Robustness of External Annotation for Web-Page Clipping: Empirical Evaluation with Evolving Real-Life Web Documents

Masahiro Hori
Faculty of Informatics, Kansai University
2-1-1 Ryozenji-cho, Takatsuki-shi
Osaka 569-1095, Japan
horim@res.kutc.kansai-u.ac.jp

Mari Abe[*]
IBM Tokyo Research Laboratory
1623-14 Shimotsuruma, Yamato-shi
Kanagawa 242-8502, Japan
maria@jp.ibm.com

Kouichi Ono
IBM Tokyo Research Laboratory
1623-14 Shimotsuruma, Yamato-shi
Kanagawa 242-8502, Japan
onono@jp.ibm.com

ABSTRACT

Web metadata is crucial for providing machine-understandable descriptions of Web resources, and has a number of applications such as discovery, qualification, and adaptation of Web documents. While annotations are often embedded into a Web document, annotations can also be associated externally by means of addressing expressions represented with the XPath language. However, there has been little empirical study of robust pointing using XPath expressions, in spite of the increasing prevalence of the XPath language not only for use with XSLT, but also in emerging content adaptation systems. The objective of this study is to evaluate the robustness of XPath expressions especially the ones that can be used for the generation of external annotations, and draw practical implications to the reliable use of external annotation.

1. INTRODUCTION

Web annotation is crucial for providing not only human-readable remarks, but also machine-understandable descriptions, and has a number of applications such as discovery, qualification, and adaptation of Web contents [19]. As more and more Web-enabled personal devices are becoming available for connecting to the Internet, the same Web documents need to be rendered differently on different client devices. Adaptation of Web document to delivery context is thus crucial for transparent Web access, which may depend on client capabilities, network connectivity, or user preferences [8]. The long-term goal of our research is to establish technologies of customizing Web documents suitable for delivery context. The customization or document adaptation requires annotation that indicates the ways of modifying the document at hand.

Annotations can be embedded into a Web document as inline annotations, which are often created as extra attributes of document elements. Most existing HTML browsers ignore unknown attributes added to HTML elements, without being bothered by the proprietary inline annotations. Because of its simplicity, inline annotation has been often adopted as a way of associating annotation with HTML documents [22, 25, 10, 12]. An advantage of the inline approach is the ease of annotation maintenance without the bookkeeping task of associating annotations with their target document. The inline approach, however, requires annotators to have document ownership because annotated documents need to be modified whenever inline annotations are created or revised.

On the other hand, the external annotation approach [13] does not suffer from these issues related to document ownership. The important point of the external annotation approach is that it facilitates the sharing and reuse of annotations across Web documents. In addition, the mixing of content and metadata is not desirable with regard to the design guideline that content should be separated from presentation. Therefore, it is assumed in this study that such metadata is maintained separately from a target document, and exploited dynamically at runtime by a content adaptation engine.

Since Web documents may change over time, it is not always obvious what kinds of addressing expression keep pointing the same target element regardless of the document changes. It was reported that a key complaint in the use of electronic annotation was the situation in which an annotation cannot point any portion of a target document [4]. These are aspects of the issue related to robust positioning, which has been investigated in a couple of empirical studies [24, 3]. However, there has been little empirical study of robust pointing using XPath expressions, in spite of the increasing prevalence of the XPath language not only for use with XSLT, but also in emerging content adaptation systems [13, 26, 23, 2].

The objective of this study is to evaluate the robustness

[*]This author also belongs to Graduate School of Science and Technology, Keio University.

of XPath expressions especially the ones that can be used for the generation of external annotations, and draw practical implications to the reliable use of external annotation. In the next section, we introduce variations in annotation tools, on the basis of two authoring methods (annotation by selection and by example) as well as the different roles of annotations for assertion and transformation. Section 3 explains an annotation language for Web page clipping, which has been adopted for commercially available software products, and its application to page clipping for small-screen devices and portal site development. In Section 4, we present an empirical evaluation of the robustness of XPath expressions with regard to the changes in real-life HTML pages. In particular, it was investigated to what extent those expressions continued to point at the same nodes in the modified pages during the observation period of one year and three months. Finally, we discuss the advantages and limitations of the XPath expressions taking account of the generation of external annotations.

2. VARIATIONS IN ANNOTATION TOOLS

An annotation in general declares properties that qualify a particular portion of a target document. In some cases, however, annotations may indicate structural changes for the annotated portion of a target document. In order to clarify the distinction of these two roles, the former is called *assertional annotation*, while the latter *transformational annotation* [14]. Note here that this distinction is not exclusive, because every annotation is intrinsically an assertion.

Transformational annotation has been used for Web content adaptation, in which structural changes of a target document are needed [13, 23, 26, 28]. In contrast to assertional annotation languages (such as Dublin Core Metadata [7]), transformational annotation languages (such as XSLT [28]) are more like programming languages, and not necessary easy for annotators to create transformational annotations by using conventional annotation by selection approach.

