Proceedings of the Fifth IASTED International Conference
WEB-BASED EDUCATION
January 23-25, 2006, Puerto Vallarta, Mexico

AN APPROACH FOR INTEROPERABLE AND CUSTOMIZABLE WEB-BASED MATHEMATICS EDUCATION

David Chiu
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210, USA
chiud@cse.ohio-state.edu

Paul S. Wang
Department of Computer Science
Kent State University
Kent, OH 44242, USA
pwang@cs.kent.edu

ABSTRACT
Classroom trials at Kimpton Middle School demonstrated advantages of the WME (Web-based Mathematics Education) system and obtained positive feedback as well as suggestions from teachers and students. Based on the prototype site for Kimpton, an interdisciplinary team is developing a model WME site making it directly deployable in different schools. Components within WME sites are interoperable and easily customizable by teachers. Reported are the model site design, organization, architecture and implementation. Features that support portability, component interoperability and easy user customization are emphasized. These features are important in making WME practical, efficient, and effective for mathematics education.

KEY WORDS
WME, Web-based Mathematics Education, customization of lessons

1. Introduction

Students in the US are falling seriously behind other countries in mathematics tests. Furthermore, recommendations for mathematics education by distinguished groups of scientists and educators have been for the most part ignored [8]. According to [10], "The fastest growing jobs require much higher math, language, and reasoning abilities than current jobs, while slowly growing jobs require less." It is clear that students need to master mathematics more deeply and at higher levels than they do at present.

The increased curricular demands on students mean mathematics teachers are required to know more mathematics themselves. This coupled with the push for high stakes tests has caused shortages in highly qualified mathematics teachers. The amount of cognitive, organizational, and emotional work needed to be a successful mathematics teacher, already high, is growing. What can be done to support mathematics teachers and help them become more efficient and productive? Can we find a practical and effective way to use modern technology to assist the teaching and learning of mathematics?

At the Institute for Computational Mathematics, an interdisciplinary team of mathematicians, computer scientists, education researchers, Web developers, and middle school teachers is building a *Web-based Mathematics Education* (WME) system by an innovative combination of open Internet technologies. Figure 1 illustrates the WME concept. WME can deliver, via the Internet or a LAN (wired or wireless), classroom-ready lessons that are well-prepared, interesting, effective, as well as interoperable. In addition to allowing multimedia content and hyperlinks, lesson pages feature in-page *manipulatives* to help students understand and explore mathematical concepts and skills through hands-on activities.

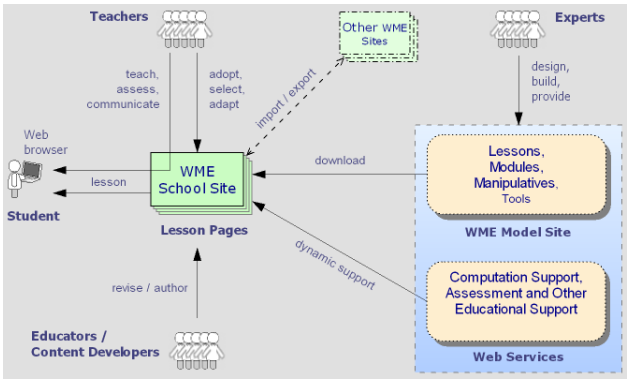

Figure 1: The WME Concept

WME is different from existing approaches and aims to be a modern, practical, efficient and effective Web-based system to support mathematics education and learning [6]. The WME system conforms to open standards, works with regular browsers, delivers integrated and complete lessons, enables easy customization, provides systematic access to client-side and server-side support, and allows independently developed WME components to interoperate seamlessly. In short WME seeks to create a *Web for Mathematics Education*. Figure 2 illustrates some of the contents and services WME can combine and

integrate for effective Web-based mathematics education. See [6] for details and references on related work.

1.1 WME Pilot Site

With funding from the Ohio Board of Regents (OBR) the ICM group began, in early 2004, a pilot project to study how the WME technologies can best be tailored and applied for actual mathematics education in schools. The pilot project at Kimpton Middle School (Munroe Falls, Ohio) conducted in-class trials at the 7th grade with two mathematics teachers and over 100 students that spanned several math classes. Our gained experiences, student and teacher feedback, and early lessons learned have been reported in [5].

