Communications of JSSAC (2012)
Vol. 1, pp. 75 – 91

Design and Implementation of a WEB-browser Tool BrEdiMa* for Mathematical Expressions

Yasuhito Nakano†
Hirokazu Murao

Department of Computer Science, The University of Electro-Communications

Daisuke Morimitsu

Department of Computer Science, The University of Electro-Communications

(Received 29/May/2009 Accepted 22/Nov/2009)

Abstract

We have developed a new software tool, called BrEdiMa, to input and edit mathematical expressions on WEB browsers. BrEdiMa is implemented completely in JavaScript and therefore runs on most of the existing browsers, which implies it can be used independently of the types of WEB pages. It can be used both as an attached program to some HTML pages placed on servers, and also as userscripts on client side to extend the ability of browsers independently of any specific WEB pages. This paper describes the software design of BrEdiMa, and explains how it was implemented in some details and what WEB technologies are integrated into and how. To show its extensibility, we also explain how it can be utilized as WEB tools for blog systems and Wiki clones.

Keywords: WEB-tool, WEB-browser, editing, Greasemonkey

1 Introduction

In the last decade, communication via Internet has expanded explosively, and is still expanding. Most information transmitted and communicated is based on plain text, which can be input only with keyboard. On the other hand, mathematical expressions, which rarely appear in daily communication but are commonly used in education and academics, are difficult to input, and consequently, they are not suited for daily and smooth communication. Mathematical expressions are rarely treated in WEB pages, and even if treated, they are in their rendered images. This situation seems far from casual writing, and motivated us to develop yet another tool to support input and edit of mathematical expressions on the WEB. The tool is to be used on WEB browsers, and should

*`http://bredima.sourceforge.jp`

†Currently, Yahoo Japan Corporation.

be of light-weight, easy to use or suited for casual use, and, most importantly, highly conformable to Internet and other WEB tools. Some may think that TEX/LATEX is a common language to describe, and possibly to communicate, mathematical expressions. However, in the current internet community, most of users are accustomed to usual input method of word processing software, and therefore, input method in WYSIWYG style will be preferred to input by character strings of formatting commands.

There are many tools developed for similar purposes and available in public as listed later. Some of them are available as commercial products, and some of them require plugins for handling or display. Therefore, in order to install and use them, we usually have to go through some steps, which can often be a nuisance for novices. With this recognition in mind, we decided to develop our own tool for WEB browsers. It is called BrEdiMa, named from the related words of browser, edit and math. Our policy on developing the tool is easy to install with no hurdle, simple and easy to use with comprehensive interface, good accordance with browsers and no conflict with other browser tools. All these objectives are attained with BrEdiMa, we think, and BrEdiMa will be characterized by the following features.

- Installation is easy, because it assumes no prerequisite condition as plugins for some extended capability, and therefore it should basically work on most of existing browsers because it is implemented using only standard capabilities of JavaScript and by default, only HTML elements and images are used for display.

- Simplicity and conciseness: BrEdiMa treats a simplified set of mathematical objects up to high-school level, which can be understood by non-experts of mathematics, and the manipulating operations for editing are integrated into a simple GUI, where editing can be done in WYSIWYG style of two-dimensional layout.

- Output in MathML[1] is supported. If a browser used supports MathML, MathML output can be used for preview.

- Extensibility: BrEdiMa prepares well-defined API in JavaScript, and therefore, it can be applied in various ways.

For more details, refer to [2].

The features stated above describe the core part of BrEdiMa, and are nothing more than catchphrases to advertise our programming effort from the developers' point of view, where even users might ought to be termed supposed-users. Nowadays, most internet or WEB-browser users have no interest in technologies of computers, network and programming or their technical characters. Our target users would have little concern about most of the above features, but would have a concern about what can be done with BrEdiMa and whether it can be used on their own PCs or browsers. Intuitive operations introduced by GUI and two-dimensional layout of mathematical expressions as in textbooks will be of major necessity. Furthermore, most popular and common space to write texts in the internet will be blogs and Wikis for most casual users. If such authoring systems are compared to papers, writing tools like BrEdiMa are pencils and erasers. Any pencil can be used for any type of paper and various media, and will be chosen in users' favor. While it is not clear with the existing tools mentioned above whether such a separation is possible and whether they can be applied to blogs or Wikis, our BrEdiMa can be used as an independent front-end tool of browsers and will conform very well to blog systems and Wiki clones.

BrEdiMa is a JavaScript program of so-called WEB-tool used to add and support a special behavior of browsers. At the time of initial development, BrEdiMa was a unique WEB tool for mathematical expressions, and later on there has been published MathEdit[3] being developed in

a similar direction to BrEdiMa. There are two ways to install and use JavaScript programs into browsers. The conventional way is to put programs in servers with HTML files which refer to the program and to be loaded into a browser when the WEB page is browsed. Another method is rather new and to prepare JavaScript program as userscripts on client side. To which page to apply will be specified by users. The latest sophisticated browsers provide extended facilities to execute JavaScript programs as userscripts to add some capabilities. Greasemonkey is one such example for the browser Mozilla Firefox. The latest version BrEdiMa is fully revised to conform to Greasemonkey userscripts, and thus can be used in both ways. In the sense of the above comparison to paper-and-pencils, use as userscripts will bear an important role, because it can be easily replaced by better-made tools, if any, promptly, which will reflect the software quality of the tools. In either case, source codes of JavaScript programs must be open to users, and therefore, it is possible for users to modify and improve by themselves.