It is simple for annotation authors or annotators to indicate a location to be annotated and create an assertion as annotation content. This is an approach that we call *annotation by selection*, and is adopted by existing annotation tools [1, 5, 10, 13, 17, 23, 11]. On the other hand, for transformational annotations, it is easier for annotators to modify a target document toward the desired results of the customization, rather than to indicate the ways of modifications declaratively as assertional annotations. This is a basic idea behind an approach what we call *annotation by example*, which was originally proposed in our previous work on the generation of XSLT rules [18].

According to the distinctions of annotation authoring methods and the roles of annotation, Table 1 summarizes variations in annotation tools. Annotators can select a portion of document to be annotated and declare properties on the selected portion as assertional annotation. This type of annotation tools support assertional annotation by selection [Table 1(a)], and most of the existing annotation tools fall into this category. Even when annotations are used for structural changes of a target document, it is possible for authors to create transformational annotations by selecting portions to be changed and declare instructions of transformations as annotations. This type of annotation tools support transformational annotation by selection [Table 1(b)].

In order to create transformational annotations, however, annotation by example would be much easier for annotators, because the annotators can work with a concrete example and create a desired result interactively with the example. In particular, the example-based method allows annotators to generate transformational annotations on the basis of annotators' editing operations conducted to come up with a desired result. This type of annotation tools follows transformational annotation by example [Table 1(c)]. This example-based method is particularly useful for the transformational annotations, and would not make sense for assertional annotations, because it is not intuitive for annotators to indicate assertional annotations as results of structural changes of a target document.

Table 1: Variations in annotation tool design

		Authoring methods	
		By selection	By example
Roles of annotation	Assertion	(a)	N/A
	Transformation	(b)	(c)

The core part of the tool configuration is independent of any particular views and editors, and consists of two document object models (DOMs) [9]: one for a target document, and another for an annotation document. It is assumed here that the creation of an annotation document is a primary task of annotators, and the annotators are not allowed to modify a target document. The assertional annotation by selection is the most typical and provide a comprehensive way of annotation. The details of this type of annotation tool is reported in another article [1, 16]. According to our interests in customization of Web documents, this paper focuses on the transformational annotation, and in particular the two advanced approaches to generating annotations by selection [Table 1(b)] and by example [Table 1(c)].

2.1 Transformational Annotation by Selection

Figure 1 depicts the configuration for transformational annotation by selection. This type of annotation tools relies on a target document viewer, because portions of a target document can only be selected without any modification. With this type of annotation tools, first an annotator opens a target document to be customized. The annotator then selects portions of the target document by using a document viewer [Figure 1 (a)], and indicates how each of the selected portion to be modified (*e.g.*, remove and enlarge). Transformational annotation can then be generated [Figure 1 (b)] on the basis of the portions of a target document selected by an annotator.

2.2 Transformational Annotation by Example

If a person knows how to perform a task to be executed by a computer, perhaps the person's knowledge can somehow be exploited for the creation of a program to perform the task. This is the idea behind *programming by example* [20]. Programming by example is a natural approach to generating transformational annotation for page designers or novice programmers, because such users need only work with examples of how to transform a document at hand, and are given with generated annotations that can replicate the

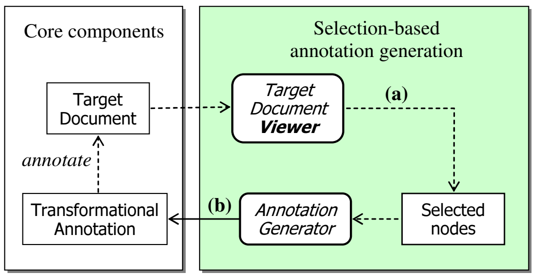


Figure 1: Tool configuration for transformational annotation by selection

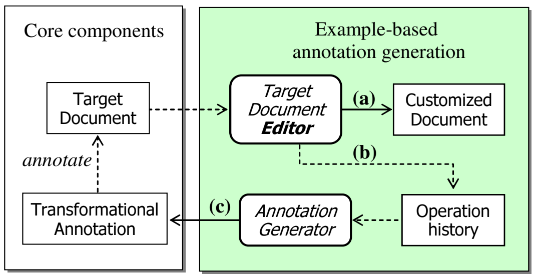


Figure 2: Tool configuration for transformational annotation by example

same transformation.

A configuration of the example-based annotation tool is depicted in Figure 2. This type of annotation tools relies on a target document editor rather than a viewer in contrast to the case of annotation by selection (see Figure 1). With this type of annotation tools, first an annotator opens a target document to be customized (*e.g.*, an HTML file). The annotator then edits the document by using the full capabilities of a WYSIWYG authoring tool [Figure 2 (a)]. Although the annotator's editing actions are recorded into an operation history [Figure 2 (b)], the annotator does not have to care about the recording process behind the scenes. When the editing is finished, the annotator will have a customized document. At the same time, the annotation generator creates transformational annotation for the document customization [Figure 2 (c)], which can be used by a runtime engine (*e.g.*, XSLT processor) to replicate the transformation from the initial target document to the customized document. Further details on the annotation generation procedure are reported in the other articles [14, 18].

It is possible for both selection-based and example-based approaches to generate transformational annotations, but the selection-based approach is limited in the kinds of annotation constructs to be generated as compared with the example-based approach, because the expressiveness of annotators' selection on a document viewer is far more limited than that of annotators' full editing capability on a document editor.

