Knowledge-based Links for Automatic Interaction with Programming Online Judges

Guojin Zhu
Dept. of Computer Science, Donghua University, Shanghai 201620, China
Email: gjzhu.dhu@163.com

Yefeng Chen
Dept. of Computer Science, Donghua University, Shanghai 201620, China
Email: chenyefeng5@sina.com

Abstract—**Programming online judges are computing resources with pre-designed test data. Recently, an increasing attention has been paid on integration of online judges into a tutoring system. However, it is difficult for a local system to automatic interaction with remote online judges on the web to share their computation, because these computing resources are designed for human users only. To address such issue, a novel link is proposed here, called framed link, which consists of a frame-based representation for the knowledge about how to interact with the computing resource that the link points to. With the framed link, three remote online judges including their internal test data are integrated successfully into a local system for programming courses.**

Index Terms—**Framed Link, Automatic Interaction, Online Judge, Frame-System, Remote Resource Integration**

I. Introduction

There are lots of computing resources with internal data on the web that are valuable for local computer systems. Like the majority body of the web [1], most of these computing resources, however, are designed for human user only. A human user can interact with the computing resource easily by filling the web form with the data to be computed, clicking the hyperlink to submit the data to the computing resource, and reading the web page at the computing resource site that contains the results of the computation, whereas a machine is hardly able to do these things without human being assistances. In simple words, it is difficult for local systems to interact with these computing resources on the web to share their computation including their internal data.

An online judge [2-4], for instance, is such a computing resource that is designed for human users only. Students from anywhere in the world can submit their program source codes to the online judge, which will compile the submitted source codes to check whether there are grammatical errors in them or not. If there is no grammatical error in a submitted program, the online judge will run the executable code of the submitted program and compare the output data from this running procedure with pre-designed test data. The results of the judgment will be displayed on a web page whenever a student asks the online judge for the test results of his or her submitted programs.

There are many online judges on the web, among which UVa OJ [5], PKU OJ [6], and Timus OJ[1] are distinguished representatives. These computing resources are valuable for programming teaching and learning [7]. Teachers can select suitable programming problems from online judges for their students to practice programming. However, it is inconvenient for teachers to check the program source codes that their students submit to online judges on the web. One possible way to avoid this inconvenience is to build a local tutoring system that can share the resources on the remote online judges. When receiving a program submitted by a student, the local tutoring system is supposed to forward the submitted program to a respective remote online judge and save to its local database the submitted program together with the testing result which is obtained from the remote online judge. From the local tutoring system, teachers can easily obtain the information about the achievements of their students in the programming practice situation. Nevertheless, how does a local tutoring system to interact autonomously with remote online judges that are designed for human users only?

To address the issue above, we propose a novel link, called *framed link*, which consists of a frame. The frame encodes the knowledge about how to interact with the computing sources. The content of the frame is a description of a specific computing resource that the framed link points to, which probably includes the website address of the computing resource, the web forms required to fill with the data to be submitted and the format of the page that contains the results produced by the computing resource. By using the knowledge encoded in the frame, a machine can interact automatically with the computing resource that the framed link points to without human being assistances. This makes it possible for a local system to interact with the computing resources on the web to share their internal data that are

Corresponding author: Guojin Zhu; E-mail: gjzhu.dhu@163.com.
1. http://acm.timus.ru/

involved in their computation. In this paper, we take online judges as an example to show how the framed link works.

II. The Model of Automatic Interaction with Online Judges

It is difficult for a local tutoring system to access to a remote online judge because the latter is designed for human users only. Thus, an application programming interface (API) is needed between the local tutoring system and the remote online judge. The API is supposed to provide the local tutoring system with operations necessary to interact with the remote online judge. These operations enable the local tutoring system to log in to a remote online judge, to send submitted programs to the remote online judge, and to query the results of the judgments. The model of using an application programming interface for interaction with computing resources on the web is shown in Fig. 1, where UVa OJ, PKU OJ, and Timus OJ are remote online judges.

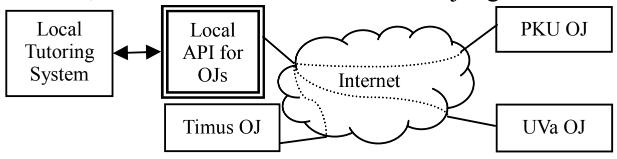


Figure 1. The local API provides the local tutoring system with resources on remote online judges.

A. Operations of Interaction

Consider a programming problem entitled *A + B Problem* on the PKU OJ. The ID number of this problem is 1000. Fig. 2 shows a program written in the C++ language as a solution to this *A + B Problem*. To submit the program, one needs to log in to the PKU OJ first. After submitting the program, one can query the judgment result from the PKU OJ. To mimic these actions of a human user, we design three corresponding operations for the local API as listed in TABLE I.

The operation api.login(n) is designed for the local tutoring system to log in to a remote online judge. The parameter n is an identifier of a remote online judge, e.g., $n = PKU$. By calling the operation api.login(PKU), the local tutoring system can log in to the PKU OJ.

The operation api.submit(s, p, l) enables the local tutoring system to submit a program to its last logged-in online judge. The parameter s is the source code of the program to be submitted to the logged-in online judge, the parameter p is the ID number of the problem that the submitted program is supposed to solve, and the parameter l is the ID number of the language in which the submitted program is written. Upon logging in to the PKU OJ, the tutoring system can submit the source code in Fig. 2 by calling the operation api.submit(S_{a+b}, 1000, 4), where S_{a+b} denotes the source code in Fig. 2, 1000 is the ID number of the problem *A + B Problem*, and 4 is the ID number of the C++ language in the PKU OJ.

The operation api.query() will read the web page at the remote online judge site that containing the judgments of submitted programs, and return the verdict of the last submitted program to the local tutoring system. For example, the local tutoring system can get the verdict of the source code for the problem *A + B Problem* by calling the operation api.query() following the operation api.submit(S_{a+b}, 1000, 4).

The same local API is applicable to different remote online judges. Fig. 3 shows another program that is also written in the C++ language. The program solves a programming problem on the UVa OJ, which is entitled *Power of Cryptography*. The local tutoring system can employ the same local API to submit this program to the UVa OJ by calling the submitting operation api.submit(S_{power}, 113, 3) after a log-in operation api.login(UVa), where S_{power} denotes the source code in Fig. 3, 113 is the ID number of the problem *Power of Cryptography*, and 3 is the ID number of the C++ language in the UVa OJ. After these two operations, the local tutoring system can get the verdict of the source code for the problem *Power of Cryptography* by calling the operation api.query().

```
#include<iostream>
using namespace std;
int main(){
   int a,b;
   while(cin>>a>>b){
      cout<<a+b<<endl;
   }
   return 0;
}
```

Figure 2. The program for the problem *A + B Problem*.

```
#include<iostream>
#include<math.h>
using namespace std;
int main(void) {
  for (double dNum, dPow;
     cin>>dPow>>dNum;
     cout<<(int)(pow(dNum,1.0/dPow)+0.5)<<endl
  );
  return 0;
}
```

Figure 3. The program for the problem *Power of Cryptography*.