Related works. The existing tools take various forms depending on the methods for input and rendering. A key factor of the difference is the technology for rendering. It is closely related with which side of server or client to process, and also the language to use for implementation. A well-known technique used in mimeTeX [5] for rendering is the image generation by means of TEX via CGI. There are similar examples such as imgTeX [6] and texvc [7]. ASCIIMathML, implemented in JavaScript, converts plain texts of mathematical expressions into MathML expressions and relies on browsers or plugins rendering. The W3C graphics standard SVG, supported by Firefox and a plugin by Adobe for Internet Explorer, is used by a GUI-editor sMArTH along with the application logic implemented in JavaScript. Flash can be an alternative technology for rendering and used by a commercial GUI editor MathIWYG. For more softwares and details, refer to the software list page of the WEB page of the W3C MathML[8], and the corresponding pages linked from the list.

Organization of the paper. The rest of the paper is organized as follows. Section 2 describes the overall design of BrEdiMa, in relation with technologies used, and Section 3 discusses implementation details, which include internal data structures and objects, two-dimensional layout of display, and editing mechanisms. Section 4 explains how to integrate WEB-browser tools. Finally, Section 5 summarizes the paper.

2 Overall design and technologies used

As was already explained, our primary design goal is as simple as realization of intuitive operations and easy introduction and installation. For this purpose, our software must be free of charge, which implies no relying on non-free softwares, may not request to browsers additional capabilities such as plugins, and should realize smart GUI and editing in WYSIWYG style. In order for effective development, it is important to fix design policy of technical aspects and overall design of the software, based on the investigation of required and appropriate technologies and with unbiased future prospects. In our case, they are as follows.

(a) programming: use JavaScript as a programming language and program in object-oriented style.

(b) internal data structures of mathematical expressions: represent mathematical expressions internally using MathML-like objects.

(c) user-interface: realize two-dimensional input with GUI.

(d) rendering: real-time rendering with full ability of formatting .

(e) extensibility: modularize the whole program to be utilized in various ways, and also enable it usable with other capabilities or utilities, such as AJAX.

Some more details are explained below.

(a) Programming

Programming in JavaScript enables us, without reloading Web pages, to generate and modify HTML elements dynamically on the client side, and also to communicate with server, mainly for obtaining rendered images. Recent advent of GoogleMaps and its famous technology, called AJAX(Asynchronous JavaScript and XML)[9], has revived programming in JavaScript, and has popularized serious programming in JavaScript. Formerly, it has been usual to use Java to develop such programs as our target. However, recent accomplishment of software development in JavaScript revealed that it not only equips with sufficient programming capabilities but is suited for casual use and also for extending browsers' capabilities. If we develop our tool in JavaScript with AJAX-ready interface, we can use it as an user interface or a front-end with editing facility for other software, e.g., database retrieving system.

Notice that, although JavaScript is used in common to various browsers, it is browser dependent in some implementation details. We target most of popular browsers, including Internet Explorer on Windows, Opera, and Mozilla Firefox.

The object-oriented property of JavaScript is termed "*prototype based*", and is different from "*class based*" in that a new object is created from the existing one, not as an *instance* of a template *class*. As explained in Subsection 3.1, we introduce a notion of class to represent a set of objects composed of multiple parts of mathematical expressions, by preparing an existent object used only for a template. Inheritance is implemented by preparing a function to copy some part of a parent object into the corresponding part of a child object.

(b) Internal data structures of mathematical expressions

We define some kinds of objects as described in Subsection 3.1 to represent mathematical structures and also to support two-dimensional layout. We represent mathematical expressions internally by those objects structured hierarchically, which just follows the structure of MathML's presentation markup, while the whole document of a page is represented and handled dynamically as a straight DOM tree[10].

The structure defined by MathML provides sufficient capability to represent any mathematical expressions, and also makes it easy to output in various ways.

In the case of making the internal data structures externally visible, we represent them in a standard method of JavaScript Object Notation, JSON[11].

(c) User-interface

Mathematical expressions have specific structures and are displayed in two dimensions. Layout of expression parts can be specified by one-dimensional character sequence as in LaTeX, however, casual users would prefer WYSIWYG editing style. Therefore, two-dimensional input and edit in association with graphical user-interface, GUI for short, will be of primary necessity. GUI must spread a work area to construct mathematical expressions from subparts arranged in two-dimensions, and also, must have a list of buttons for special symbols and mathematical structures to insert. In addition, preview facility during editing will be useful to check the expressions under construction, and is expected to be present. Preview with typeset image will give look-and-feel of

the expression inserted in a text, while the source in LaTeX or MathML will indicates the precise structure of layout of the subexpressions. In order to display all those areas, GUI must occupy some large amount of display area on the browser, and maintain multiple display areas, at least for the buttons and the work area, while the preview area may be optional. Notice that the size, height and width, of the work area would have to change, and may grow beyond those of the browser's display area, if the input expression becomes very large. Appropriate treatment for those cases of large-scale expressions will require full typesetting capability, and therefore is far beyond our scope.

After the experiments with our first version, which lists all the icon images of buttons horizontally and arranges the areas for the list, the work area and previews of the rendered image and the source without overlapping, using a full WEB page, we have arrived at a smart resolution for compact arrangement of display. Fig. 1 of the screen shot of the normal state of BrEdiMa indicates the resolution.