Encouraged by the positive reactions from both students and teachers, we proceeded to build a prototype WME Web site to implement new ideas and improvements to teach a variety of additional topics and to perform more in-class trials. The topics include *integers*, *percentages*, *proportions*, *length and area*, *number relations*, *fractions*, *probabilities*, and *understanding data*. These are a subset from the *Ohio Academic Content Standards* [11] which closely follows the NCTM standards [9]. These trials have continued and are still on-going.

Figure 2: WME Integration

Component interoperability and easy customization by teachers are among the most important features of WME. The goal is to provide simple download and installation of WME sites for schools and to give teachers the ability to modify and tailor many parts of the classroom-ready materials to suit their own teaching needs. The WME site deployment process is illustrated in Figure 3. It is furthermore possible for teachers to publish back to the WME system their improved lesson pages in order to share with others.

We begin by presenting the design and organization of the *WME model site*, site portability, and component independence in the next section. Sections 3 and 4 describe our approach to achieve interoperability and customization of WME manipulatives. Section 5 explains how the many components of WME can be made customizable to users. Section 6 concludes and discusses future opportunities in this area.

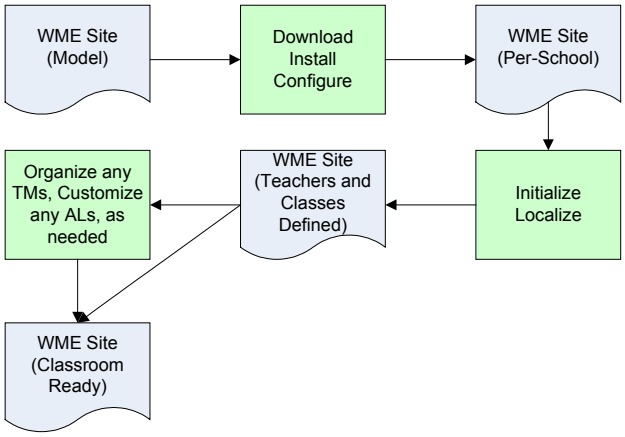

Figure 3: WME Site Deployment

2. WME Components and the Model Site Design

WME interoperability aims to make many of its inner components *easily* deployable into or removable from any WME system. Our approach to supporting this interoperability relies on hierarchically organizing WME into self-contained components, making them maximally independent of other components while explicitly defining their connections and dependence on related components. Such plug-and-play components are also made customizable by exposing their flexibility for user-defined changes. Our conclusions leading to the current WME component structure were reported in [1, 3, 4].

A complete copy of the model site can be easily downloaded and installed on any standard Web server. With some simple data entry, the site is further refined and tailored to a particular school. The model site contains two parts:

- Site administration and operations support. These include
  - Modifying school and WME system settings
  - User management
  - Classroom management
  - Topic module management
  - Lesson plan management for teachers

  A detailed description of these is given in the last section regarding customization.
- Educational content --- These are organized as topic modules (TM) under one directory. Each TM contains a sequence of active lesson (AL) pages and each AL focuses on teaching a particular mathematical concept or skill. ALs are further

broken down into independent lesson parts (LP), which contains both fixed and editable content.

Following the breakdown in Figure 4, at the root of the WME component hierarchy lays a TM. A TM is similar to a chapter in a book. It covers a topical area (such as integers or percentages) and includes an ordered sequence of interactive active lesson pages, or ALs (think of these as sections in a chapter). TM interoperability means any TM can be installed (or removed) and guarantees that the TM will operate with the host site seamlessly and becomes immediately available for teachers. Likewise, removal of any TM will not affect the normal functioning of the rest of the WME site.

