

Meme Media Technologies for the ad hoc Federation of Intellectual Resources over the Web

Yuzuru TANAKA and Jun FUJIMA

Meme Media Laboratory, Hokkaido University,
N.13 W.8, Sapporo 060-8628, Japan.
{tanaka, fujima}@meme.hokudai.ac.jp

Abstract: The Web works as an open repository of intellectual resources published in the form of Web documents and Web applications. Such intellectual resources include not only multimedia contents but also application tools and services. Pervasive computing denotes an open system of intellectual resources in which users can dynamically select and interoperate some of these intellectual resources to perform their jobs satisfying their dynamically changing demands. In pervasive computing, the ad hoc definition and/or execution of interoperation among computing resources is called federation. While the integration denotes interoperation among computing resources with standard interoperation interfaces, federation denotes interoperation among computing resources without a priori designed interoperation interfaces. We define knowledge federation as federation of intellectual resources published in the form of Web documents, i.e., Web applications. This paper uses the meme media technologies developed by our group to propose a new framework for the ad hoc federation of intellectual resources over the Web, and shows the detail mechanism of this framework.

1 Introduction

Some intellectual resources over the Web are computing resources such as computation services and database services. The Web works as an open repository of computing resources. It treats each computing resource as a service, and publishes it as a Web application. Pervasive computing denotes an open system of computing resources in which users can dynamically select and interoperate some of these computing resources to perform their jobs satisfying their dynamically changing demands. Such computing resources include not only services on the Web, but also embedded and/or mobile computing resources connected to the Internet through wireless communication. In pervasive computing, the ad hoc definition and/or execution of interoperation among computing resources is called federation. While the integration denotes interoperation among computing resources with standard interoperation interfaces, federation denotes interoperation among computing resources without a priori designed interoperation interfaces. We define knowledge federation as federation of computing resources published in the form of Web documents, i.e., Web applications.

Federation over the Web is attracting the attention for interdisciplinary and international

advanced reuse and interoperation of heterogeneous intellectual resources especially in scientific simulations [MST00], digital libraries [FJH01], and research activities [BB01]. It may be classified into two types: federation defined by programs and federation by users. Most studies on federation focused on the former type. Their approach is based on both the proposal of a standard communication protocol with a language to use it and a repository with a matching mechanism between service providing programs and service consuming programs [EFGK03]. The origin of such an idea can be found in [Gel92]. Federation of this type over the Web uses Web service technologies. In this paper, we focus on the latter type, i.e., federation defined by users. Computing resources over the Web in this case must be visible to users and can be directly manipulated by users. Therefore, this paper mainly focuses on the ad hoc federation of arbitrary Web applications, namely the ad hoc knowledge federation over the Web.

Meme media technologies proposed and developed by our group [IT03, TI89, Tan96, Tan03], when applied to the Web, can federate arbitrarily selected Web applications with each other in an ad hoc manner by extracting arbitrary input forms and output contents from these Web pages, and by defining interoperation among them only through direct manipulation. They work as ad hoc knowledge federation technologies over the Web. Meme media technologies make the Web or its arbitrary subsystem work as a pervasive computing environment. They can be further applied to federation of computing resources distributed over ubiquitous computing environments including embedded and/or mobile computing resources if these resource are made accessible through the Web in some way. This makes such environments also work as pervasive computing environments.

This paper reviews and reinterprets meme media technologies from a new view point of knowledge federation over the Web and pervasive computing environments. We will propose a general framework for clipping arbitrary Web contents and functionally recombining such clips based on their original IO relationships in the original Web pages. This framework called C3W allows us to define a mathematical equation among clips extracted from mutually independent Web navigations for their interoperation. In our previous papers on the application of meme media technologies to the Web, we proposed how to extract Web contents as live clips [TIK04], how we can wrap Web applications into meme media objects by arbitrarily specifying some input forms and output portions to work as IO ports [IT03]. We also proposed the C3W framework without showing the details of its internal mechanism [FLHT04]. In our recent paper [TIF05], we tried to unify all these frameworks into a single but complicated framework. Here in this paper, we will show a simple extended C3W framework for clipping Web contents and defining their interoperation. We will also show that this framework allows us to embed such Web clips in an MS Word document and/or an MS Excel sheet without losing their interoperability.