- The list of buttons, most of which are used not very often, are classified into a few groups and compacted into menu items.
- Preview with rendered image is useful, especially with real-time update, and a certain amount of display space for it is allotted to make it visible simultaneously with the editing expression in the work area. The space itself will appear and disappear by clicking the preview button.
- On the other hand, the source listings in LaTeX and MathML are treated optional and displayed on request by the corresponding button on a separate sheet overlaid upon the work area, because the details of display structure of the expression is rarely checked while editing.

Fig.2 and Fig.3 are the sample screen shots from the current version of BrEdiMa.

(d) Rendering

For real-time preview to check the editing expression, we prefer typesetting quality formatted in two-dimensions, as is done with LaTeX. As for the full rendering capability of mathematical expressions, we rely on the existing free software; browsers' capability to render MathML, maybe with the help of plugins, and the image generation by mimeTeX. In either case, it is possible to catch up immediately with the the change of the editing expression. Direct use of MathML is useful because it enables us the followings:

(1) to change arbitrarily the size like other plain texts,

(2) to print in printers' full resolution, and

(3) to reuse and edit by dragging and copying by mouse.

However, while graphic image is supported by most of the existing browsers, support of MathML is quite limited; Firefox, with the request of additional fonts in some situations, provides a good support, and Internet Explorer and Opera need the help of plugin, such as MathPlayer. We use both methods, and they are switched by users' choice. If the image generation by the use of mimeTeX is chosen, no additional plugins and fonts is required. Correspondingly, we support output both in MathML format and in LaTeX. Rendering facility is also used for entity references of special symbols and extended characters which cannot be input direct from keyboard.

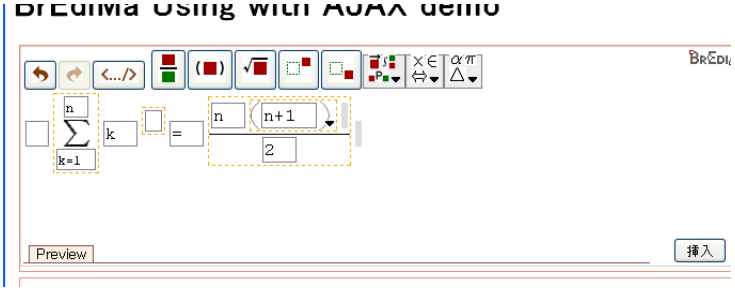

Fig. 1: Screen shot of the normal state of BrEdiMa.

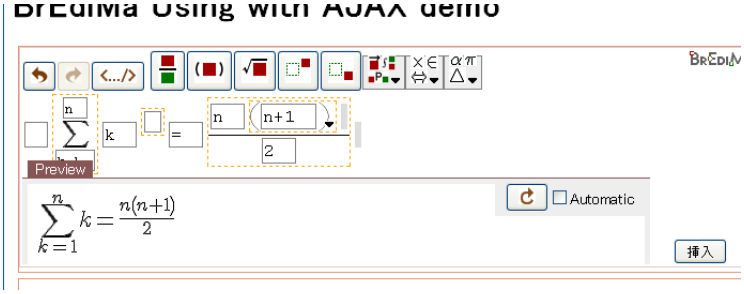

Fig. 2: Editing with preview of rendered image.

Fig. 3: Display the source in MathML of the editing expression.

(e) Extensibility

In the current situation that the treatment of mathematical expressions on the WEB is not fixed yet although its format has been almost fixed to MathML, putting editing function into front-end software makes it easy to apply to various kinds of WEB pages as blogs and Wikis, and will be more promising for future extensions rather than to build it into specific softwares. The choice of writing tool ought to be independent of the type of written pages or media, as any pens or pencils can be used for any type of ordinary papers. However, treatment for mathematical expression editing seems too heavy to be integrated into a single independent front-end software as Japanese input system, because it must equip with facilities of two-dimensional layout and further of rendering with almost typesetting quality, and also it requires display space spreading in two-dimensions. In our case that the tool is used with WEB browsers, it should work in collaboration with a browser in which it executes, as its extension, with some help of standard WEB technology if required.

While such an extension was implemented as so-called plugin formerly, nowadays, sophisticated browsers provide, as such extensions, general-purpose environments to execute users' programs. The most famous will be Greasemonkey [4], an add-on for Mozilla Firefox, to execute JavaScript programs, called userscript, prepared in some fixed format and coded under some simple restrictions[12]. For Internet Explorer, there have been developed equivalent environment such as Greasemonkey for IE[13] and Trixie [14], or similar extensions for programming such as Turnabout and GreasemonkIE. Notice that such userscripts are introduced and activated by users on client side on their own responsibility, and to which WEB page to apply is determined by users. Greasemonkey is chosen for our first-time experiment, and we will cleanly modularize the BrEdiMa program so as to match Greasemonkey and to be easily adapted to various kinds of WEB pages, blog systems and Wiki clones. Note that userscripts are readable and editable by users, which enables users to extend and apply or modify in their own ways. At the same time, it is demanded for BrEdiMa to be independent from, not to interfere and to be able to coexist with other similar extension tools in order to retain future extensibility.