TABLE I.
Three Operations of the Local API

Operation	Comment
api.login(n)	To log in to the remote online judge that the parameter n identifies.
api.submit(s, p, l)	To submit a problem solution to the last logged-in online judge. The parameter p identifies the problem that is supposed to be solved by the program source code s. The parameter l indicates the language in which the program is written.
api.query()	To make a query for the testing result of the last submitted solution.

B. Automatic Interaction by using Framed Links

How does the same local API enable a local machine to interact with different remote computing resources? The answer is the framed link, see Fig. 4. It is similar to a situation in which the same browser enables a human user to read pages at different websites. When the local tutoring system calls the log-in operation api.login(PKU), the local API will select the PKU home frame which is embedded with three framed likes, each pointing to a

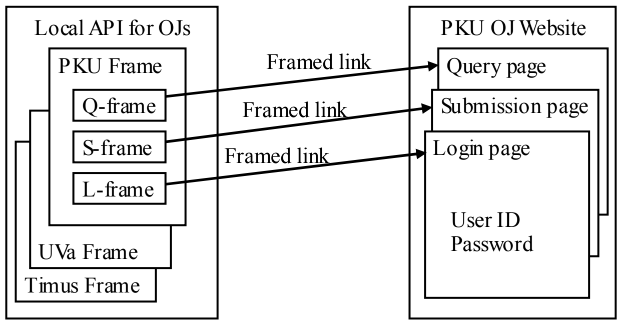


Figure 4. The model of interaction by using framed links.

respective page at the PKU OJ website. When the local tutoring system calls the operation api.login(*UVa*), the local API will select the UVa home frame for logging in to the UVa OJ.

The element *L-frame* embedded in the PKU home frame specifies the link to the login page at the PKU OJ website. It encodes the knowledge about how to interact with the login page that the link points to. Using this knowledge, the operation api.login(*PKU*) can enable the local tutoring system to log in to the PKU OJ.

When the local tutoring system calls the submitting operation api.submit(S_{a+b}, 1000, 4) following the operation api.login(*PKU*), the local API will employ the knowledge which is encoded by the element *S-frame* in the PKU home frame to fill the web form on the submission page of the PKU OJ with the respective parameter values S_{a+b}, 1000 and 4. After filling the web forms, the local API will click the button on the submission page of the PKU OJ to complete the submitting action.

The knowledge encoded by the element *Q-frame* in the PKU home frame enables the operation api.query() to read the query page of the PKU OJ and return the verdict of the last submitted program.

TABLE II compares the framed link with the hyperlink. The user of the hyperlink is a human being, whereas the user of the framed link is a machine. The hyperlink is embedded in a webpage which usually includes a description about how to use the computing resource that the hyperlink points to, whereas the framed link is embedded in a home frame which describes functions of a remote online judge. Moreover, the browser is the tool that enables a human user to click the hyperlink, whereas the local API is the tool that enables a local machine to employ the framed link by calling an operation of the local API.

TABLE II.
FRAMED LINKS VS. HYPERLINKS

User	Tool	Description	Linkage
Human	*Browser*	*Pages*	*Hyperlinks*
Machine	*API*	*Frames*	*Framed links*

III. THE FRAMED LINK

A link specified by a frame is called a framed link. More specifically, a framed link is such a link that consists of a frame which contains the knowledge about how to interact with the resource that the link points to. The frame here is a data-structure proposed by Marvin Minsky for representing a stereotyped situation [8]. For the framed link, the frame is composed of some slots, each of which has a name, a marker and an assignment [9].

According to Marvin Minsky, attached to each frame are several kinds of information. Some of this information is about how to use the frame. Some is about what one can expect to happen next. Some is about what to do if these expectations are not confirmed. These kinds of attached information are needed for clicking a framed link. We express the information attached to a frame as an algorithm of a clicking procedure attached to its framed link.

There are three kinds of frames for links to remote online judges, which point to login pages, submission pages and query pages, respectively.

A. The Link to a Login Page

In order to log in to an online judge, one needs to enter a user name (e.g., *pkuuser*) and a password (e.g., *pkupass*) on the login page of the online judge. Fig. 5 shows the source code of the login page of the PKU OJ, which is written in the Hypertext Markup Language (HTML [10]). The uniform resource identifier (URI [11]) of this login page is http://poj.org/login. In the first line in Fig. 5, we can find the word *POST*, which will instruct the web browser to use the post method of the Hypertext Transfer Protocol (HTTP [12]) to pass the user name and password to the PKU OJ when a user logs in to this online judge. In the HTTP post method [13, 14], the user name and the password are encoded as key-value pairs (also known as name-value pairs). For the PKU OJ, the key for the user name is *user_id1* and the key for the password is *password1*, which could be found in the third line and the sixth line in Fig. 5, respectively. The values of the user name and the password are what the user enters on the login page (e.g., *pkuuser* and *pkupass*, respectively). All these kinds of information should be encoded in the frame, named as *L-frame*, which specifies the framed link to the login page of a remote online judge.

An L-frame consists of five slots, named *LoginURI*, *UserKey*, *UserValue*, *PasswordKey* and *PasswordValue*. The assignment at the slot *LoginURI* is supposed to be a URI of a login page. The assignments for the pair of slots *UserKey* and *UserValue* are supposed to be a key-value pair for the user name, and the assignments for the other pair of slots *PasswordKey* and *PasswordValue* are

```
<form method=POST action=login>
<table><tr><td>User ID:</td>
  <td><input type=text name=user_id1 size=10 style='font-
family:monospace'></td>
  </tr><tr><td>Password:</td>
  <td><input type=password name=password1 size=10 style='font-
family:monospace'></td>
</tr></table>
<input type=Submit value=login name=B1>&nbsp;&nbsp;
<a href=register target=_parent>Register</a>
<input type=hidden name=url value=.>
</form>
```

Figure 5. The HTML code of the login page of the PKU OJ.

supposed to be a key-value pair for the password. TABLE III lists all the five slots of an L-frame together with their respective assignments for a link to the login page of the PKU OJ.