3. ANNOTATION-BASED DOCUMENT ADAPTATION

Web pages for e-commerce, for example, contain a lot of information such as details of products, product images, and numerous links to other areas of the site, when the pages are created for the desktop computers. However, it may be necessary to deliver portions of this page for users to access through a Web-enabled phone rather than a desktop browser. In such a case, the images and nested HTML tables prepared for a nicely laid out page are a hindrance rather than help. The sheer amount of information becomes unwieldy in the small display, and potentially expensive depending on the user's wireless service.

Transformational annotations provide additional information about Web documents, so that an adaptation engine can make better decisions on the content transformation. The role of annotations here is to provide explicit semantics that can be understood by a content adaptation engine [15].

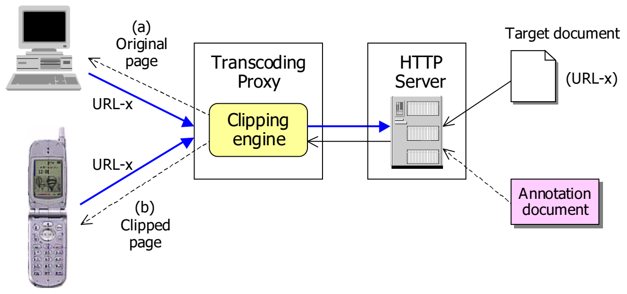


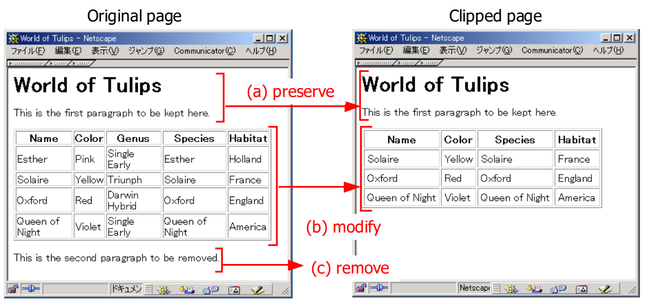
Figure 3: Overview of an annotation-based transcoding

3.1 Page Clipping for Small-Screen Devices

An overview of an annotation-based transcoding process is depicted in Figure 3. Upon receipt of a request from a client, a Web document is retrieved from a content server. Taking account of the capabilities of the client specified in the HTTP request header, a transcoding proxy selects one or more transcoding modules. When a selected transcoding module requires an annotation document, an annotation file is also retrieved from a content server, which may or may not be the same server that retrieved the Web document. The transcoding module may simply return the original document, if a client agent has the rendering capabilities compatible with ordinary desktop computers [Figure 3 (a)]. Alternatively, the original document may be returned with modification, so that the original content can fit into a small screen device [Figure 3 (b)]. The decisions about the content adaptation are made taking account of the client capabilities specified in the HTTP request header.

Content adaptation can be done by using an annotation-based page-clipping engine [26]. At content delivery time, the page-clipping engine may modify the original document with reference to page-clipping annotations and client profiles sent over HTTP. The main idea in the page-clipping annotation language is the notion of a clipping state. By using `<keep>` and `<remove>` elements in the annotation descriptions, users can specify the clipping state to indicate whether the content being processed should be preserved or removed.

As a simple example, an HTML page and its clipped results are shown in Figure 4. In this example, the header and the first paragraph are preserved as shown in Figure 4(a). The table element is modified by deleting the third column and the second row. The cell-padding attribute of the table is increased, so that each table cell can be provided with



World of Tulips - Netscape

World of Tulips

This is the first paragraph to be kept here.

Name	Color	Genus	Species	Habitat
Esther	Pink	Single Early	Esther	Holland
Solaire	Yellow	Triunph	Solaire	France
Oxford	Red	Darwin Hybrid	Oxford	England
Queen of Night	Violet	Single Early	Queen of Night	America

This is the second paragraph to be removed.

World of Tulips - Netscape

World of Tulips

This is the first paragraph to be kept here.

Name	Color	Species	Habitat
Solaire	Yellow	Solaire	France
Oxford	Red	Oxford	England
Queen of Night	Violet	Queen of Night	America

Figure 4: Simple example of an HTML page clipping

```
<?xml version='1.0' ?>
<annot version="2.0">
  <!-- (a) Set the default clipping state to 'keep' -->
  <description take-effect="before"
               target="/HTML[1]/BODY[1]/*[1]">
    <keep/>
  </description>

  <!-- (b) Remove a column and a row of the first -->
  <!--     table, and change a cellpadding         -->
  <!--     attribute value                         -->
  <description take-effect="before"
               target="/HTML[1]/BODY[1]/TABLE[1]">
    <keep/>
    <table>
      <column index="3" clipping="remove"/>
      <column index="*" clipping="keep"/>
      <row index="2" clipping="remove"/>
      <row index="*" clipping="keep"/>
    </table>
    <insertattribute name="cellpadding" value="4"/>
  </description>

  <!-- (c) Set the clipping state to 'remove' -->
  <description take-effect="before"
               target="/HTML[1]/BODY[1]/P[2]">
    <remove/>
  </description>
  <!-- (d) Set the clipping state back to 'keep' -->
  <description take-effect="after"
               target="/HTML[1]/BODY[1]/P[2]">
    <keep/>
  </description>
</annot>
```

Figure 5: Example of page-clipping annotations

margin space [Figure 4(b)]. In addition, the whole of the second paragraph is removed as shown in Figure 4(c). All the structural changes in HTML documents can be easily done by using a WYSIWYG HTML editor.

