Build Interactive Mathematics Web Services Using Ajax Technology and Automated Reasoning Tools

Zhenbing Zeng
zbzeng@sei.ecnu.edu.cn
Software Engineering Institute
East China Normal University
200062 Shanghai, China

Hongguang Fu
fu_hongguang@hotmail.com
Laboratory for Automated Reasoning and Programming
Chengdu Institute of Computer Applications, Chinese Academy of Sciences
610041 Chengdu, China

Abstract: In this paper we show a method to build mathematics web services through integrating the newly developed JavaScript technology Ajax with automated reasoning tools like the symbolic algebra and dynamic geometry. We investigate some typical interactive learning scenarios and propose a Perl format to describe the process. The scripts of this format can be automatically transformed to MathML files. Using the JavaScript commands embedded in the MathML files the Ajax engine enables the users to interact with server. The JavaScript also communicates with the automated reasoning programs in server and client machines.

1. Introduction

Recently, a new pattern of computer-aided learning has been developed by applying web services technologies. Using XML, SOAP, WSDL and UDDI open standards, web services provide a standard way to integrate Web-based applications over an Internet protocol backbone. In Web services, XML is used to tag the data, SOAP (Simple Object Access Protocol) is used to transfer the data, WSDL (Web Services Description Language) is used for describing the services available and UDDI (Universal Description, Discovery and Integration) is used for listing what services are available. One successful example of computer-aided learning in mathematics is CALM [1], a project developed in the Department of Mathematics at Heriot-Watt University in Edinburgh. The current system CUE of CALM project includes a flexible assessment engine and is capable of delivering test on a personal computer or over the web. Another example of mathematics web service is a 24-month project MONET [2, 3] done in the Department of Computer Sciences at Bath University. The project investigated the mechanisms for describing a mathematical web service which on the one hand provide sufficient detail to allow for automatic service discovery, while on the other hand are not too complicated for practical use. MONET uses OpenMath to represent mathematical problem ontology and uses Description Logic and OWL (Web Ontology Language) in mathematical service matching.

As it is in the traditional process of mathematics education, conceptual recognition and problem solving are the cores of any web-based mathematics learning system. In many mathematics web services the symbolic computation programs (like Maple and Mathematica) play the role of

teachers. For example, Maple is used in CATT programme [4], iSchloar 2004 [5], MapleTA [6], MONET to create exercises and answers. SureMath [7] is a symbolic algebra program specifically designed for solving word problems. R. Milner proposed an object model for creating mathematical content on web by integrating symbolic computation programs and dynamic geometry software (like Cinderella and Geometer's Sketchpad) in [8]. Since late of 1990, some packages with symbolic computation, dynamics geometry and automated theorem proving modules have been developed and used to provide mechanical solutions and certain abilities of human-machine interaction in mathematics education software (e.g., Geometry Expert [9], Super Sketchpad [10], and MathXP [11]). We call this kind software "automated reasoning tools" in this paper. It is worth to indicate that various non-commercial freeware are available for constructing mathematics web services now.

The structure shown in Fig.1 illustrates a typical structure of web-based mathematics teaching and learning systems (see [12, 13, 14]). In this model, teachers write course materials (examples, glossary of definitions and additional course notes) in LaTeX or MathML format. The student requests and answers are sent by CGI, applets or JavaScript. The automated reasoning tools (AR tools) like Maple, NAG are invoked through middleware like CORBA, DCOM or Java Remote Method Invocation.

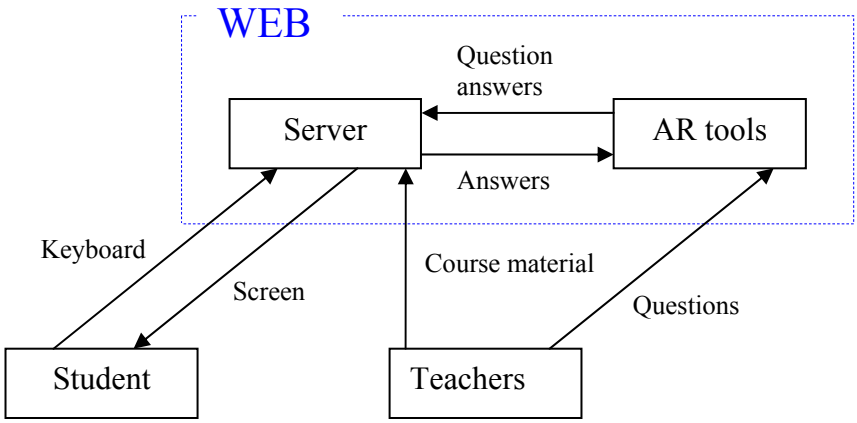

Figure 1. The model of web-based mathematics learning systems, AR tools stands for automated reasoning tools.

There are two primary drawbacks to current Web service models for mathematics. The first is the lack of a description language which is suitable for human-machine interactive operation and is also easy for mathematics teachers to use.