Mathematical expressions within our scope

The final decision we made is on the kinds of mathematical expressions to be handled with our software. We treat only those elementary expressions that appear in textbooks of high-school level. Notations and symbols used in higher-level mathematics are not fixed, and used and defined freely (differ person-to-person). We may expect that those who require such special notations often be experts of TEX/LATEX, and they cannot be a main target of our software.

3 Implementation details

3.1 Internal data structures and objects

To represent mathematical expressions of editing mode in JavaScript, we define four types of objects, *MathInput* object, *MathRow* object, *Container* object and *Token* object.

- *MathInput* object corresponds to a token element of MathML and is a set of character objects which can be input direct from keyboard. Input is done via *textarea* element of HTML as plain text. Key input is monitored to detect the characters of parentheses `(` and `)`, uparrow `^` for superscripts and underscore `_` for subscripts, and convert them to the corresponding *Container* objects; parentheses to *MathFenced* objects, and super- and sub-scripts to *MathSup* and *Math-*

Sub. Text strings corresponding to special symbols or extended characters, to be represented by *MathString* object, are also detected.

- *MathRow* object corresponds to MathML's `mrow` element, and in some cases to braces in LaTeX. Internally, every *MathRow* object holds an array of other kinds of objects which compose a single mathematical expression.
- *Container* object is used to represent a class of objects that have mathematical structure, such as rational expressions, and contains at least one *MathRow* object of the constituents. Every constituent is input as a corresponding *MathRow* object.
- *Token* object represents such token elements of MathML that do not correspond to any single character input from keyboard and are therefore hardly handled by *MathInput* object. In an HTML document, it generates a `SPAN` element containing a symbol or a character to display. *Token* objects are further classified into *MathSymbol* and *MathString* objects. The former is for entity references which cannot be input direct by keyboard, and the latter for special names of mathematical functions such as sin, cos, log and so on. While for every *MathSymbol* object BrEdiMa prepares an input button in the editing area, *MathString* objects are input by detecting and replacing the *MathInput* objects of special text strings created in response to the incoming plain texts. In order to display *MathSymbol* objects, we define an object *Symbol* which holds the respective method of rendering, and attach its occurrence to each *MathSymbol* object.

Every mathematical expressions treated in BrEdiMa is represented internally using these objects. The structured data may be represented by character strings using JSON [11]. JSON, which abbreviates "JavaScript Object Notation", is a lightweight data-interchange format, and is used in BrEdiMa to save and restore mathematical expressions into and from external files. Especially, to parse JSON data, a library program is provided and used in BrEdiMa.

3.2 Two-dimensional layout of input area

Input and editing is done in the work area of the BrEdiMa screen. Each object occupies its own rectangular area in it, and is displayed by arranging its descendant objects inside the rectangular area, recursively following the object hierarchy. The rectangular area of *Container* object is displayed with dotted lines, as can be seen in the screen shots (e.g. Fig.'s 1,2 and 7), to indicate the area and the contained objects, which will be helpful in editing.

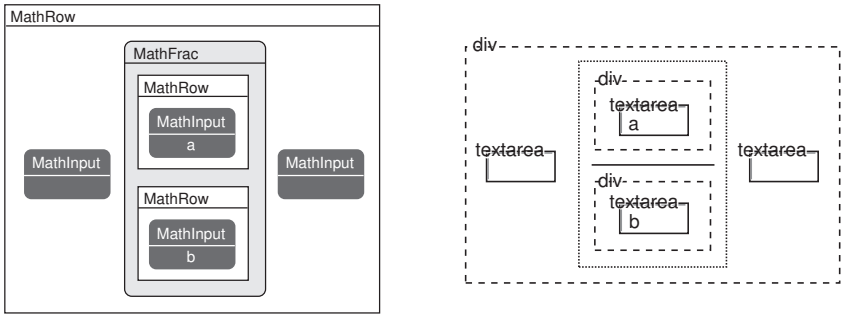

Fig. 4: The structures of objects and HTML elements of $\frac{a}{b}$.

Fig.4 gives a simple example to depict the internal structure of objects and the display layout. Each *Container* object generates a `DIV` element as its own area in an HTML document, and

arranges the elements of its children inside the `DIV`. The arrangement depends on the kind of *Container* object. For example, *Container* object of rational expressions arrange two sets of elements contained in the *MathRow* objects of the numerator and the denominator aligned vertically, and that of square root places the elements of the *MathRow* object on the right of a radical sign. Each *MathRow* object also generates a `DIV` element, and arranges the elements of its children inside the `DIV` from left to right.

Notice that inside the work area, two different font sizes are used to render *MathInput* objects and *Token* objects. The style of the expression being edited is expected as similar to the formatted image as possible, however, too-small images are hard to recognize and edit. Also, we rarely need multiple sizes because expressions we treat are limited to those appear in high school text in our implementation. We decided to use only two sizes. Compare the visibility and the understandability of the following, and we will see that our choice (the middle) is appropriate:

$$a^{b^c}, a^{b^c} \text{ and } a^{b^c}.$$

In this situation, vertical positioning plays an important role for understandability. We compute the heights of the expressions above and below the level, and arrange the expressions in a *MathRow* object so as to horizontally align their centers.

Adjustment of object layout

Every time the content changes, the current *MathInput* object adjusts the size and the alignment and redraws the content to get a new layout. This adjustment is repeated for all its ancestors, until the outermost object is reached. This can be done by calling a method `layout()` which performs the adjustment of its own object and then calls the method `layout()` of its parent. Fig.5 depicts this process of adjustment; (A) two '0's are added in the denominator, (B) the method of the Container object of the rational expression realigns the numerator and the denominator and change the size of the display, and (C) the size change in (B) affects the positions of the objects on the right.

