Transclusions in an HTML-Based Environment†

Josef Kolbitsch[1] and Hermann Maurer[2]

[1] Graz University of Technology, Austria
[2] Institute for Information Systems and Computer Media, Graz University of Technology, Austria

Abstract

Transclusions are an advanced technique for the inclusion of existing content into new documents without the need to duplicate it. Although originally described in the early 1960s, transclusions have still not been made available to users and authors on the world wide web.

This paper describes the prototype implementation of a system that allows users to write articles that may contain transclusions. The system offers a simple web-based interface where users can compose new articles. With a simple button the user has the ability to insert a transclusion from any HTML page available on the world wide web.

While other approaches introduce new markups for the HTML specification, make use of technologies such as XML and XLink or employ authoring systems that internally support transclusions and can generate web pages as output, this implementation solely relies on the techniques provided by an HTML-based environment. Therefore HTML, Javascript, the Document Object Model, CGI scripts and HTTP are the core technologies utilised in the prototype.

Keywords: Hypertext, Transclusions, Xanalogical Structure, Authoring Systems, Publishing Systems, Web-Based Applications.

1 Introduction

In 1965, Ted Nelson presented *"a file structure for the complex, the changing and the indeterminate",* in which he introduced the term hypertext (see [20]). One of the fundamental concepts in Nelson's notion of hypertext is a technique called *transclusions.* Transclusions allow authors to include portions of existing documents into their own articles without duplicating them. Basically, a transclusion in document A is a reference to a portion of the content of a potentially remote document B that is virtually included into document A (see Figure 1).

1.1 Background of Transclusions

Transclusions are designed as complete replacement for all *cut-and-paste* mechanisms in use. Nelson argues that cut-and-paste is not what people actually want to do but that it is a restriction imposed upon authors by the nature of paper. Writers actually do not want to make a copy of an existing document, cut out the piece they want to reuse and paste it in their document. They want to include the original content and let readers know what the source and the context of the quote is (e.g., [21]).

Reference lists at the end of a scientific publication, for instance, are usually not what is intended by writers and desired by readers. They are rather a pragmatic solution to the problem that both the source and the context of the quotation are lost by copying-and-pasting a portion of content printed on paper.

What used to be physical restrictions of paper was embraced by most computing systems in an attempt to resemble the work environments and common processes in offices (cf., [33]). Therefore most current graphical operating systems make use of metaphors such as a desktop, folders and documents; a document has to be put in exactly one folder; there is a clipboard, and content from a different document is included using copy-and-paste mechanisms (see [23]).

† This paper was supported by the Styria Professorship for Revolutionary Media Technologies.

1.2 Implications of Transclusions

Transclusions are, however, not only a mere replacement for copy-and-paste. They assure that the original context of a quotation is preserved and can provide a visible link to the source of the transclusion. Ted Nelson's approach to realising this functionality is based on *transpointing windows* (e.g., [22]).

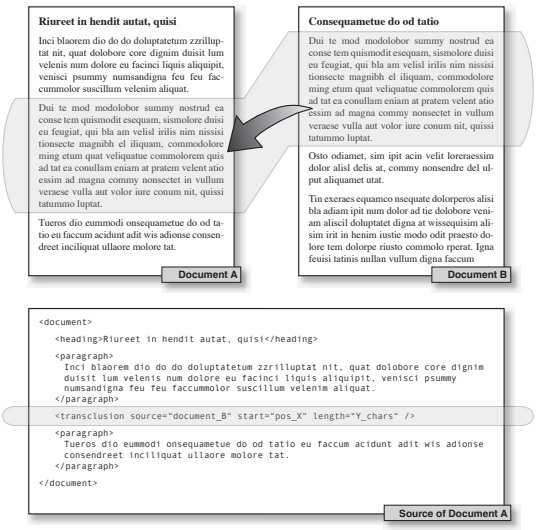

Figure 1. Exemplary transclusion. Part of document B (top right) is transcluded into document A (top left). Bottom: the source code of document A does not contain the actual text of document 2 but only the data required to retrieve it from the original document.

Moreover, authors of documents can be notified when their articles are transcluded. Thus they can, for instance, find out about other researchers in the same area. Authors using transclusions, on the other hand, can be informed automatically about modifications in source documents (see [14]).

Apart from obvious improvements in authoring and publishing systems, transclusions can also offer a solution to copyright issues experienced today on the world wide web: authors include content into their documents by means of transclusions. Whenever a reader views a transclusion a note about the rights associated with the transcluded content is added, and a micropayment is made to the corresponding owner ([25]). Nelson names this model *transcopyright* (see [24]).

