Submission for the publication in WWWJ

Title: **Meme Media for clipping and combining Web Resources**

Authors: **Yuzuru Tanaka, Kimihito Ito, and Jun Fujima**

Affliation: **Meme Media Laboratory, Hokkaido University**

Address: **N-13, W-8, Sapporo, 060-8628 Japan**

e-mail: **{tanaka, itok, fujima}@meme.hokudai.ac.jp**

Tel: Int+81-11-706-7256
Fax: Int+81-11-706-7808

Abstract:

The publication and reuse of intellectual resources using the Web technologies provide no support for us to clip out any portion of Web pages, to combine them together for their local reuse, nor to publish the newly composed object as a new Web page for its reuse by other people. This paper shows how the meme-media architecture is applied to the Web to provide such support for us. This makes the Web work as a shared repository not only for publishing intellectual resources, but also for their collaborative reediting. We will propose a general framework for clipping arbitrary Web contents as live objects, for defining IO ports on such a clip, and for the recombination and linkage of such clips based on both the original and some user-defined relationships among them. In our previous works, we proposed two separate frameworks for these three purposes; one works for the first two, and the other for the last. Here we will propose a unified framework for these three purposes, as well as its detailed internal mechanisms. Then we show how it can be easily applied to various legacy Web applications to develop innovative services.

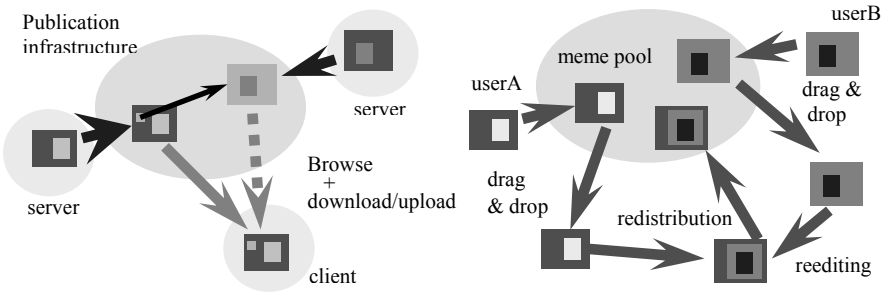
Keyword: Web technology, meme media, Web application linkage, Web clipping, interoperability, direct manipulation

Running head: Meme media for Web resources

1 Introduction

During the last decade, we have observed the rapid accumulation of intellectual resources on the Web. These intellectual resources include not only multimedia documents, but also application tools running on the client side, and services provided by remote servers. Today, from the Web, you can almost obtain whatever information items, application tools, or services you may think of.

The publication and reuse of intellectual resources using the Web technologies can be characterized by the schematic model in Figure 1 (a). The Web publication uses a compound document representation of intellectual resources. Compound documents denote documents with embedded contents such as multimedia contents, visual application tools, and/or interactive services provided by servers. Such a compound document published on the Web defines a Web page. The model in Figure 1 (a) has no support for us to extract any portion of published Web pages, to combine them together for their local reuse, nor to publish the newly defined composite object as a new Web page. The composition here means not only textual combination but functional federation of embedded tools and services. We need some support to reedit and to redistribute Web resources for their further reuse and new intellectual-resource creation.



(a)Current Web technologies (b)Meme media technologies

Fig. 1. The publication and reuse of intellectual resources

It is widely recognized that a large portion of our paperwork consists of taking some portions of already existing documents, and rearranging their copies in different formats on different forms. Since the reediting is so fundamental in our daily information processing, personal computers introduced the copy-and-paste operation as the most fundamental operation. We need to make this operation applicable not only to multimedia documents but also to documents with embedded tools and services.

Figure 1 (b) shows a new model for the worldwide publication, reediting and redistribution of intellectual resources. As in the case of the Web, you can publish a set of your intellectual resources as a compound document into a worldwide publication

repository. You can use a browser to view such documents published by other people. In addition to these operations, you can clip out any portions of viewed documents as reusable components, combine them together to define a new compound document for your own use, and publish this new compound document into the repository for its reuse by other people. This new model of publishing, reediting and redistributing intellectual resources assumes that all these operations can be performed only through direct manipulation. Meme-media technologies proposed by our group [17, 18, 12, 19], when applied to the Web, realize this new model, and make the Web work as a meme pool of intellectual resources. They provide the direct manipulation operations necessary for reediting and redistributing intellectual resources.

'Meme' is a word coined by Richard Dawkins. He pointed out a similarity between genetic evolution of biological species and cultural evolution of knowledge and art, and coined the word 'meme' to denote the cultural counterpart of a biological gene. As biological genes are replicated, recombined, mutated, and naturally selected, ideas are replicated, recombined, modified with new fragments, and selected by people. The acceleration of memetic cultural evolution requires media to externalize memes, and to distribute them among people. Such media allows people to re-edit their knowledge contents and redistribute them, and hence may be called meme media. They work as knowledge media for the re-editing and redistribution of intellectual assets.

Web Service technologies can provide us with similar functions for the functional linkage among Web applications [15, 13, 1, 4]. Web Service technologies enable us to interoperate services published over the Web. However, they assume that the API (Application Program Interface) library to access such a service is a priori provided by its server side. You need to write a program to interoperate more than one Web service. Meme-media technologies, on the other hand, provide only the client-side direct manipulation operations for users to reedit intellectual resources embedded in Web pages, to define a new combination of them together with their functional linkage, and to republish the result as a new Web page. In addition, meme-media technologies are applicable not only to the Web, but also to local objects. Meme media can wrap any documents and tools, including also any Web services, and make each of them work as interoperable meme-media object. You can easily combine Web resources with local tools and Web services.

This paper shows how the meme-media architecture is applied to the Web to make it work as meme pools. This makes the Web work as a shared repository not only for publishing intellectual resources, but also for their collaborative reediting. We will propose a general framework for clipping arbitrary Web contents as live objects, for defining IO ports on such a clip, and for the recombination and linkage of such clips. In our previous works, we proposed two separate frameworks for these three purposes; one works for the first two [19, 20, 12], and the other for the last [9]. The mechanism for the latter is not reported elsewhere yet. Here we will propose a unified framework for these three purposes, as well as its detailed internal mechanisms. Then we show how it can be easily applied to various legacy Web applications to develop innovative services.

2 Wrapping Web Resources as Meme Media Objects

2.1 Meme-Media Architecture IntelligentPad

IntelligentPad is a two-dimensional representation meme-media architecture. Its architecture can be roughly summarized as follows for our current purpose. Instead of directly dealing with component objects, IntelligentPad wraps each object with a standard pad wrapper, i.e., a software module with a standard visual representation and a standard functional linkage interface, and treats it as a media object called a pad. Each pad has both a standard user interface and a standard connection interface. The user interface of every pad has a card like view on the screen and a standard set of operations like 'move', 'resize', 'copy', 'paste', and 'peel'. As a connection interface, every pad provides a list of slots that works as IO ports, and a standard set of messages 'set', 'gimme', and 'update'. Each pad defines one of its slots as its primary slot. Most pads allow users to change their primary slot assignments.

You may paste a pad on another pad to define a parent-child relationship between these two pads. The former becomes a child of the latter. When you paste a pad on another, you can select one of the slots provided by the parent pad, and connect the child pad to this selected slot. The selected slot is called the connection slot. Using a 'set' message, each child pad can set the value of its primary slot to the connection slot of its parent pad. Using a 'gimme' message, each child pad can read the connection slot value of its parent pad, and update its primary slot with this value. Whenever a pad has a state change, it sends an 'update' message to each of its child pads to notify this state change. Whenever a pad receives an 'update' message, it sends a 'gimme' message to its parent pad. For each slot connection, you can independently enable or disable each of the three standard messages. By pasting pads on another pad and specifying slot connections, you may easily define both a compound document layout and functional linkage among these pads.