The second drawback is that the most web page of mathematics contents available now are prepared using the traditional form, and the traditional web pages does not support human-machine interaction in a "user-friendly" manner, as the requests to the server for more information are typically answered by the web server delivering a new page and the user has to wait until that page has loaded although the new page contains many of the same element in many case. A change has happened to this traditional model recently with using of a new technology called Ajax. Ajax (Asynchronous JavaScript And XML) allows navigating web pages while the page itself - quietly and unobtrusively - sends requests to the server for more data, and can use that data to update the user interface without the user having to wait for a new page or a page refresh. It is apparent the mathematics web service built with Ajax technology will have more characteristics in aspect of

human-machine interaction.

In this paper we propose a method to build interactive mathematics web service by integrating of Ajax technology and automated reasoning tools. In Section 2 we observe a typical use case scenario of web-based mathematics teaching and learning, and coming up with requirement for building mathematics web service. In Section 3 we introduce a Perl script format for describing the human-machine interaction. In Section 4 we will discuss the frame for building mathematics web service with Ajax, symbolic computation and dynamic geometry.

2. A Typical Use Case Scenario and Requirements for Mathematics Web Services

We investigate the general requirements of interactive mathematics web services by analyzing the following typical use case scenario.

In the use case, we're going to follow a scenario where a grade 7 student [the user] learns to formulate and solve equation from an application problem.

`> user:` selects the topic and a specific problem from the GUI (Graphic User Interface) of a web-based learning system.

`> machine:` displays the selected exercise problem stored in the server, with the widgets (in blue colour) for user to give instruction for browsing and selecting.

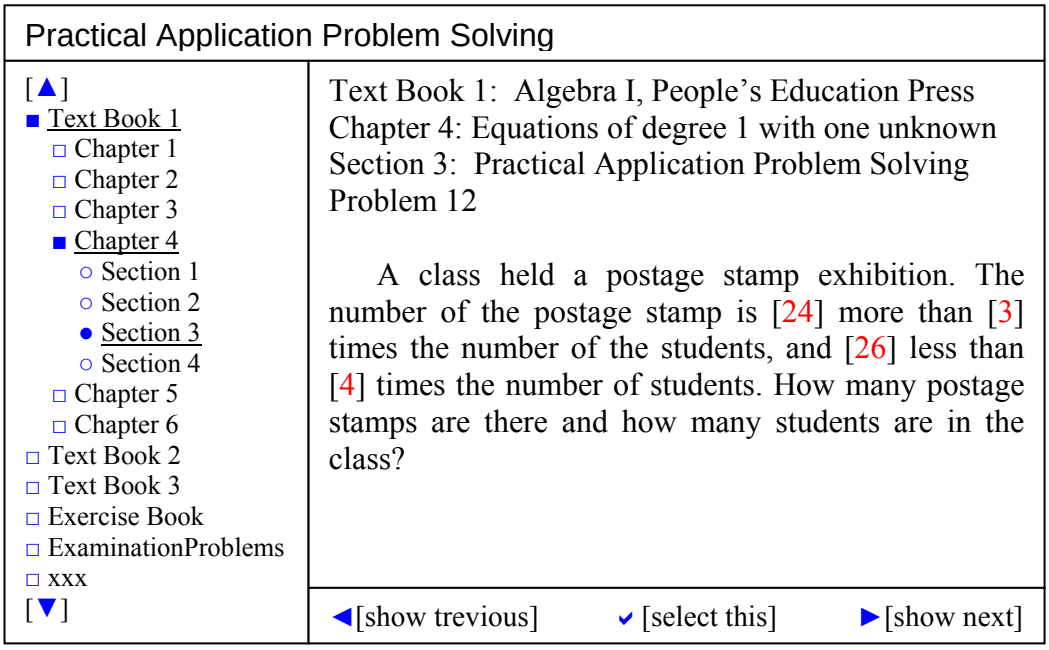

Figure 2: An example of the interface for problem browsing and selecting. The Ajax engine uses Javascript commands embedded in the source code of the page to send the request to the server when user gives instruction and then use the received data to update interface without a page refresh

`> user:` browses the content of the exercise problems by clicking the widgets (◀ and ▶), clicks editable text to change, and selects a specific exercise by clicking ✔,

> machine: displays the selected problem in the work sheet (a bitmap window), with the widgets for user to edit the text boxes, solves the problem interactively, or shows the solution.

