Contributing to Mathematics Lessons Authoring System (MLAS)

A Web-based Application Programming Interface

Samer F. Khasawneh
Department of Mathematics and Computer Science
The College of Wooster
Wooster, Ohio, USA
skhasawneh@wooster.edu

Abdulelah A. Algosaibi
Department of Computer Science
Kent State University
Kent, Ohio, USA
aalgosai@kent.edu

Abstract—**In this paper, we give a broad overview of a Web-based system, MLAS, that enables teachers, who do not necessarily know how to program, to dynamically author and deploy mathematical lessons on the Web. Our primary focus, however, is on a feature inside MLAS; an on-Web Application Programming Interface (API). It is possible through this API to embed educational objects by any developer with XHTML/ JavaScript expertise. With an intuitive and easy-to-use Graphical User Interface (GUI), programmers can deploy their own code and add new materials to be used by anybody using MLAS. The API utilizes the simple, yet powerful, site architecture which guarantees structured content placement and retrieval between the application and the back-end MySQL database.**

Keywords- API; GUI; Web Technologies.

I. Introduction

Web-based Mathematics Education (WME) [1, 2], which started in 2003, is a mathematics education system that uses the Web to promote the quality of education. It aims to deliver classroom ready, dynamic, and hands-on lessons and modules to teachers and students. In WME, mathematical lessons are offered as collection of Web pages using cutting-edge Web standards such as PHP [9], JavaScript [7], Document Object Model (DOM) [8], and MySQL [12]. These web pages are connected through a giant architecture that allows easy interoperability and sharing across different schools participating in the WME project. While those WME lessons have proved to be helpful resources and students found them fun to use, from a programming perspective, the process of creating a lesson is considered to be long and challenging. Each WME lesson needs to be hand-coded and should comply with a number of requirements and follow certain protocols in order to work as intended.

The on-Web MLAS [3, 14] is an independent work under the big WME project. MLAS features a well-organized architecture that abstracts all aspects of manipulating the contents. The process of creating a lesson in MLAS is automated and content-rich lessons can be created with few mouse clicks without any programming know-how. Editing lessons enjoys the same simplicity and assumes no advanced computer skills.

Because MLAS can be thought of as a content management system for mathematics education, it might be relevant to explain some terminologies that will be used in this paper. *Manipulatives*, in general, are any objects, such as coins, tiles, and even a paper that is cut or folded, used to help students understand abstract math concepts such as fractions and percentages in an active, hands-on approach. With the advent of the Web and based on its potential effect in enhancing the quality of education in general, and the math subject in particular, a new term has come into existence, "*virtual manipulatives*". This term refers to those manipulatives that cannot be "touched" but rather can be "seen" on a computer screen, allowing students to explore them using computer hardware, such as a mouse and keyboard [4].

Existing Web-based systems, including MLAS and WME, assume that a mathematical lesson is a collection of virtual manipulatives. MLAS features a library of customizable virtual manipulatives. When authoring a lesson, a teacher may include one or more interactive virtual manipulatives. The manipulatives can be customized and can interact with one another or questions and comments in the lesson page. MLAS offers a growing library of virtual manipulatives that are fully customizable, editable, and reusable.

Due to the nature of the MLAS project and the need to have dedicated people adding new manipulatives, we think that it is necessary to let others contribute to expand those manipulatives of MLAS.

An *Application Programming Interface* (API) is a specification intended to be used as an interface by software components to communicate with each other. An API may include specifications for routines, data structures, object classes, and variables [5]. Through the Web-based API MLAS offers, the MLAS library is easily expandable by adding new manipulatives contributed by developers and other experts.

MLAS supports two views: teacher view and student view. Obviously, a teacher is the one who controls the form in which a lesson would look to students. A lesson in the authoring stage where customization is possible is the teacher view, while the view of the "final product" in the lesson page where no customization is allowed is the student view.

This paper is organized as follows. Section II presents the related work in the field. Section III gives detailed overview of our API including its features and capabilities. Section IV shows case study on how to embed an external work to be as if it was natively supported by MLAS. We conclude this work by presenting our views on possible development and enhancements.