Figure 5 shows an annotation document that realizes the page clipping illustrated in Figure 4. This transformational annotation can actually be generated by using the example-based annotation generation tool [14]. The `<description>` element prescribes a unit of an annotation statement in the annotation language. The `target` attribute is set to an XPath expression, and identifies the node on which the annotation will be applied, and the `take-effect` attribute indicates whether the annotation is applied before or after the target node. By specifying the value of `target` attribute as **/HTML**[1]**/BODY**[1]**/***[1] [Figure 5(a)], the clipping state is activated after the first element after the first `<BODY>` element, which in this case is an `<H1>`. The `<keep>` element in Figure 5(a) indicates that all the document elements encountered are preserved, until otherwise instructed by another annotation statement. The clipping state is changed to 'remove' just before the second `<P>` element [Figure 5(c)], and changed back to 'keep' after the `<P>` element [Figure 5(d)]. As results, the second paragraph element indicated by **/HTML**[1]**/BODY**[1]**/P**[2] is removed while preserving the elements just before and after the removed element.

Since HTML tables can often be complex elements to clip, the annotation language provides special-purpose elements to make table clipping easier. The `<row>` and `<column>` elements allow user to clip rows and columns without relying on complicated XPath expressions. The table-clipping elements are used in the description shown in Figure 5(b). This description sets the clipping state to 'keep' just before the first table element, and also changes the value of `cellpadding` attribute to 4 by using the `<insertattribute>` element. The `name` attribute of `<insertattribute>` can be specified with an arbitrary name of an attribute available for a target document.

In addition, the description element [Figure 5(b)] declares that the third column, which is indicated by the `index` value of the `<column>` element, is discarded, while the remaining columns are preserved. Note here that the wildcard character to indicate multiple columns (`index="*"`). If a wildcard is specified, all rows (or columns) will be affected, except for those specifically indicated by a separate `<row>` (or `<column>`) element. So, all rows but the second are preserved for the target table.

3.2 Page Clipping for Portal Site Development

Annotation-based page clipping is a useful technique for the adaptation of existing HTML documents to varieties of small-screen devices, but the advantages are not limited to device adaptation. Another promising application of the page clipping technology is the use in Web portals. Web portals are becoming an increasingly popular technology, since it can provide a single point of comprehensive, integrated access to both Web data and applications. However, each of the Web data or application is for the most cases provided assuming to be presented on a desktop browser, and would be too spacious to fit into a small area in a portal page. Page clipping is thus useful for Web pages that are aggregated into a portal site.

Figure 6 illustrates the process of creating a portal page with an annotation-based clipping portlet. Portlets are specialized servlets that plug into and run in portals, and allow to generate dynamic contents. When a portal server receives

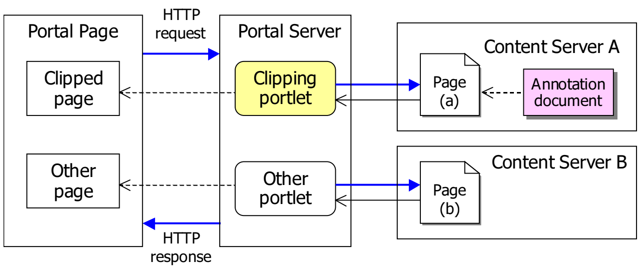


Figure 6: Creation of a portal page with annotation-based clipping portlet

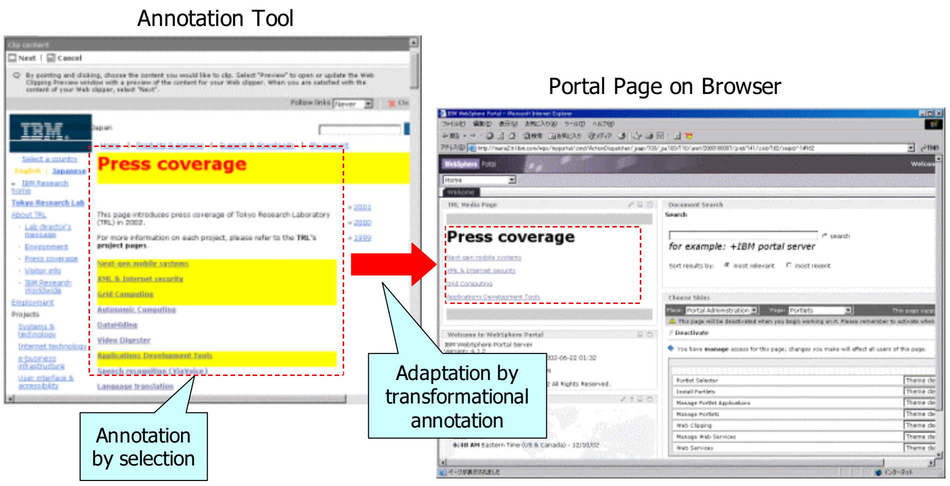


Figure 7: Annotation tool for Web clipping portlet

an HTTP request, the server dispatches the request to each portlet aggregated in the page, and collects the results into a portal page to be returned (Figure 6).