2.2 Clipping and Reediting of Web Resources

Web documents are defined in HTML format. An HTML view denotes an arbitrary HTML document portion represented in the HTML document format. The pad wrapper to wrap an arbitrary portion of a Web document specifies an arbitrary HTML view and renders any HTML document. We call this pad wrapper an HTMLviewPad. Its rendering function is implemented by wrapping a legacy Web browser Internet Explorer. The specification of an arbitrary HTML view over a given HTML document requires the capability of editing the internal representation of HTML documents, namely, DOM trees [22]. The DOM tree representation allows you to identify any HTML-document portion, which corresponds to a DOM tree node, with its XPath expression [20] such as /HTML[1]/BODY[1]/TABLE[1]/TR[2]/TD[2]. The precise definition of DOM and XPath is beyond the scope of this paper. For the details, we refer to their specification documents [22, 20].

The definition of an HTML view consists of a source document specification, and a sequence of view editing operations. A source document specification uses the

document URL. Its retrieval is performed by a function 'GETHTML' in such a way as

```
url =  'http://www.abc.com/index.html';
doc=url.GETHTML()
```

The retrieved document is kept in DOM format. The editing of an HTML view is a sequence of DOM tree manipulation operations selected out of the followings:

CLIP(*node*)
: To delete all the nodes other than the sub tree with the specified node as its root.

DELETE(*node*)
: To delete the sub tree with the specified node as its root.

INSERT(*node*, *HTML_view*, *location*)
: To insert a given DOM tree (HTML view) at the specified relative location of the specified node.
: You may select the relative location out of CHILD, PARENT, BEFORE, and AFTER.

An HTML view is specified as follows:

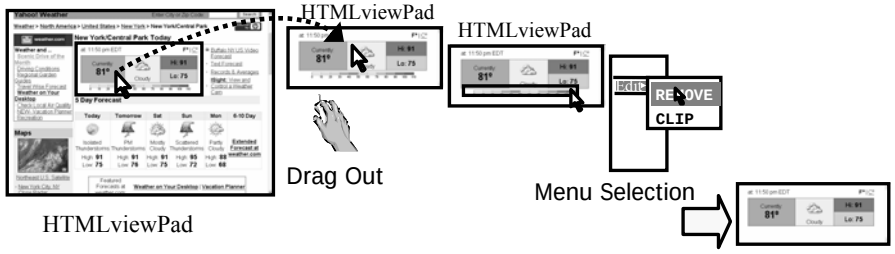
defined-view = *source-view*.*DOM-tree-operation*(*node*),

where *source-view* may be a Web document or another HTML document, and *node* is specified by its path expression. This code is called the view editing code of the HTML view. The following is an example view definition code.

```
view1 = doc
      .CLIP('/HTML/BODY/TABLE[1]')
      .CLIP('/TABLE[1]/TR[1]')
      .DELETE('/TR[1]/TD[2]');
```

After the first CLIP operation, the node /TABLE[1]/TR[1] corresponds to the node /HTML/BODY/TABLE[1]/TR[1] of the original document doc. The former path expression /TABLE[1]/TR[1] is called the relative path expression.

An HTMLviewPad with a view editing code can execute this code to recover the edit result when necessary. Its loading from a server or a local disk to a desktop is such a case.



(a) CLIP operation (b) DELETE operation

Fig. 2. Direct Manipulations for extracting and removing views

2.3 Direct Editing of HTML Views

Instead of specifying a relative path expression to identify a DOM tree node, we will make the HTMLviewPad to dynamically frame different extractable document portions for different mouse locations so that its user may move the mouse cursor around to see every extractable document portion. When the HTMLviewPad frames what you want to clip out, you can drag the mouse to create another HTMLviewPad with this clipped-out document portion. The new HTMLviewPad renders the clipped-out DOM tree on itself. Figure 2 (a) shows an example clip operation, which internally generates the following view edit code.

```
url = 'http://www.abc.com/index.html';
view = url.GETHTML()
          .CLIP('/HTML/BODY/TABLE[1]');
```

The HTMLviewPad provides a pop-up menu of view-edit operations including CLIP, DELETE and INSERT. After you select an arbitrary portion, you may select either CLIP or DELETE. Figure 2 (b) shows an example delete operation, which generates the following code.

```
url = 'http://www.abc.com/index.html';
view = url.GETHTML()
    .CLIP('/HTML/BODY/TABLE[1]')
    .DELETE('/TABLE[1]/TR[2]');
```

The INSERT operation uses two HTMLviewPads showing a source HTML document and a target one. You may first specify INSERT operation from the menu, and specify the insertion location on the target document by directly specifying a document portion and then specifying relative location from a menu including CHILD, PARENT, BEFORE, and AFTER. Then, you may directly select a document portion

on the source document, and drag and drop this portion on the target document. Figure 3 shows an example insert operation, which generates the following code, where the target HTMLviewPad uses a different name space to merge the edit code of the dragged-out HTMLviewPad to its own edit code:

```
A::view =A::url.GETHTML()
          .CLIP('/HTML/BODY/…/TD[2]/…/TABLE[1]')
          .DELETE('/TABLE[1]/TR[2]');
view = url.GETHTML()
  .CLIP('/HTML/BODY/…/TD[1]/…/TABLE[1]')
  .DELETE('/TABLE[1]/TR[2]')
  .INSERT('/TABLE[1]', A::view, AFTER);
```

The dropped HTMLviewPad is deleted after the insertion.

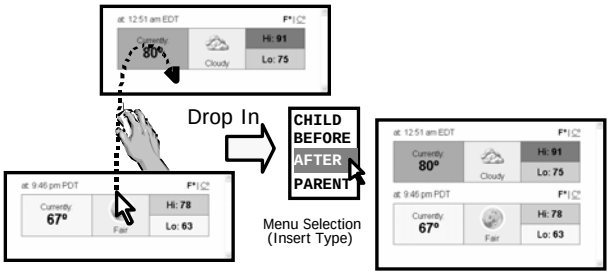


Fig. 3. Direct manipulation for inserting a view in another view

2.4 Automatic Generation of Default Slots

Each HTMLviewPad has the following four default slots. The #UpdateInterval slot specifies the time interval for the periodical polling of referenced HTTP servers. A view defined over a Web document refreshes its contents by periodically retrieving this Web document in an HTTP server. This is done by periodically executing its view editing code. The #SourceURL slot stores the URL of the source document. The #ViewEditingCode slot stores the view editing code. The #DestinationURL slot is used to hold a destination URL with or without a query to access a new page. The HTMLviewPad updates itself by executing its view editing code whenever one of the following conditions occurs: (1) the #SourceURL slot or the #ViewEditingCode slot is accessed with a set message, (2) the interval timer invokes the polling, (3) a user specifies its update, or (4) it becomes active after its loading from a file.

Depending on the type of the node to clip out, the HTMLviewPad automatically generates some additional default slots. For this purpose, we use the following view editing operations:

HTMLSLOT(*slot_name*, *node*)

To create a slot that holds the HTML view of the specified node.

TEXTSLOT(*slot_name*, *node*)

To create a slot that holds the text value of the specified node.

NUMSLOT(*slot_name*, *node*)

To create a slot that holds a numerical value converted to and from its textual representation in the specified node.

TABLESLOT(*slot_name*, *node*)

To create a slot that holds a CSV representation of the specified tabular node.

HREFSLOT(*slot_name*, *node*)

To create a slot that holds the href value defined by the specified anchor node.

INPUTSLOT(*slot_name*, *node*)

To create a slot that holds the input value to the specified input form node.

SUBMITSLOT(*slot_name*, *node*)

To create a slot that sends a set message with the value of the #DestinationURL slot whenever it receives a set message.

All these operations return the same HTML view as their recipient. Each slot created by one of the first four operations associates its slot value with the corresponding node value, and allows a pad that is connected to it to input a new value using a set message to rewrite this node value, and/or to read out this node value by sending a gimme message. An html slot created by the first operation can be connected to an HTMLviewPad, which may send a gimme message to read out the slot value and to render this HTML view on itself. Slots of the second, third, and fourth types can be respectively connected to text I/O pads, numerical I/O pads, and table I/O pads.