II. RELATED WORK

In this field, it been somehow difficult to find a project that implement this kind of API. The reason behind it, as we researched most of the work [15][16][17][18], is that consider as project who software company take care of its maintenance (adding, deleting or editing a feature). Such softwares suffers allowing users themselves to contribute on systems. In this work, it is essential that teacher has the ability to author hands-on, fully customizable virtual manipulatives e.g. copy old web lesson and paste it at API. We built MLAS project with API feature in order to support ability to be expanded or shrinked in terms of future virtual manipulatives or removing unnecessary one. All that without necessarily programming skills. The research done on [19] aimed to solve the the cost of time and financial issues in the way of developing reusable personalized e-Learning content with appropriate metadata. In here, we are considering API helps in the reusability of manipulatives. In this context, as learning style, the idea of building reusable pedagogical components that transferable to other learning style have been introduced in [20]. Our work is different in the way of solving mentioned issues. Technically, the API architecture built in the way increase the level of the degree customization and reduce implementation overhead.

III. API OVERVIEW AND SPECIFICATION

It is possible through MLAS to embed any number of manipulatives by any developer with XHTML [11] and JavaScript expertise. With an intuitive and easy-to-use GUI, programmers can deploy their own code and add manipulatives. Such manipulatives would appear in the manipulative library so any user can use them

There are basically little to no limitations at all through the self-guided interface. XHTML, JavaScript, and CSS [13] contents can be uploaded to the server and their contents will be stored in our MySQL database. Alternatively, these contents can be written directly into designated text boxes. Because a manipulative often engages the use of dynamic resources, the interface also allows our users to upload any type of resource they wish (*.swf*, *.class*, *.jpg*, etc.). In short, this is a very intuitive work that strengthens MLAS and expands its usability.

It is worth to mention that throughout our literature search, we were unsuccessful trying to find a system that provides such noble feature MLAS offers.

As appears in Fig. 1, the user interface has three separate code segments for the user to fill-in with XHTML and/or JavaScript. The organization and order of these boxes were selected and arranged in a way that we think is very convenient and easy to follow. Therefore we have separated the JavaScript-only box from other boxes. We also gave the users the opportunity to directly type or even copy-and-paste contents in designated areas.

Fig. 2 shows the requirements that have to be taken into account once a user wishes to submit a manipulative. All these requirements appear underneath the form on the same interface.

Requirements:
1. In Box 1: You need to enter only HTML inside it. Javascript code can be entered inside HTML events like onclick.
2. In Box 2: You must make explicit call to the Javascript function inside the HTML events.
    * Parameter 1: String of HTML elements that will be saved in the DB and which will be seen by the students. This may also contain Javascript inside the HTML events.
    * Parameter 2: String that tells the type of your manipulative.
    * Parameter 3: The string name of your manipulative.
3. In Box 3: Define your needed Javascript functions.
4. Give your functions unique name to avoid conflicts.
5. Resources can be applets, flash files, etc. They get uploaded to "uploaded_files" directory.

Figure 2. List of requirements.

The first box is to be filled with HTML only and may have some JavaScript inside any HTML event such as *onmouseover* or *onclick*. This aids in manipulative flexibility and the multi-form property of all manipulatives. However, this HTML forms the teacher view of a manipulative and hence it is not yet ready to be previewed on the lesson page. Technically, a manipulative content has to be saved in the database in order for the script to be able to pull up its content and display it. At this point, the HTML here can only be appended to a parent element in the DOM tree and as a result would disappear with every page refresh

The second box should have JavaScript code embedded to capture specific behavior from the HTML in the first box. For instance, if the HTML from the box above had two HTML checkboxes and one is initially checked, such as:

```
<input checked="checked" type="checkbox" id="cb1" />First
<br/>
<input type="checkbox" id="cb2"/>Second <br />
```