1.3 Transclusions and HTML

The Hypertext Markup Language (HTML, [7]) is a relatively simple language for describing platform independent hypertext pages. In the early stages of its development, the focus of HTML has been on style and graphical presentation rather than on functionality and underlying paradigms. Therefore many innovative ideas such as bidirectional hyperlinks and issues already known at that time including broken links were not considered in the implementation (e.g., [27]).

In principle, transclusions are used in HTML. Designated markups including `<img>`, `<object>` and `<embed>` incorporate content such as images, Java applets and animations into HTML pages by means of linking. Thus, these elements basically make use of the concept of transclusions.

Transclusions in HTML are very limited, though. Only certain media such as images can be virtually included, whereas textual content, in general, cannot be transcluded. Moreover, the transclusion mechanisms available in HTML can only be applied to entire documents. Fine-grained transclusions such as a small spatial selection of an image are not implemented.

2 Attempts to Implement Transclusions

Although the idea of transclusions has been proposed some forty years ago, only a few attempts to implement this advanced technique have been made. The following sections give an overview of several notable approaches to the realisation of transclusions.

2.1 Xanadu

Transclusions are an integral part of Xanadu, Ted Nelson's original hypertext system (e.g., [21]). Their implementation relies on a document model, though, that is radically different from what is widely used today. In Xanadu, documents *(versions)* do not contain content but references to the actual content. Content is both stored and referenced with the highest granularity possible—on the level of single characters. All content is retained in content repositories.

Any document is made up of a list of references to content stored in the system, e.g., a document consists of "characters 124 to 729 and 1276 to 1301 from the repository". When content from document B is transcluded into document A, the corresponding references to the actual content in the repositories are added to the reference list of document A.

Thus, the creation and retrieval of transclusions in Xanadu are trivial list operations. Ted Nelson also details a number of functions related to transclusions and the handling of situations in which documents are modified or large portions of documents are deleted (e.g., [25]). Basically, these functions can be seen as more complex list operations.

2.2 Proposal to Amend the HTML Specification

Since the idea of transclusions is already present in HTML for media such as images and multimedia animations, a sen-

sible approach to text-based transclusions is to introduce a new tag that allows users to transclude text. Therefore [28] suggests an amendment to the HTML specification. A new markup, `<text>`, is proposed with the intention to offer an element of the same significance as `<img>` or `<embed>`.

The main attributes of the markup are the URI of the source document and the start position and length of the text to be transcluded. The web browser analyses the tag, loads the source document of the transclusion, extracts the portion of text given by the attributes of the `<text>` tag, and inserts it into the document. Thus, a transclusion is handled in a similar way as an inline image.

Although the proposal seems rational it has not been accepted, and no web browser to date has the feature implemented.

2.3 Transclusions with IFrames and Embedded Objects

The recommendation for HTML 4 includes markups for inline frames and embedded objects (e.g., [7]). Both inline frames and embedded objects define areas within a given HTML document that can be used to display potentially remote resources. Inline frames can merely contain HTML pages and images, whereas objects may contain resources of arbitrary type.

Transcluding document A into document B can, for instance, be achieved by inserting an `<object>` tag with a reference to document A into document B ([16]). The capabilities of this technique are rather limited, though. Only entire documents can be referenced. Moreover, the context is lost because a link from the document containing the embedded object to the source of the transclusion is not provided by these markups. Therefore this approach is not well suited for realising transclusions.

2.4 XML-Based Transclusions

The Extensible Markup Language, XML, is a flexible language for describing documents that contain structured information (see [31]). In contrast to other markup languages such as HTML, where both syntax and semantics are determined, neither a set of tags nor the semantics are defined in XML. Therefore, XML per se does not contain a distinct markup for links; a separate linking language is used instead.

XLink, the XML Linking Language (e.g., [30]), provides a framework for describing the syntax and semantics of even complex linking structures between resources. An XLink link typically contains a number of attributes that describe, for instance, what resource is to be loaded and when it is to be displayed.

Three attributes are essential for the implementation of transclusions using XLink:

- `href`: the document to be loaded. Set to the source document of the transclusion;
- `actuate`: when the resource is to be loaded. When set to `onLoad`, the resource is loaded when the document containing the XLink link is loaded;
- `show`: in which manner the resource is to be displayed. When set to `embed`, the resource is displayed practically instead of the XLink tag.

The skeleton of the XLink link shown in listing 1 transcludes the entire document `source.xml` into the document containing the link at the position of the link. A similar approach to transclusions in XML is described by [29].

```
<my:transclusion
   xmlns:my="http://www.kolbitsch.org/"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xlink:type="simple"
   xlink:href="source.xml"
   xlink:actuate="onLoad"
   xlink:show="embed" />
```

Listing 1: Fragmentary transclusion with XLink.

With the XML Pointer Language (XPointer, [32]) fragments of XML documents can be identified and addressed as well. Thus, a combination of XML, XLink and XPointer can be employed to make the use of fine-grained transclusions in XML-based environments possible (see [16; 15]).