We have also already developed a technology for the semiautomatic generation of a meme media object from an arbitrary Web service with its WSDL description. This technology automatically analyses the WSDL description and shows a list of input and output ports of the Web service for us to select some of them to work as IO ports of the generated meme media object. Such a meme media object works as a proxy object of the Web service, and can be easily combined with any other meme media objects including Web clips and their federations. Therefore, meme media technologies enable us to federate any clips of any

Web applications and any Web services.

Our extended C3W framework for the knowledge federation over the Web can be also considered as a Web-based GRID computing technology, which allows us to easily extract information retrieval services, database services, and/or simulation services as meme media objects from arbitrary Web applications an/or Web services over the Internet and/or intranetworks, and to make them interoperate with each other by combining them together only through direct manipulation without any programming.

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. An HTML fragment denotes what becomes an HTML view if it is enclosed by HTML tags. Here in this pa-

per, we will not distinguish HTML fragments from HTML views. 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 [JSH03]. The DOM tree representation allows you to identify any HTML-document portion, which corresponds to a DOM tree node, with its HTML Path expression [CD99] such as /HTML[1]/BODY[1]/TABLE[1]/TR[2]/TD[2]. The precise definition of DOM and HTML Path is out of the scope of this paper. For the details, you may refer to their specification documents [CD99, JSH03].

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 ]');
```

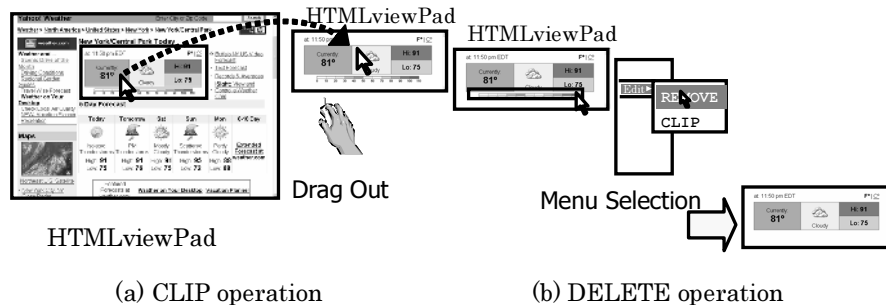


Figure 1: Direct Manipulations for extracting and removing views

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.

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 1 (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 1 (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]');
```

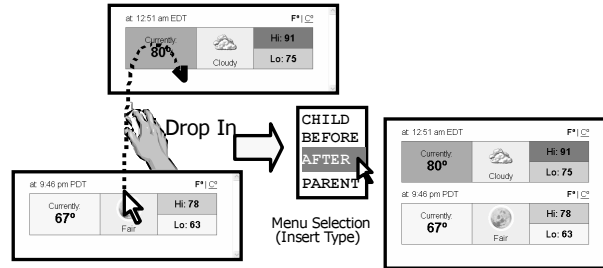


Figure 2: Direct manipulation for inserting a view in another view

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 2 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.

2.4 Automatic Generation of Default Slots

Each HTMLviewPad has the following two default slots. The #SourceURL slot stores the URL of the source document. The #ViewEditingCode slot stores the view editing code. 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) a user specifies its update, or (3) it becomes active after its loading from a file.

When you clip out a node with a relative path expression π from an HTML view with a view editing code σ , the HTMLviewPad automatically creates a new HTMLviewPad with the following view editing code:

```
view =  $\sigma$ .CLIP( $\pi$ );
```

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 Recombination and Linkage of Web Resource Clips

3.1 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. The framework described in the sequel is called C3W, and proposed by our group in 2004 [FLHT04]. Here we show its detailed mechanism. Figure 3 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. The HTMLviewPad internally holds a sequence of such operations as a view editing code including also event operations. 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.

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.

These clips have the following view editing codes:

```
Clip1 view = url.GETHTML()  
          .CLIP( $\pi$ )
```

```
Clip2 view = url.GETHTML()  
          .CLIP( $\pi_1$ )
```