Attached to each L-frame is the same clicking procedure for completing the log-in action. The clicking procedure is composed of constructing an HTTP post method and executing the constructed method. Fig. 6 shows the algorithm of the clicking procedure attached to the link that points to the login page, where an HTTP post method is constructed in the first five lines and executed in the last line. All the five slots of an L-frame can find their names in the first three lines in the algorithm, which means that the assignment value at each slot of the L-frame will be passed to the HTTP post method during the clicking procedure. In the second line from the bottom in Fig. 5, we can see a *hidden* field [15] in the login page of PKU OJ. The key-value pair of the hidden field should also be passed to the HTTP post method. The method getHiddens(*LoginURI*, *PasswordKey*) in the fourth line of the algorithm is designed to extract key-value pairs of all the hidden fields between the pair of form tags that includes the assignment value at the slot *PasswordKey* on the page whose URI is the assignment value at the slot *LoginURI*. It uses the HTTP get method to fetch the page before it is able to extract the hidden fields. The extracted key-value pairs of hidden fields are added to the HTTP post method in the fifth line.

Notice that no assignment value at any slot appears in the clicking algorithm. This means that the same clicking procedure without any modification of its code is applicable to different assignments for the L-frame. For example, the same clicking procedure can be applied to the L-frame in TABLE IV for logging in to the UVa OJ.

TABLE III.
AN L-FRAME FOR A LINK TO THE PKU OJ LOGIN PAGE

Name	Marker	Assignment
LoginURI	*URI for login page*	*http://poj.org/login*
UserKey	*Key for user name*	*user_id1*
UserValue	*Value for user name*	*pkuuser*
PasswordKey	*Key for password*	*password1*
PasswordValue	*Value for password*	*pkupass*

```
Algorithm login.clicking()
 (1) post = new PostMethod(LoginURI);
 (2) post.addParameter(UserKey, UserValue);
 (3) post.addParameter(PasswordKey, PasswordValue);
 (4) hiddens = getHiddens(LoginURI, PasswordKey);
 (5) post.addParameter(hiddens);
 (6) client.executeMethod(post);
End login.clicking
```

Figure 6. The algorithm attached to the L-frame.

TABLE IV
AN L-FRAME FOR A LINK TO THE UVA OJ LOGIN PAGE

Name	Marker	Assignment
LoginURI	*URI for login page*	*http://uva.onlinejudge.org/index.php?option=com_comprofiler&task=login*
UserKey	*Key for user name*	*username*
UserValue	*Value for user name*	*uvauser*
PasswordKey	*Key for password*	*passwd*
PasswordValue	*Value for password*	*uvapass*

B. The Link to a Submission Page

A frame for a link to a submission page is called an *S-frame*. In order to submit a program (e.g., the S_{a+b} in Fig. 2) to an online judge (e.g., Timus OJ), one needs to enter on the submission page of the online judge the program together with the ID number (e.g., 1000) of the problem that the program is supposed to solve and the ID number (e.g., 10) of the language (e.g., C++) in which the program is written. In some online judge (e.g., Timus OJ), one may also need to enter the user ID number on the submission page. These kinds of information are then passed in the key-value pair format to the HTTP post method before the submission is actually sent to the remote online judge. Thus, an S-frame needs four pairs of slots for the four key-value pairs corresponding to the program, the problem ID number, the language ID number and the user ID number, respectively. TABLE V lists all the slots of an S-frame together with their respective assignments for a link to the submission page of the Timus OJ, where the assignment value at the slot *SubmitURI* is the URI http://acm.timus.ru/submit.aspx of the submission page of the Timus OJ, and the assignment value at the slot *LanguageValue* is the ID number 10 of the C++ language on the Timus OJ.

Attached to each S-frame is the same clicking procedure for completing the submission action. Like the one for login, the clicking procedure for submission is also composed of constructing an HTTP post method and executing the constructed method. Fig. 7 shows the algorithm of the clicking procedure attached to the link that points to the submission page, where an HTTP post method is constructed in the first seven lines and executed in the last line. As in the algorithm for login, all the ten slots of an S-frame can find their names in the algorithm for submission, which means that the assignment value at each slot of the S-frame will be

TABLE V.
AN S-FRAME WITH ITS SLOT ASSIGNMENTS FOR A LINK TO THE SUBMISSION PAGE OF THE TIMUS OJ

Name	Marker	Assignment
SubmiURI	*URI for submission page*	*http://acm.timus.ru/submit.aspx*
OptionString	*Optional part in URI*	*NULL*
LanguageKey	*Key for language*	*Language*
LanguageValue	*Value for language*	10
ProblemKey	*Key for problem id*	*ProblemNum*
ProblemValue	*Value for problem id*	1000
SourceKey	*Key for source code*	*Source*
SourceValue	*Value for source code*	S_{a+b} in Figure 2
UserKey	*Key for user id*	*JudgeID*
UserValue	*Value for user id*	*123702WX*

```
Algorithm submit.clicking()
 (1) post = new PostMethod(SubmitURI + OptionString);
 (2) post.addParameter(LanguageKey, LanguageValue);
 (3) post.addParameter(ProblemKey, ProblemValue);
 (4) post.addParameter(SourceKey, SourceValue);
 (5) post.addParameter(UserKey, UserValue) if UserKey ≠ NULL;
 (6) hiddens = getHiddens(SubmitURI, ProblemKey);
 (7) post.addParameters(hiddens);
 (8) client.executeMethod(post);
End submit.clicking
```

Figure 7. The algorithm attached to the S-frame.

passed to the HTTP post method during the clicking procedure. Note that there is a condition in the fifth line to check if the slot *UserKey* has a non-null value before the assignments for the pair of slots *UserKey* and *UserValue* is added to the HTTP post method, because some online judge (e.g., PKU OJ or UVa OJ) do not require the user ID number for submission of programs (see TABLE VI). The method getHiddens(*SubmitURI*, *ProblemKey*) in the sixth line of the algorithm extracts key-value pairs of all the hidden fields between the pair of form tags that includes the assignment value at the slot *ProblemKey* on the page whose URI is the assignment value at the slot *SubmitURI*. In some online judge (e.g., UVa OJ), the HTTP post method for the submission needs an optional string appended to the URI of the submission page. For this reason, we design a slot *OptionString* to keep this optional string for appending it to the URI from the slot *SubmitURI* in the first line of the algorithm.

Also notice that no assignment value at any slot appears in the clicking algorithm for submission. This means that the same clicking procedure without any modification of its code is applicable to different assignments (e.g., in TABLE VI) for the S-frame.