The jQuery [6] statement below can be used in the second box of the GUI and should pop-up the ID of the selected checkbox.

```
    if($('#cb1').attr('checked')==true){
            alert('cb1!');
}
    else {

            alert('cb2!');}
```

To make the manipulative available for display, a programmer has to make an explicit call to the JavaScript function "*save*", which is defined by MLAS. This allows for a manipulative to be saved in the appropriate database table through some PHP and Asynchronous JavaScript And Xml (AJAX) [10] implementations. The *save* function takes three string parameters: The first parameter is the manipulative HTML content to be saved in the database and thus to be previewed in the lesson page. It could be related to the HTML from the first box, but they are not necessarily the same. The second and third string parameters do not affect the functionality and behavior of a manipulative, but rather are needed to describe its overall meaning. The third parameter, in particular, can be any text and will appear in the "*progress menu*" that documents all user activities.

The third box allows the user to define the JavaScript functions needed. Very often, HTML events will make explicit calls to JavaScript functions which can be defined in this box. Users can include all the functions their manipulatives need in this box. To avoid name conflicts, we ask our users to choose unique names for their functions. Since the user might not always need to define his/her functions, this box can be left unfilled.

Once the user finishes filling out the necessary items in the HTML form and submits it, everything else gets taken care of by MLAS. The resources get saved in a dedicated directory on the server and the user's code gets saved in a secure database. Then through PHP, we extract that code from the database and encapsulate it, along with other pieces of data, under a couple of JavaScript functions – One function will be triggered once the manipulative is called from the manipulative library (the *show* function), while the other will be called once the teacher is happy with the manipulative and decides to have it included in the lesson (the *submit* function). Fig. 3 explains the process in details.

From Fig. 3, we can see that we use PHP to write two JavaScript functions to encapsulate the user inputs which is already saved into the database. These files can then be included in the lesson page and both functions will be ready to be called once the manipulative is in use.

A typical user cannot distinguish between a user-added and an admin-added manipulative. A user-chosen manipulative image and name would appear inside the manipulative library as if they were natively supported by MLAS. MLAS also includes all the necessary code needed to make user-added manipulatives appear and interact seamlessly in their enclosing pages.

Since user's HTML input is allowed, this implies that interactive contents can be inserted to MLAS and will be supported as well. For example Java applets and flash files can be embedded to the system with the use of the appropriate HTML tags. Below is a simple example of how an applet can be embedded to MLAS. In the applet context, only the compiled version of the Java program (*.class* extension) needs to be uploaded to the server so the browser can display the applet (assuming the browser has the Java plug-in installed).

IV. Case Study

In this section we present a simple case to show how smoothly embedding an manipulative applet in MLAS. The applet that we will be showing is very simple. It is a single button labeled as "This button doesn't do anything." For this case, the only resources we need are the applet image and the applet *.class* file. We begin by giving our manipulative a name (say, Applet Example), and then we upload the manipulative image and the *.class* file (Assume named, *ExampleApplet.class*) through the user interface.

To embed an applet in a Web page, the HTML <*applet*> tag needs to be used, with the *codebase* attribute indicating the directory on the server where the *.class* file exists, and the *code* attribute to denote the name of the *.class* file itself. Other attributes, such as *width* and *height*, might be used to control the size of the applet. Therefore, the first HTML box can be filled with the following HTML segment.

```
<applet   codebase = "uploaded_files/"
          code=" ExampleApplet.class"
          width = "400" height = "50" > </applet>
```

The GUI in Fig. 2 clearly indicates that all user uploads go inside "*uploaded_files*" directory. Therefore that directory is referenced in the *codebase* attribute above.