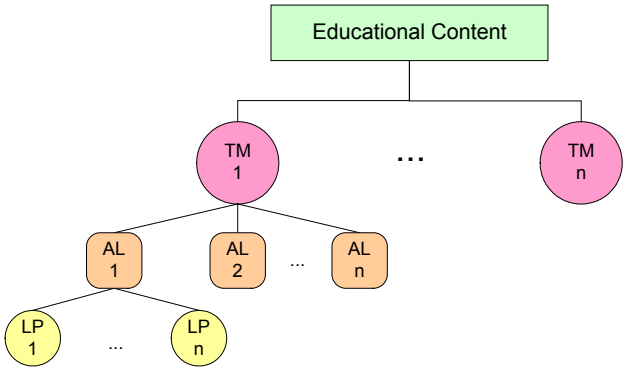

Figure 4: WME Component Breakdown

An AL is a self-contained unit that teaches a particular mathematical concept or skill and helps strengthen the student's understanding through hands-on activities and manipulation of tools, games, answering questions, or other interactions within the page. From an instructional standpoint, AL usage is non-linear. That is, a teacher can choose *where and when* to use an AL without predetermined constraints. For example, each TM may come with a large set of ALs relating to a particular area covered by the TM and with a suggested typical sequence of ALs to use for the classroom, but a teacher can easily change the sequence, add/remove ALs, or even introduce an AL from another module. AL interoperability states that an AL can be *dropped into a TM* at any time and its availability for use is immediate. Removal of ALs from a TM works as expected.

At the lowest tier in our TM hierarchy is the lesson part, or LP. These broken-up units contain the actual lesson content, including text, markup text (normally XHTML and MathML [12]), graphics, and manipulatives. LPs inside an AL are displayed one at a time to help focus student attention on the learning tasks at hand. Customizable contents in an LP are so designated and supported by JavaScript-defined editors.

The purpose of this component-based organization is to provide a sound structure of supporting customization. For example, teachers may tailor/edit ALs in simple but important ways such as rewording contextual phrases, modifying mathematical formulas, replacing images, adjusting parameters to manipulatives, and adding questions in the right places on a lesson page to collect student answers. Interoperability at this level includes the ability to add, remove, edit, and replace any LP unit in an active lesson.

The hierarchical structure has an underlying advantage in that, as long as these component nodes are self-sufficient, they can be added under their parent node for interoperation. Because TMs are modules that can be dropped in and out of any WME system, the model site anticipates a standard file structure for TMs. Due to this sensitivity, it is without question that their file structure must be followed rigorously upon creation. Figure 5 illustrates the TM file structure. Starting from the TM root directory, we see that each TM carries its own configuration file, *conf.xml,* which contains information that is read upon installation of the TM. Information contained in this file includes its module name, author, and a default list of inclusive ALs. The *index.php file, img/, css/,* and *script/* directories are meant to contain all necessary files of each type, thereby allowing the TM to be self-sufficient and independent from any WME server resources. AL directories are observed similarly. ALs themselves contain an inner configuration file, *conf.xml*, which contains information about the active lesson page such as a list of default LPs that make up its content, and what questions are attached to the LPs. LPs are themselves self-sufficient and independent from any TM resources because, as we mentioned before, they are mostly blocks of text or markup that defines/leads to content. Manipulatives are also independently deployable. Due to their complexity, however, we will next discuss manipulatives further with an in-depth look at their architecture to support their self-sufficiency.

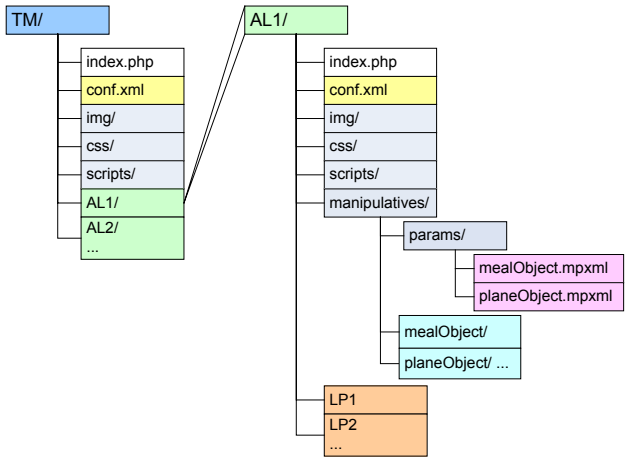

Figure 5: The Topic Module File Structure

3. The Manipulative Architecture and Interoperability

Manipulatives are hands-on exercises often designed as a game to help communicate a mathematical concept to a student both visually and interactively. Like Question Units, manipulatives themselves contain inner elements. These are parameter files such that, when supplied to a manipulative, can affect the manipulative's display, computation, and outcome.