Figure 7 shows a screen of an annotation tool for clipping portlet in the left, and a portal page that includes the clipped page in the right. This annotation tool allows a user to select the portions of the original page to be removed in the portal page [*cf.*, Figure 1 (a)], and the annotation generator creates page-clipping annotations from the selected nodes [*cf.*, Figure 1 (b)].

The selection-based annotation generation was actually adopted for a software product of an annotation tool for a portal server, and extensively used in the development of a supplier portal of an automotive company. In this case, the automotive company extensively used the page-clipping portlet with the annotation tool solely for the simple `<keep>` and `<remove>` clipping operations. The primary reason for the customer's choice was just the simplicity of the authoring process without advanced annotation constructs for page clipping. Since the automotive company needs to aggregate several thousands of existing pages into the portal site, it was not practical to create sophisticated clipping annotations for page by page, and it was reasonable to provide just simple clipping capability to remove headers and side menus in the original documents that were created for browsers on desktop computers.

4. EMPIRICAL EVALUATION

The page-clipping annotation explained in the previous section is adopted for commercially available software products, and has been used for a number of real applications for Web document adaptation. Moreover, the above-mentioned annotation generation tools have been provided as toolkits for the software products: the example-based annotation tool for transcoding proxy [26] and the selection-based annotation tool for portal server development [6].

4.1 Single-Node Pointing Expression

External annotations generated by these two tools use a type of addressing expressions that points to at most one particular node, and will not point at anything if the particular node is not found. This type of addressing expressions is what we call *single-node pointing expressions*, which are suitable for automatic generation of addressing expressions, because single-node pointing expressions can be generated solely on the basis of focal nodes without human intervention.

Although the XPath language [27] provides thirteen axes (such as *ancestor*, *descendant*, *following*, and *preceding*) for specifying the direction of node-set selection in a location step, there exist only the two axes, namely, the *child* and *descendant* axes that can point to every element from a document root element, using only one kind of axis for every location step with position number predicate. Therefore, there are the only two kinds of single-node pointing expressions created by using only one axis. One is to create an expression pointing to an element of the target document using a sequence of child-position location steps (*ChildPosSeq*). Another is for pointing to an element by means of a descendant position (*DescendantPos*).

The *ChildPosSeq* expression simply follows the hierarchy of DOM tree from the root to a target element, and points to at most one element by a sequence of child positions (*e.g.*, **/html**[1]**/body**[1]**/table**[2]**/tbody**[1]**/tr**[1]**/td**[2]). The *DescendantPos* expression, on the other hand, indicates the number of an element with the same tag name in the document order among all the descendant nodes from the root node, and points to at most one element by a descendant position (*e.g.*, **/descendant::table**[8]).

The two annotation tools mentioned above have been using the *ChildPosSeq* expression, and the *DescendantPos* was not used for the generation of addressing expressions. In the remainder of this section, we present an empirical evaluation of the robustness of XPath expressions, in order to draw practical implications to the reliable use of external annotation.

4.2 Evaluation Method and Results

Table 2 shows the basic data of the observed HTML pages. The pages A and B are a corporate top page and a product page of the same company. The page C is a top page of a news media company, while the page D belongs to a software company. These pages were saved each day during about the

Table 2: Basic data of the observed HTML pages

Page	URI	Number of nodes per page [ave. (max, min)]	Depth of document tree [ave. (max, min)]
A	www.ibm.com/	393 (441, 348)	21 (21, 20)
B	www.ibm.com/products/	709 (758, 623)	27 (30, 20)
C	public.wsj.com/	952 (1333, 433)	22 (24, 21)
D	java.sun.com/	909 (1311, 325)	22 (29, 15)

The number of sample pages is 540 for each reference page.

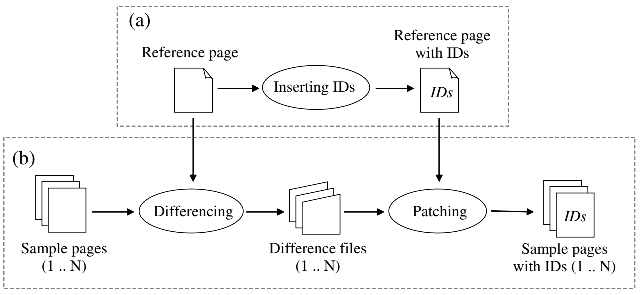


Figure 8: Process of node ID insertion

period of one year and three months, and 540 pages for each are collected as samples for this investigation.

In order to make sure that an XPath expression actually points to the same node in a reference page, it is necessary to know the node-to-node mapping between the reference page and other sample pages with the same URI that are collected during the observation period. For the purpose of this empirical study, an ID attribute was added to every DOM node of the reference and sample pages. Note that the most of existing HTML pages are not well-formed, and therefore cannot be reliably pointed to by XPath expressions. It is assumed here that both the reference page and sample pages are parsed by an HTML parser in advance, and converted to DOM trees before the node ID insertion process.