In order to properly display the applet in the lesson page, we need to have the above applet tag to be the first parameter of the *save* function. Since, in this case, the teacher and student views are similar, nothing else needs to be added to the second box. That can be something like:

```
var applet = '<appletcodebase = "uploaded_files/" ';
applet +='code ="ExampleApplet.class"';
applet +='width="400" height="50"></applet>';

save (applet,"Applet","Applet added!");
```

All other fields of the HTML form can be left blank, and the user can now proceed. The applet image will now be available in the manipulative library together and a click on that image will show the applet in the lesson (Fig. 4)

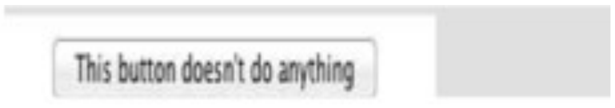


Figure 4. The applet appears in the lesson.

To test this API, we have also embedded a numerous lessons and manipulatives. The time to add such one was fast with no problem encouraged. Requirements for the API ensure that MLAS work smoothly without affecting of removing or adding manipulative. We were effortlessly able to import any desired content and have them work perfectly under MLAS's framework. Table 1 below shows the time taken on seconds to Add/Remove a manipulative with respect to expert or Non-Expert. As Non-expert user, our sampling included teacher with modest to no expertise. The other way, adding/removing without API, to deal with manipulative is to go to the row php page write necessary piece code and test it. In compare that with API usage, the difference noticeable in term of time and effort. The time ratio in the table shows the deference ratio and how much time this technique save. This ratio represent how this API saved time to write necessary piece of code e.g. PHP/MySql and test its result and presentation. The reader is referred to [14] for more examples.

TABLE 1. shows the cost in time for applying API feature to MLAS framework.

	Add with API	Add without API	Def. ratio	Rem. with API	Rem. without API	Def. ratio
Expert	31	54	42.6%	14	24	41.6%
Non-Expert	84	2167	96.12%	35	808	95.6%

V. CONCLUSION AND FUTURE WORK

In this work, we briefly introduced an on-Web lesson authoring system, MLAS, for mathematics education. Our system permits teachers and experts to access and author dynamic mathematical lessons without any programming know-how. To achieve this, MLAS offers a growing library of virtual manipulatives that can be extended through a well-organized API. The architecture of the system and its collaborative constituents make it easy to share and exchange content both within and beyond MLAS.

We are constantly trying to reduce the requirements of our API to make it even better. As a possible future work, it might be useful to add a feature that allows the automatic extraction of the HTML/JavaScript code of any external manipulative to be inserted into the API. Further, we are in the process of empowering the entire MLAS with the upcoming HTML5 standards. With HTML5, MLAS would have more dynamic features and be more up-to-date with the Web standards.

REFERENCES

[1] P. Wang, M. Mikusa, S. Al-shomrani, D. Chiu, X. Lai, and X. Zou. Features and advantages of WME: a Web-based mathematics education system. In Proceedings of the IEEE Southeast Conference. Florida, USA, 2008. pages 621-629.

[2] P. Wang, M. Mikusa, S. Al-Shomrani, X. Lai, X. Zou, and Zeller. "WME: a Web-based Mathematics Education System for Teaching and Learning." ICME 11 – TSG 22 Theme 3 the 11th International Congress on Mathematical Education. Mexico, July 2008.

[3] S. Khasawneh and P. Wang. "Overview of Mathematics Lessons Authoring System (MLAS)". Proceedings of CSEDU 2012, Porto, Portugal, pp. 48-54, April 2012.

[4] CITEd Research Center, Learning Mathematics with virtual manipulatives, http://www.cited.org/index.aspx. Retrieved: Jan, 2013.

[5] Application Programming Interface. http://en.wikipedia.org/wiki/Application_programming_interface. Retrieved: Jan, 2013.

[6] jQuery. http://www.jquery.com Retrieved: Jan, 2013.

[7] JavaScript. http://en.wikipedia.org/wiki/JavaScript. Retrieved: Jan, 2013.

[8] Document Object Model (DOM). Technical report, http://www.w3.org/DOM/. Retrieved: Jan, 2013.

[9] PHP: Hypertext preprocessor. Technical report, http://www.php.net/ Retrieved: Jan, 2013.

[10] Asynchronous JavaScript and Xml (AJAX). Technical report, http://developer.mozilla.org/en/docs/AJAX.