2.5 Recent Projects Involving Transclusions

Currently, several mostly academic projects that experiment with transclusions exist. The following paragraphs introduce three selected systems.

The University of Nottingham, UK, has proposed a technology-based learning environment that adapts to its users (see [19]). The information retained in the system is organised in small "chunks" that are stored as XML files.

Since the system is adaptive, lessons are not static but assembled dynamically on the basis of a lesson plan. When a user requests a particular lesson, appropriate chunks of information are retrieved and included into a virtual document by means of transclusion. Thus, the system facilitates the reuse of small pieces of information for a number of lessons or for students with various differing standards of knowledge.

At the University of Bologna, Italy, researchers attempt to combine existing software products such as the Internet Explorer and Microsoft Word to offer a collaborative editing environment for the world wide web (see [3]). The implemented tool, XanaWord, allows users to edit any web page

they view in their web browser—even if they do not have write permissions for the resource.

Any page displayed in Internet Explorer can be opened with a word processor such as Microsoft Word, where the user can make arbitrary changes to the document. When the user saves the page, only the changes to the original document are stored in the XanaWord repository. Whenever a document is retrieved from the repository, the modifications made by the user and the content from the original resource are included in a dynamically generated document by means of transclusion. Finally, the dynamic document is sent to the user's web browser.

The Institute for Information Systems and Computer Media in Graz, Austria, proposed an environment capable of handling transclusions in various output document formats (see [14]). The system includes three components:

- the Latex typesetting system that allows users to create documents and save them in a number of document formats including Postscript, PDF and HTML;
- an extension to Latex that allows users to create transclusions; and
- a Hyperwave Information Server (see [9]) that handles issues such as linking and versioning.

In the proposed environment users can insert a special markup that designates a transclusion in Latex documents. Then, the user has to upload the file to a Hyperwave Information Server that extracts links and saves them in a link database, etc. When the document is requested by a user, the transclusions and links are inserted into the file saved on the server. The resulting intermediate file is processed by Latex in order to generate the requested document format. Ultimately, the document containing the transclusion is sent to the client.

3 Implementation

In contrast to several approaches to transclusions illustrated above, this project does not present a proposal but an actual implementation of a system that lets users take advantage of transclusions. It is designed as part of a larger system that offers communities instruments to work actively with content from digital libraries and electronic encyclopaedias (see [12]). Thus, a prototype is implemented offering a tool for authoring new articles that can contain transclusions. It is available online at [11].

3.1 Design Goals and Requirements

The environment for creating and retrieving transclusions aims at facilitating the reuse of information readily available on the Web—even by novice users. Therefore a number of design goals have to be taken into consideration:

- ease of use: the tool for making transclusions must be as easy to use as traditional copy-and-paste mechanisms;
- use of any document on the web: not only documents from a closed repository but basically any web page may be the source of a transclusion;
- level of granularity: any portion of text may be transcluded from a document, from a single character to the entire content of a page.

Browser plug-ins or special software tools should not be required. Therefore, this implementation of transclusions solely relies on technologies available and widely utilised on the world wide web:

- HTML: transclusions can be made from any HTML formatted document available on the web. Moreover, documents containing transclusions are presented to the reader as traditional HTML documents (see [7]);
- Javascript, DOM: internally, most current web browsers represent HTML pages as trees of objects. The underlying technology is the Document Object Model (DOM, [4; 5]). Javascript is used to access individual objects in the DOM tree of the HTML page to be transcluded (see [26; 6]) and enables fine-grained transclusions;
- HTTP: documents containing transclusions are transmitted to the readers using the Hypertext Transport Protocol (see [8]).

3.2 System Overview

Since the system for creating and retrieving transclusions consists of a number of components, a brief overview is given. The following description is made in the order of actions taken by a user in authoring and reading a document including transclusions.

Two fundamental actions can be distinguished in the system: the creation of a transclusion when the article is authored, and its evaluation when the page containing the transclusion is to be displayed. When the user wants to create an article with a transclusion the web browser presents a frameset with two frames. One frame contains a conventional area for authoring HTML content and an additional button for adding a transclusion. When the user presses this button, the URL of a page can be entered, and the corresponding page is loaded through an HTTP proxy application into the second frame. The user can either transclude content from this page or can use the second frame to browse to a different page—again through the HTTP proxy application. The document to be transcluded is complemented with a button that inserts the transclusion into the text area of the first frame, when pressed. An illustration of the two frames is given in figure 5.

The user selects the portion of text to be transcluded in the second frame and presses the button to have the transclusion actually inserted into the article. The button calls a Javascript function that determines the start and end positions of the selection made. Together with the URL of the page in the second frame these values are used to generate an intermediate markup that is inserted into the article (see section 3.4).