TABLE VI.
ASSIGNMENTS AT EVERY SLOT OF THE S-FRAME FOR LINKS TO RESPECTIVE SUBMISSION PAGES OF THE PKU OJ AND THE UVA OJ

Name	Assignment (PKU)	Assignment (UVa)
SubmiURI	*http://poj.org/submit*	*http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=25*
OptionString	*NULL*	*&page=save_submission*
LanguageKey	*language*	*language*
LanguageValue	4	3
Problemkey	*problem_id*	*localid*
ProblemValue	1000	113
SourceKey	*source*	*code*
SourceValue	S_{a+b} in Figure 2	S_{power} in Figure 3
UserKey	*NULL*	*NULL*
UserValue	*NULL*	*NULL*

C. The Link to a Query Page

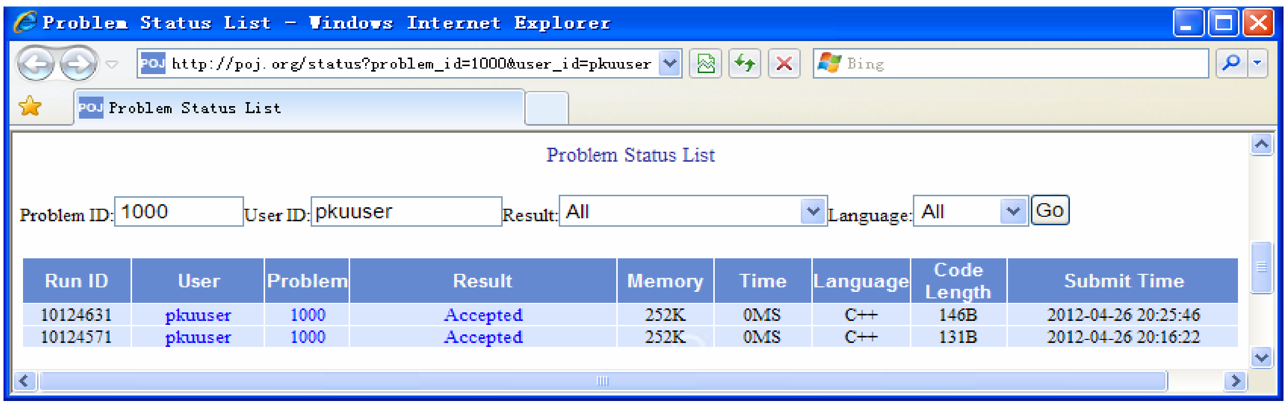
After submitting a program to an online judge, one can find the verdict of the program on the query page of the online judge. Fig. 8 shows the query page of the PKU OJ. To query the verdict, one can enter the ID number (e.g., 1000) of the problem that the program is supposed to solve into the text box labeled “Problem ID” on the query page, enter his or her user name (e.g., *pkuuser*) into the text box labeled “User ID”, and finally click the button labeled “Go” on the rightmost below the title “Problem Status List”. The result of the query is shown in the table, where the fourth column of the first line under the heading line displays the verdict (e.g., *Accepted*) of the last submission (see the number 10124631 under the heading “Run ID”) for the concerned problem. In the address bar, we can find that the URI of the query page is http://poj.org/status. This URI has a query string [16] “problem_id=1000&user_id=pkuuser” after the question mark ‘?’, which consists of two key-value pairs for the problem ID number and the user name, respectively.

Thus, a frame for a link to a query page, named as *Q-frame*, needs two pairs of slots for the two key-value pairs corresponding to the problem ID number and the user name, respectively. TABLE VII lists all the slots of a Q-frame together with their respective assignments for a link to the query page of the PKU OJ, where the assignment value 1000 at the slot *ProblemValue* is the ID number of the problem *A* + *B Problem* on the PKU OJ, and the assignment values 4, 5 and 6 at the slots *Verdict*, *Memory* and *Time*, respectively, indicate the fourth, fifth and sixth columns of the table in which the verdicts and code qualities are displayed on the query page as shown in Fig. 8.

Attached to each Q-frame is the same clicking procedure for completing the query action. In comparison with those for login and submission, the clicking procedure for query is required to return the verdict of the last submission after constructing an HTTP get method and executing the constructed method. Fig. 9 shows the algorithm of the clicking procedure attached to the link

TABLE VII.
A Q-FRAME FOR A LINK TO THE QUERY PAGE OF THE PKU OJ

Name	Marker	Assignment
QueryURI	*URI for query page*	*http://poj.org/status*
Verdict	*Column number for the verdict*	4
Memory	*Column number for the memory*	5
Time	*Column number for the time*	6
UserKey	*Key for user id in query string*	*user_id*
UserValue	*Value for user id in query string*	*pkuuser*
ProblemKey	*Key for problem id in query string*	*problem_id*
ProblemValue	*Value for problem id in query string*	1000


Problem Status List - Windows Internet Explorer
http://poj.org/status?problem_id=1000&user_id=pkuuser
Bing
Problem Status List

Problem Status List

Problem ID: 1000 User ID: pkuuser Result: All Language: All Go

Run ID	User	Problem	Result	Memory	Time	Language	Code Length	Submit Time
10124631	pkuuser	1000	Accepted	252K	0MS	C++	146B	2012-04-26 20:25:46
10124571	pkuuser	1000	Accepted	252K	0MS	C++	131B	2012-04-26 20:16:22

Figure 8. A query page of the PKU OJ.

```
Algorithm query.clicking()
  (1) get = new GetMethod(QueryURI);
  (2) queryString = get.getQueryString();
  (3) userString = getQueryString(UserKey, UserValue);
  (4) problemString = getQueryString(ProblemKey, ProblemValue);
  (5) queryString = queryString+"&"+userString+"&"+problemString;
  (6) get.setQueryString(queryString);
  (7) client.executeMethod(get);
  (8) queryPage = get.getResponseBodyAsString();
  (9) result = getResult(queryPage, Verdict, Memory, Time);
  (10) return result;
End query.clicking
```

Figure 9. The algorithm attached to the Q-frame.

that points to the query page, where an HTTP get method is constructed in the first six lines and executed in the seventh line. The two key-value pairs for the user name and the problem ID number are encoded in the third and fourth lines, respectively, and added to the query string in the fifth line. In the sixth line, the query string is passed to the HTTP get method. The last three lines are designed for obtaining the verdict from the query page. The method get.getResponseBodyAsString() returns the HTML source code of the query page, which is assigned to the variable *queryPage* in the eighth line. In the ninth line, the method getResult(*queryPage*, *Verdict*, *Memory*, *Time*) extracts as the result the top values in the columns, indicated by the assignment values at the slots *Verdict*, *Memory* and *Time* respectively, from the query page contained in the variable *queryPage* (e.g., the word *Accepted* at the fourth column, the value *252K* at the fifth column, and the value *0MS* at the sixth column of the first line in the table on the query page of the PKU OJ, see Fig. 8). The result is returned in the last line of the algorithm.