An href slot created by .HREFSLOT operation, when accessed by a set message with a new href value, will rewrite the href value of the corresponding HTML node. It also returns the href value when accessed by a gimme message.

A slot created by .INPUT SLOT, when accessed by a set message, will use the parameter value as a new form input, and hold this value. It also returns its current value when accessed by a gimme message. A slot created by .SUBMITSLOT operation, when accessed by a set message, will execute the editing code with the current slot values. This slot neglects gimme messages.

When you clip out a general node such as text node or paragraph node, the HTMLviewPad automatically creates a new HTMLviewPad, defines a default text-value slot in this new pad, and sets the text in the selected node to this slot. For an HTML view with a view editing code σ, and a text node with a relative path expression π, the clipping of this node generates a new HTMLviewPad with the following editing code:

```
view = σ.CLIP(π)
        .TEXTSLOT(slot_name, '/');
```

We use '/' to specify the root node. The slot name is specified by the user when he clips this node. This slot works as the default primary slot of this clip.

If the text is a numerical string, the HTMLviewPad converts this string to a numerical value, and sets this value to the automatically created slot. For an HTML view with a view editing code σ, and a numerical text node with a relative path expression π, the clipping of this node generates a new HTMLviewPad with the following editing code:

```
view = - σ.CLIP(π)
         .NUMSLOT(slot_name, '/');
```

This slot works as the default primary slot of this clip. Figure 4 shows Yahoo Finance Web page showing the current Nikkei average stock price in real time. You may clip out the Nikkei average index as a pad working as a live copy, and paste it to a DataBufferPad with its connection to #input slot. The clipped-out pad has a function to periodically polling the source server to update its HTML view. A DataBufferPad associates each #input slot input with its input time, and outputs this pair in CSV (Comma-Separated Value) format. You may paste this composite pad on a TablePad with its connection to #data slot. A TablePad adds every #data slot input at the end of the list stored in CSV format. You may paste this TablePad on a ChartPad with its connection to #tabelInput slot. A ChartPad redraws a chart whenever it receives a new #tableInput slot value.

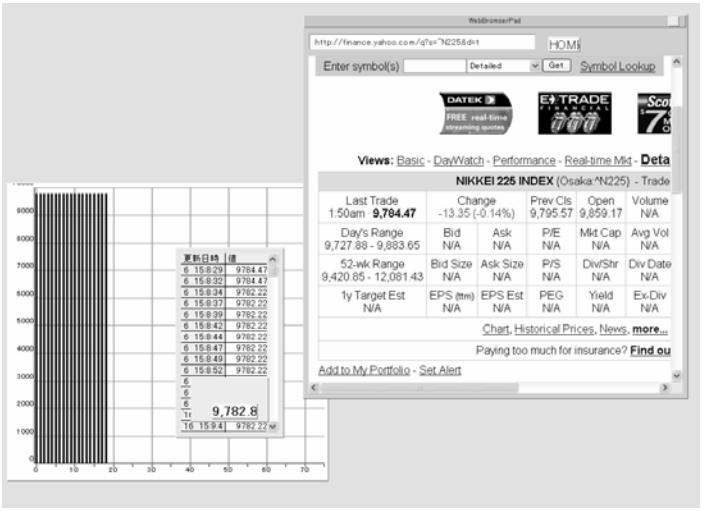


Fig. 4. The real-time drawing of a stock price chart using a live copy pad

When you clip out a table node such as </HTML/…/TABLE>, the HTMLviewPad creates a new HTMLviewPad, converts the table value to its CSV representation, and generates a new default slot with this value in the new pad. For an HTML view with a view editing code σ, and a tabular node with a relative path expression π, the clipping of this node generates a new HTMLviewPad with the following editing code:

```
view = σ.CLIP(π)
         .TABLESLOT(slot_name, '/');
```

The slot name is specified by the user when he clips this node. This slot works as the default primary slot of this clip.

When a clip of one of the above three types receives a set message, it sends a set message to its parent with the current primary slot value, and also sends an update message to its child pads. When it receives an update message from its parent pad, it sends a gimme message to get a new HTML view to render on itself.

For an HTML view with a view editing code σ, and an anchor node π, the clipping of this node generates a new HTMLviewPad with the following editing code to create three default slots:

```
view = σ
       .CLIP(π)
        .TEXTSLOT(slot_name1, '/')
        .HREFSLOT (slot_name2, '/')
        .SUBMITSLOT(slot_name3, '/');
```

The slot names are specified by the user when he clips this node. When this clip receives either a set message to its third slot or a mouse click on the clipped anchor, it calculates the destination URL from the href slot value, and stores this URL in the #DestinationURL slot, which works as its default primary slot. Then it sends a set message to its parent with its primary slot value, and also sends update messages to its child pads. If it receives an update message from its parent, it sends a gimme message to get a new HTML view to render on itself.

For example, let us consider a case in which we clip out an anchor defined as follows:

```
<A href = ./next.html>
Next Page
</A>
```

The first slot holds 'Next Page', while the second slot holds './next.html' as its value. Whenever the anchor is clicked or a set message is sent to the third slot, the destination URL is calculated, and is set to the #DestinationURL slot. Then this HTMLviewPad sends a set message to its parent pad with the value of its third slot as its parameter value, and also sends an update message to each of its child pads if any.

For an HTML view with a view editing code σ, and a form node with a relative path expression π, the clipping of this node generates a new HTMLviewPad with the following editing code to create two default slots:

```
view =σ.CLIP(π)
        .INPUTSLOT(slot_name1, π1)
        .SUBMITSLOT(slot_name2, π2);
```

The #DestinationURL slot works as the default primary slot of this clip. When this clip receives a set message to its second slot, it creates a query to the server from the first slot value, and updates #DestinationURL slot with the concatenation of the server URL and this query. Then it sends a set message to its parent with its primary slot value, and also sends update messages to its child pads. If it receives an update message from its parent, it sends a gimme message to get a new HTML view to render on itself.

For example, let us consider a case in which we clip out a form defined as follows:

```
<FORM action='./search'>
<INPUT Type=txt name=keyword >
<INPUT Type=submit value='search'>
</FORM>
```

The first slot holds the input keyword. Whenever we submit a form input or the second slot receives a set message, the HTMLviewPad calculates the destination URL with a query '/search?keyword=<the first slot value>', and sets this value to the #DestinationURL slot. Then the HTMLviewPad sends a set message to its parent pad with the value of its primary slot as its parameter value, and also sends an update message to each of its child pads if any.

Figure 5 shows a Yahoo Maps Web page. You may make live copies of its map display portion, its zooming control panel, and its shift control panel, all as pads, and paste the two control panels on the map display with their connections to #SourceURL slot of the map display. In these slot connections, we only enable set messages and disable gimme and update messages. Whenever you click some button on either of these control panels, the control panel sets the URL of the requested page, and sends this URL to #SourceURL slot of the map display. Such a URL may include a query to the specified server. Then the map display accesses the requested page with a new map, and clips the map portion to display.

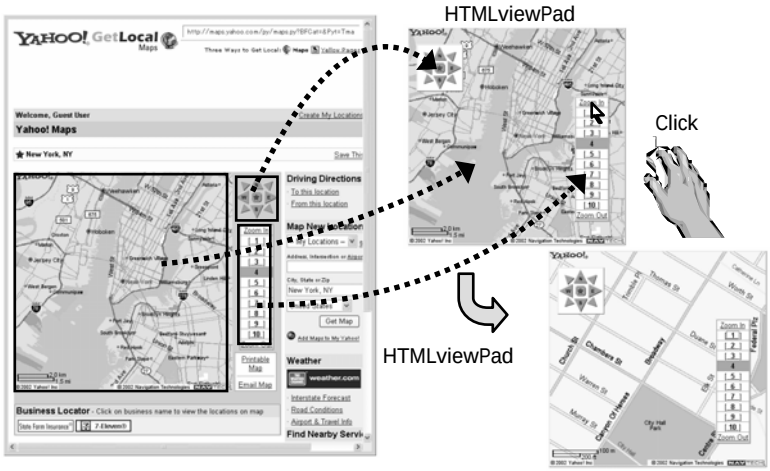


Fig. 5. Composition of a map tool using a map service and its control panels

For a non atomic node corresponding to an internal node of a DOM tree, its clipping generates a new HTMLviewPad with the following view editing code:

```
view =σ.CLIP(π)
        .HTMLSLOT(slot_name, '/');
```

The slot name is specified by the user when he clips this node. This slot works as the default primary slot of this clip. This clip accepts no set message. When this clip receives a gimme message to this slot, it returns this slot value. When a user operation on this clip updates its HTML view, it sends the new HTML view to its parent, and also sends an update message to each of its child pads.