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

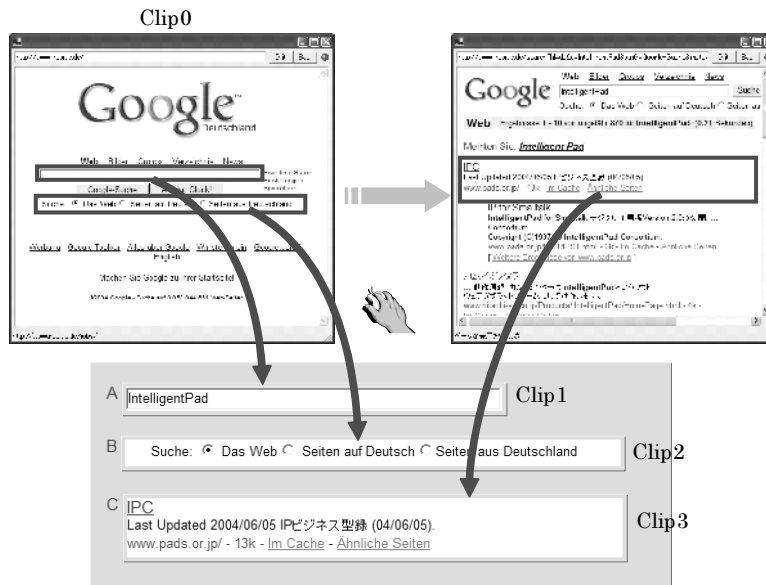


Figure 3: Clips extracted from a single navigation and their recombination on a ClipboardPad.

```
Clip0 view = url.GETHTML()
```

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()
             .SET( $\pi$ , *)
             .SET( $\pi1$ , *)
             .SUBMIT( $\pi2$ );
```

where $SET(\pi, *)$ and $SET(\pi1, *)$ denote that the current HTML views at nodes π and $\pi1$ are set to the nodes π and $\pi1$. This substitution may seem to be redundant. However, in the later discussion, these parameter values are connected to the Clip1 and Clip2, and are updated by 'set' messages from these clips that may have changed their HTML views through user's keyword input and radio button selection.

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()
        .SET( $\pi$ , *)
        .SET( $\pi 1$ , *)
        .SUBMIT( $\pi 2$ )
        .CLIP( $\pi 3$ )

```

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. 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. 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 $\pi 1$ with the new cell 'B'. This entry value becomes as follows:

```

view = url.GETHTML()
        .CELL('A',  $\pi$ )
        .CELL('B',  $\pi 1$ );