Again, notice that no assignment value at any slot appears in the clicking algorithm for query. This means that the same clicking procedure without any modification of its code is applicable to different assignments (e.g., in TABLE VIII) for the Q-frame. Comparing TABLE VIII with TABLE VII, we can find that the assignment value at the slot *QueryURI* for the Timus OJ has already had a query string. i.e., *space=1*. This query string is extracted in the second line in Fig. 9 so that other key-value pairs can be appended to it easily in the fifth line.

TABLE VIII.
ASSIGNMENTS AT EVERY SLOT OF THE Q-FRAME FOR LINKS TO RESPECTIVE QUERY PAGES OF THE TIMUS OJ AND THE UVA OJ

Name	Assignment(Timus)	Assignment(UVa)
QueryURI	*http://acm.timus.ru/status.aspx?space=1*	*http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=9*
Verdict	6	4
Memory	9	*NULL*
Time	8	6
UserKey	*author*	*NULL*
UserValue	*123702WX*	*NULL*
ProblemKey	*num*	*NULL*
ProblemValue	1000	*NULL*

IV. HOME FRAMES FOR ONLINE JUDGES

Just as a hyperlink is embedded in a webpage, a framed link is embedded in a frame that describes a remote online judge. To distinguish the frame for an online judge from the frame for a framed link, we call the former the home frame of the online judge, and the latter a sub-frame of the home frame. Each online judge can be represented by its home frame. The home frame encodes enough knowledge for interaction with its respective online judge. All the home frames have the same structure, so that the same API could be applied to them without any modification of its implementation. However, different online judges have different assignments for their home frames, which make it possible for the same API to provide a local machine with resources on different remote online judges. When a local machine calls a log-in operation api.login(*n*), the local API will select the home frame of the remote online judge whose identifier is the value of the parameter *n*. By using the framed links embedded in the home frame, the local API enables the local machine to interact with the remote online judge that the parameter *n* identifies.

A. Structure of Representation

A home frame consists of six slots. Three of them can be filled with a corresponding framed link. TABLE IX lists all the six slots of the home frame together with their respective assignments for the PKU OJ. The three slots *LoginFrame*, *SubmitFrame* and *QueryFrame* are assigned with three corresponding sub-frames, i.e., an *L-frame* for a link to a login page, an *S-frame* for a link to a submission page and a *Q-frame* for a link to a query page at the website of the PKU OJ. The assignment value *PKU* at the slot *SiteName* is the identifier of the PKU OJ. The slot *Compliers* is assigned with a list of pairs, each of which consists of a name and an ID number of a possible compiler on the PKU OJ. The assignment of the slot *Verdicts* is a list of terms, each of which indicates a possible result of the judgment that the PKU OJ may make.

TABLE X shows assignments at every slot for respective home frames of the Timus OJ and the UVa OJ. Comparing TABLE X with TABLE IX, we can find that each slot name of the home frame keeps unchanged, whereas the slot assignment value may differ from one online judge to another. The slot *LoginFrame* of the home frame for the Timus OJ is assigned with NULL, which means that there is no need to log in to the Timus OJ.

TABLE IX.
A HOME FRAME WITH AN ASSIGNMENT FOR PKU OJ

Name	Marker	Assignment
SiteName	*Online judge name*	*PKU*
LoginFrame	*Frame for login*	An L-frame as in TABLE III
SubmitFrame	*Frame for submission*	An S-frame as in TABLE VI
QueryFrame	*Frame for query*	A Q-frame as in TABLE VII
Compliers	*List of name-number pairs for compilers*	{(*G++*, 0), (*GCC*, 1), (*Java*, 2), (*Pascal*, 3), (*C++*, 4), (*C*, 5), (*Fortran*, 6)}
Verdicts	*List of verdict terms*	{*Wrong Answer*, *Compile Error*, *Accepted*, *Runtime Error*, *Time Limit Exceeded*, *Presentation Error*, *Output Limit Exceeded*, *Memory Limit Exceeded*}

TABLE X.
RESPECTIVE ASSIGNMENTS FOR HOME FRAMES OF TIMUS OJ AND UVA OJ

Name	Assignment (Timus)	Assignment (Uva)
SiteName	*Timus*	*UVa*
LoginFrame	*NULL*	A L-frame as in TABLE IV
SubmitFrame	An S-frame in Table 5	An S-frame as in TABLE VI
QueryFrame	A Q-frame in Table 8	A Q-frame as in TABLE VIII
Compliers	{(*Java*, 7), (*Pascal*, 3), (*C*++, 10), (*C*, 9), (*C#*, 11)}	{(*C*, 1), (*Java*, 2), (*C*++, 3), (*Pascal*, 4)}
Verdicts	{*Wrong answer*, *Compilation error*, *Accepted*, *Runtime error*, *Time limit exceeded*, *Presentation error*, *Output limit exceeded*, *Memory limit exceeded*}	{*Wrong answer, Compilation error, Accepted, Runtime error, Time limit exceeded*, *Presentation error*, *Output limit exceeded*, *Memory limit exceeded*, *Can't be judged*, *In queue*}

B. Default Assignments

In the opinion of Marvin Minsky, frames are never stored in long-term memory with unassigned terminal values. Instead, frames are stored with weakly-bound default assignments at every slot [8]. For this reason, each home frame together with its sub-frames is stored with default assignments at every slot. The default assignments are attached loosely to their slots, so that they can be easily displaced by new items that fit better the current situation. For example, the assignment at the slot *SourceValue* will be displaced by a new version of the submitted solution that is updated at the current time. Moreover, the assignments at the slots *LogginURI*, *SubmitURI* and *QueryURI* can be easily displaced by new URIs of respective pages when the online judge changes their URIs (actually, the UVa OJ and the PKU OJ have changed these URIs). This dynamic feature makes frames different from web pages, whose contents are usually unchanged.