3 Interactive Definition of Slots on Web Resource Clips

3.1 Slot definition on clips and composition with clips

Our HTMLviewPad allows users to visually specify any HTML-node to work as a slot. In its node specification mode, an HTMLviewPad frames different extractable document portions of its content document for different mouse locations so that its user may change the mouse location to see every selectable document portion. When the HTMLviewPad frames what you want to work as a slot, you can click the mouse to pop up a menu to specify the type of the slot you like to create. Then it pops up a dialog box for you to name this slot. The specified HTML node works as a slot of this type with this slot name. Since each clipped-out Web component uses an HTMLviewPad to render its contents, it also allows users to specify any of its portion to work as its slot. The HTMLviewPad internally holds a sequence of these operations as a view editing code including slot-defining operations and event operations. Slot-defining operations include HTMLSLOT, TEXTSLOT, NUMSLOT, TABLESLOT, HREFSLOT, DESTSLOT, INPUTSLOT, and SUBMITSLOT, while event operations include the following three operations:

CLICK(*anchor_node*)	to return the destination Web page of this anchor
SET(*form_node*, *input*)	to set an input value to the specified form input node
SUBMIT(*submit_node*)	to return the output page by sending the corresponding server a query specified by the current input-form values.

Figure 6 (a) shows an HTMLviewPad showing a Yahoo Web page with an embeded Web application to convert US dollar to Japanese yen based on the current exchange rate. On this pad, you first input a dollar amount, say '200', to the dollar input form. Then you can visually specify this input form to work as a form input slot.

The HTML-path of this input-form is represented as HTML[1]/BODY[1]/DIV[1]/FORM[1]/INPUT[1]. As a result of your interactive slot definition, the following view editing code is generated and stored in the HTMLviewPad. You may name this slot as #dollarAmount.

```
url = 'http://...';
view = url.GETHTML()
    .SET('/HTML[1]/BODY[1]/DIV[1]/FORM[1]/INPUT[1]', '200')
    .INPUTSLOT('#dollarAmount',
        '/HTML[1]/BODY[1]/DIV[1]/FORM[1]/INPUT[1]');
```

Now you can click the conversion button on this Web page to submit a query to the corresponding server. As a result, the HTMLviewPad updates its page and shows the equivalent yen amount. This adds .SUBMIT('/HTML[1]/BODY[1] /DIV[1]/FORM[2]') operation at the end of the view editing code. Then you can visually specify the portion showing the equivalent yen amount to work as a text slots. The HTML-path of the selected output text portion is represented as /HTML[1]/BODY[1]/TABLE[1]/TR[1]/TD[2]/A[1]. You may name this slot as #yenAmount. This interactive slot definition adds .TEXTSLOT('#yenAmount', '/HTML[1]/BODY[1]/TABLE[1]/TR[1]/TD[2]/A[1]') operation.

The HTMLviewPad allows you to suspend the rendering of its contents. In this mode, you may use an HTMLviewPad as a blank sheet pad with an arbitrary size. You can use the following HTML view operation to control the rendering.

```
NORENDER( )
        to suspend the rendering of the HTML view, and to return the
        recipient HTML view.
```

As a result of your interactive slot definition described above, the following view editing code is generated and stored in the HTMLviewPad.

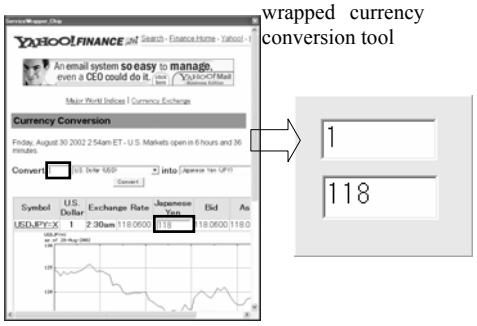
```
url = 'http://…';
view = url.GETHTML( )
    .SET('/HTML[1]/BODY[1]/DIV[1]/FORM[1]/INPUT[1]',
        '200')
    .INPUTSLOT('#dollarAmount',
        '/HTML[1]/BODY[1]/DIV[1]/FORM[1]/INPUT[1]')
    .SUBMIT('/HTML[1]/BODY[1]/DIV[1]/FORM[2]')
    .TEXTSLOT('#yenAmount',
        '/HTML[1]/BODY[1]/TABLE[1]/TR[1]/TD[2]/A[1]')
    .NORENDER( );
```

When # dollarAmount slot is accessed by a set message with a new textual value, the HTMLviewPad scans this view editing code to find out that this value is an alternative value for the second parameter of the corresponding SET operation, and executes the view editing code with this alternative value.

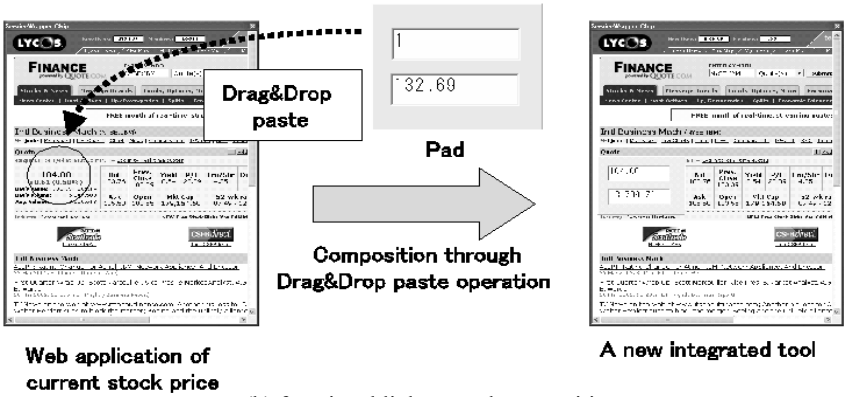
Figure 6 (a) shows, on its right hand side, a currency rate converter pad. You can define this pad from the above mentioned Web page just by defining two slots, suspending the rendering, resizing the HTMLviewPad, and pasting two text IO pads with their connections to #dollarAmount and #yenAmount slots.

Such a pad wraps a Web application, providing slots for the original's input forms and output text strings. We call such a pad a wrapped Web application. Since a wrapped Web application is such a pad that allows you to change its primary slot assignment, you may specify any one of its slots to work as a primary slot.

Figure 6 (b) shows an HTMLviewPad showing a Lycos' Web page for a real-time stock-price browsing service. You can wrap this page defining a slot for the current stock price. Then you can paste the wrapped currency conversion Web application with its #dollarAmount slot specified as its primary slot on this wrapped Lycos stock-price page. You may connect the conversion pad to the newly defined current stock price slot as shown in this figure. For the input of different company names, you may use the input form of the original Web page. Since this Web application uses the same page layout for different companies, the same path expression correctly identifies the current stock-price information part for every different company.



(a) wrapping



(b) functional linkage and composition

Fig. 6. Direct manipulation for wrapping and linking of Web applications

3.2 Slot definition during a navigation across Web resources

Figure 7 shows a Web application of NCBI's PubMed service. This application uses three pages for its input and output. On the first page, it requests users to input data. Its second page displays a list of result items. Users may select one of these items and click its anchor to jump to the third page, where they will get the output data of this application.