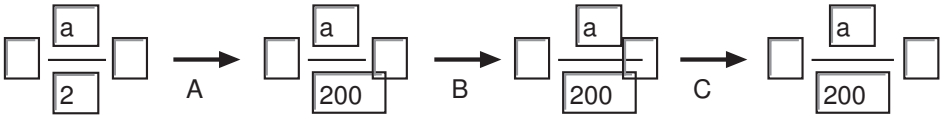

Fig. 5: Process to adjust object layout.

3.3 Editing: insertion and deletion of objects

For *MathInput* object, we use *textarea* element of HTML instead of a usual method by an *input* element with `type="text"`>, in order to maintain the cursor position without being disturbed by the transfer of the focus. We suppress the emerging of scroll bar for *textarea*, by the help of CSS.

A new object is inserted by clicking an object button or by typing a special character at cursor's location. Then, the text string of the *MathInput* object on cursor is partitioned and a new object is inserted between the parts, as shown in Fig.6, which is performed by an `insert()` method of the *MathInput* object.

For every *Container* object and for every *Token* object, we put *MathInput* objects on the left and the right, so as to enable insertion at arbitrary places in the edited expression. This flexibility may seem inconsistent with structured editing, but is important to ease editing and to attain higher

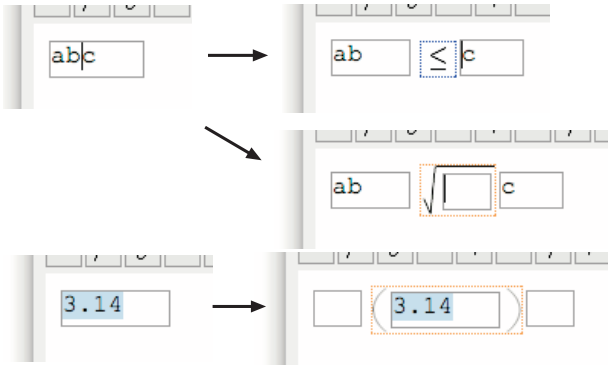

Fig. 6: Insertion of a *Token* object (top), a *Container* object (middle) and bracket-pair object for a selected text (bottom).

usability. *MathInput* objects serve to showing possible places for insertion, not merely as place holders for input. The above flexibility slightly complicates the control of cursor movement. Outbound cursor movement from a *MathInput* object by cursor (arrow) key invokes the corresponding method of the *Container* object of the parent. This invocation is transmitted to the ancestors until an appropriate neighboring *MathInput* object is found in the *Container* object, in which case the method `setCursor(0)` of the object is invoked to set a new cursor position.

Backspace at the left edge of the *textarea* of a *MathInput* object deletes an object on the left. This is done by invoking the method of the parent *MathRow* object for delete, which will call the method `removeObj()` for removal of the target object. If the target is a *Container* object, all the contained objects are deleted, as shown in Fig.7.

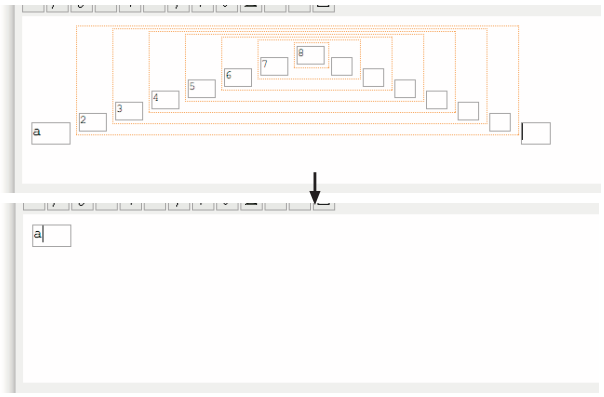

Fig. 7: Deletion of a *Container* object having multi-level descendant.

3.4 Editing: undo and redo

With usual softwares with GUI, the history of all the operations executed in response to the events from keyboard or mouse will enable undo and redo of editing. Undo will be realized by performing

the inverting operations from the last. In our cases with BrEdiMa, a JavaScript program running on WEB browsers, events are handled mainly by browsers, and script programs can detect and handle only a limited kinds of events. In order to realize full undo/redo capability, on every event, the data structures of the current working object before and after the event will be memorized. BrEdiMa's internal operations for editing can be classified into two groups; insertion/deletion of a *Container* or *Token* object into/from a *MathRow* object, and insertion/deletion of characters in the input of a *Token* object for typed characters.

As for the operations of the former group, because they are implemented completely as a single function in JavaScript and within JavaScript's capability, the working object to be kept will be easily obtained, and we have only to save the *MathRow* object being manipulated. The data structure prepared for input and output in JSON style will be used to retain the objects. For the operations of the latter group, copy from clipboards, drag-and-drop, and so on must be considered, besides character input from keyboard that can be treated up to a point with JavaScript programs, and all possible operations can hardly be treated properly. Our simple resolution is to check and detect the change of the input of the working *Token* object, which gives rise to an event, every time there arises any of the input operations and events under the control of our program such as typing of a single character or mouse-cursor movement, by comparing the current state of the object and the previous saved for the moment. If any change is detected, both data will be retained as a history for future undo/redo. This simple method cannot be complete, but turned out satisfactory in practice.