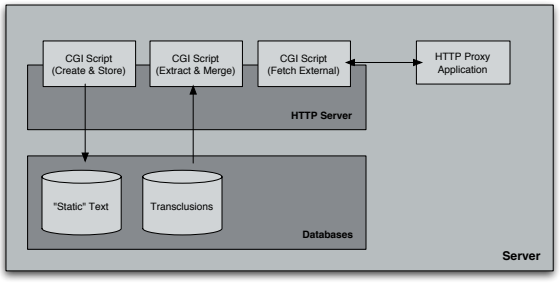

Figure 2. Overview of the server-side components.

Once a user has finished authoring an article and chooses to save the new document, the contents of the text area including the intermediate transclusion markup are sent to a CGI script on the server. The server stores the "static" text and the values provided through the transclusion tag to a database. In addition to this, metadata on the source of the transclusion is collected and stored in the database.

Whenever the article containing the transclusion is requested, a second CGI script is invoked. The script retrieves the contents of the article and the parameters of the transclusion tag from the database. The parameters defining the transclusion are used to load the original page from its original location. If it is unchanged, the transcluded portion of text is extracted from the original page, combined with the static text and sent to the web browser of the client (see Figure 5).

The next section gives a brief introduction to the architecture of the implementation and its components.

3.3 System Architecture

Our implementation of transclusions follows a classic client-server paradigm. A conventional HTTP server, a relational database, several server-side CGI programs, a non-transparent HTTP proxy application and client-side Javascript code are the main components of the system.

The CGI script "Create and Store" in figure 2 receives the data submitted by users. It analyses the content of the article, extracts transclusions and stores both content and transclusions in the internal database of the system (see section 3.4). The "Extract and Merge" script, on the other hand, reads the content of an article together with the information on the transclusion from the database, fetches the source document of the transclusion, assembles the complete article and sends it to the user (see section 3.5).

The third CGI script, "Fetch External", is utilised during the authoring process for loading the page to be transcluded. This programme is basically necessary to insert a button and a small portion of Javascript code into the corresponding page (see section 4.1). It relies on a specialised, non-transparent proxy application developed for this project.

In the current prototype implementation the relational database consists of only two tables. While one table contains the static content of the article, the other one stores detailed information on the transclusion as well as a rich set of metadata and a fingerprint of the source document.

3.4 Creating a Transclusion

As explained above, the interface for authoring new articles consists of a frameset with a frame for writing an article in an HTML form and a separate frame for displaying the content to be transcluded (Figure 5). When users wish to insert a transclusion they select the portion of text with the pointer device and click on the button provided in the window.

The button calls a Javascript function which is essentially the only operation carried out on the client computer. It accesses the document object model to determine the exact start and end positions of the selection made by the user

```
<transclusion src="{url}"
   atag="{tag}" aindex="{int}" aoffset="{int}"
   ftag="{tag}" findex="{int}" foffset="{int}" />


<transclusion src="http://www.kolbitsch.org/about/"
   atag="H1" aindex="1" aoffset="0"
   ftag="P" findex="4" foffset="29" />
```

Listing 2. Syntax of a intermediate transclusion tag (top) and an example (bottom).

and generates an intermediate tag that is inserted into the article. The Javascript interface to selections provided by most browsers is somewhat peculiar in that it determines these start and end positions in the way the user actually marked the text. I.e., the *anchor* of the selection is the position where the user clicked to indicate the beginning of the selection. Then, the user drags the mouse, for instance, to the end of the selection and releases the mouse button to denote the end of the selection. The end position is the *focus*. In the following paragraphs, anchor and focus are denoted by prefixes "a" and "f".

The syntax of the newly introduced `<transclusion>` markup with its seven parameters is rather complex (see listing 2). This level of detail is required to be able to determine the exact start and end positions of transclusions, though. Values in curly braces describe the type of attribute values:

- `src`: the URL of the document to be transcluded;

- `atag`, `ftag`: the names of the tags in which the transclusion starts and ends, e.g., "P" for a paragraph;
- `aindex`, `findex`: the index of the tags in which the transclusion starts and ends, e.g., the seventh paragraph in the document;
- `aoffset`, `foffset`: the offset within the start and end tags, e.g., the transclusion starts at the second character of the seventh paragraph in the document.

The exemplary tag shown in listing 2, for instance, describes a transclusion that starts at the first character of the second H1 heading and ends at the 30th character of the fifth paragraph in the given document.

When the article containing the transclusion is saved by the user, the data is sent to the server, and the "Create and Store" CGI program is invoked (see Figure 3). It extracts the transclusion from the article, determines the attributes of the transclusion and writes the information to the database. The `<transclusion>` markup in the original article is replaced with a transclusion object that refers to the transclusion stored in the database.