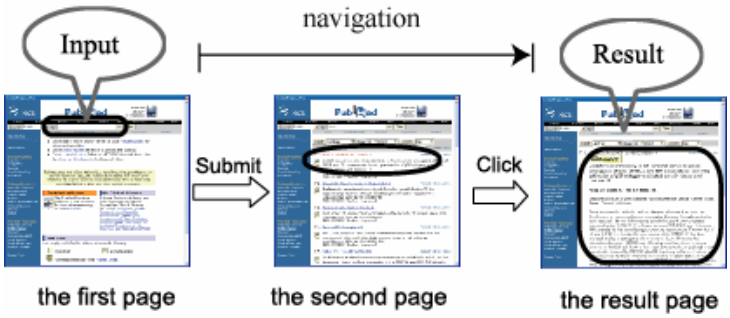


Fig. 7. A model of navigations within PubMed service

As shown in Figure 8, you may first opens the PubMed search page, and input a search keyword 'kinase' to the keyword input form on the first page, visually specify this form to work as an input slot named #keyword. Now you may click the search button to submit a query to the server. As a result, you will jump to the second page. There, you may click the first candidate of the search result list to jump to the third page. On the third page, you may specify the paper abstract portion to work as a slot named #paperabstract, and then hide the background HTML view. Now you may freely resize the HTMLviewPad. You may paste two text pads with their connections to these two slots.

As a result of these interactive operations, the HTMLviewPad creates the following view editing code to wrap this Web application.

```
url = 'http://www.ncbi.nlm.nih.gov/entrez/';
view = url.GETHTML()
    .SET('/HTML[1]/BODY[1]/FORM[1]/…/INPUT[1]',
          'kinase')
    .INPUTSLOT('#keyword', '/HTML[1]/…/FORM[1]/../INPUT[1]')
    .SUBMIT('/HTML[1]/…/FORM[1]')
    .CLICK('/HTML[1]/BODY[1]/…/TABLE[2]/…/FONT[1]/A[1]')
    .TEXTSLOT('#paperabstract',
       '/HTML[1]/…/TABLE[2]/…/TR[2]/TD[3]/DL[1]/DD[1]')
    .NORENDER();
```

If the value of some input-form slot is changed, then the HTMLviewPad automatically executes the view editing code.

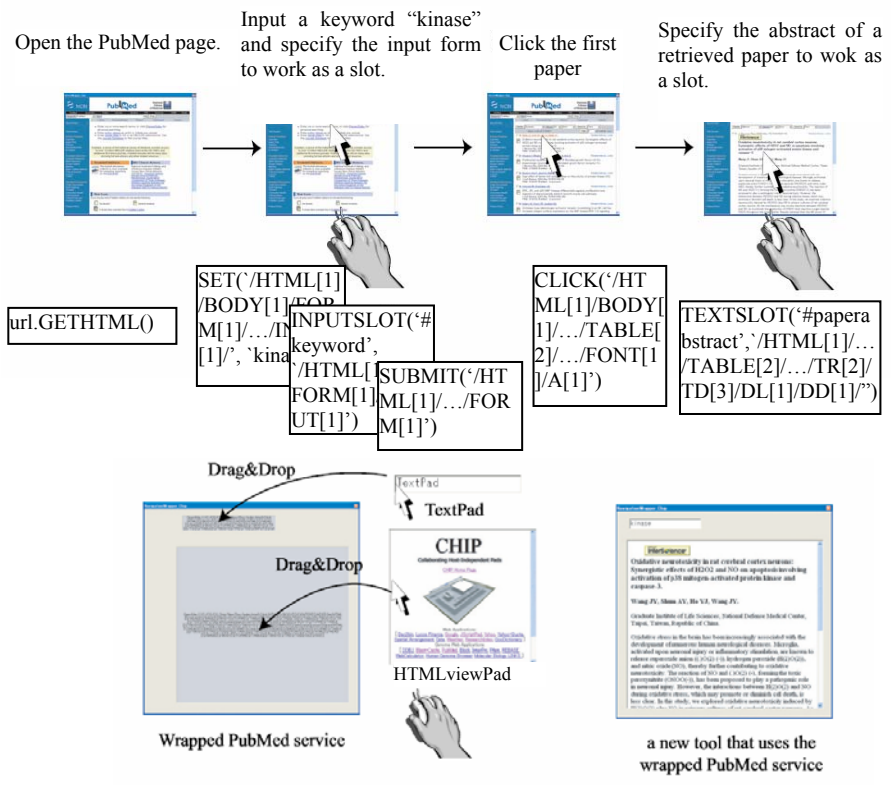


Fig. 8. An Example Sequence of Mouse operations for defining an HTMLview

4 Recombination and Linkage of Web Resource Clips

4.1 Recombination of Web clips and their linkage

Here we consider the clipping of more than one HTML node from more than one Web page visited through a single navigation, and their functional recombination based on the functional linkage relationship among these nodes in this navigation. Figure 9 shows a Google Web page, and the clipping of the keyword input form, and the first search result through a search navigation with 'IntelligentPad' as a search keyword. The Web browser used here is also an HTMLviewPad. We can use its node specification mode to clip out HTML nodes only through mouse operation. In this

example, we first clipped out the keyword input form, and the search area selector as two new HTMLviewPads from the Google home page.

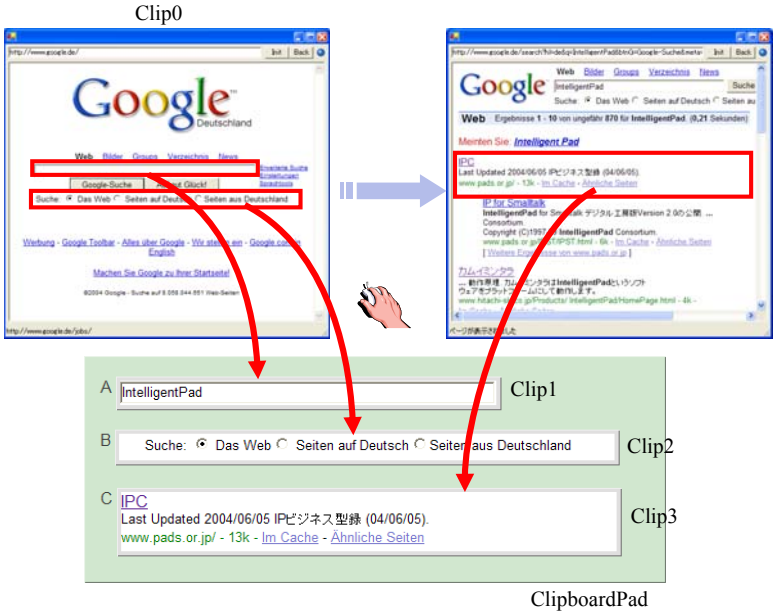


Fig. 9. Clips extracted from a single navigation and their recombination on a ClipboardPad.

These clips have the following view editing codes:

```
Clip1  view=url  .GETHTML()
                 .CLIP(π)
                 .INPUTSLOT(#keyword, '/')
                 .SUBMITSLOT(#submit, '/');

Clip2 view=url   .GETHTML()
                 .CLIP(π1)
                 .HTMLSLOT(#selector, '/');
```

After these clips are clipped out, the HTMLviewPad Clip0 showing the Google home page has the following view editing code:

```
Clip0    view=url.GETHTML()
                 .INPUTSLOT(#keyword, π)
                 .SUBMITSLOT(#submit, π)
                 .HTMLSLOT(#selector, π1);
```

Then we input a search keyword ‘IntelligentPad’ on this Google home page, set a selector, and click the search button to obtain the first search result page. This changes the view editing code of Clip0 as

```
Clip0   view=url.GETHTML()
                .INPUTSLOT(#keyword, π)
                .SUBMITSLOT(#submit, π)
                .HTMLSLOT(#selector, π1)
                .SET(π, ‘IntelligentPad’)
                .SET(π1, *)
                .SUBMIT(π2);
```

where .SET(π1, *) denotes that the current HTML view at node π1 is set to the node π1. This substitution may seem to be redundant. However, in the later discussion, this parameter value is connected to the Clip2, and is updated by a set message from this clip that may have changed its HTML view through user’s interaction.