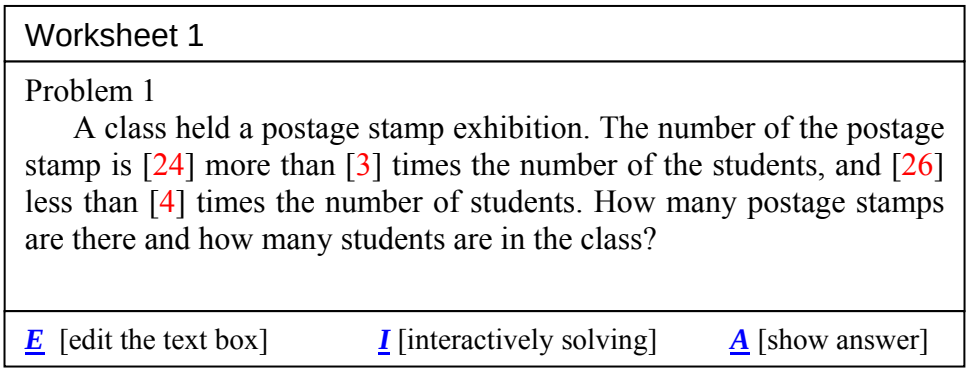

Figure 3. The interface of a worksheet. The red texts in the red bracketsare editable by the user. Widgets ***I*** and ***A*** are for selecting the solving methods. The widget ***E*** is active in default.

> user: clicks ***I*** to start the process of interactive solving immediately, or changes the numbers in the four text boxes and then clicks ***I*** to start the process of interactive solving, or clicks ***A*** to see the whole answer for this problem.

> machine: checks the changed data and displays the error information if any change is not appropriate in a pop-up-dialog window or directly in the worksheet bitmap, as shown in Figure 4 (which corresponds to [-2], [u], [3] and [0] in the four text boxes, respectively). In former case, active the widget ***E*** , and in the later, active the edit widget immediately after the pop-up-dialog window is closed. If there is no error, display the interactive solving interface or the answer in the worksheet.

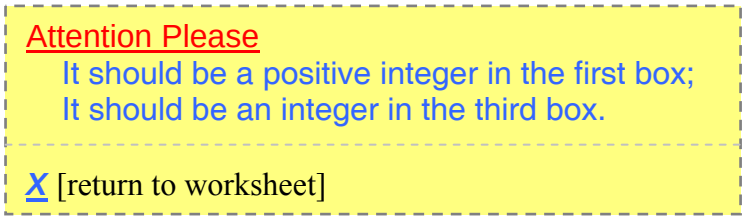

Figure 4: A pop-up-dialog window for error message. This window is on the top of the worksheet.

> user: if there is no change made or no error has been found in the input, clicks ***I*** to start the process of interactive solving or clicks ***A*** to see the whole solution. (For simplicity we omit the later event and assume that the text boxes have not been changed).

> machine: displays the instruction for interactive solving as in Figure 5 if the user clicked ***I*** with two text boxes (inside the brackets) for user to write.

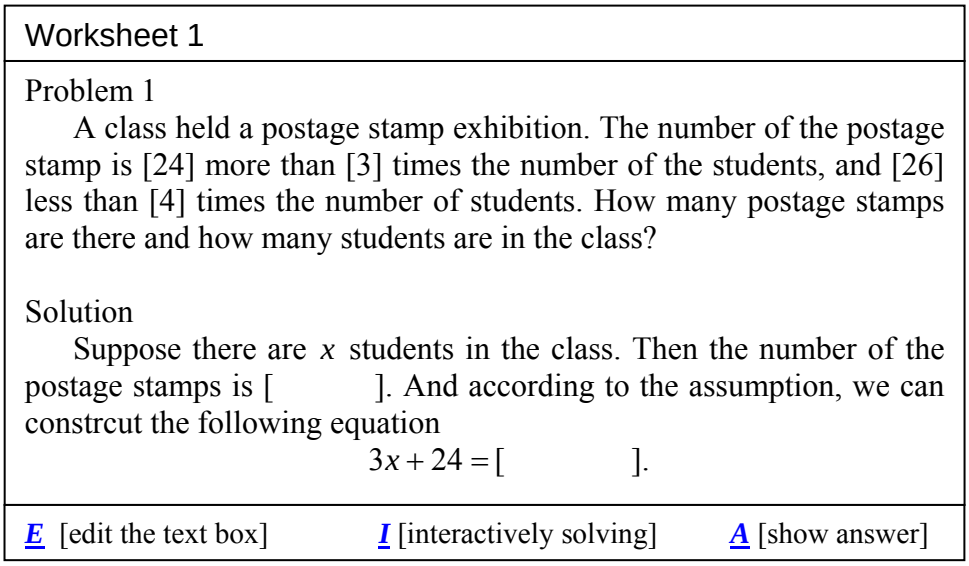

Figure 5. The instruction for interactive solving. User clicks the widget ***I*** to check if the inputs are correct.

`> user:` inputs some text (mathematics expressions) in the text boxes and clicks the widget ***I*** to check if the inputs are correct, or clicks the widget ***A*** to see the answer for this step. Type mathematics expression in plain text and the system changes it into MathML format automatically.

`> machine:` displays the mathematics expression in normal format; checks the expression written by the user, if there is any error, displays an error information as in Figure 6, otherwise displays the next instruction for interactive solving as in Figure 7; or displays the correct answer for this step if ***A*** is clicked.