Figure 8 illustrates the process of node ID insertion, which can be done in the following two steps. The fist step is to add unique identifiers as an attribute value (*e.g.*, `uid="N27"`) to every node in the reference page [Figure 8 (a)]. Note that comment and text nodes were excluded from the ID insertion, because an attribute cannot be added to those nodes. The second step consists of differencing and patching [Figure 8 (b)]. The DOM-tree difference was calculated taking account of the changes in each sample page as compared with a reference page with the same URL. Each difference file consists of a sequence of edit operations that transform the reference page into a corresponding sample page. The difference files are then applied to the reference page with unique identifiers. Finally, as results of the patching, we can obtain a set of sample pages with unique identifiers. As a differencing and merging tool for XML documents, we used the 3DM tool [21].

It is straightforward to insert ID attributes to all the DOM nodes in the reference pages. However, due to the changes in Web pages over time, some nodes will be inserted into or deleted from the documents. The newly inserted nodes do not have any ID attribute, because they cannot be mapped from any node in the reference page. In addition, like the most tree differencing and merging tools, the 3DM algorithm regards an attribute-value change as deletion of the modified node followed by insertion of the same node with updated attribute value. This means the ID attributes disappear when a node is updated as results of an attribute-value change.

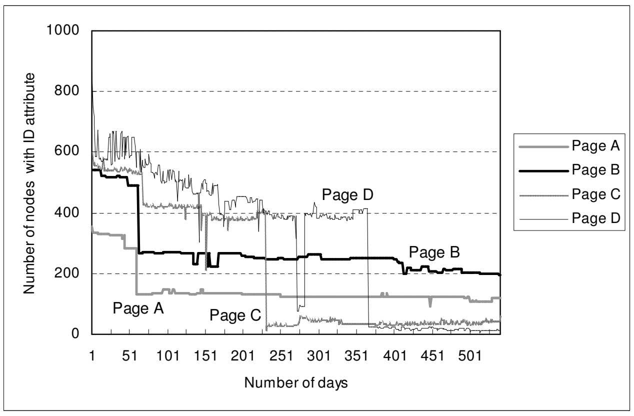


Figure 9: Number of nodes with ID attributes during the observation period

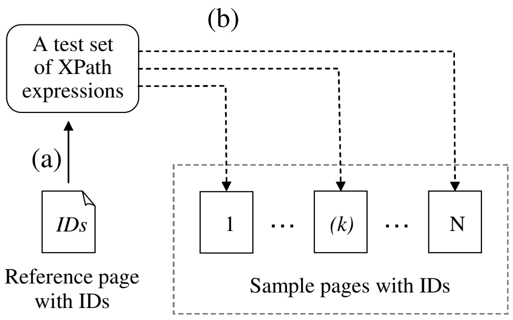


Figure 10: Process of testing XPath expressions

Figure 9 shows the numbers of nodes ID attributes for each sample page along the 540-day observation period. For the pages A and B, there is a sudden decrease in the number of nodes with IDs just after the 50th day due to the small style changes. In addition, since the site design for the pages C and D has been changed, the numbers of nodes with IDs were suddenly decreased in the page C after the 232nd day and the page D after the 367th day. The numbers in the page D tentatively decreased during about 10 days from the 272nd day. This is because the site design was temporally changed due to the annual developer conference[1] held by that company.

The number of nodes with IDs decreases as results of the deletion, insertion, and update of DOM nodes. Note here that the differencing algorithm was always applied to obtain node-to-node mappings from a reference page to a sample page. Therefore, the difference from the reference page would not necessarily increase monotonically, but may be reduced later if the page were modified again to be closer to the reference page. This is why the number of nodes with IDs increases at some points in the observed period.