Now the Clip0 shows the second page. From this page, we clipped out the first search result as another new HTMLviewPad Clip3.

```
Clip3   view=url.GETHTML()
                .INPUTSLOT(#keyword, π)
                .SUBMITSLOT(#submit, π)
                .HTMLSLOT(#selector, π1)
                .SET(π, ‘IntelligentPad’)
                .SET(π1, *)
                .SUBMIT(π2)
                .CLIP(π3)
                .HTMLSLOT(#first_candidate, ‘/’);
```

Each of these clips is pasted on the same special pad called a ClipboardPad immediately after it is clipped out. This ClipboardPad is used to make these clips operate with each other based on their functional linkage relationship in the navigation. It holds a list of view editing codes, each of which corresponds to a single navigation. When we clip out and paste Clip1 on a ClipboardPad, it creates a new entry in the view editing code list, and puts the view editing code of Clip0 at this time with all the slot creating operations deleted. This entry value becomes as follows:

```
view=url.GETHTML();
```

The ClipboardPad associates this clip a new cell name ‘A’, creates a corresponding slot #A, and connects Clip1 to this slot. It associate the slot #A with the node π. This adds one more operation .CELL (‘A’, π) at the end of the above code using the following new operation:

CELL(*cell_name*, *node*)

To associate the specified cell with the specified node, and to return the same HTML view as the recipient.

When we paste Clip2 on the same ClipboardPad, it searches the list for an entry with such a code that is a prefix of the code σ obtained from the view editing code of Clip0 at this time by deleting all the slot creating operations. This comparison neglects all the CELL operations in the code stored in each entry of the code list. The ClipboardPad updates this entry with the code σ, inserts all the CELL operations in the old entry code at the same positions in σ, and adds one more CELL operation to associate the node π1 with the new cell 'B'. This entry value becomes as follows:

```
view=url.GETHTML()
        .CELL('A', π)
        .CELL('B', π1);
```

The ClipboardPad connects Clip2 to the slot #B.

When we finally paste Clip3, the same entry of the code list becomes as follows:

```
view=url.GETHTML()
        .CELL('A', π)
        .CELL('B', π1)
        .SET(π, 'IntelligentPad')
        .SET(π1, *)
        .SUBMIT(π2)
        .CELL('C', π3);
```

The CliupboardPad connects Clip3 to the slot #C.

When we input some keyword on Clip1, it sends a set massage to the slot #A with its primary slot value, which is the value of the slot #keyword, i.e., the input keyword. The ClipboardPad scans CELL operations for the slot #A, finds the associated node π, searches for a SET operation with π as its first parameter, replaces its second parameter with the input keyword, and executes this view editing code. When we select an item on Clip2, it sends a set message to the slot #B with its updated HTML view. This values is used to rewrite the second parameter of SET(π1, *). Then this ClipboardPad executes the view editing code.

Whenever the Clipboard executes a view edit code, it sends update messages to all the clips working as output devices. Each of thease clips sends a gimme message to its connection slot. The ClipboardPad searches for the node associated with this connection slot, and returns the HTML view of this node.

These mechanisms allow clips obtained during a single navigation to interoperate with each other based on their original functional relationships in the navigation.

4.2 Linkage between Web clips from different navigations

A ClipboardPad is also used to define functional linkage among different clip sets clipped out from different navigations. When we paste a clip on a ClipboardPad, the

ClipboadPad automatically gives it a unique cell name such as ‘A’, ‘B’, ‘C’, ‘A1’, ‘B1’, ‘A2’. Like a spreadsheet tool, it allows us to relate an input cell X clipped out from a navigation to other cells Y1, Y2, …, Yk clipped out from another different navigation just by specifying an equation X←*f* (Y1, Y2, …, Yk), where *f* is an arbitrary computable function. Whenever one of the cells Y1, Y2, …, Yk is updated, the ClipboardPad computes *f* (Y1, Y2, …, Yk), and updates the cell X with this new value. This triggers the execution of the view editing code in which the cell X is defined, and updates all the output clips involved in this view editing code. Some of these updated cells may be linked to other cells through some equations, which may trigger further updates of the ClipboadPad.

Figure 10 shows two navigations, one starting with CNN Money Stock Quote page, and the other with Yahoo Finance Currency Conversion page. From the former, we clipped out the company code input form and the stock quote output, and put them in this order on a ClipboardPad, which associated them with two cells A and B. From the latter, we clipped out the dollar amount input, the currency selector, and the converted amount output, and put them in this order on the same ClipboardPad, which associated them with three more cells C, D, and E. Then we input a substitution expression ←B to the cell C. The composite tool enables us to retrieve the current stock quote of an arbitrarily selected company in an arbitrarily selected currency.

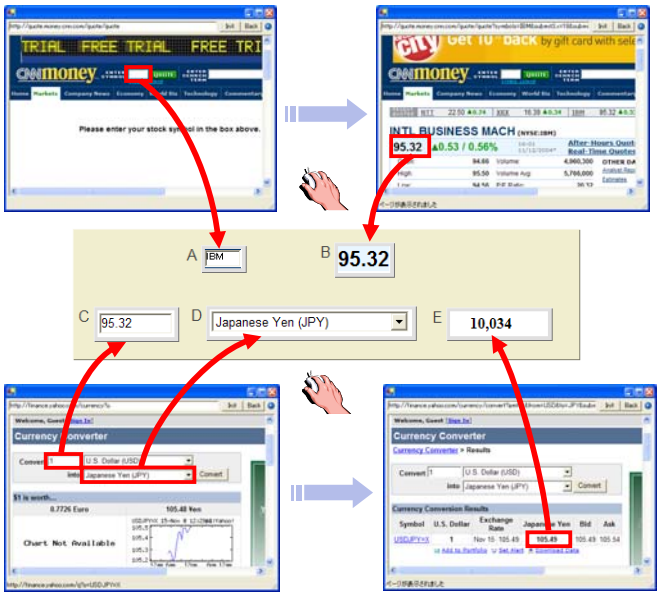


Fig. 10. Clips extracted from more than one navigation and their recombination on a ClipboardPad.

5 World wide Repository of Meme-Media Objects

Meme media technologies for clipping and recombining Web resources can be applied to arbitrary Web pages, and hence, any legacy Web applications such as Google and Wiki services. They also enable us to make any legacy applications interoperable with each other just by developing their HTML view interfaces.

Here we will apply meme media technologies to Wiki service to make it work as a worldwide repository of pads for sharing and exchanging them, i.e., as a meme pool. Wiki is a piece of server software that allows users to freely create and edit Web page content using any Web browser. Wiki supports hyperlinks and has a simple text syntax for creating new pages and crosslinks between internal pages on the fly. Wiki is unusual among group communication mechanisms in that it allows the organization of contributions to be edited in addition to the content itself.

In order to make a meme pool system from Wiki, you can access a Wiki page using an HTMLviewPad, and extract the URI input, the HTML input form, refresh button, and the output page as Web clips, and paste them on a ClipboardPad. You need to paste a PadSaverLoaderPad on the same ClipboardPad, which then creates a new cell with its connection to this pad. Now, you can relate the input and output of this cell respectively to the cell for the input form clip and the one for the extracted output page clip. A PadSaverLoaderPad makes conversion between a pad on itself and its save format representation in XML. Suppose that the PadSaverLoaderPad, the extracted HTML input form clip, and the extracted output page clip are assigned with cell names A, B, and C. The relationship among them is defined as follows. We define the equation for the cell A as ←C, and the equation for the cell B as ←A. People can access any page specifying its URL, drag-and-drop arbitrary composite pads to and from the PadSaverLoaderPad on the composed pad to upload and download them to and from the corresponding Wiki server. Each page is shown by the PadSaverLoaderPad. This meme pool system based on Wiki technologies is called a Wiki piazza. Figure 11 shows a Wiki piazza. Users may manipulate and/or edit some pads on an arbitrary page of a Wiki piazza to update their states. Another user accessing the same page can share the updated pads by just clicking the reload button to retrieve this page again from the corresponding server. For a jump from a page to another page in a Wiki piazza system, we can use an anchor pad that can be pasted on a Wiki piazza page. This anchor pad holds a URL that can be set through its #refURL slot, and, when it is clicked, sets this URL to the #URL slot of the Wiki piazza, i.e., to the #URL slot of its base ClipboadPad.