4 Integration as a WEB-browser tool

Initially, BrEdiMa was implemented as a material for programming experiment in JavaScript. The success of this experiment has driven us to proceed to the next step. Document writing environments used easily and daily appropriate to check how far our initial objective is accomplished will be blogs and Wikis. Next good attempt would be to adapt BrEdiMa to the existing WEB environment, and we develop a plugin for a Wiki clone. This attempt will be useful not only for checking the ease and comfort of using BrEdiMa, but for investigating how the program should be organized and what settings ought to be left modifiable. This investigation can be exploited for further development for various blogs and Wikis.

4.1 Simple experiment: development of a Wiki plugin

The Wiki clone chosen as our target is FreeStyleWiki, which equips with a LaTeX plugin and seems easy to introduce new plugins because it is highly modularized. The plugin we implemented consists of two major parts. The one is a button of '*input of mathematical expressions*' on Wiki's editing page to initiate BrEdiMa with opening a new window, and the other is a button of '*return expression and quit*' on BrEdiMa to quit and exit from the running BrEdiMa. The expression returned is in LaTeX adjusted to the LaTeX plugin, and inserted at cursor's position on the Wiki page for editing. The plugin in such a pure sense that it extends the action of a Wiki program consists only of the addition of the button, whose unique code piece amounts to only 10 lines at most. Also, the modification of our tool is done with several lines, and is almost independent of the types of Wiki.

Totally, the amount of code implemented is quite limited, and its Wiki-dependent part is very small. From our experiment, it turned out that BrEdiMa can be adapted, with a small amount of change, to software tools, including WEB tools, with similar properties, and that the codes need change are summarized as simple as the following.

- preparation of a clickable button to invoke BrEdiMa,
- settings of the locations of mimeTeX server and the images of icons and others used in BrEdiMa, and
- preparation of a clickable button or a method to quit BrEdiMa by emitting the editing result to the parent WEB page, which requires the following mechanisms:
  - conversion of mathematical expressions to appropriate forms, character strings in LaTeX with some slight extensions or in MathML.
  - methods to probe the *textarea* where the resulting string is put into,
  - and to insert the string.

4.2 Greasemonkey userscript for blogs and Wikis

One of the world-wide most popular WEB-browsers (or, according to some reputation, the unique such browser) is Mozilla firefox. This browser has an add-on facility to customize or extend. Greasemonkey is one such add-on that allows users to install scripts, called (Greasemonkey) userscript, written in JavaScript. While JavaScript programs are usually included in or attached to HTML pages and therefore kept on server side, userscripts are developed independently of any particular WEB pages or their designs, and prepared on browsing side of clients. Namely, userscripts provide a method to modify or add some capabilities to WEB pages, without changing the originals, by manipulating the DOM trees of the pages.

Recently, blogs and Wikis have become a simple and easy-to-use communication tool, so that they allow anonymous users on client side to edit and write documents easily without worrying about server-side matters. As its extension, we might have a desire for a simple but good writing tool at our own browser side. Treating mathematical expressions is popular for some people, but is not necessary for everyone. So, with a mathematical writing tool, installation as a userscript will be a good choice. Preparation of Greasemonkey userscript to introduce BrEdiMa to blogs and Wikis as an input method will be useful and also a good practice of adaptation of BrEdiMa. We consider how it can be realized.

4.2.1 Adaptation to Greasemonkey

There is some restrictions on JavaScript programming for Greasemonkey userscript, but they are quite limited. Based on the following considerations, the BrEdiMa program was fully revised to become such a single module that can be utilized both for a Greasemonkey userscript and for the ones to be installed on server side.

- A userscript must be contained in a single file.
- Eliminate the use of HTML and CSS used for display, and program completely in JavaScript; replace HTML elements by corresponding DOM manipulations, and for style-sheets, generate dynamically by program to be inserted into the HEAD elements of DOM trees.
- As for images of icons of buttons, reference externally to the files installed separately on some server, and the URL is to be written in the userscript as a configuration parameter. This external reference might be eliminated by generating required images by JavaScript program using canvas elements or alternatives of HTML, supported by current major browsers. However, using such non-standard capabilities requires the use of library to wrap the capability, and also, complicates the script. This is the rationale of our decision.

- Add a specific prefix string to the external names of variables, functions, and so on used in the userscript, in order to construct its own namespace and to prevent name conflicts with other scripts. Note that introduction of scripts by users (and by users' responsibility) might increase a chance of name conflicts.

As mentioned above, the latest BrEdiMa provides only a single version to be used in common for userscripts and for server-side scripts, and for which script to use must be specified as an argument at the time of creation of a new instance of the BrEdiMa object. There are some other setting parameters prepared for each instance to change the configuration.

4.2.2 Sample userscript for blog, Hatena::Diary

We explain how to describe Greasemonkey userscript briefly, using the one for a well-known Japanese blog site, Hatena::Diary in Fig.8. In the following, $\ell.xx$ indicates the line number or the line numbered with xx in the figure. We need the following information in advance.

- Can the target sites of blog or Wiki treat mathematical expressions? Do the sites have prepared any method to render mathematical expressions, such as plugin to make use of mimeTeX or to obtain the formatted image of mathematical expressions, or do the sites accept and render MathML expressions?
- If possible, in what style mathematical expressions are to be represented, LaTeX or MathML or some other format?
- To what part of the DOM tree of WEB page does the input expression ought to be added?
- Specification of target URL's of the script, which can be included in the file or specified in runtime configuration.