[1]The 2002 JavaOne Conference

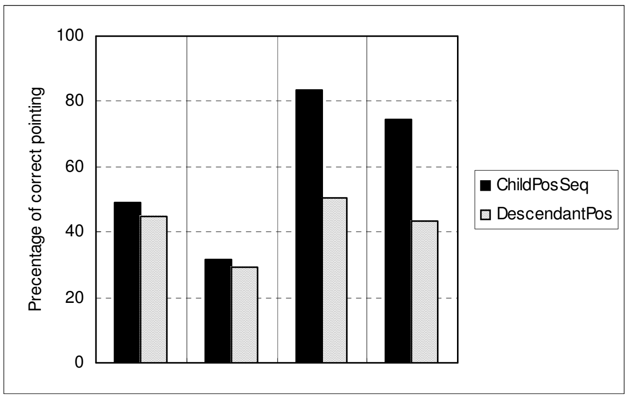


Figure 11: Percentage of correctly pointed nodes for each XPath expression

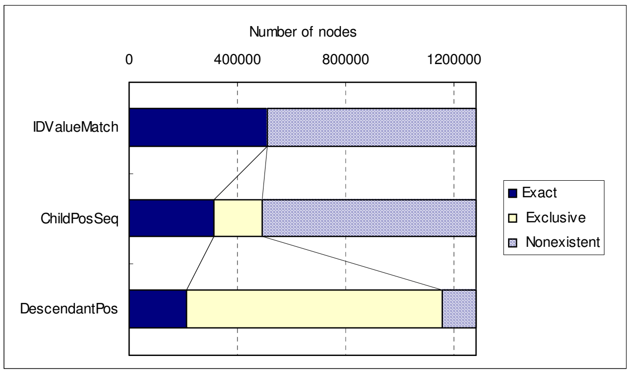


Figure 12: Number of nodes with detailed distinction of pointing status

Every DOM node with a unique identifier in a reference page was regarded as a reference node. For each reference node, we created *ChildPosSeq* and *DescendantPos* expressions, so that they can point to the corresponding reference node. This set of XPath expressions constituted a test set [Figure 10 (a)]. Each expression in the test set was then applied to sample pages with the same URI [Figure 10 (b)]. If an XPath expression actually points to a node with the same ID as the reference node, the expression was regarded as correct in the sample page. Note that an XPath expression may point to multiple nodes in a target document. However, since this evaluation only deals with single-node pointing expressions, correct expressions must point to the only one node with the same ID in a sample page, and must not point to any other nodes in the sample page.

Figure 11 shows the percentages of nodes that are correctly pointed to by each type of expressions. The *ChildPosSeq* always scores higher percentage than *DescendantPos* expression. In the next section, we will further investigate the robustness of the two types of expressions.

5. DISCUSSION

A correctly pointing expression points to a target node that is identical with the reference target node as identified by the ID-attribute value, and does not points to any other nodes other than the reference target. Here we call this correct pointing as *exact* pointing. When an XPath expression does not correctly point to a target node, the ways of incorrect pointing can be categorized into three types: *nonexistent*, *inclusive*, and *exclusive*. Nonexistent pointing is when an XPath expression points to nothing. Inclusive pointing is when an expression points to a node set that includes not only the target node but also nodes other than the target. Exclusive pointing is when an expression points to a node or a set of nodes that does not include the target node at all.

Figure 12 divides the nodes up by pointing status. The total number of tested nodes was more than a million (1,280,880), counting all the HTML elements included in the sample pages throughout the entire observation period. The *IDValueMatch* expressions rely on the ID-attribute value of the target node. An example of the *IDValueMatch* expression is `//*[@uid='N35']`, where the "uid" is the name of the ID attribute we created in the ID-insertion process (Figure 8). The *IDValueMatch* corresponds to a situation when every node of a document at hand is provided with an unique identifier. Although the attribute name "uid" might be accidentally duplicated in some sample pages, no duplication was not found within the pages examined for this study.

Besides the numbers of the exact (correct) pointing, one of the notable feature in Figure 12 is the large number of the exclusive pointing errors for the *DescendantPos*. An exclusive pointing error means that an XPath expression points to a node or a set of nodes that does not include the target node at all.

If we consider only exact pointing, this example is not so serious as simply a case of incorrect pointing. However, when we use XPath expressions for external annotations, we need to further think about the actual influences of such incorrect pointing with regard to the behavior of the runtime engines such as XSLT processors for XSLT stylesheets. Therefore, it is also important to investigate the robustness of XPath expressions taking account of application scenarios.

6. CONCLUDING REMARKS

In this paper, we presented variations in annotation tools, and explained the two types of tools that generate transformational annotation for Web document clipping. Since the transformational annotations are descriptions of the ways of modifying the document at hand, the semantics of the document adaptation can easily be indicated through annotator's demonstration or editing actions to obtain the desired result of adaptation. Although the example-based annotation tool is the most sophisticated approach to creating transformational annotation, it may suffer from difficulties in inferring the annotator's intension behind the editing process. On the other hand, the selection-based annotation is a simpler approach, and limited in the capabilities of annotation generation. However, it is noteworthy that the simplicity was a real advantage for use in the development of clipping portlet, because it was not practical to create sophisticated clipping annotations for several thousands of pages to be aggregated into a supplier portal.

Finally, the empirical study presented in this paper is tightly bound to the specific HTML pages we investigated, and we know that our sample sizes are too small to have statistical validity. Therefore, we are not claiming that the results can easily be generalized to all the other kinds of HTML documents. However, this empirical study is an important step towards establishing hypotheses regarding phenomenon that may hinder the practical use of external annotations or metadata that exploit XPath expressions.

7. REFERENCES

[1] Abe, M. and Hori, M.: A visual approach to authoring XPath expressions. *Proceedings of Extreme Markup Languages 2001*, pp. 1–14 Montréal, Canada (2001).

[2] Asakawa, C. and Takagi, H.: Transcoding system for non-visual Web access (2): annotation-based transcoding. *Sixteenth International Conference on Technologies and Persons with Disabilities (CSUN2001)* (2001).

[3] Brush, A. J., Bargeron, D., Gupta, A., and Cadiz, J. J.: Robust annotation positioning in digital documents. *Proceedings of the 2001 ACM Conference on Human Factors in Computing Systems (CHI 2001)*, pp. 285–292, Seattle, Washington (2001).

[4] Cadiz, J. J., Gupta, A., and Grudin, J.: Using Web annotations for asynchronous collaboration around documents. *Proceedings of ACM 2000 Conference on Computer Supported Cooperative Work (CSCW 2000)*, pp. 309–318, Philadelphia, PA (2000).