[11] XHTML. http://en.wikipedia.org/wiki/XHTML Retrieved: Jan, 2013.

[12] MySQL. http://en.wikipedia.org/wiki/MySQL. Retrieved: Jan, 2013.

[13] Cascading Style Sheet (CSS) to style HTML elements . http://en.wikipedia.org/wiki/Cascading_Style_sheet. Retrieved: Jan, 2013.

[14] S. Khasawneh. "A Web-based Lessons Authoring System for Mathemtics Education". PhD dissertation. 2012

[15] http://www.articulate.com/ Retrieved: Jan, 2013.

[16] http://www.courselab.com/. Retrieved: Jan, 2013.

[17] http://www.elicitus.com/. Retrieved: Jan, 2013.

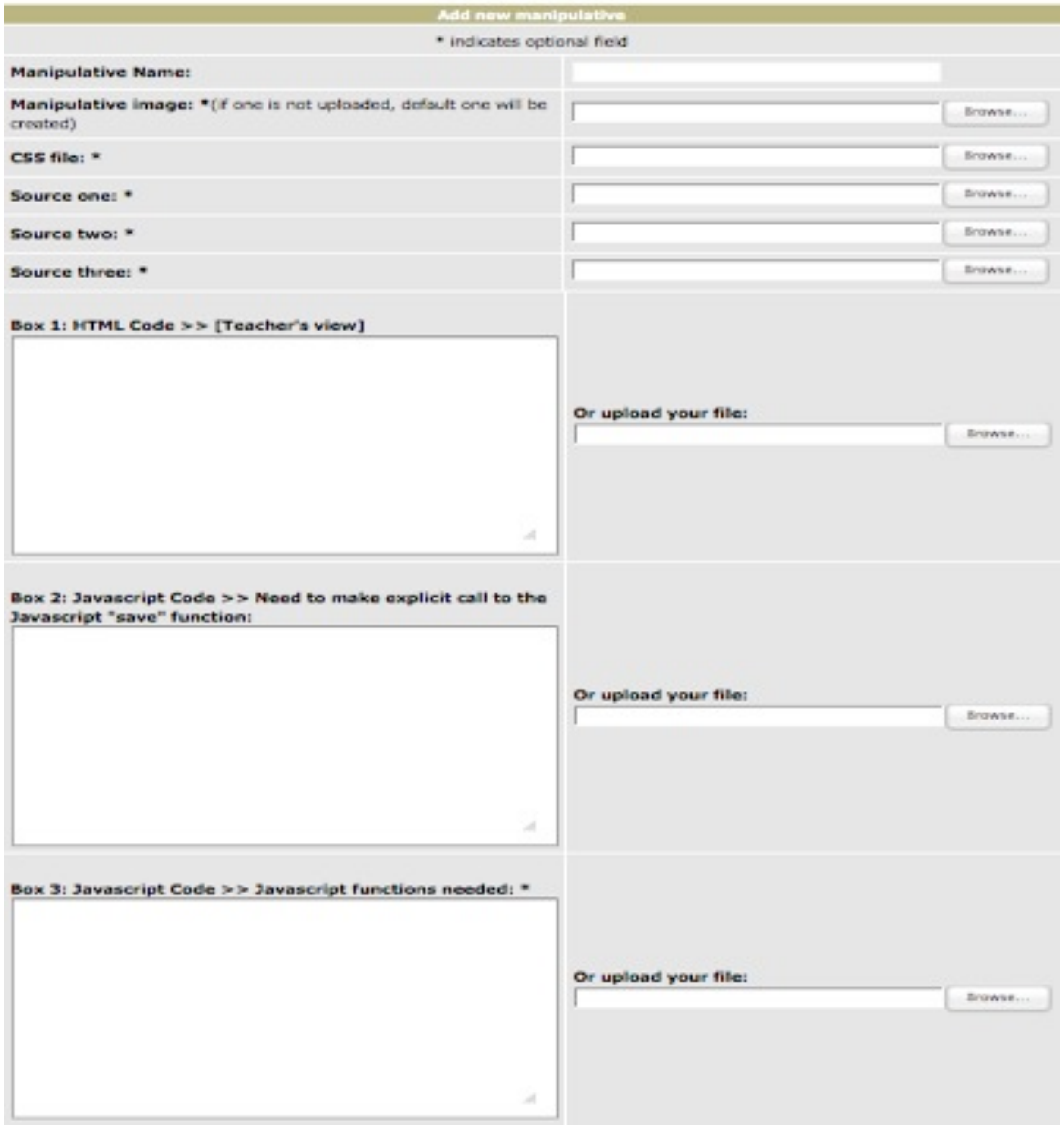
[18] http://www.ispringsolutions.com/. Retrieved: Jan, 2013.