A userscript file is composed of three portions.

(a) Meta data of the script (ℓ.1~6), where ℓ.5 specifies the target URL's.

(b) Definition of BrEdiMa itself(ℓ.8). By convention, the body of a program is encoded into a single line even if it is lengthy.

(c) Description of various settings (ℓ.10~41). The statements for settings are packed into a single function with no name (ℓ.34~40), will be executed at the loading time(ℓ.11) of the file of the script. Brief explanations are included as comments in the listings, and some more details are given below. Also, refer to the BrEdiMa reference prepared online at [15].

  - Specification of default values of the class, mainly for external resources (ℓ.17~19); a mimeTeX server and the images of icons and others used by BrEdiMa. Configuration required when used in a Greasemonkey userscript is to be done in the class, not in each instance(ℓ.20).
  - Installation of a button to be used for invoking BrEdiMa (ℓ.24~29). ℓ.24~26 generates an image element used as the button. The image is listed along with other edit buttons (ℓ.29) defaulted by the Hatena::Diary pages, where the HTML element of the list is identified by the name `edit-buttons`. The button image is connected with the BrEdiMa program next.

```
// ==UserScript==
// @name          BrEdiMa GM template
// @namespace     http://bredima.sourceforge.jp/
// @description   Includes GUI Math Editor into ??? edit page.
// @include       http://d.hatena.ne.jp/*/edit*
// ==/UserScript==

Bredima= ... /* Definition of BrEdiMa itself */

window.addEventListener(
   'load',
   function() {
      // check applicability.
      if (!document.getElementById('edit-buttons')) return;
      // default values of the class:
      // URLs of mimeTeX server and of the images used by BrEdiMa.
      Bredima.setConfig('uri_mimetex', '/cgi-bin/mimetex.cgi');
      Bredima.setConfig('uri_img',
                        'http://bredima.sourceforge.jp/pub/img/');
      Bredima.setConfig('isGM', true);
                           // used in a Greasemonkey userscript.

      // generate an image of a button to start BrEdiMa.
      var button = document.createElement('img');
      button.src =
          'http://bredima.sourceforge.jp/pub/pubimg/gm_hatena.png';
      button.style.marginRight = '5px';
      // put the button in an appropriate place in the page.
      document.getElementById('edit-buttons').appendChild(button);
      // generate a BrEdiMa object connected with the button.
      var bd = new Bredima(button, 'float');
      bd.setConfig('use_button', true);
      // treatment of input mathematical expression.
      bd.onsubmit = function() {
         var form = document.forms.namedItem('edit');
         var area = form.elements.namedItem('body');
                                       // location to insert
         Bredima.insertTo(area, '[tex:' + bd.toLatex() + ']');
      }
   },
   false);
```

Fig. 8: Greasemonkey userscript of BrEdiMa for a Japanese blog Hatena::Diary

- Generation of an instance of the BrEdiMa object (ℓ.31~32). As in ℓ.31, in the case that *'float'* is given as the second argument, a new BrEdiMa instance created will be connected to the image element of the first argument for a startup button; clicking the image invokes the BrEdiMa program and opens a new editing area in a 'floating' style. In other cases, the first argument is a DIV element of the DOM of the page, to which an editing area will be attached. To start the program after the initialization by opening a new editing area, an instance method must used explicitly, in this case.

  Default settings of the new instance can be changed(ℓ.32). Turning on the switch *'use_button'* lets an "insert" button emerge in the right bottom of the editing area for insertion of the edit result to the DOM of the page in a way specified next. Clicking this button can be detected by the *onsubmit* event of the instance, thus enabling to specify how to treat as in ℓ.34.
- Treatment of the mathematical expression of the input/edit result (ℓ.35~36). Call to the method *insertTo()* causes the insertion of a text string of the second argument into the area of the first argument. In this case of ℓ.38, the LaTeX representation of the input expression surrounded by "*[tex:*" and "*]*" will be inserted into Hatena::Diary's editing area, the *textarea* element with name *body* in the *form* element with name *edit*.

4.2.3 Tips for more extensions

Userscripts for some kinds of Wiki clones can be easily developed as a slight modification of the userscript of Fig.8. Major parts that require changes are the location of the BrEdiMa start button, the insert area and the format of the mathematical expressions of the result, and the change depends on the kind of Wiki and differs Wiki-to-Wiki. The format of mathematical expressions is most obvious. The following table summarizes the format.

blog/wiki	format
Hatena::Diary	`[tex:` *MathExpr-in-LaTeX* `]`
FreeStylewiki	`{{ mimetex` *MathExpr-in-LaTeX* `}}`
Mediawiki	`<math>` *MathExpr-in-LaTeX* `</math>`

Notice that LaTeX format is used in common, although the standard format of mathematical expressions on the WEB has been fixed to MathML. The reasons for this will be the facts that MathML is supported by a few browsers, natively or by means of plugins, and that the problem of rendering as its consequence causes blog and Wiki systems treat mathematical expressions as extended features not as basic. More significant problem is the input of MathML expressions. MathML is not for hand-writing, and is not suited as it is for such casual text-writing systems. Representation in LaTeX is easy to write and render, but not very popular except for experts or in academic. In this situation, BrEdiMa can be a good choice of software bridging non-experts to those casual systems over the difficulties mentioned above.