Directly below, Figure 6 shows a manipulative in the *Dinning Out* lesson page inside the *Percentages* topic module. Like many other manipulatives in WME, it is implemented in JavaScript through the Document Object Model (DOM). We will revisit this example time and again in order to illustrate our approach for interoperability and customization.

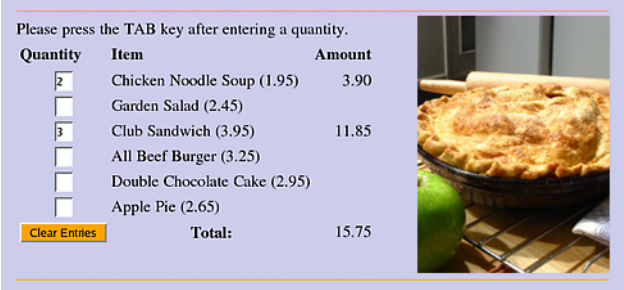

Figure 6: A Sample Manipulative

To achieve interoperability of WME manipulatives, each manipulative must be rendered into a self-contained program and object instance so they can be easily deployed in any AL and at any WME site. In fact, an AL may contain one or more instances of the same manipulative and the same manipulative may be deployed in multiple ALs. The containing Web page must also be able to interact with any manipulative instance through a well-defined interface. Furthermore, each manipulative must be customizable by teachers on a per-instance, per-course, and per-teacher basis. We will describe how JavaScript-defined manipulatives achieve these goals. Other types of manipulatives can follow a similar approach. Each manipulative is self-contained and has its own directory for all of its files. A JavaScript source file, *MealOrder.js* for example, which defines, among other things, a constructor, MealOrder. The *MealOrder.js* file is loaded into any page that will use the manipulative. To create a manipulative instance, we call the constructor:

```
american = new MealOrder('american');
```

The first argument to the manipulative constructor is always the variable name in JavaScript that holds the instance being created. If a second argument is supplied, then it must be an associative array of name-value pairs.

A manipulative instance, such as *american*, is deployed in a Web page by the code

```
<div class="manipulative" id="manip1">
  <script type="text/javascript">
    american.deploy("manip1");
   </script>
</div>
```

In this example, *manip1* is the ID of the * * element where the manipulative is deployed on the page. The class attribute *manipulative* allows us to set up CSS style rules for consistently rendering all manipulatives and easily customizing the style for different WME sites. Each manipulative must define these instance methods:

- **reset()** --- Re-initializes the manipulative instance.
- **getArg(***name***)** --- Returns the value of the named parameter.
- **getArgs()** --- Returns an associative array of the values of all parameters (in the same form as the array used to call the constructor).
- **setArg(***name*, *value***)** --- Sets the named parameter to the given value.
- **getProperties()** --- Returns an associative array of the values of all instance properties made accessible.
- **getProperty(***name***)** --- Returns the named instance property. For example, *total* (the total amount of the bill) is an instance property of a *MealOrder* instance.

Such instance methods can be accessed from anywhere in a Web page for interaction with the manipulative. For example, immediately after the *american* manipulative, the dinning out AL presents several questions based on the meal order. Such a question might be: "If *50%* of the cost of your meal actually pays for labor at the restaurant, what is the labor cost of your meal?" Here, the *50%* part in the question is customizable by the teacher so that a different percentage can be used. Its source code:

```
<span class="pageparameter"
      id="labor_percent">50</span>
```

This editable percentage value and the menu order total are automatically passed to the verifier with the code:

```
american.getProperty('total');

document.getElementById('labor_percent').fi
rstChild.nodeValue;
```

This architecture allows a manipulative to be easily deployed in another AL or even to a different WME site.