```

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',  $\pi$ )
        .CELL('B',  $\pi 1$ )
        .SET( $\pi$ , *)
        .SET( $\pi 1$ , *)
        .SUBMIT( $\pi 2$ )
        .CELL('C',  $\pi 3$ );

```

The ClipboardPad connects Clip3 to the slot #C.

When we input some keyword on Clip1, it sends a ‘set’ message to the slot #A with its updated HTML view ‘<input value=’IntelligentPad’ id=... , type=text ... />’. 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 HTML view sent by Clip1, 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 ($\pi 1, *$). Then this ClipboardPad executes the view editing code.

Whenever the Clipboard executes a view editing code, it sends ‘update’ messages to all the clips working as output devices. Each of these 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.

3.2 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 ClipboardPad 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 Y_1, Y_2, \dots, Y_k clipped out from another different navigation just by specifying an equation $X \leftarrow f(Y_1, Y_2, \dots, Y_k)$, where f is an arbitrary computable function. Whenever one of the cells Y_1, Y_2, \dots, Y_k is updated, the ClipboardPad computes $f(Y_1, Y_2, \dots, Y_k)$, 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 ClipboardPad.

A ClipboardPad also allows us to create a new slot, which will be automatically given a new slot name. Such a slot initially has no functionality. To define the functionality of such a slot #X, you may specify an equation either $X \leftarrow f(Y)$ or $Y \leftarrow g(X)$ using an existing slot #Y created by the pasting of a Web clip. If the equation $X \leftarrow f(Y)$ is defined, then the value of the slot #X is defined from the value of #Y. If the equation $Y \leftarrow g(X)$ is defined, then this new slot #X works as an input slot, and its value determines the value of the slot #Y.

Our C3W framework provides two special functions *setTextValue* and *extractTextValue* to be used in slot defining equations. For a newly defined text type input slot #X1 and a slot #X2 created for a Web clip corresponding to an input form, the equation $X2 \leftarrow \text{setTextValue}(\cdot/@\text{value}, X1)$ sets the value X1 to the specified attribute *./@value* of the HTML node specified by X2. In the example in Figure 3, you may create a new text input slot #D and specify the equation ‘ $\leftarrow \text{setTextValue}(\cdot/@\text{value}, D)$ ’ for the cell ‘A’.

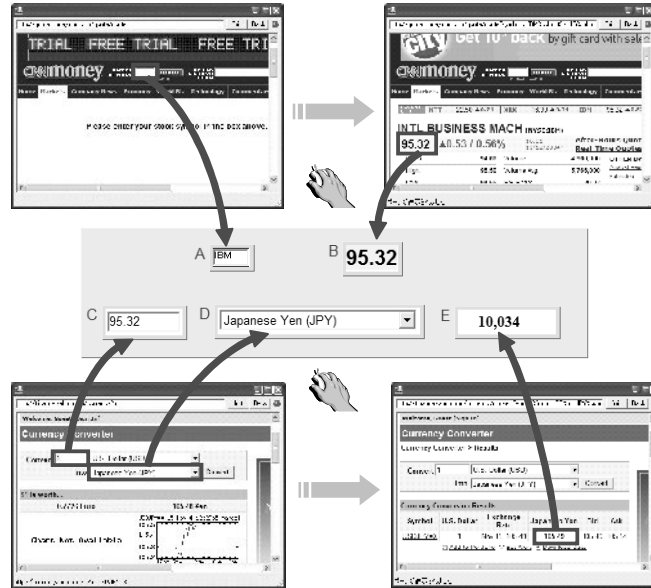


Figure 4: Clips extracted from more than one navigation and their recombination on a ClipboardPad.

Then the slot #D works as a keyword input slot.

For a newly defined text type output slot #Y1 and a slot #Y2 created for a Web clip, the equation $Y1 \leftarrow extractTxtValue(Y2)$ performs `//text()` for the HTML node specified by the current value of the slot #Y2 to obtain a set of text sub nodes, concatenates their text strings using a space as the delimiter, and sets this concatenated text to the slot #Y1. The operator `//text()` lists all the text strings under the specified HTML node.

Figure 4 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 define a new cell 'F', input a substitution expression $\leftarrow extractTxtValue(B)$ to the cell F, and input another substitution expression $\leftarrow setTxtValue(./@value, F)$ 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.

3.3 Extraction of list indices

Using our C3W framework, we can extract the search keyword input form as a Web clip pad from the first Google Search page, paste it on a ClipboardPad, input a list of search keywords and click the search button in the original Google Search page to obtain the search result page, then click the first candidate url anchor on the search result page to obtain the target page, extract this whole target page as a Web clip, and paste it on the same ClipboardPad. This process constructs a new tool that automatically retrieves the first candidate Web page for each input keyword list. This tool, however, shows only the first candidate Web page. In order to scan all the search result pages one after another, we need to introduce a cursor function to this tool. A cursor has a state with a positive integer value i , and specifies the i -th candidate in some list. Here we will show how we can introduce cursor control mechanism to scan all the search result candidates one after another.

Each search result page of Google Search has a list of search result url anchors. The first and the second candidates in this list have almost the same HTML path expression with different indices at only one location. These two path expressions have the forms $\pi a[1]\pi b$ and $\pi a[2]\pi b$ for some HTML path expressions πa and πb .

Our HTRMLviewPad allows you to extract $\pi a[x]$ from two specified nodes with path expressions $\pi' = \pi a[1]\pi b$ and $\pi'' = \pi a[2]\pi b$. After clipping out the keyword input form from Google Search page and clicking the search button to obtain the search result page, you may temporarily change the mode of the HTRMLviewPad to its index extraction mode. The HTRMLviewPad in this mode allows you to specify the first and second candidates at the HTML nodes π' and π'' in the search result page through mouse operations, and extracts $\pi a[x]$ from π' and π'' . This extraction generates a special pad with the path expression $\pi a[x]$. If π' and π'' do not satisfy the above condition, this extraction fails. You may drag and drop this pad on the ClipboardPad to define a new cell 'B'. This changes the view editing code of the ClipboardPad from

```
view = url.GETHTML()
      .CELL('A',  $\pi$ );
to
view = url.GETHTML()
      .CELL('A',  $\pi$ )
      .SET( $\pi$ , *)
      .SUBMIT( $\pi 2$ )
      .INDEXCELL('B',  $\pi a$ )
      .INDEXSET( $\pi a$ , *);
```

The dropped pad will be deleted, and only the cell, namely the slot, is defined. The operation `INDEXCELL('B', πa)` defines a new cell with 'B' as its name, and associates this cell with the index after the path expression πa . The operation `INDEXSET(πa , *)` sets the index after the path expression πa to the current index value. If the slot #B receives a new positive integer i , this replaces the current path expression $\pi a[-]$ with $\pi a[i]$, and replaces all the appearances of $\pi a[-]$ as prefixes of path expressions in the subsequent part of the view editing code with $\pi a[i]$.

After changing the mode of the HTMLviewPad to the clipping-out mode, you may click the first candidate anchor at the node π' to obtain the destination page. Then you can clip out this whole page and paste it on the same ClipboardPad. This clip Clip4 has the following view editing code.