[19] O. Conlan, D. Dagger, and V. Wade. "Towards a standards-based approach to e-Learning personalization using reusable learning objects". In: Driscoll, M. and Reeves, T.C. (eds.) Proceedings of World Conference on E-Learning AACE. Montreal, Canada, October 15-19, 2002. PP. 210-217.

[20] C. Bruen and O. Conlan. "Dynamic Adaptive ICT Support for learning Styles – A Development Framework for re-useable learning resources for different learning styles & requirements". Proceedings of the ITTE 2002, Annual Conference of the Association of Information Technology for Teacher Education. 2002 pp. 1238-1241. Chesapeake, VA

```
// Assume the manipulatives table is fetched from the database and stored in $manip variable
//Now, also define a file variable and have a file ready (manipulatives.js) to accept content write

$file = fopen("manipulatives.js", "w");

//Go through the manipulatives table one record by another and place each manipulative in a JavaScript function
foreach($manip as $m) {

//Define a PHP variable that contains some JavaScript code,
// which later will be appended to the page. $m['name'] here refers to the name of the manipulative
        $function = "function show_". str_replace(' ', '_', $m['name']) . "() {\n";
//Assuming user directly placed code in box ($m['html'] is user's first box content). Then, perform some code cleaning
        $m['html'] = str_replace('\'', '\"', $m['html']);
        $m['html'] = str_replace(array("\r\n", "\r", "\n"), ' ', $m['html']);
//Add the buttons to allow work saving or cancellation
        $submit = '<input type="button" value="Proceed" onclick="submit_'. str_replace(' ', '_', $m['name']) .'();";
        $submit .= 'class="sbmt"><input onclick="cancel()" type="button" value="Cancel">';
//Finally allow user's code to append to the page and end of the show function

        $function .= '$("#page_element").append(\'<div>'. $m['html'] . ' ' . $submit. ' </div>\');';
        $function .= "} \n";
//Write content to file
        fwrite($file, $function);
// Define the submit_$m['name'] which will contain the contents of box 2 including
// the call to save function that will save code in the database
        $function = "function submit_". str_replace(' ', '_', $m['name']) . "() {\n"
//Similarly, do code cleaning as we are assuming user has put code directly in box 2
        $m['js'] = str_replace(array("\r\n", "\r", "\n"), ' ', $m['js']);

        $function .= $m['js'] . "\n";

        $function .= "}\n"; //end of the submit function
//Write content to file and then close it
        fwrite($file, $function);
        fclose($file);
}//end of foreach loop. Finally, the file has to be included in the page for this to work
echo '<script type="text/javascript" src=" manipulatives.js"></script>';
```

Figure 1. The API.



Add new manipulative

* indicates optional field

Manipulative Name:	
Manipulative image: *(if one is not uploaded, default one will be created)	Browse...
CSS file: *	Browse...
Source one: *	Browse...
Source two: *	Browse...
Source three: *	Browse...
Box 1: HTML Code >> [Teacher's view]	Or upload your file: Browse...
Box 2: Javascript Code >> Need to make explicit call to the Javascript "save" function:	Or upload your file: Browse...
Box 3: Javascript Code >> Javascript functions needed: *	Or upload your file: Browse...

Figure 3. PHP handling user’s input.