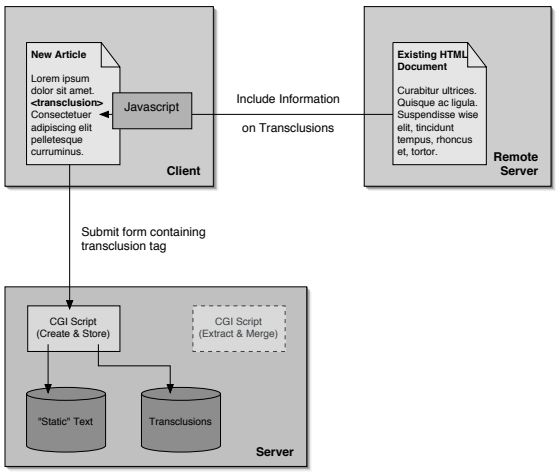

Figure 3. Simplified schematic illustration of the process of creating a new article containing a transclusion.

The source document of the transclusion is not stored in the internal repository. However, its URL, the creation and modification dates as well as an MD5 hash value of the entire page content are retained as fingerprint. These values are necessary to determine if the source document has changed when the transclusion is retrieved.

It should be noted that, in contrast to [28], where an amendment to the HTML specification is suggested, the `<transclusion>` markup in this implementation is only used during the authoring process. It is inserted when the user makes a transclusion, is evaluated by the system and replaced by a transclusion object. When the page containing the transclusion is to be displayed, the transclusion object is replaced with the corresponding content from the original page (see below). Hence, the `<transclusion>` tag is only visible within the system but not externally to the user.

3.5 Retrieving a Transclusion

The "logic" of our implementation mainly lies within the component that retrieves transclusions. Whenever an article is requested, its body is analysed for the presence of transclusion objects. For each transclusion object the following steps have to be carried out (see also Figure 4):

- resolve the object and retrieve the information on both the transclusion and on the source document from the database;
- check if the given URL of the source document can be loaded;
- if it can be retrieved check if the metadata, i.e., the creation and modification dates as well as the MD5 hash values, have changed;
- if the fingerprint of the source document is valid, retrieve the resource and extract the portion of text determined by the start and end positions of the transclusion;
- replace the transclusion object in the article with the transcluded content;
- if any of the operations above fails, insert an apologetic error message.

Every transclusion is formatted in a way that readers can distinguish between authentic and transcluded content. In figure 5, transcluded text is highlighted using a light-gray background. Transclusions are complemented with a hyperlink to the original source of the content.

4 Issues Encountered

During the implementation and evaluation of the prototype a number of difficulties were experienced. A few substantial issues are addressed in the following sections.

4.1 Javascript Restrictions

As described in section 3.4, our implementation relies on Javascript code that detects which portions of a document are selected by the user; when the user presses a button, the start and end positions of the selection are determined.

Restrictions imposed by the security mechanisms of most modern web browsers (e.g., [18]) prevent Javascript functions from accessing selections in "foreign" frames and documents. This means that the button that reads the user's selection has to be present in the same frame as the selection.

Since our premise was that transclusions can be made from any HTML document on the Web, we have to make sure that the Javascript code required is inserted in any page the user wants to transclude. The approach in the current implementation is to use of a non-transparent proxy application. So when users enter the URL of the page they wish to transclude, the page is not loaded directly by the web browser but by a CGI script on the server that acts an HTTP proxy. The CGI script appends the demanded Javascript code and sends the document to the client.

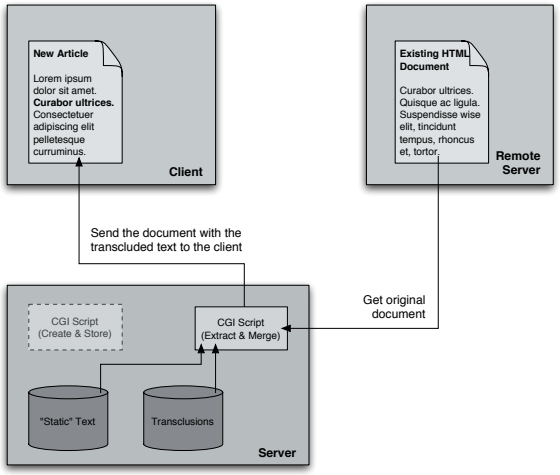

Figure 4. Handling a request for a page containing a transclusion (simplification).

The proxy application could be omitted if transclusions were only made in documents from an internal repository such as an online journal or a content management system. The system generating the documents could automatically insert the essential Javascript code when the resource is requested, for example, with a particular parameter.

4.2 Browser Specific Implementation

The function for accessing the user's selection poses yet another problem. Different implementations of the corresponding function exist in the various web browsers available today. The Mozilla family accesses the selection through the `document.getSelection()` method, whereas Internet Explorer, for instance, uses a dedicated `document.selection` object (e.g., [10]).