Fig.11 A worldwide repository of pads developed by applying meme media technologies to Wiki.

A Wiki piazza system allows people not only to publish and share contents represented as pads, but also to compose new contents by combining components of those pads already published in it. People can publish such newly composed contents into the same Wiki piazza system. The collaborative reediting and redistribution of contents in a shared publishing repository by a community or a society of people will accelerate the memetic evolution of contents in this repository, and make it work as a meme pool.

6 Redistribution and Publication of Meme-Media Objects as Web resources

Whenever you save a Web clip extracted from a Web page, the system saves only the pad type, namely 'HTMLviewPad', the values of all the slots, and, for each user-defined slot, its path expression, its name, and its value. Copies of such a Web clip share the same meta information with the original. They execute their view editing codes and access the source Web pages whenever they need to update themselves. This is an important feature from a copyright point of view, since every update of such a copy requires the access of original Web resources in the related servers. The redistribution of a Web clip across the Internet needs to send only its save format representation.

For the reediting of extracted Web contents, our framework provides two methods. One of them allows you to insert an HTML view into another HTML view without any functional linkage. The other allows us to paste an HTML view as a pad on another HTML view as a pad with a slot connection between them. The former composition results in a new HTML view, while the latter composition is no longer an HTML view.

In order to publish composed documents and/or tools as HTML documents in the Web, we need to convert non HTML view compositions to HTML views. We call

such a conversion a *flattening* operation. While there may be several different methods to flatten composite pads, we chose the use of IFrame (inline frame) representation of composite pads. An HTML view representation of a composite pad can be inserted as a new IFrame element in the HTML view representation of the parent pad. This insertion defines a new HTML view composition.

Non HTML view composition treats HTML views as pads, and combines them through slot connections. A script language, such as JavaScript, enables an HTML view to be connected to another HTML view with a slot connection between them. Using the same visual operation to insert an HTML view in another HTML view, you can visually insert a composite pad as a new HTML element in an arbitrary HTML view. For the HTML representation of an HTML view working as a pad, we use script variables to represent its slots, its primary slot, its parent pad, the parent's slot it is connected, and the list of child pads.

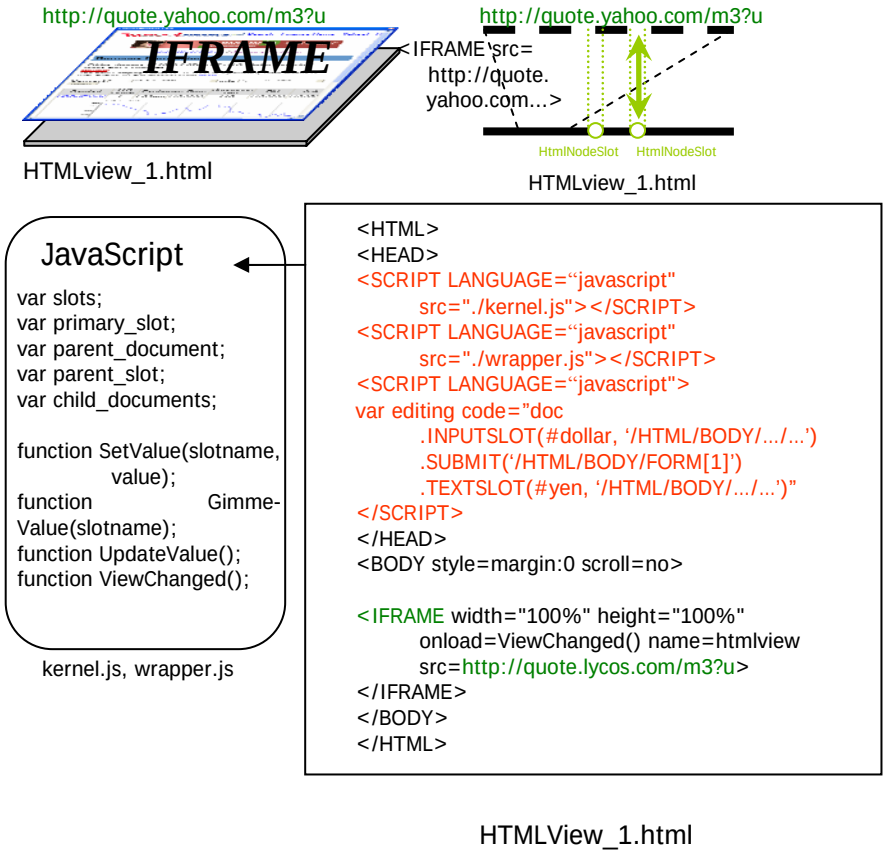
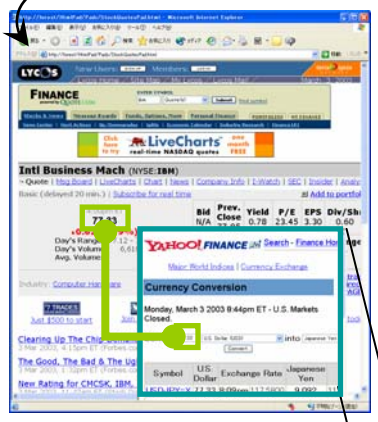


Fig. 12. A JavaScript program to define slots in an HTML view

As shown in Figure 12, we use JavaScript to define SetValue function to set a new value to a specified slot, GimmeValue function to read out the value of a specified slot, and UpdateValue function to update the primary slot value and to invoke every

child pad's UpdateValue function. To update the primary slot value, we define a script program to invoke the parent's GimmeValue function with the connection slot as its parameter, and to set the return value to the own primary slot. Figure 12 shows an HTML view defined with two slots #dollar and #yen and the three standard functions. This HTML view works as a wrapper of currency conversion provided by Yahoo.




```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="javascript"
        src="./kernel.js"></SCRIPT>
<SCRIPT LANGUAGE="javascript"
        src="./wrapper.js"></SCRIPT>
</HEAD>
<BODY onload=Initialize() style=margin:0
        scroll=no>

<IFRAME width="100%" height="100%"
        onload=ViewChanged()
        name=htmlview
        src=/HTMLPad/Proxy/proxy.aspx?url
        =http://finance.lycos.com/home/stocks
        /quotes.asp%3Fsymbols%3DNYSE
        %253AIBM%26origsymbols
        %3Dibm&encoding=
        ></IFRAME>

<IFRAME
        style="width:500px;height:500px;
        position:absolute;top:400;left:300"
        src=./YahooFinancePad.html
        ></IFRAME>

<SCRIPT>
   var editingcode="doc
   .NODESLOT(#price,/HTML/BODY/.../...)";
   function Initialize(){
      Con-
     nect("#price",document.frames.item(1).docume
     nt,  "#dollar");
     Sync();
         }
</SCRIPT>
</BODY>
</HTML>
```

HTML +JavaScript

Fig. 13. A flattening of composite HTML view

Figure 13 shows an HTML view composition with two HTML views corresponding to two Web clips; one works as a child pad of the other. The parent HTML view is the stock quote service with a slot, #price. The child HTML view works as a currency conversion with its slot #dollar and # yen. This HTML view composition is represented as a single HTML view that embeds the two HTML views using <IFRAME> tags. Its HTML representation also uses script codes using <SCRIPT> tags to define slot connection linkages among embedded I-frames. The composed HTML view works exactly the same as the composite pad combining the pad representations of these two HTML views. Users may use a legacy Web browser to display this composite view and to play with it. Our framework uses this mechanism to flatten a composite pad that uses only Web clip pads.