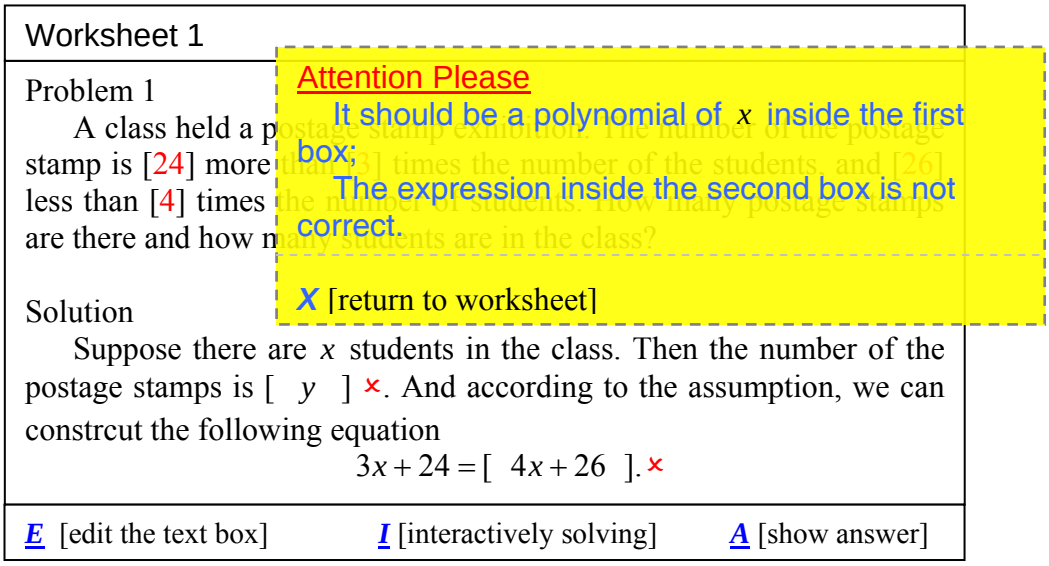

Figure 6. The expressions written in the text boxes are not correct.

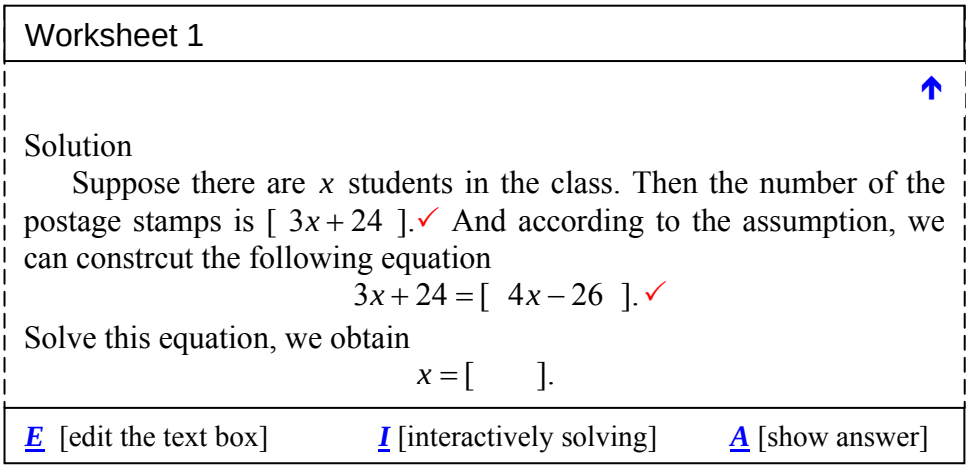

Figure 7. The instruction for the next step is displayed if the expressions written in the text boxes are correct.

`> user:` inputs the solution to the equation $3x+24=4x-26$ in the text box and clicks the widget ***I*** to check if it's correct, or clicks the widget ***A*** to see the answer for this step.

`> machine:` displays the next step for user if the solution written in the text box is correct, otherwise displays an error information; displays the correct answer for this step if the user clicked ***A***.

Refer to "process-2" (shadowed text) in the Perl scripts next section for this step. In Section 4 we will show an implementation of this interactive process with Ajax technology. Do this process until the following final step.

`> user:` inputs text inside the text boxes, clicks the widget ***I*** to check if it's correct, or clicks the widget ***A*** to see the answer for this step, until the machine shows the finish information.

`> machine:` displays the finish information, as shown in Figure 8, if the user input in the last round of interactive solving process are all correct.