```
Clip4 view = url.GETHTML()
            .SET( $\pi$ , *)
            .SUBMIT( $\pi 2$ )
            .INDEXSET( $\pi a$ , *)
            .CLICK( $\pi'$ )
            .CLIP(/)
```

The view editing code of the ClipboardPad becomes as follows

```
view = url.GETHTML()
       .CELL('A',  $\pi$ )
       .SET( $\pi$ , *)
       .SUBMIT( $\pi 2$ )
       .INDEXCELL('B',  $\pi a$ )
       .CLICK( $\pi'$ )
       .CELL('C', /)
```

The slot #B enables us to specify an index integer i to obtain the i -th candidate Web page on Clip4. The input i to the slot #B replaces all the appearances of $\pi a[1]$ to $\pi a[i]$. The operation `CLICK(π')` becomes `CLICK($\pi a[i]\pi b$)`, and opens the i -th candidate Web page, whose HTML view is sent to Clip4.

4 Repository and Lookup Services for Knowledge Federation

Knowledge federation requires not only ad hoc interoperability among computing resources but also two kinds of services, i.e., a repository service and a lookup service. A repository service is used to register available computing resources. In our approach to knowledge federation, computing resources are represented as pads. We need a repository service for pads. A lookup service is used to find out a desired computing resource from a repository of available computing resources, and to get its reference as a proxy object as well as a method to access it through the proxy. In our approach, each pad representing a computing resource over the Web is its proxy object. We only need a lookup service of the pad repository for desired pad. Here in this section, we will propose Piazza as such a pad repository service, and show that it can be easily instantaneously constructed by applying the C3W framework to Wiki service. We will also show that the C3W framework is also applied to Wiki Search and/or Google Search to construct a lookup service for Piazza.

Our C3W framework can be applied to arbitrary Web pages, and hence, any legacy Web applications such as Google and Wiki services. It also enables 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 world-wide 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 C , A , and B . The relationship among them is defined as follows. We define the equation for the cell C as $\leftarrow extractTextValue(B)$, and the equation for the cell A as $\leftarrow C$. 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 PadSaverLoader-Pad. This meme pool system based on Wiki technologies is called a Wiki Piazza. Figure 5 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 ClipboardPad.

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.

For the lookup service for Wiki Piazza, we can apply our C3W framework to Wiki Search service. Possible search keywords in this case are types of component pads and parent-child pairs of component pad types. A parent-child pair of component pad types specifies a pair of such two pad types that the latter of them is pasted on the former in all the desired composite pads. The save format of each composite pad also includes these keywords as its substrings. We developed a special pad KeywordExtractionPad that extracts these two different kinds of keyword information from a pad put on itself, and sends them to its parent pad using a 'set' message. Then, in the first step, we clipped out the keyword input form of Wiki Search service to paste it on a ClipboardPad, and created a cell 'A'. In the

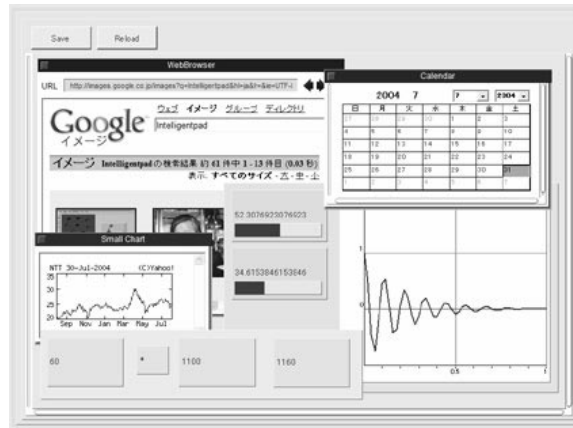


Figure 5: A worldwide repository of pads developed by applying meme media technologies to Wiki.

second step, we specified the first and the second search result candidates to extract the list index, and defined a new cell ‘B’ for this list index on the same ClipboardPad. Then we clicked the first candidate to obtain the first result page, and clipped out the content of this page to paste it on the same ClipboardPad. This defined another new cell ‘C’. Then we paste a KeywordExtractionPad, a text pad, and a PadSaverLoaderPad on the ClipboardPad with their connections respectively to the slots #A, #B, and #C. The composite pad allows us to drop an example composite pad as a query, and returns a set of different Wiki Piazza pages with such composite pads including the specified set of component pads and the specified set of parent-child relationships. The slot #B is used to change the index to scan through all the candidate pages.