We use the Extensible Markup Language (XML [17, 18]) to describe home frames together with default assignments at every slot. Fig. 10 shows an XML version of the home frame for the PKU OJ. Each slot is represented by a tag pair, which has the same name as the slot. Between the opening and closing tags is the respective default assignment value for the slot. For example, the slot *SiteName* is represented by a pair of the opening tag <SiteName> and the closing tag </SiteName>, between which is the respective default assignment value *PKU*. The tag pairs for respective slots in the L-frame are embedded between the pair of tags <LoginFrame> and </LoginFrame>, which means that the L-frame is embedded in the home frame as the assignment of the slot *LoginFrame*. The S-frame and the Q-frame are embedded in the home frame in the same way as the L-frame. Each of slots including those in sub-frames is filled with its respective default assignment value in the XML version of the home frame. All home frames in the XML version forms a frame-system.

```
<?xml version="1.0" encoding="UTF-8"?>
<website>
<SiteName>PKU</SiteName>
<LoginFrame>
  <LoginURI>http://poj.org/login</LoginURI>
  <UserKey>user_id1</UserKey>
  <UserValue>pkuuser</UserValue>
  <PasswordKey>password1</PasswordKey>
  <PasswordValue>pkupass</PasswordValue>
</LoginFrame>
<SubmitFrame>
  <SubmitURI>http://poj.org/submit</SubmitURI>
  <OptionString></OptionString>
  <UserKey></UserKey>
  <UserValue></UserValue>
  <LanguageKey>language</LanguageKey>
  <LanguageValue>4</LanguageValue>
  <ProblemKey>problem_id</ProblemKey>
  <ProblemValue>1000</ProblemValue>
  <SourceKey>source</SourceKey>
  <SourceValue> #include&lt;iostream&gt; using namespace std;
   int main() {int a,b;while(cin&gt;&gt;a&gt;&gt;b)
   {cout&lt;&lt;a+b&lt;&lt;endl;} return 0; }</SourceValue>
</SubmitFrame>
<QueryFrame>
  <QueryURI>http://poj.org/status</QueryURI>
  <Verdict>4</ Verdict >
  <Memory>5</Memory>
  <Time>6</Time>
  <UserKey>user_id</UserKey>
  <UserValue>pkuuser</UserValue>
  <ProblemKey>problem_id</ProblemKey>
  <ProblemValue>1000</ProblemValue>
</QueryFrame>
<Compilers>
   <name>G++</name><value>0</value><name>GCC</name><valu
   e>1</value><name>Java</name><value>2</value><name>Pascal
   </name><value>3</value><name>C++</name><value>4</value>
   <name>C</name><value>5</value><name>Fortran</name><valu
   e>6</value>
</Compilers>
<Verdicts>
   <name>Wrong Answer</name><name>Compile
   Error</name><name> Accepted</name><name>Runtime
   Error</name><name>Time Limit
   Exceeded</name><name>Presentation Error</name>
   <name>Output Limit Exceeded</name><name>Memory Limit
   Exceeded</name>
</Verdicts >
</website>
```

Figure 10. An XML version of the home frame for PKU OJ.

C. Attached Procedures

Attached to each home frame are three procedures as the three API operations api.login(*n*), api.submit(*s*, *p*, *l*) and api.query(), respectively. Fig. 11 shows the algorithm of the procedure api.login(*n*). The method findOJFrame(*n*) in the first line of the algorithm is designed to select from the frame-system the home frame whose slot *SiteName* has the same value as the parameter *n*. The variable *OJFrame* keeps the selected frame as the current home frame. The slot assignments in the current home frame are from the XML file of the selected frame. For example, the procedure api.login(*PKU*) will select the home frame of the PKU OJ as the current home frame, leading to that the slots of the current home frame are filled with default assignments described in Fig. 10. In the second line of the algorithm, it tries to fetch from the slot *LoginFrame* the L-frame embedded in the current home frame. The fetched L-frame is passed to the variable *link*, which represents a framed link to a login page if it exists. This framed link is "clicked" in the last line of the algorithm by calling the procedure link.clicking(), which is attached to the fetched L-frame (i.e., the procedure login.clicking()

```
Algorithm api.login(n)
 (1) OJFrame = findOJFrame(n);
 (2) link = OJFrame.LoginFrame;
 (3) link.clicking() if link ≠ NULL;
End api.login
```

Figure 11. The algorithm of the procedure api.login(*n*).

in Fig. 6). In the case of the PKU OJ, the clicking procedure will use the knowledge encoded in the L-frame as in TABLE III to submit the corresponding user name and password to the PKU OJ for completing the log-in action. The same thing is true for the procedure api.login(*UVa*) to log in to the UVa OJ. However, the procedure api.login(*Timus*) runs differently because the slot *LoginFrame* has a null value in the home frame of the Timus OJ (see TABLE X). The effect of the procedure api.login(*Timus*) is to select the home frame of the Timus OJ as the current home frame, enabling its following procedures api.submit(s, p, l) and api.query() to access to the Timus OJ.

The algorithm of the procedure api.submit(s, p, l) is shown in Fig. 12. In the first line of the algorithm, it fetches from the slot *SubmitFrame* the S-frame embedded in the current home frame. The default assignments at the three slots *SourceValue*, *ProblemValue* and *LanguageValue* in the fetched S-frame are displaced by the values of the three parameters s, p and l in the second, third and fourth lines, respectively. The fetched S-frame is passed to the variable *link*, which represents a framed link to a submission page. In the fifth line, this framed link is "clicked" by calling the procedure link.clicking(), which is attached to the fetched S-frame (i.e., the procedure submit.clicking() in Fig. 7). After an operation api.login(*Timus*), for example, the procedure api.submit(S_{a+b}, 1000, 10) will substitute the three parameter values S_{a+b}, 1000 and 10 for the default assignments at the three slots *SourceValue*, *ProblemValue* and *LanguageValue* of the S-frame embedded in the home frame of the Timus OJ, respectively. The clicking procedure in the fifth line of the algorithm employs the knowledge encoded in the S-frame as in TABLE V, to submit to the Timus OJ the source code S_{a+b} together with the problem ID number 1000 and the language ID number 4 as a solution to the problem *A + B Problem*. In the last line of the algorithm, the value of the parameter p (e.g., the problem ID number 1000) substitutes for the default assignment at the slot *ProblemValue* of the Q-frame embedded in the current home frame, so that the procedure api.query() following the procedure api.submit(s, p, l) is able to know from the Q-frame the ID number of the problem that the last submitted program is supposed to solve.

Fig. 13 shows the algorithm of the procedure api.query(). In the first line of the algorithm, it fetches from the slot *QueryFrame* the Q-frame embedded in the current home frame. The fetched Q-frame is passed to the variable *link*, which represents a framed link to a query page. In the second line, this framed link is "clicked" by calling the procedure link.clicking(), which is attached to the fetched Q-frame (i.e., the procedure query.clicking() in Fig. 9). The clicking procedure will use the knowledge encoded in the Q-frame (e.g., in TABLE VII) to send to the remote online judge a request for the pointed query page (e.g., in Fig. 8) that contains the test results of submitted programs. Furthermore, the clicking procedure will extract from the query page the verdict (e.g., *Accepted*) and the code quality information (e.g., *252K* for memory and *0MS* for run time) of the last submission. The extracted verdict and code quality information are returned to the caller (e.g., a local system) in the last line of the algorithm.

```
Algorithm api.submit(s, p, l)
 // OJFrame keeps the current home frame
 (1) link = OJFrame.SubmitFrame;
 (2) link.SourceValue = s;
 (3) link.ProblemValue = p;
 (4) link.LanguageValue = l;
 (5) link.clicking();
 (6) OJFrame.QueryFrame.ProblemValue = p;
End api.submit
```

Figure 12. The algorithm of the procedure api.submit(s, q, l).

```
Algorithm api.query()
 // OJFrame keeps the current home frame
 (1) link = OJFrame.QueryFrame;
 (2) result = link.clicking();
 (3) return result;
End api.query
```

Figure 13. The algorithm of the procedure api.query().