4. Manipulative Customization Support

A WME manipulative is made customizable by allowing settable parameters to be passed as arguments to its

constructor. Therefore, manipulatives can be programmed with much flexibility and generality. For example, we can make another *MealOrder* instance that displays an Italian menu (Figure 7) by passing to the constructor a menu and a picture as indicated by the following JavaScript code:

```
m = {
    'Italian Wedding Soup': 2.25,
    'Caesar Salad': 3.45,
     'Macaroni and Cheese': 3.45,
     'spaghetti with Meat Balls': 4.65,
     'Tiramisu': 2.95
};

im = "Italian_Waiter.png";

//values passed as associative array
p = {'menu': m, 'image': im};

//instantiate manipulative object
italian = new MealOrder('italian', p);
```

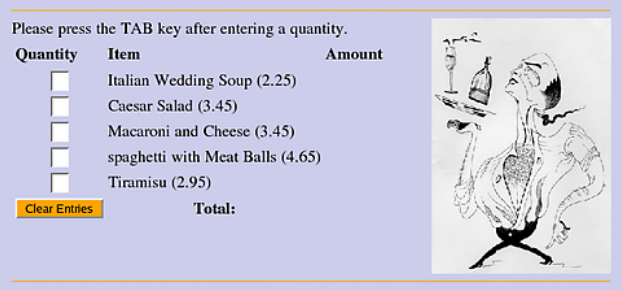

Figure 7: A Customized Manipulative

The manipulative architecture must support user-level customization as well. Each manipulative provides these static (class-wide) properties to help usage and customization:

- **instances** --- Is an array of ids (strings) for all existing instances.
- **displayType** --- Is either "block" or "inline".
- **location**---Gives manipulative directory location relative to site root.
- **document** --- Is a text string providing a brief API documentation.
- **help()** --- Returns a string for end-user help.
- **help("***topic***")** --- Returns a string for end-user help relating to the given topic.
- **defaultArgs** --- Is an associative array for the default arguments when the constructor is called with no arguments.
- An argument description file in XML---Provides information on parameter name, meaning, allowable values, to help provide end-user customization of constructor arguments. We describe this XML file next, and how it is used to provide manipulative customization on a per-instance, class, and teacher basis in the next section.

While WME developers might find such customizations powerful and convenient, we cannot make presumptions on teachers' familiarity with programming. Thus, we allow teachers to customize manipulatives using an easy GUI tool we have developed, which we describe late in the next section.

5. User Customization of WME

Every school may potentially have different grade levels, teachers, and even curricula. WME allows these settings to be made on a per-school basis. Another type of customization involves teachers' adjustments to their lesson plans and pages. This section gives an overview of the dual levels of WME customization.

5.1 Level 1 Customization: Schoolwide Administration

Information that varies across schools is unavoidable. This means that WME must be able to be tailored to every school on this level. In WME sites, there are three types of users: administrator, teacher, and student. Administrators have full control over the WME site. They are allowed the following actions:

- School Settings --- Modify school-specific data such as street address, pictures, logos, etc.
- WME Site Settings --- Modify WME site configuration such as database connectivity info.
- Grade Level Management --- Add/Delete/Modify grade levels in this school (e.g. in Kimpton Middle School, we only have 7th and 8th grade levels).
- User Management --- Add/Delete/Modify accounts, and grant user statuses.
- Course Management --- Add/Delete/Modify courses. Constitute what course is to be taught at which grade level and who is to be teaching it.
- Topic Module Management --- Install/Delete TMs for teachers to use in their classrooms.

Upon installation of the WME model site, a user account, *root*, is natively provided. This account is simply for administrators to initiate site settings, and after creation of administrative account, it is recommended that this account be deleted.

5.2 Level 2 Customization: Lesson Management

This level of customization is available only to teachers. In order to expose Level 2 customization in the WME site, the administrators will have already created teacher accounts, grade levels, courses, and installed TMs and ALs that came with the site. When these are done, the teacher, upon login, will be taken to their administrative area for lesson management. After selecting which course to manage, the teacher can decide which TM to use for that course. At this point the TM will contain original

copies of ALs, and the teacher may decide to use them as-is, or to customize any selected ALs.