5 Embedding Clips in MS Word and Excel Documents

Recently, we have extended our C3W framework so that Web clips can be embedded as objects in MS Word and Excel documents. We developed a special pad object MSPad that can be treated as an MS Word object and/or an MS Excel object, and can be embedded into MS Word and MS Excel documents. MSPads that have been independently embedded in the same Word document work as shared copies of the same MSPad. MSPads also work as ClipboardPads. Each MSPad may have only one child pad. In order to embed Web clips in a word document, you can first embed an MSPad in the word document for each Web clip, and paste this clip on this MSPad. You may also define an appropriate slot, namely a cell, on an embedded MSPad with its relationship to another cell created for a Web clip, and paste a text pad with its connection to this slot. This allows you to use text pads instead of HTMLviewPads. You may also hide the pad borderline to make the embedded contents look naturally as if they are parts of the ordinary Word documents. Figure 6 shows an example Word document with several embedded Web clips interoperating with each other.

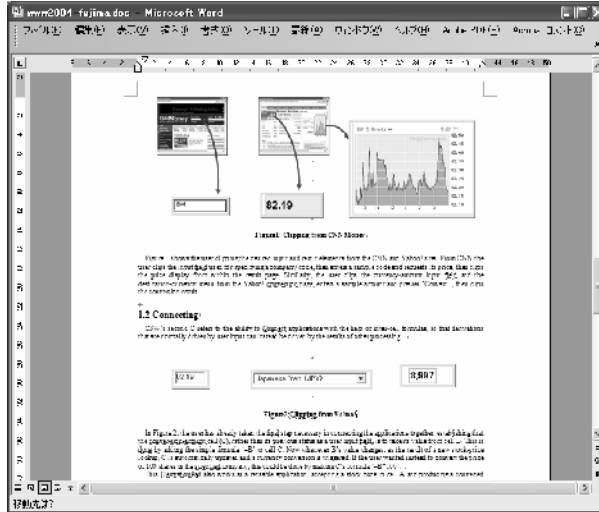


Figure 6: An MS Word document with 6 interoperable embedded Web clips extracted from CNN Money Stock Quote pages.

Each MSPad can be also embedded in any Excel cell. It keeps the value of its slot to which its child pad is connected always equal to the Excel cell value. MSPads embedded in the same Excel sheet work as shared copies of the same MSPad, however each of them may have a different child pad and a different slot. They only share the code list. Figure 7 shows an example Excel sheet with several embedded Web clips interoperating with each other.

6 Related work

Web Service technologies such as SOAP (Simple Object Access Protocol) [BEK⁺00] enable us to interoperate different 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 more than one Web service. Our technologies, on the other hand, provide the client-side direct manipulation operations for users to re-edit intellectual resources embedded in Web pages for the visual definition of their arbitrary new layout combination as well as their interoperation.

The Semantic Web [BLHL01] is an extension of the Web in which Web contents are associated with explicit machine-processable semantics. The Semantic Web Service technologies [ABH⁺01], which integrate Web Service technologies and Semantic Web technologies, aim to automate Web service composition. In our approach, on the other hand, we do not aim to automate any composition of Web applications. We focus on how instantaneously users can create wrappers for Web applications when the users want to reuse them in some functional combinations with other applications.

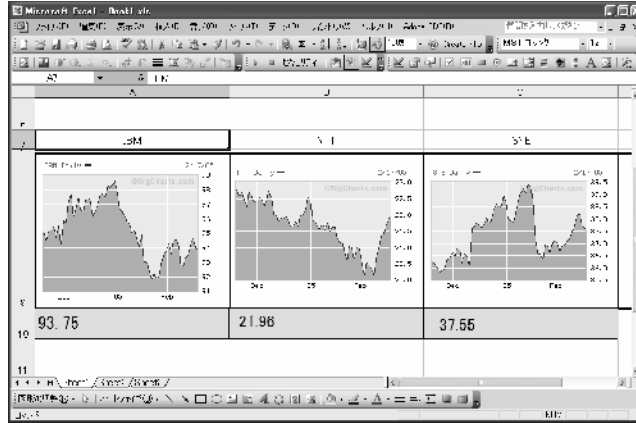


Figure 7: An MS Excel sheet with interoperable embedded Web clips extracted from CNN Money Stock Quote pages.

There are a few preceding research studies that adopted programming-by-demonstration (PBD, for short) technologies on Web pages. Internet Scrapbook [SK98] 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 [BD99, BDP00] 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 [AFKL00] and WebView [FKL01] provide familiar man-machine interfaces to record and replay users' actions. Users can create and update their 'smart bookmarks', which are shortcuts to Web contents and represent 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 parameter values of 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 whether each input form value is to be requested at the playback time or to be a priori stored at the demonstration time. However, in WebView, it seems difficult for end-users to create a new view that integrates different Web applications.

Sometimes Web applications may revise the format of their front-end HTML pages. There are lots of preceding research studies on the induction of a Web wrapper from a given set of example extractions from an arbitrary Web page. However, there are hardly any other