[5] Denoue, L. and Vignollet, L.: An annotation tool for Web browsers and its applications to information retrieval. *Proceedings of the 6th Conference on Content-Based Multimedia Information Access (RIAO 2000)*, Paris, France (2000).

[6] DeWitt, S. : Basic Web Clipping Using WebSphere Portal Version 4.1. *IBM WebSphere Developer Domain*, http://www7b.software.ibm.com/wsdd/library/techarticles/0206_dewitt/dewitt.html (2002).

[7] Dublin Core Metadata Element Set, Version 1.1: Reference Description. *Dublin Core Metadata Initiative, Recommendation*, http://dublincore.org/documents/dces/ (1999).

[8] Device Independence Principles. *W3C Working Draft*, http://www.w3.org/TR/di-princ/ (2001).

[9] Document Object Model (DOM) Level 1 Specification Version 1.0. *W3C Recommendation*, http://www.w3.org/TR/REC-DOM-Level-1/ (1998).

[10] Erdmann, M., Maedche, A., Schnurr, H.-P., and Staab, S.: From manual to semi-automatic semantic annotation: about ontology-based text annotation tools. *Proceedings of the COLING 2000 Workshop on Semantic Annotation and Intelligent Content*, Luxembourg (2000).

[11] Handschuh, S. and Staab, S.: Authoring and annotation of Web pages in CREAM. *Proceedings of the 11th International World Wide Web Conference*, pp. 462–473, Honolulu, Hawaii (2002).

[12] Heflin, J. and Hendler, J.: Semantic interoperability on the Web. *Proceedings of Extreme Markup Languages 2000*, pp. 111–120 (2000).

[13] Hori, M., Kondo, G., Ono, K., Hirose, S., and Singhal, S.: Annotation-based Web content transcoding. *Proceedings of the 9th International World Wide Web Conference*, pp. 197–211, Amsterdam, Netherlands (2000).

[14] Hori, M., Ono, K., Koyanagi, T., and Abe, M.: Annotation by transformation for the automatic generation of content customization metadata. In F. Mattern and M. Naghshineh (Eds.) *Pervasive Computing, First International Conference, Pervasive 2002*, Lecture Notes in Computer Science 2414, pp. 267–281, Zurich, Switzerland (2002).

[15] Hori, M.: Semantic annotation for Web content adaptation. In D. Fensel, J. Hendler, H. Lieberman, and W. Whalster (Eds), *Spinning the Semantic Web*, pp. 542–573, MIT Press, Boston, MA (2002).

[16] Hori, M., Abe, M. and Ono, K.: Extensible framework of authoring tools for Web document annotation. *Proceedings of International Workshop on Semantic Web Foundations and Application Technologies (SWFAT)*, pp. 1-8, Nara, Japan (2003).

[17] Kahan, J. and Koivunen, M.-R.: Annotea: an open RDF infrastructure for shared Web annotations. *Proceedings of the 10th International World Wide Web Conference*, pp. 623–632, Hong Kong (2001).

[18] Koyanagi, T., Ono, K., and Hori, M.: Demonstrational Interface for XSLT Stylesheet Generation. *Markup Languages: Theory & Practice*, **2**(2): 133–152 (2001).

[19] Lassila, O.: Web metadata: a matter of semantics. *IEEE Internet Computing*, **2**(4): 30–37 (1998).

[20] Lieberman, H. (Ed.): *Your Wish is My Command: Programming by example*. Morgan Kaufmann Publishers, San Francisco (2001).

[21] Lindholm, T.: A 3-way merging algorithm for synchronizing ordered trees – The 3DM merging and differencing tool for XML. *Master Thesis, Department of Computer Science, Helsinki University of Technology* (2001).

[22] Mea, V. D., Beltrami, C. A., Roberto, V., and Brunato, D.: HTML generation and semantic markup for telepathology. *Proceedings of the 5th International World Wide Web Conference*, pp. 1085–1094, Paris, France (1996).

[23] Nagao, K., Shirai, Y., and Kevin, S.: Semantic annotation and transcoding: making Web content more accessible. *IEEE Multimedia*, **8**(2): 69–81 (2001).

[24] Phelps, T. A. and Wilensky, R.: Robust intra-document locations. *Proceedings of the 9th International World Wide Web Conference*, pp. 105–118, Amsterdam, Netherlands (2000).

[25] Rousseau, J. F., Macias, A. G., de Lima, J. V., and Duda, A.: User adaptable multimedia presentations for the World Wide Web. *Proceedings of the 8th International World Wide Web Conference*, pp. 195–212, Toronto, Canada (1999).

[26] Spinks, R., Topol, B., Seekamp, C., and Ims, S.: Document clipping with annotation. *IBM developerWorks*, http://www.ibm.com/developerworks/ibm/library/ibm-clip/
(2001).

[27] XML Path Language (XPath) Version 1.0. *W3C Recommendation*, http://www.w3.org/TR/xpath (1999).

[28] XSL Transformations (XSLT) Version 1.0. *W3C Recommendation*, http://www.w3.org/TR/xslt (1999).