Due to the use of the `document.getSelection()` method in our implementation, the prototype is only compatible with Mozilla-based browsers. With minor modifications in the client-side Javascript code, however, the prototype should work with a wide range of web browsers including Internet Explorer.

4.3 Modified Documents and Unavailable Resources

Similar to broken links in web pages, documents that are modified and resources that become unavailable can pose a problem for transclusions. One reason for this deficiency is the use of Uniform Resource Locators on the world wide web (URLs, [2]).

URLs identify an object *and* describe its physical location. Defining the physical location of a document determines that only one instance of the document may exist at a time. Different resource identification and allocation mechanisms allow for multiple locations of the same document, i.e., several instances of the same document may exist in different physical locations. When a resource with a certain object identifier is requested, it is retrieved from one of the locations that retain a copy (e.g., [27]). This can, for instance, be the location with the fastest network connection, the one with the lowest load, or the one with the shortest distance.

The design of Xanadu takes a similar approach, in which a resource may exist in several locations (e.g., [25]). Thus, when a transclusion is requested and one instance of the source data becomes unavailable, it is retrieved from another repository containing the same information.

We propose an analogue mechanism that makes use of the Wayback Machine (e.g., [13]), a very large archive of currently about forty billion web pages, and local caching. When a transclusion is requested whose source document has undergone major changes or has become unavailable, the Wayback machine is queried for the resource. The query includes the URL and the creation date of the transclusion as access date.

Alternatively, a local cache or Google Cache can be employed. Local caching means that a copy of a resource has to be made when it is transcluded; the local copy is retained in an internal repository of the system. In case of local caches, however, legal issues may arise. [1], for instance, discusses whether services such as Google Cache are in conflict with German copyright laws.

Figure 6 illustrates the suggested retrieval strategy for transclusions: when the original source of a transclusion is available and has not been changed, it is retrieved from the original location. Otherwise an attempt is made to load the page from the Wayback Machine or from a similar cache. If this attempt fails as well, the user is notified that the transclusion cannot be made at this time.

5 Discussion

The implementation of transclusions in a purely HTML-based environment has shown interesting perspectives, and various aspects need to be investigated in detail and require

further research. A few selected topics are pointed out in the following sections.

5.1 Robustness

The prototype we presented offers ease of use and relative overall stability. The robustness, however, can still be improved. Under certain conditions, for example, transclusions can be imprecise. Content transcluded from a document by the "Extract and Merge" component can be slightly different from what a user originally selected—a few characters too many or too little are extracted.

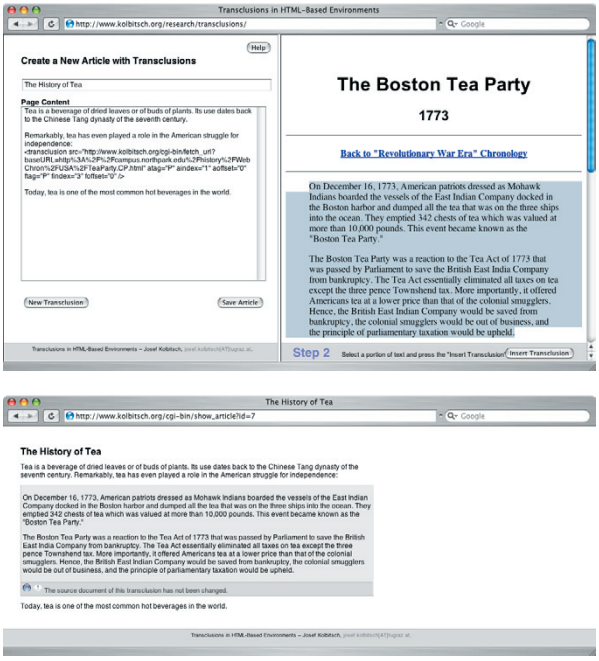

Figure 5. Screenshots from a prototype of transclusions in an HTML-based authoring environment.

An issue that generally affects the robustness of our implementation and demands in-depth analysis is modified content. The shortcoming partly arises from an optimization that improves the system performance. When modifications in the source document of a transclusion are to be detected, only the creation and modification dates as well as the content length in the HTTP header of the resource are scanned. Some servers do not return these values at all, though, and a small percentage of hosts return invalid date values. So if creation and modification dates or the content length are not available, the entire resource is retrieved and an MD5 hash value is generated. When the content to be retrieved is very large, the system load is high or the network connection is slow, it might take too long to calculate the hash value. In this case, the process might terminate with a time-out signal, and the transclusion cannot be made.