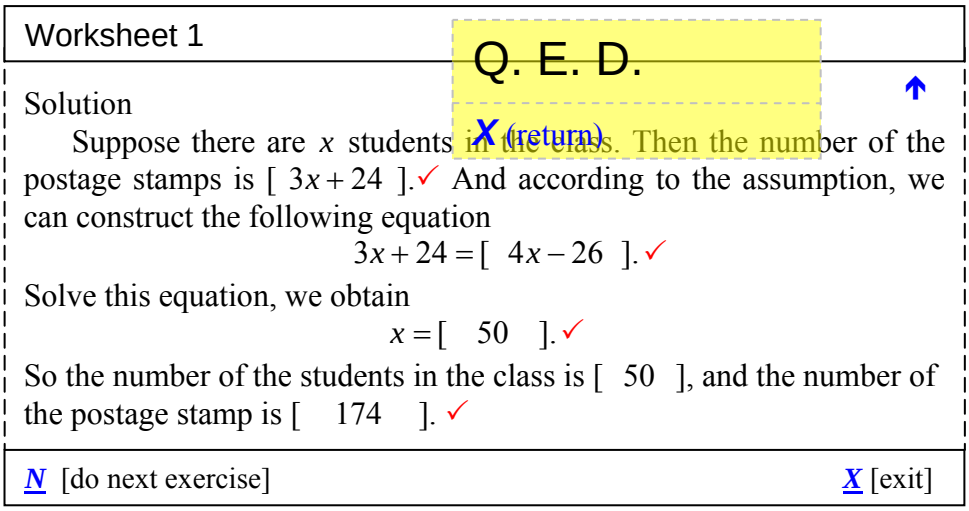

Figure 8. The finish information.

The above scenario illustrates the three important aspects of an interactive mathematics web service:

(1) a knowledge-based metadata catalog service which supports users to browse and review mathematics contents of a broad categorization of the problem area(s) according to some agreed taxonomy in some suitable ontology;

(2) an interactive worksheet which supports users to change parameters of exercises, inquires answers and instruction, manipulate mathematics expression, draw geometry and submit solution;

(3) an easy-to-use programming language that can be used to describe the human-machine interactive processes in mathematics learning.

The implementing of (1) means the realization of a large-scale knowledge engineering which needs the extensive the close cooperation between experienced mathematics teachers and software engineering.

In view of computer network technology, Ajax is an appropriate technology for construct the real-time interactive system satisfying the requirements in (2). Using this technology we can build interfaces to our web applications which are much more like those our users are used to from their desktop applications. Mathematics expressions and geometry images can be populated automatically with data retrieved from the server, data grids can be sorted or paginated, and server-side databases can be queried and edited - all without the user having to wait for pages to load.

Operations on mathematics expression (including checking two expressions for equality, checking an expression for a pattern match, solving equations, evaluating and computing with expressions) can be implemented with symbolic computation software and remote invoke method (like CORBA).

MathML makes it possible to develop Web-based applications for displaying mathematical content. There are standard tools for converting LaTeX equations to MathML. It becomes easy to create dynamic math web sites featuring interactive equations using JavaScript with MathML. See [15, 16, 17].

The above standard technologies, Ajax, Maple (Mathematica, Geometer's Sketchpad or the freeware with similar functions) and MathML provide a substantial for the interactive worksheet mentioned in (2).

As we will show in next section, Perl is a suitable language for (3).

3. A Proposal for the Format of the Script Describing Interactive Web Services

In this section we introduce a method to describe the interactive mathematics learning in web. Our proposal is to use Perl (Practical Extraction and Report Language) and then transform the Perl script into a MathML file. Perl is a popular language interactive web-based application. What is more important is that Perl is relatively easy to master for teachers without programming experience. We will not go to detail for Perl, see [18, 19] for Perl programming and examples of Perl application. For simplicity, we only give the segment of the Perl scripts for "process-1" and "process-2" in the interactive process of the use case scenario presented in Section 2.

```
#prb12.pl, perl script for interactive solving
#initial-process: process-0(omitted)

#process-1
#variables
@e=();
@f=();
$x="x";
```

```
#display content
print “Solution:\n
Suppose there are $x students in the class. Then the number of
the postage stamps is [$e]. And according to the assumption, we
can construct the following equation\n
  3*x+24=[$f].”;
#answer
@answer1=($e==”$m*$x+$a” or $e==”$n*$x+$b”) && $f==”$n*$x+$b”;
#error information
%errorinformation0=(
not($e==”$m*$x+$a” or $e==”$n*$x+$b” or not(polynomialp($e,$x))
==> ”In the first text box there should be a polynomial of $x”,
not($e==”$m*$x+$a” or $e==”$n*$x+$b” or polynomialp($e,$x))
==>”The expression in the first text box is not correct”,
not($f==”$n*$x+b”) ==>”The expression in the second text box is
not correct”);

#process-2
#An Implementation of this step will shown in the Section 4
#variable
$g=();
#display
print “solve this equation, we obtain \n
$x=[$g].”;
#answer
@answer2=”$g==($m-$n)/($b-$a)”;
#note: the answer can also be written in a function form
$answer=proc(g,a,b,m,n);
#error information
%error-information2=(
not($g==($m-$n)/($b-$a)) ==>”The answer you written is not
correct”);

#process-3(omitted)
#process-4(omitted)

#end-solution-1
#end-file
```

4. A Prototype Implementation for the Interactive Web Service with Ajax Technology

In this section we show how to use Ajax to realize the interactive scenario we have shown in the previous sections. For simplicity we just explain the implementation method for the process-2 in the Perl script. Figure 9 describes the outline of the AJAX interaction in the implementation.