research studies that allow end-users to wrap more than one Web application and to define interoperation among them in the same environment.

W4F [SA01], which is a semi-automatic wrapper generator, provides a visual support tool to define an extraction. The system can create a wrapper class written in Java from user's demonstrations. To use this wrapper class, users need to write program codes. DEbyE [GLdSRN00] provides a more powerful visual support tool for the wrapping of Web applications. DEbyE stores the extracted text portions in an XML repository. Users need to use another XML tool to combine extracted data from Web applications. LEXIKON [GJLT00] can learn an underlying relation among objects within a Web page from a user-specified ordered set of text strings to extract. It provides no GUI tool for the join of two extracted relations.

7 Conclusion

Meme-media architectures work as the enabling technologies for the ad hoc federation of intellectual resources including multimedia contents, application tools and services embedded in Web documents published over the Internet and/or intra network systems. Meme media technologies provide another way of defining interoperations among different Web resources, which, in de facto standard approach, are defined using Web Service and Semantic Web technologies. The former requires no programming but clipping and pasting of input forms and some output portions of existing Web application pages for the definition of their interoperation, while the latter requires programming to define a new composition. Meme media technologies, however, do not exclude the use of these de facto standard technologies in their paradigm. We have already developed a Web-service wrapper, which, when given a Web service with its WSDL description, analyzes this description, and automatically creates a pad that works as a proxy of this Web service. This tool shows its user all the I/O functions of the target Web service, and allows him to select some of them to work as slots. Such a proxy pad can be also combined with any pads. Meme media technologies add a new functional dimension to standard Web technologies without losing any of their functions.

Meme media objects composed by our clipping and combining framework are not robust against the DOM-tree structural change of source Web pages. They may need to be re-defined whenever such structural changes may occur in some of their source Web pages. Meme media technologies do not aim at the automatic composition of Web resources. They aim at the flexible, instantaneous, ad hoc definition and execution of their interoperation.

Meme media technologies also allow us to embed Web clips in MS Word documents and Excel sheets without losing their interoperability in the original Web pages. This provides a document-based architecture for the compilation of related live information items obtained from more than one mutually independent Web applications. An example is a portfolio document showing live stock quotes of more than one company and their weighted sum together with the statistics and movements of these values. This frame-

work also works as a Web-based GRID computing framework to federate more than one computing resources represented as Web applications and/or Web services.

References

- [ABH⁺01] Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Sheila A. McIlraith, Srinu Narayanan, Massimo Paolucci, Terence Payne, Katia Sycara, and Honglei Zeng. "DAML-S: Semantic Markup For Web Services". In *Proceedings of the International Semantic Web Workshop*, 2001.
- [AFKL00] Vinod Anupam, Juliana Freire, Bharat Kumar, and Daniel Lieuwen. Automating Web navigation with the WebVCR. In *Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications networking*, pages 503–517, Amsterdam, The Netherlands, The Netherlands, 2000. North-Holland Publishing Co.
- [BB01] Michael J. Bass and Margret Branschofsky. DSpace at MIT: meeting the challenges. In *JCDL '01: Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries*, page 468, New York, NY, USA, 2001. ACM Press.
- [BD99] Mathias Bauer and Dietmar Dengler. InfoBeans-Configuration of Personalized Information. In *Proceedings of the International Conference on Intelligent User Interfaces*, pages 153–156, 1999.