As pointed out above, modifications in transclusion sources are a general problem. Especially dynamic content such as pages from a content management system or from a digital library can be critical. In many cases, these documents contain advertisements or other frequently changing information such as references to the most recent articles. Although the actual content of the document is not altered, the system component that analyses the state of transclusion sources would detect a modification.

It is desirable to have modifications in documents and their importance—was only an advertisement changed or has the meaning of the article changed?—detected automatically. However, this functionality is presently computationally not feasible. [14] suggests leaving the decision to the user: despite the modifications the content from the altered source document is transcluded, and the user has to determine if the transclusion and the context are still appropriate.

In any case, authors of modified transclusion sources should be notified that the content they virtually included into their articles might not be suitable anymore, and that it has to be reviewed.

5.2 Aspects of the Design

The design of the implementation, the use of a transclusion object in particular, open up exciting opportunities. Since the transclusion object is associated directly with the source document of a transclusion, it is possible to determine which other articles in the system include content from the same source. This information indicates that the corresponding articles might deal with a similar topic and that they could be of interest for both authors and readers. More importantly, this information denotes that the authors of these articles might work in a similar area. In a scientific setting, for instance, these authors can be researchers working on similar projects. Thus, information exchange can be enabled. From a more general perspective, collaboration can be fostered and organisational knowledge management can be facilitated (e.g., [17]).

Therefore we propose a simple function like *"Which other articles transclude the same document?"* or *"Who else uses the same document?"* that can help readers and writers discover new information.

This principle can be applied in the "opposite direction" as well. Authors can easily find out which other articles in the system transclude the articles they produced. This information can basically be used for the same purposes as pointed out above. Hence, we propose another function that complements every article in the system: *"Which articles in the system transclude this article"* or simply put, *"Who transcludes 'us'?"*

In a more sophisticated approach, the system could proactively point out resources and authors that are related to the article being displayed.

5.3 Aspects of the Proxy Application

Our implementation relies on a non-transparent HTTP proxy application that makes it possible to insert a small portion of Javascript code into every page the user wishes to transclude. Although the application was initially intended for a very specific purpose, its design is so flexible that it a whole range of other, largely unrelated applications become feasible.

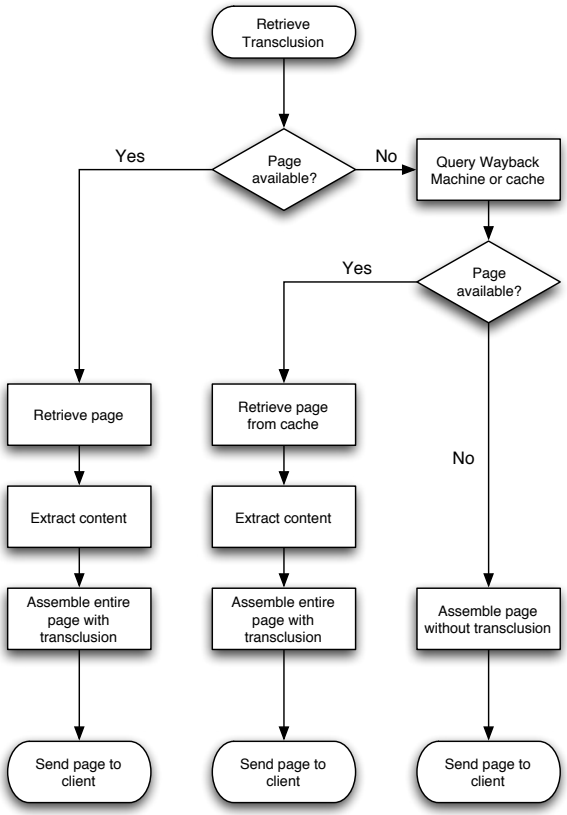

Figure 6: Flow chart of a document retrieval strategy where the source document of a transclusion can become unavailable or can be modified.

Blacklisting of words and hyperlinks, highlighting of text and dynamic insertion of annotations are just a few simple examples. Advanced techniques may include dynamic adaptation of content, on-the-fly insertion of complementing information, etc.

These novel ideas need to be explored in detail. A comprehensive analysis and an evaluation of early results will be presented in a forthcoming publication.

6 Conclusion

This paper briefly outlined Ted Nelson's notion of hypertext and one of its prime concepts—transclusions. Although HTML has been influenced by the notion of transclusions for the inclusion of external objects such as images, they have not been implemented consistently. Therefore a number of proposals have been made on how to implement transclusions with the technologies available today. A few of the most important approaches have been discussed.

Based merely on the technologies provided by a web-based environment, we have designed a system that offers users to author articles that may contain transclusions. A first prototype utilises plain HTML, Javascript and server-side components including CGI scripts and a specialised HTTP proxy application.

Although a number of issues were encountered during the implementation phase, we have been capable of collecting valuable results that make us confident that we can enhance the current prototype and increase its robustness and stability.