As for the location to insert the resulting expression, an appropriate *textarea* can be probed as follows, so far as blog or Wiki systems are concerned, because only one *textarea* element of a form element is used in the editing page of those systems.

```
var form = document.forms[0];  var area;
for (var i=0; i<=document.forms[0].elements.length; i++) {
    if (document.forms[0].elements[i].type=="textarea") {
        area = form.elements[i];
        break;
    }
}
```

If it fails, we have only to add some unique id to the *textarea* and refer to the area by the name.

Currently, the input format of expressions supported by BrEdiMa is limited to JSON because BrEdiMa does not have parsing capability, and thus, reediting expressions once stored is possible only for those in JSON. This means that BrEdiMa cannot be used directly for editing the mathematical expressions stored in the blog or Wiki pages. The problem of reediting in blog and Wiki is not so simple as only with this format problem. In the editing mode of blog or Wiki, all the contents of a page, including mathematical expressions, are treated simply as a plain text, within a *textarea*. We need a mechanism to extract mathematical expressions in the edited text, on which BrEdiMa can focus. Furthermore, there may exists in the text multiple instances of mathematical expressions. We also need a mechanism to choose and change the focus. A smart possible mechanism will be something like drag-and-edit; character string in the dragged area will be sent to some program and replaced by its result. Usually mathematical expressions contained in a document text are treated separately from the normal texts as in TEX family. From this viewpoint, we immediately notice that in the edit mode of blog and Wiki, an extended mode to edit each occurrence of mathematical expression is necessary. In order to realize, considerable modification is necessary upon the existing blog and Wiki systems and their design. The test example for extension with AJAX described next might give a direction for resolution.

In order to test the coexistence or the collaboration with AJAX, we have implemented a WEB page for remote editing. The WEB page lists up the mathematical expressions stored on the server in both LATEX and JSON, where the images of the expressions generated by some mimeTeX server are used for display, and once one of the expressions is chosen to edit, a new BrEdiMa will be invoked with the corresponding JSON data and the image on display will be replaced by BrEdiMa's editing area. On completion of editing, the result will be sent back to and stored in the server. The communication with the server is done asynchronously (without page transition, of course) using AJAX ordinarily; no particular programming is required. The actual working page is available at the demonstration page on the distribution site.

Another important programming item is handling of events. On completion of editing indicated by clicking the insert button, onsubmit event will arise with a BrEdiMa instance, which should be handled as in the former example in Fig. 8. Also, any change in BrEdiMa's input area can be detected by onchange event, which enables real-time check of the input.

5 Summary

We implemented a new WEB-browser tool, BrEdiMa, for mathematical expressions, and its details in design, implementation and applications are described in this paper. It is characterized by the use of JavaScript in object-oriented programming style, and the alternative use of MathML and mimeTeX for image generation. All editing and input functions, except the image generation by LATEX, are implemented by JavaScript and will be executed on a client side. Therefore, it is expected to be used in various scenes of WEB services. Especially, our effort to implement userscript for blog systems and Wiki clones will enlarge the chance and utilization. Also, because the JavaScript source code is necessarily open, we may expect its improvement by users. Even the latest version of BrEdiMa cannot be complete and lacks many standard capabilities for editing. To utilize BrEdiMa more widely and deeply, further investigation will be required, especially for collaboration with browsers and other tools. A general mechanism for cut-edit-and-splice of mathematical expressions will be an interesting subject to consider and experiment.

References

[1] Mathematical Markup Language(MathML) Version 2.0. W3C Recommendation 21 October 2003. *http://www.w3.org/TR/MathML2/*.

[2] Y.Nakano and H.Murao: BrEdiMa: Yet Another Web-browser Tool for Mathematical Expressions. *http://www.activemath.org/~paul/MathUI06/proceedings/BrEdiMa.html*.

[3] Wei Su, P.S.Wang and Lian Li. Entering and Editing Mathematical Expressions on the Web. Mathematical User-Interfaces Workshop 2008. *http://www.activemath.org/workshops/MathUI/08/proceedings/WeiWangLi-MathEdit.html*.

[4] Greasespot. *http://www.greasespot.net/*.

[5] mimeTeX manual. John Forkosh Associates, Inc. *http://www.forkosh.com/mimetex.html*.

[6] K.Nakamura. imgTeX. *http://www.eaflux.com/imgtex/index.html.en*.

[7] Texvc. Wikipedia, the free encyclopedia. *http://en.wikipedia.org/wiki/Texvc*.

[8] W3C Math Home. *http://www.w3.org/Math/*.

[9] J.J.Garret. Ajax: A New Approach to Web Applications. *http://www.adaptivepath.com/publications/essays/archives/000385.php*

[10] Document Object Model (DOM). W3C. *http://www.w3.org/DOM/*.

[11] Introducing JSON. *http://www.json.org/json-ja.html*.

[12] Dive Into Greasemonkey. *http://diveintogreasemonkey.org*.

[13] Greasemonkey for IE Home Page. *http://www.gm4ie.com/*.

[14] Trixie, Teaching an old browser new tricks. *http://www.bhelpuri.net/Trixie/*.

[15] Official distribution site of BrEdiMa. *http://bredima.sourceforge.jp*.