TM Customization: Inside each TM, a teacher can pick and choose from the available ALs and decide on their sequencing. Thus, the resulting TM will contain only the ALs picked by the particular teacher and in the desired sequence for teaching the particular class at hand.

Figure 8 shows the TM customization interface for the Statistics module. The ALs will be listed in the customized TM by the integers their left. To remove a AL, simply leave the sequence number blank. By following any of the AL links on the list, the user is transferred to its customization interface.

AL Customization: After deciding the AL ordering and usage in the previous step, teachers can further customize its content. In the AL customization interface, a teacher can add content, reword text, reorder and manage and pose questions, and customize any manipulatives that are instantiated inside the lesson (notice that all of these operations are done on LP units that we prescribed to the architecture).

The modifications by teachers are stored in a per-WME site database and are dynamically applied to the AL or TM delivered depending on the teacher and class designations of the user (teacher or student). Currently, we utilize PHP and MySQL database for our WME sites. But the WME site design does not depend on these particular technologies.

Going further into page content customization, we look to controlling in-page manipulatives. In the following, we first discuss how manipulative customization is supported, then its interface for editing.

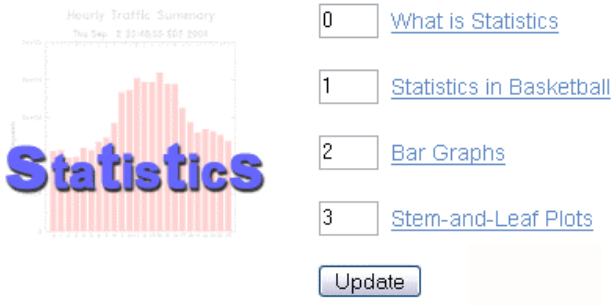

Figure 8: TM Customization Interface

Manipulative Customization with MPXML: A manipulative's display and inner computation can be altered on a per-object instance basis by passing parameters to its constructor. The modified parameter values must be recorded in the site database. The question now was how to store these parameters for every manipulative instance.

At first, a general colon-delimited format was considered for storing these values, but it proved to be too general when we wanted some semantics describing the parameter values. We then considered an XML-defined structure for the job. Other XML applications that can be used to describe source code and parameter data include srcML [2] and WSDL [7], but these seemed to be overkill for our simple purpose of editing and storing manipulative parameter values. In the end, we compromised and developed MPXML (Manipulative Parameters XML). This package includes:

- *The MPXML Specification* --- Used to describe manipulative parameters in a way that is generalized and independent from programming languages. Upon manipulative deployment, these values are read and used to instantiate the object instance.
- *A 2-Way JavaScript Translator* --- Because MPXML is programming language independent, a parser must be written to tailor to the language of choice (in our case, JavaScript). This 2-way parser takes JavaScript code and constructs an MPXML, and in the other direction, takes MPXML data and builds the JavaScript constructor call.

The MPXML definition is quite straightforward:

```
<manip_params>
<!-- The parameter is always treated as an
       associative array -->
  <param type="array">
     ...
     <!-- List of parameter values -->
     <param type="string"> ...  </param>
     ...
  </param>
</manip_params>
```