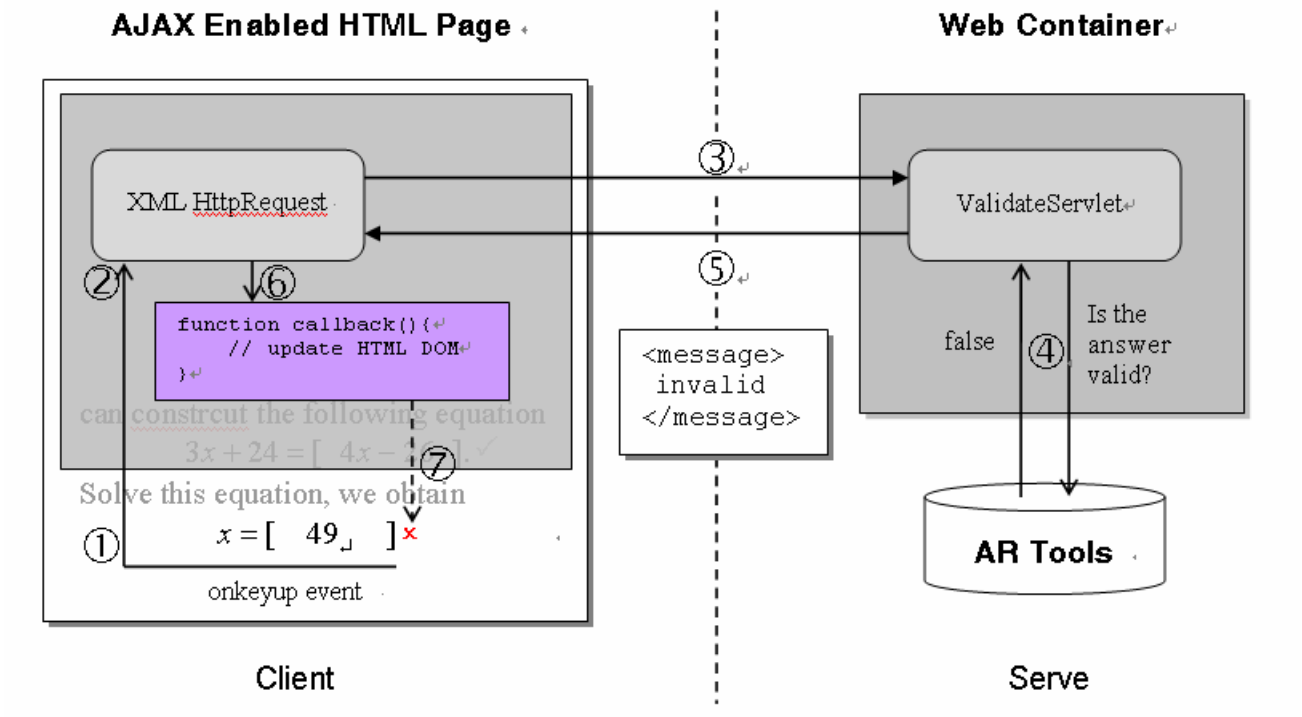

Figure 9. The outline of an AJAX interaction

The interaction contains the following seven steps.

(1) A client event occurs;
(2) An XMLHttpRequest object is created and configured;
(3) The XMLHttpRequest object makes a call;
(4) The request is processed by the ValidateServlet via CORBA with AR Tools;
(5) The ValidateServlet returns an XML document containing the result;
(6) The XMLHttpRequest object calls the callback() function and processes the result;
(7) The HTML DOM is updated.

We give an explanation to each step in more detail with fragment of code.

Step 1: JavaScript functions are called as the result of an event. In this case, the function validate() is mapped to an onkeyup event on text box (a form component). This form element will call the validate() function each time the user presses a key in the form field.

```
<input type="text"
    id="textbox"
    name="id"
    onkeyup="validate();" >
```

Step 2: The validate() function creates an XMLHttpRequest object and calls the open function on the object. The open function requires three arguments: the HTTP method, which is GET or POST; the URL of the server-side component that the object will interact with; and a Boolean indicating whether or not the call will be made asynchronously. If an interaction is set as asynchronous (true), a callback function must be specified. The callback function for this interaction is set with the statement req.onreadystatechange = callback;.

```
var req;
function validate() {
    var idField = document.getElementById("textbox");
```

```
    var url = "validate?id=" + escape(idField.value);
    if (window.XMLHttpRequest) {//for non IE browser
        req = new XMLHttpRequest();
    } else if (window.ActiveXObject) {//IE 5.0 and up
        req = new ActiveXObject("Microsoft.XMLHTTP");
    }
    req.open("GET", url, true);
    req.onreadystatechange = callback;
    req.send(null);
}
```

Step 3: The XMLHttpRequest object makes a call when the statement req.send(null); is reached. In the case of an HTTP GET, this content may be null or left blank. When this function is called on the XMLHttpRequest object, the call to the URL that was set during the configuration of the object is called. In this example, the data that is posted (id) is included as a URL parameter. Use an HTTP GET when the request is idempotent, meaning that two duplicate requests will return the same results, and an HTTP POST method when the data sent to the server will affect the server-side application state. In this example we use HTTP GET.

