprear.

Gundavaram S., *CGI Programming on the World Wide Web*, O'Reilly & Associates, Inc. 1996.

Michalski R. S., Wnek J., Kaufman K., Bloedorn E., Lee S. W., "AQ18: A Multi-Goal Inductive Learning Environment", *Reports of the Machine Learning and Inference Laboratory* , George Mason University 1996 To appear.

Weiss S.M. and Kulikowski C.A., *Computers that Learn*, Morgan Kaufmann Publishers, San Mateo, CA, 1991

Wnek J., Kaufman K., Bloedorn E., Michalski R. S., "Inductive Learning System AQ15c: The Method and User's Guide" *Reports of the Machine Learning and Inference Laboratory*, George Mason University 1996.

APPENDIX

CGI PROGRAM SOURCE CODES: AQ-INTERFACE.C-

```
/* aq-interface.c                                    */
/* uncgi program for AQ interface          */
/* written by Seok Won Lee                 */
/* Sep. 5th 1996                                     */

#include <stdio.h>
#include <stdlib.h>

void main (void)
{
  FILE *fp;  /* file pointer fro aq-in */
  char *ex1, *ex2, *ex3, *ex6, *ex7, *domain_t, *var_t, *name_t,
             *structure_t, *inhypo_t, *event_t, *child_t, *tevent_t, *ex8,
             *b, *c, *d, *e, *q, *p, *n, *i, *s, *v, *confusion;
  float ex4, ex5;

    printf ("Content-type: text/html\n\n");

    uncgi();

    ex1 = getenv("WWW_ex1");
    ex2 = getenv("WWW_ex2");
    ex3 = getenv("WWW_ex3");
    ex4 = atof(getenv("WWW_ex1"));
    ex5 = atof(getenv("WWW_ex1"));
    ex6 = getenv("WWW_ex6");
    ex7 = getenv("WWW_ex7");
    domain_t = getenv("WWW_domain_t");
    var_t = getenv("WWW_var_t");
    name_t = getenv("WWW_name_t");
    structure_t = getenv("WWW_structure_t");
    inhypo_t = getenv("WWW_inhypo_t");
    event_t = getenv("WWW_event_t");
    child_t = getenv("WWW_child_t");
    tevent_t = getenv("WWW_tevent_t");
    ex8 = getenv("WWW_ex8");
    b = getenv("WWW_b");
    c = getenv("WWW_c");
    d = getenv("WWW_d");
    e = getenv("WWW_e");
    q = getenv("WWW_q");
    p = getenv("WWW_p");
    n = getenv("WWW_n");
    i = getenv("WWW_i");
    s = getenv("WWW_s");
    v = getenv("WWW_v");
    confusion = getenv("WWW_confusion");

/* writing aq-input file */

 if ((fp = fopen("/tmp/aq.inp1","w"))==NULL) {
     printf("Cannot open aq input file\n");
     exit(1);
  }
 fprintf(fp, "parameters\n");
 fprintf(fp, "trim mode ambig criteria trunc wts maxstar verbose echo
test\n");
```

```
while(*ex1 != '\0') {
fprintf(fp, "%c", *ex1); ++ex1; }
fprintf(fp, "  ");

while(*ex2 != '\0') {
fprintf(fp, "%c", *ex2); ++ex2; }
fprintf(fp, "  ");

while(*ex3 != '\0') {
fprintf(fp, "%c", *ex3); ++ex3; }
fprintf(fp, "  ");

fprintf(fp, "myself  ");

while(*ex6 != '\0') {
fprintf(fp, "%c", *ex6); ++ex6; }
fprintf(fp, "  ");

while(*ex8 != '\0') {
fprintf(fp, "%c", *ex8); ++ex8; }

fprintf(fp, "   10   0  ");

if(p != NULL) fprintf(fp, "p");
if(v != NULL) fprintf(fp, "v");
if(d != NULL) fprintf(fp, "d");
if(n != NULL) fprintf(fp, "n");
if(c != NULL) fprintf(fp, "c");
if(s != NULL) fprintf(fp, "s");
if(i != NULL) fprintf(fp, "i");
if(b != NULL) fprintf(fp, "b");
if(e != NULL) fprintf(fp, "e");
if(q != NULL) fprintf(fp, "q");

while(*ex7 != '\0') {
fprintf(fp, "  %c", *ex7); ++ex7; }
if(confusion != NULL)  fprintf(fp, "c");

fprintf(fp, "\n\n");  /* end of parameters table */

while (*domain_t != '\0') {
  if (*domain_t != ')
fprintf(fp, "%c", *domain_t); ++domain_t;  }
fprintf(fp, "\n\n");

while (*var_t != '\0') {
  if(*var_t != ')
fprintf(fp, "%c", *var_t); ++var_t; }
fprintf(fp, "\n\n");

while(*name_t != '\0') {
  if (*name_t != ')
fprintf(fp, "%c", *name_t); ++name_t; }
fprintf(fp, "\n\n");

while(*structure_t != '\0') {
  if (*structure_t != ')
fprintf(fp, "%c", *structure_t); ++structure_t; }
fprintf(fp, "\n\n");

fprintf(fp, "myself-criteria\n");
fprintf(fp, "criterion  tolerance\n");
fprintf(fp, "maxnew     %.2f  \n", ex4);
fprintf(fp, "minsel     %.2f  \n\n", ex5);
```

```
 while(*inhypo_t != '\0') {
   if (*inhypo_t != ')
 fprintf(fp, "%c", *inhypo_t);  ++inhypo_t; }
 fprintf(fp, "\n\n");

 while(*event_t != '\0') {
  if (*event_t != ')   fprintf(fp, "%c", *event_t);
  ++event_t; }
 fprintf(fp, "\n\n");

 while(*child_t != '\0') {
   if (*child_t != ')
 fprintf(fp, "%c", *child_t); ++child_t; }
 fprintf(fp, "\n\n");

 while(*tevent_t != '\0') {
   if (*tevent_t != ')
 fprintf(fp, "%c", *tevent_t);  ++tevent_t; }
 fprintf(fp, "\n\n");      /* end of tables */

 fclose(fp);

/* running AQ with given input */

system("/student/swlee/public_html/aq.run < /tmp/aq.inp1 > /tmp/aq.out1 ");

system("cat /tmp/aq.out1 | dos2unix > /tmp/aq.output1");

/*************************************************************/
    Graph stuff
/*************************************************************/

system("cp /tmp/aq.inp1 /tmp/aqnew.inp");
system("/tmp/aqe.out /tmp/aqnew 1 3 70");
system("cat /tmp/run00.out | /student/swlee/public_html/first_space1 | sed
-f /student/swlee/public_html/second_space > /tmp/aqeresult1.dat");
system("/usr/local/bin/matlab < /tmp/aqescript.m > /tmp/qqq");

system(" cp /tmp/aqe-test.gif /student/swlee/public_html/.");

/* display output */

printf("<h3><A href=\"http://www.site.gmu.edu/~swlee/tmp\"> Please click
here to see the results! </A></h3> <br>");

printf("<h3><A href=\"http://www.site.gmu.edu/~swlee/aqe-test.gif\"> Please
click here to see the error graphs! </A></h3> <br>");

 exit(0);

}
```