V. Application

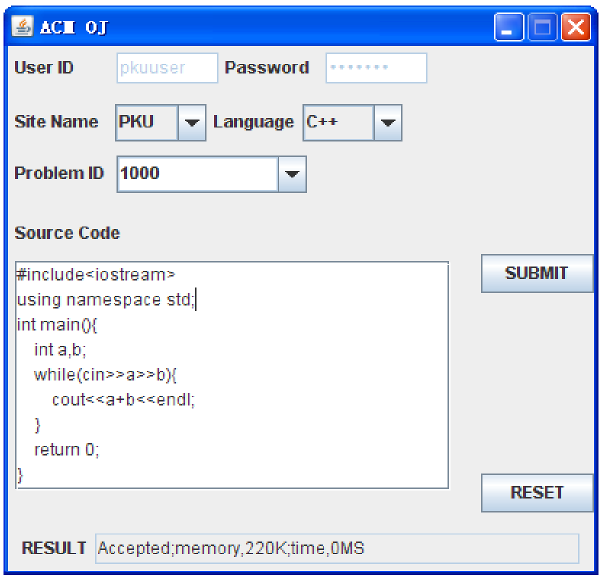
We have implemented the local API for remote online judges by using the Java language. The home frame of each online judge is saved as an XML file into a local directory. The API will load from the local directory all the XML files into its frame-system. When necessary, the API will fetch the default assignment at the slot *SiteName* of each home frame in its frame-system to produce a name list of online judges whose XML files are in the local directory.

On a basis of the implemented API, we have developed a prototype of a single platform for operating multiple online judges. The platform enables a human user to visit different online judges in the same way. Furthermore, we have integrated several remote online judges into a local system, which include the PKU OJ, the UVa OJ and the Timus OJ.

A. A Single Platform for Multiple Online Judges

Fig. 14 shows a snapshot of the single operating platform for multiple online judges. When it starts up, the platform will ask the API for a name list of online judges that the API is able to interact with. The name list is passed to the drop-down list labeled "Site Name", from which one can select an online judge by its name. Each time an online judge is selected, the default assignment list at the slot *Compilers* of the respective home frame will be passed to the drop-down list labeled "Language", so that one can select from this drop-down list a language in which his or her program is written. In addition, the default assignment values at slots *SourceValue* and *ProblemValue* will be displayed in the text box labeled "Source Code" and the editable drop-down list labeled "Problem ID", respectively.

To submit a program (e.g., S_{a+b} in Fig. 2) to an online judge (e.g., PKU OJ), one can select the online judge name (e.g., *PKU*) from the drop-down list labeled "Site Name", copy the program source code into the text box labeled "Source Code", select a language name (e.g., C++)



```
#include<iostream>
using namespace std;
int main(){
   int a,b;
   while(cin>>a>>b){
      cout<<a+b<<endl;
   }
   return 0;
}
```

RESULT Accepted;memory,220K;time,0MS

Figure 14. A snapshot of the single operating platform for multiple online judges.

from the drop-down list labeled "Language", enter the problem ID number (e.g., 1000) into the editable drop-down list labeled "Problem ID", and finally click the button labeled "SUBMIT".

After the submit button is clicked, the selected online judge name (e.g., *PKU*) is passed from the drop-down list labeled "Site Name" to the parameter *n* of the log-in operation api.login(*n*). The copied source code (e.g., S_{a+b}) is passed to the parameter *s* of the operation api.submit(*s*, *p*, *l*) from the text box labeled "Source Code", the entered problem ID number (e.g., 1000) to the parameter *p* from the editable drop-down list labeled "Problem ID", and the ID number (e.g., 4) of the selected language (e.g., C++) to the parameter *l* from the drop-down list labeled "Language". The platform then calls the operation api.login(*n*) to log in to the selected online judge (e.g., the PKU OJ), followed by the operation api.submit(*s*, *p*, *l*) to submit the source code (e.g., $s = S_{a+b}$) together with the entered problem ID number (e.g., $p = 1000$) and the ID number (e.g., $l = 4$) of the selected language (e.g., C++). Later after the two operations above, the platform will call the operation api.query() to send to the online judge a request for the verdict of the submitted program. The returned verdict together with the code quality information will be shown on the text box labeled "RESULT".

B. *Integration of Online Judges into a Local System*

We have integrated several remote online judges into a local system. The local system, called Donghua University Online Judge (DHUOJ[19]), was successfully applied to the online preliminary contest and the onsite contest of the 2009 ACM/ICPC Asia Regional Shanghai Site, in which over a thousand teams participated. Teachers can put their exercise papers on DHUOJ for students to practice programming. However, it usually takes much time to produce programming problems for the exercise papers. To save the time, teachers may download their selected problem statements from remote online judges, but it is difficult to find the test data set for the downloaded problems because the test data are not published on the remote online judges. For this reason, we applied the framed link technology to integration of remote online judges for the DHUOJ to share these computing resources together with their pre-designed test data.

TABLE XI shows a list of problems in an exercise paper on the integrated system. The first column of the table lists problem names in the exercise paper. The second column and the fourth column list the problem ID numbers in DHUOJ and PKU OJ, respectively. In the third column are the problem titles. With this table, the integrated system can forward the programs that the DHUOJ receives to the PKU OJ that the framed link points to. Before forwarding the programs to the remote online judge, the integrated system will transform the local language identifiers into the remote language identifiers with a mapping table as in TABLE XII. Similarly, the integrated system will transform the verdict received from a remote online judge into a local version with a verdict mapping table.