Step 4: Use CORBA or other remote invoke methods to get the solution of the equation $3x + 24 = 4x - 26$ from an AR tool (here a symbolic computation program). This part is not related to Ajax technology so we will not go to detail discussion.

Step 5: The ValidateServlet returns an XML document containing the results. In this example, if the user answer is "49", the ValidateServlet will write an XML document to the response containing a message element with the value of invalid.

```
response.setContentType("text/xml");
response.setHeader("Cache-Control", "no-cache");
response.getWriter().write("invalid");
```

Step 6: The XMLHttpRequest object calls the callback() function and processes the result. The XMLHttpRequest object was configured to call the callback() function when there are changes to the readyState of the XMLHttpRequest object. The parameter readyState can be one of the following:

0: Uninitialized, open() has not been called;
1: Loading, send() has not been called;
2: Loaded, send() has been called;
3: Interactive, send() has been called, but the request has not been satisfied;
4: Completed, request has been satisfied.

The parameter status can be one of the following values

404: server not found;
500: error in the Server program;
200: OK, a successful HTTP interaction.

In this example we assume that readyState is 4 and the HTTP status code is 200.

```
function callback() {
    if (req.readyState == 4) {
        if (req.status == 200) {
            processMessage()
        }
    }
}
```

The function processMessage() updates the HTML DOM (an object representation of the documents being displayed by the browser, Document Object Model,) based on the message returned from the ValidateServlet. Following is the XML document of the message:

```
<message>
   invalid
</message>
```

Step 7: Update the HTML DOM by using document.getElementById() to gain the reference to the element appearing in the HTML document and using JavaScript to modify the element's attributes.

```
<script type="text/javascript">
function processMessage(message) {
    mdiv = document.getElementById("messageText");
    if (message == "invalid") {
       mdiv.innerHTML = "<div style=\"color:red\">×</ div>";
    } else {
       mdiv.innerHTML = "<div style=\"color:red\">✓</ div>";
    }
}
</script>
<body>
<div id="messageText"></div>
</body>
```

Figure 10 shows the user interface corresponding to the above Ajax interaction in a prototype implementation.

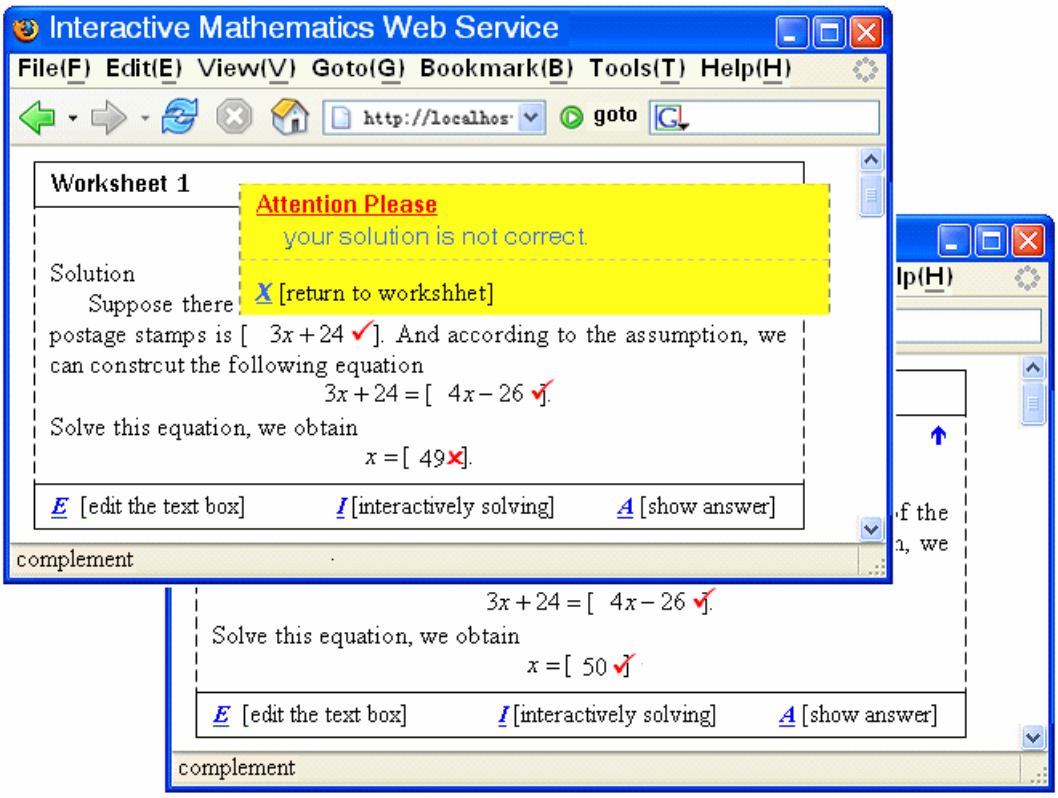

Figure 10. The user interface corresponding to process-2.