7 Related work

Web Service technologies such as SOAP (Simple Object Access Protocol) [21] enable us to interoperate services published over the Web. However, they assume that the API (Application Program Interface) library to access such a service is *a priori* provided using the WSDL. Users need to write a program to interoperate between more than one Web service. Our technologies, on the other hand, provide only the client-side direct manipulation operations for users to re-edit intellectual resources embedded in Web pages, and to define a new combination of them together with their interoperation.

The Semantic Web is an extension of the current Web in which Web contents are associated with explicit machine-processable semantics. The Semantic Web Services, which is integration of Web Service technologies and Semantic Web technologies, aims to provide the automation of Web service composition. In our approach, on the other hand, we do not aim to provide the automatic composition of Web applications. We focus on how instantaneously users can create wrappers for Web applications when the users need to reuse them in functional combinations with other applications.

There are a few preceding research studies that adopt programming by demonstration (PBD, for short) technologies on Web pages. Internet Scrapbook[16] allows us to re-edit Web documents by demonstrating how to change the layout of a web page into a customized one. Internet Scrapbook applies the same editing rule whenever the Web page is accessed for refreshing. They enable us to change layouts, but not to extract any components, nor to functionally connect them together.

Bauer and Dengler[5,6] have also introduced a PBD method in which even naive users can configure their own Web based information services satisfying their individual information needs. They have implemented the method into InfoBeans. By accessing an InfoBox with an ordinary Web browser, users can wrap Web applications. By connecting channels among InfoBeans on the InfoBox, users can also integrate them functionally together. However, it seems difficult for users to reuse a part of composite Web applications defined by other users.

WebVCR[3] and WebView[8] provide familiar interface to record and replay users' actions. Users can create and update their 'smart bookmarks', which are shortcuts to Web contents that require a series of Web-browsing actions, by pressing a record button on their Web browser. Smart bookmarks can therefore be used to record hard-to-reach Web pages that have no fixed URLs. However, WebVCR does not support the definition of I/O ports for Web applications. For example, end-users could not modify the parameters for an input-form. WebView allows us to define customized views of Web contents. When a user records a smart bookmark, he or she can indicate if some field in a form is to be requested at playback time, rather than stored with the bookmark. However it seems difficult for end-users to create a new view that integrates different Web applications.

Sometimes Web applications revise the format of front-end HTML pages. There are lots of preceding research studies on the induction of Web wrapper by extraction examples from Web pages. However, there are a few research studies that allow end-users to wrap Web applications and to define functional linkage in the same environment.

W4F[14], which is a semi-automatic wrapper generator, provides a GUI support tool to define an extraction. The system creates a wrapper class written in Java from user's demonstration. To use this wrapper class, users need to write program codes. DEbyE[10] provides more powerful GUI support tool for the wrapping of Web applications. DEbyE stores the extracted text portions in XML repository. Users have to use another XML tool to combine extracted data from Web applications. LExIKON[11] learns an underlying relation among objects within a Web page from a user-specified ordered set of text strings. There is no GUI support tool for the join of two extracted relations.

8 Conclusion

Meme-media architectures work as the enabling technologies for interdisciplinary and international availability, distribution and exchange of intellectual assets including information, knowledge, ideas, pieces of work, and tools in reeditable and redistributable organic forms. When applied to the Web, they make the Web work as a shared repository not only for publishing intellectual resources, but also for their collaborative reediting and reorganization. Meme-media technologies allow us to reedit multimedia documents with embedded tools and services, and to combine their functions through copy-and-paste manipulations of these documents. Meme media over the Web will make the Web work as a meme pool, and significantly accelerate the evolution of memes in our societies.

References

1. R. Agrawal, R. J. Bayardo, D. Gruhl, and S. Papadimitriou, "Vinci: A Service-oriented Architecture for Rapid Development of Web Applications," in Proc. the tenth international conference on World Wide Web, Hong Kong, Hong Kong, 2001, pp. 355-365.
2. A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila, D. L. Martin, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng, "DAML-S: Semantic Markup for Web Services.", in Proc. International Semantic Web Working Symposium (SWWS), 2001,
3. V. Anupam, J. Freire, B. Kumar, and D.F. Lieuwen, "Automating Web Navigation with the WebVCR," in Proc WWW9, Amsterdam, Netherlands, 2000, Computer Networks, vol.33, No. 1-6, pp.503-517.
4. R. Anzböck, S. Dustdar, H. Gall, "Software Configuration, Distribution, and Deployment of Web-Services," in Proc. the 14th international conference on Software engineering and knowledge engineering, Ischia, Italy, 2002, pp. 649-656.
5. M. Bauer, and D. Dengler, "InfoBeans - Configuration of Personalized Information Services," in Proc. IUI'99, Los Angeles, USA, 1999, pp. 153-156.
6. M. Bauer, D. Dengler, and G. Paul, "Instructible Agents for Web Mining," in Proc. IUI2000, New Orleans, USA, 2000, pp. 21-28.
7. T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web", Scientific American, May 2001.
8. J. Freire, B. Kumar, and D. Lieuwen, "WebViews: Accessing Personalized Web Content and Services," in Proc. WWW2001, Hong Kong, Hong Kong, 2001, pp. 576-586.
9. J. Fujima, A. Lunzer, K. Hornbæk, and Y. Tanaka, "Clip, Connect, Clone: Combining Application Elements to Build Custom Interfaces for Information Access." To appear in Proceedings of the 17th annual ACM symposium on user interface software and technology (UIST 2004), October; Santa Fe, NM, 2004.
10. P. B. Golgher, A. H. F. Laender, A. S. da Silva, and B. Ribeiro-Neto, "An Example-Based Environment for Wrapper Generation," in Proc. the 2nd International Workshop on The World Wide Web and Conceptual Modeling, Salt Lake City, USA, 2000, pp. 152-164.
11. G. Grieser, K. P. Jantke, S. Lange, and B. A. Thomas, "Unifying Approach to HTML Wrapper Representation and Learning," in Proc. Discovery Science 2000, Kyoto, Japan, 2000, pp. 50-64.
12. K. Ito and Y. Tanaka, "A Visual Environment for Web Application Composition," in Proc. 14th ACM Conference on Hypertext and Hypermedia, Nottingham, UK, 2003, pp. 184-193.
13. M. Pierce, G. Fox, C. Youn, S. Mock, K. Mueller, and O. Balsoy, "Interoperable Web Services for Computational Portals," in Proc. the 2002 ACM/IEEE conference on Supercomputing, Baltimore, USA, 2002, pp.1-12.
14. A. Sahuguet, and F. Azavant, "Building Intelligent Web Applications using Lightweight Wrappers," Data & Knowledge Engineering, vol.36, No. 3, pp. 283-316, 2001.

15. M. Stal, “Web Services: Beyond Component-based Computing,” Communications of the ACM, vol.45 10 , pp. 71-76, 2002.
16. A. Sugiura, and Y. Koseki, “Internet Scrapbook: Automating Web Browsing Tasks by Demonstration,” in Proc. the ACM Symposium on User Interface Software and Technology (UIST), San Francisco, USA, 1998, pp. 9-18.
17. Y. Tanaka and T. Imataki, “IntelligentPad: A Hypermedia System allowing Functional Composition of Active Media Objects through Direct Manipulations,” in Proc. IFIP’89, San Francisco, USA, 1989, pp.541-546.
18. Y. Tanaka, “Meme media and a world-wide meme pool,” in Proc. ACM Multi-media 96, Boston, USA, 1996, pp. 175-186.
19. Y. Tanaka, Meme Media ad Meme Market Architectures –Knowledge Media for Editing, Distributing, and Managing Intellectual Resources–, IEEE Press & Wiley-Interscience, NJ, 2003.
20. World Wide Web Consortium, “XML Path Language (XPath)”, 1999. http://www.w3.org/TR/xpath.
21. World Wide Web Consortium, “Simple Object Access Protocol (SOAP) 1.1”, 2000. http://www.w3.org/TR/SOAP/
22. World Wide Web Consortium, “Document Object Model (DOM) Level 2 HTML Specification”, 2003. http://www.w3.org/DOM/