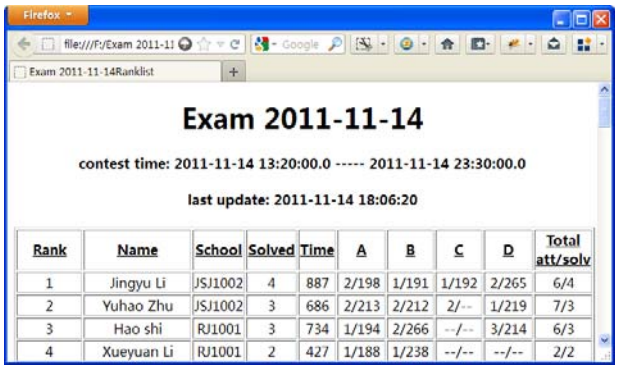
The advantage of this approach is that teachers can easily obtain the information about the achievements of their students in the programming practice situation. For example, it is easy to find that the top student has solved all the four problems *A*, *B*, *C* and *D* from the rank list as shown in Fig. 15. Furthermore, it is easy for the teacher to obtain from the local system the programs that the students have submitted for some reason (e.g., to inspect the programming style). However, it would be difficult to

TABLE XI.
A PROBLEM SET FOR ONE CLASS TEST

Local Name	Local ID	Problem Title	Remote ID	Remote OJ
A	1001	Deli Deli	3366	PKU
B	1002	Card Game Cheater	2062	PKU
C	1003	Building for UN	3566	PKU
D	1004	Java vs C++	3157	PKU

TABLE XII.
A MAPPING TABLE FROM THE DHUOJ TO THE PKU OJ

DHUOJ		PKU OJ	
Language	Language ID	Language	Language ID
CPP	0	G++	0
C	1	GCC	1
JAVA	2	Java	2
PASCAL	3	Pascal	3



Exam 2011-11-14

contest time: 2011-11-14 13:20:00.0 ----- 2011-11-14 23:30:00.0

last update: 2011-11-14 18:06:20

Rank	Name	School	Solved	Time	A	B	C	D	Total att/solv
1	Jingyu Li	JSJ1002	4	887	2/198	1/191	1/192	2/265	6/4
2	Yuhao Zhu	JSJ1002	3	686	2/213	2/212	2/--	1/219	7/3
3	Hao shi	RJ1001	3	734	1/194	2/266	--/--	3/214	6/3
4	Xueyuan Li	RJ1001	2	427	1/188	1/238	--/--	--/--	2/2

Figure 15. The status page of the DHUOJ.

get the rank list without integration of remote online judges into a local system. It would be more difficult for the teacher to know how much time that each student spends on their solved problems.

VI. Conclusion

Remote online judges together with their pre-designed test data are valuable resources for teaching and learning. However, it is difficult for a local tutoring system to share these resources, because online judges are designed for human users only and their pre-designed test data are not published. To address such issue, this paper proposed a novel link, called framed link, which consists of a frame that encodes the knowledge about how to interact with the remote online judge that the link points to. Each remote online judge is represented by its home frame. Embedded in the home frame are three framed links that point to a login page, a submission page and a query page on the remote online judge, respectively. Moreover, a local API was designed for a local system to "click" the framed links for its interaction with the remote online judge. With this framed link technology, we developed a prototype of a single platform for operating multiple online judges in the same way. Furthermore, we integrated several remote online judges into a local system, so that the local system is able to share these computing resources including their pre-designed test data. We believe that the framed link can be applied to automatic interaction with pages of other computing resources on the web to share their computation including their internal data.

Acknowledgment

This research is supported by the National Natural Science Foundation of China (NSFC) under Grant No. 60973121.

References

[1] Rudi Studer, "The Semantic Web. Enabling innovative approaches for handling information and services", Information Services & Use 29 (2009) 73–80.

[2] Andy Kurnia, Andrew Lim, Brenda Cheang, "Online Judge", Computers & Education 36 (2001) 299–315.

[3] Elena Verdúa, Luisa M. Reguerasa, María J. Verdúa, José P. Lealb, Juan P. de Castroa, Ricardo Queirósc, "A distributed system for learning programming on-line", Computers & Education,Volume 58, Issue 1, January 2012.

[4] Xiaoyu Du, Chao Yi, Yu Wei, Su Feng, Zhi Gong, "Design of Automata Online Judge", Information Engineering and Computer Science (ICIECS), 2010.

[5] Miguel A. REVILLA, Shahriar MANZOOR, Rujia LIU, "Competitive Learning in Informatics: The UVa Online Judge Experience", Olympiads in Informatics, 2008.

[6] Li Wen-xin, Guo Wei, "Peking University Online Judge and Its Applications", Journal of Changchun Post and Telecommunication Institute, 2005-S2.

[7] Georgouli K, Guerreiro P, "Integrating an Automatic Judge into an Open Source LMS", International Journal on E-Learning, 10(1), 27-42. Chesapeake, VA: AACE.

[8] Marvin Minsky, "A Framework for Representing Knowledge", MIT-AI Laboratory Memo 306, June, 1974.

[9] Chouyin Hsu, "Development of Semantic-CBR Framework for Virtual Enterprises in Project Management", Journal of Computers, Vol 6, No 3 (2011), 434-440, Mar 2011, doi:10.4304/jcp.6.3.434-440.

[10] Berners-Lee Tim, Connolly Daniel, Hypertext Markup Language (HTML) Internet Draft version 1.1, June 1993.

[11] Mealling M. (Ed.), R. Denenberg (Ed.), "Report from the Joint W3C/IETF URI Planning Interest Group: Uniform Resource Identifiers (URIs), URLs, and Uniform Resource Names (URNs): Clarifications and Recommendations", RFC 3305, August 2002.

[12] Yudong Yang, HongJiang Zhang, "HTML Page Analysis Based on Visual Cues", Document Analysis and Recognition, 2001.

[13] Berners-Lee Tim, "HyperText Transfer Protocol", World Wide Web Consortium, Retrieved 31 August 2010.

[14] Bruce A. Mah, "An Empirical Model of HTTP Network Traffic", Computer and Communications Societies, Proceedings IEEE. 1997.

[15] Berners-Lee Tim, Connolly Dan, "Hypertext Markup Language - 2.0", World Wide Web Consortium. Retrieved 15 January 2011.

[16] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", RFC 3986, January 2005.

[17] Maler, E., Yergeau, F., Paoli, J., Bray, T., C. Sperberg-McQueen, "Extensible Markup Language (XML) 1.0 (Third Edition)", World Wide Web Consortium FirstEdition REC-xml-20040204, February 2004.

[18] Christopher League and Kenjone Eng, "Schema-Based Compression of XML Data with Relax NG", Journal of Computers, 2007.

[19] Guojin Zhu, Yang Guan, "Communities of Autonomous Units for Distributed Online Judge Systems", pressed by the 2010 Second International Conference on Future Computer and Communication, IEEE Transel. Shanghai, China, 2010(9).

Guojin Zhu is an associate professor at the Department of Computer Science, Donghua University (DHU), Shanghai, China. He received his M.S. and Ph.D. degrees from DHU in 1991 and 2007, respectively. He was a visiting scholar at the Department of Computer Science and Engineering, Michigan State University, East Lansing, Michigan, USA from November 2007 to November 2008. His current research interests include semantic web, knowledge discovery, and neural computing.

Yefeng Chen is a graduate student of Computer Software and Theory at Donghua University. He was born in Jiangsu province, P. R. China in 1985, and received the bachelor degree of Information and Calculation Science in Nanjing University of Technology in 2007. His current main research interest is semantic web.