It is important note here that the `<param>` element assumes far more complexity. Its full specification can be found at http://wme.cs.kent.edu/help/docs. Below shows the MPXML description that renders the MealObject manipulative which we have been following into an Italian menu:

```
<manip_params>
  <param type="array">
    <name>
     <canonical>p</canonical>
     <contextual>
           Manipulative Parameters
     </contextual>
    </name>
    <items><!-- first item is the menu -->
     <pair>
      <key> <value>menu</value> </key>
      <param type="array">
       <!-- The parameter, p, is an
            associative array of menu
            items as keys with their
            respective prices as values -->
```

```
        <name>
         <contextual>
           Menu Items
         </contextual>
        </name>
        <items>
         <!-- First item on menu -->
         <pair>
          <key>
           <value>
            Italian Wedding Soup
           </value>
          </key>
          <param type="double">
           <name>
            <contextual>Price</contextual>
           </name>
           <value>2.25</value>
          </param>
         </pair>

         <!-- Second item on menu -->
         <pair> . . . </pair>
         <!-- More items not listed here -->
        </items>
       </param>
      </pair>
      <!-- second parameter, the image -->
      <pair>
       <key> <value>image</value> </key>
       <param type="string">
        <name>
         <contextual>
          Image to be used
         </contextual>
        </name>
        <value> Italian_Waiter.png </value>
       </param>
      </pair>
      <!-- end of associative array -->
     </items>
    <!-- end of parameter definition -->
   </param>
</manip_params>
```

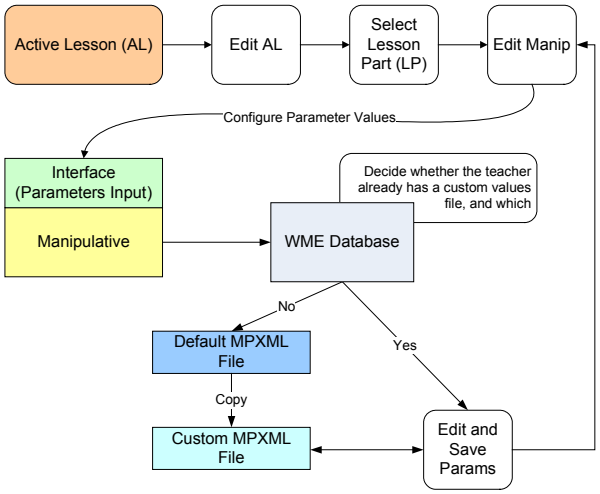

Figure 9: Decisions and Process Involved in Customizing a Manipulative

The above code will be translated into the equivalent JavaScript just above Figure 7. Note also that nowhere in the MPXML specification was the first parameter, `"Italian"` declared. This is because the first parameter is always implied as the instance variable, which we already know upon instantiation.

In order to abate complex workload and guarantee that a valid MPXML and JavaScript constructor call is generated, we created an intuitive user interface reads the corresponding MPXML file, displays the parameters that can be rewritten, and allows the user to input new values. Before saving, the user has the option of previewing and discarding the values. After committing to the new values, a new MPXML file is written, and the manipulative is reconfigured for this teacher's lesson page only. Figure 9 illustrates the process that the WME site undertakes during a manipulative modification.

The same procedures follow for American meal or any other meal menu that is created by a teacher or another source. As long as the parameter values in the MPXML file are set properly, the manipulative will be able to generate customized styles and content for each instance. Figure 10 illustrates the process of deploying a manipulative to an AL.

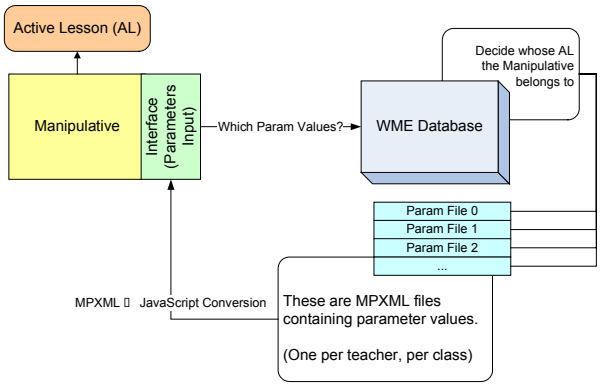

Figure 10: Manipulative Deployment

6. Conclusion and Future Work

Work reported here is part of the team effort by the WME group at ICM/Kent to create a practical, efficient and effective Web-based system to help teachers and students better teach and learn mathematics. Initial feedback from both students and teachers from trials of the pilot WME project were encouraging [5], and because of this, we were able to gain more collaboration from different teachers and schools in the area and even across continents (Lanzhou University in China). Recently, experts from the Department of Visual Communications and Design at Kent State University have joined the