5. Conclusion

We have proposed in this paper the mechanism to build the interactive mathematics web services with a prototype implementation of a specific example. Our method integrates the easy-to-use Perl language, the standard Ajax technology and the powerful symbolic computation and dynamic geometry tools.

It is worth to indicate that as a natural development of education technology in Internet, the mathematical web service and other remote learning environment can be provided by in other different ways. One applicable technology that is already used in education is MuPAD (Multi Processing Algebra Data Tool, [20]) computing server. This technology provides a remote possibility to access MuPAD (on the server side) from a web page. One advantage of this technology is graphics and animation, and export graphics to JavaView formats. In [21], an initial design of Lupin, a layered framework of PSEs construction based on Web technologies is proposed and discussed. It is also worth to mention that people can use Lisp (for example, the ACL AllegroServe and WebActions framework) for both the server (Ajax web application, [22]) and the AR tools.

Acknowledgements This work is supported in part by NKBRPC-2004CB318003 and NNSFC-10471044. The authors would like to thank Dr. Mirek Majewski and Dr. Sheng-chuan Wu for their valuable suggestions.

References

[1] C E Beevers, IT was twenty years ago today..., LTSN MSOR Maths CAA series, January 2006, http://mathstore.ac.uk/articles/maths-caa-series/jan2006/.

[2] Marc-Laurent Aird, MONET: service discovery and composition for mathematical problems. Proceedings of IEEE workshop on Agent-based Cluster and Grid Computing (at CCGrid 2003).

[3] Julian Padget, MONET: Mathematical service discovery and composition, http://monet.nag.co.uk/cocoon/monet/publicdocs/monet_talks.html

[4] Bob Broughton, Easaw Chacko, Leng Leng Lim, Computer-Aided Teaching and Testing,

[5] iScholar, www.ischolar.ca/

[6] David Fisher, Review of Maple T. A., MSOR Connections, Nov 2004, Vol. 4 No. 4, http://mathstore.ac.uk/newsletter/nov2004/pdf/mapleta.pdf

[7] SureMath, http://www.suremath.com/

[8] Robert Miner, An Object Model for Dynamic Math, http://www.mathmlconference.org/2002/presentations/miner/.

[9] Gao Xiaoshan, Zhang Jingzhong and Chou Shang-Ching. Geometry Expert, Nine Chapters Pub (in Chinese), Taiwan, 1998.

[10] Zhang Jingzhong, Master the Freeware Super Sketchpad, Science Press, China, March 2006.

[11] Fu Hongguang, Zeng Zhenbing, Hu Yufeng, Chen Changyu, Zhong Xiuqin, MathXP, Sichuan Electronic Audio Video Publish Center, China, September 2002.

[12] Dana Petcu, Cosmin Bonchis, Cornel Izbasa, Symbolic Computations based on Grid Services , International Journal of Computers, Communications & Control, Vol. I (2006), No. 1, pp. 33-39

[13] Andreas Strotmann, Wanjiku Nganga, and Olga Caprotti, Multilingual Access to Mathematical Exercise Problems,

[14] Martti Rahkila, Agent-based Method for Self-study Interactive Webbased Education, Thesis submitted in partial fulfillment of the requirements for the degree of Licentiate of Science in Technology. Helsinki University Of Technology, Department of Electrical and Communications Engineering Laboratory of Acoustics and Audio Signal Processing Espoo, May 17th, 2006

[15] Luis Alvarez-Sobreviela, REDUCE-MathML Interface, http://www.software.ibm.com/network/techexplorer/

[16] Pao-Ta Yu, Zhong-Ming Weng, Yu-Xu Zeng, Constructivism On the Web Education--On the Design of Interactive Math Editing Environment based on the MathML, http://sun.tchcvs.tc.edu.tw/cai/pdf/w4.pdf.

[17] HTML Math Overview, http://www.geom.uiuc.edu/~rminer/w3c/.

[18] Larry Wall, Tom Christiansen, Jon Orwant, Programming Perl, 3rd ed., O'Reilly Media, Inc.. (July 2000).

[19] Free Perl Scripts, www.anybrowser.com/tools.html.

[20] M. Majewski, MuPAD Graphics – Stretching Limits of Scientific Visualization, ATCM 2003, December 15-19, 2003, Chung Hua University, Hsin-Chu, Taiwan.

[21] K. Li, M. Sakai, Y. Morizane, M. Kono, and M.-T. Noda, Lupin: Towards the Framework of Web-based Problem Solving Environments, ATCM 2003, December 15-19, 2003, Chung Hua University, Hsin-Chu, Taiwan.

[22] Franz Inc., Allegroserve, Webactions and Ajax, http://www.franz.com/support/tech_corner/ajax.lhtml.