- [BDP00] Mathias Bauer, Dietmar Dengler, and Gabriele Paul. Instructible information agents for Web mining. In *IUI '00: Proceedings of the 5th international conference on Intelligent user interfaces*, pages 21–28, New York, NY, USA, 2000. ACM Press.
- [BEK⁺00] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn., Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple Object Access Protocol (SOAP) 1.1. W3C Recommendation, 2000. <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>.
- [BLHL01] T. Burners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5), May 2001.
- [CD99] James Clark and Steve DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation, 1999. <http://www.w3.org/TR/xpath>.
- [EFGK03] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.
- [FJH01] Ling Feng, Marfred A. Jeusfeld, and Jeroen Hoppenbrouwers. Towards knowledge-based digital libraries. *SIGMOD Rec.*, 30(1):41–46, 2001.
- [FKL01] Juliana Freire, Bharat Kumar, and Daniel Lieuwen. WebViews: accessing personalized web content and services. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 576–586, New York, NY, USA, 2001. ACM Press.
- [FLHT04] Jun Fujima, Aran Lunzer, Kasper Hornbæk, and Yuzuru Tanaka. Clip, connect, clone: combining application elements to build custom interfaces for information access. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 175–184, New York, NY, USA, 2004. ACM Press.

- [Gel92] D. Gelernter. *Mirror Worlds*. Oxford University Press, 1992.
- [GJLT00] Gunter Grieser, Klaus P. Jantke, Steffen Lange, and Bernd Thomas. A Unifying Approach to HTML Wrapper Representation and Learning. In *DS '00: Proceedings of the Third International Conference on Discovery Science*, pages 50–64, London, UK, 2000. Springer-Verlag.
- [GLdSRN00] Paulo Braz Golgher, Alberto H. F. Laender, Altigran Soares da Silva, and Berthier A. Ribeiro-Neto. An Example-Based Environment for Wrapper Generation. In *ER '00: Proceedings of the Workshops on Conceptual Modeling Approaches for E-Business and The World Wide Web and Conceptual Modeling*, pages 152–164, London, UK, 2000. Springer-Verlag.
- [IT03] Kimihito Ito and Yuzuru Tanaka. A visual environment for dynamic web application composition. In *HYPERTEXT '03: Proceedings of the fourteenth ACM conference on Hypertext and hypermedia*, pages 184–193, New York, NY, USA, 2003. ACM Press.
- [JSH03] Philippe Le Hegaret Johnny Stenback and Arnaud Le Hors. Document Object Model (DOM) Level 2 Specification. W3C Recommendation, 2003. <http://www.w3.org/TR/2003/REC-DOM-Level-2-HTML-20030109/>.
- [MST00] John A. Miller, Andrew F. Seila, and Junxiu Tao. Finding a substrate for federated components on the web. In *WSC '00: Proceedings of the 32nd conference on Winter simulation*, pages 1849–1854, San Diego, CA, USA, 2000. Society for Computer Simulation International.
- [SA01] Arnaud Sahuguet and Fabien Azavant. Building intelligent web applications using lightweight wrappers. *Data Knowl. Eng.*, 36(3):283–316, 2001.
- [SK98] Atsushi Sugiura and Yoshiyuki Koseki. Internet scrapbook: automating Web browsing tasks by demonstration. In *UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology*, pages 9–18, New York, NY, USA, 1998. ACM Press.
- [Tan96] Yuzuru Tanaka. Meme media and a world-wide meme pool. In *MULTIMEDIA '96: Proceedings of the fourth ACM international conference on Multimedia*, pages 175–186, New York, NY, USA, 1996. ACM Press.
- [Tan03] Yuzuru Tanaka. *Meme Media and Meme Market Architectures: Knowledge Media for Editing, Distributing, and Managing Intellectual Resources*. John Wiley & Sons, Inc., New York, NY, USA, 2003.
- [TI89] Y. Tanaka and T. Imataki. IntelligentPad: A Hypermedia System allowing Functional Composition of Active Media Objects through Direct Manipulations. In *Proceedings of the IFIP '89*, pages 541–546, San Francisco, CA, 1989.
- [TIF05] Yuzuru Tanaka, Kimihito Ito, and Jun Fujima. Meme Media for Clipping and Combining Web Resources. *World Wide Web: Internet and Web Information Systems*, 2005. (to appear).
- [TIK04] Yuzuru Tanaka, Kimihito Ito, and Daisuke Kurosaki. Meme media architectures for re-editing and redistributing intellectual assets over the web. *Int. J. Hum.-Comput. Stud.*, 60(4):489–526, 2004.