The innovative design of the transclusion structures as well as the architecture of system components open up new perspectives and can lead to more advanced functionality. Facilitating information discovery, pro-active dissemination of related content and the stimulation of community-building are only a few possibilities among others.

With the infrastructure provided by the proxy application elaborate functionality such as automatic highlighting of information, dynamic content adaptation and on-the-fly insertion of related data can be introduced. This concept can prove to be beneficial in numerous environments including electronic encyclopaedias and digital libraries.

These ideas will be discussed thoroughly in a future paper.

Acknowledgements

The first author would like to thank Edmund Haselwanter for his valuable input on the Wayback Machine.

References

[1] M. BAHR The Wayback Machine und Google Cache – eine Verletzung deutschen Urheberrechts?, http://www.jurpc.de/aufsatz/20020029.htm, (2002) Accessed May 11th, 2005.

[2] T. BERNERS-LEE Universal Resource Identifiers in WWW. A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as Used in the World-Wide-Web, *Request for Comments 1630* (1994). See also http://www.w3.org/Addressing/rfc1630.txt.

[3] A. DI IORIO and F. VITALI A Xanalogical Collaborative Editing Environment, *Proceedings of the Second International Workshop of Web Document Analysis (WDA2003),* (2003) Edinburgh, UK, August 2003. See also http://

www.csc.liv.ac.uk/~wda2003/Papers/Section_III/Paper_11.pdf.

[4] L. WOOD et al. Document Object Model (DOM) Level 1 Specification Version 1.0, http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/, (1998) Accessed March 29th, 2005.

[5] P. LE HÉGARET et al. W3C Document Object Model (DOM), http://www.w3.org/DOM/, (2005) Accessed March 29th, 2005.

[6] ECMA ECMAScript Language Specification, 3rd Edition, *Standard ECMA-262* (1999). See also http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf.

[7] D. RAGGETT et al. HTML 4.01 Specification, http://www.w3.org/TR/html4/, (1999) Accessed April 28th, 2005.

[8] R. FIELDING et al. Hypertext Transfer Protocol – HTTP/1.1, *Request for Comments 2616* (1999). See also ftp://ftp.isi.edu/in-notes/rfc2616.txt.

[9] Hyperwave, http://www.hyperwave.com/.

[10] P.-P. KOCH, JavaScript – Get selection, http://www.quirksmode.org/js/selected.html, (2004) Accessed May 10th, 2005.

[11] Transclusions in HTML-Based Environments, http://www.kolbitsch.org/research/transclusions/.

[12] J. KOLBITSCH and H. MAURER Community Building around Encyclopaedic Knowledge, (2005) to be published.

[13] R. KOMAN How the Wayback Machine Works, http://webservices.xml.com/lpt/a/ws/2002/01/18/brewster.html, (2002) Accessed May 10th, 2005.

[14] H. KROTTMAIER Transcluded Documents: Advantages of Reusing Document Fragments, *Proceedings of the 6th International ICCC/IFIP Conference on Electronic Publishing (ELPUB2002),* (2002) Karlovy Vary, Czech Republic, pp. 359-367. See also http://hkrott.iicm.edu/docs/publications/elpub-2002.pdf.

[15] H. KROTTMAIER and D. HELIC Issues of Transclusions, *Proceedings of the World Conference on E-Learning in Corporate, Government, Healthcare, & Higher Education (E-Learn 2002),* (2002) Montreal, Canada, pp. 1730-1733. See also http://coronet.iicm.edu/denis/pubs/elearn2002b.pdf.

[16] H. KROTTMAIER and H. MAURER Transclusions in the 21st Century, *Journal of Universal Computer Science,* 12 (2001), pp. 1125-1136. See also http://www.jucs.org/jucs_7_12/transclusions_in_the_21st/.

[17] H. MAURER and K. TOCHTERMANN On a New Powerful Model for Knowledge Management and its Applications, *Journal of Universal Computer Science,* 1 (2002), pp. 85-96. See also http://www.jucs.org/jucs_8_1/on_a_new_powerful/.

[18] MICROSOFT CORPORATION About Cross-Frame Scripting and Security, http://msdn.microsoft.com/workshop/author/om/xframe_scripting_security.asp, (2005) Accessed May 10th, 2005.

[19] A. MOORE et al. Personally tailored teaching in WHURLE using conditional translucion, *Proceedings of the Twelfth ACM Conference on Hypertext and Hypermedia,* (2001) Aarhus, Danmark, pp. 163-164.

[20] T. H. NELSON A File Structure for the Complex, the Changing and the Indeterminate, *Proceedings of the ACM 20th National Conference,* (1965) Cleveland, OH, U.S.A., pp. 84-100.

[21] T. H. NELSON *Literary Machines,* Mindful Press, 1981.

[22] T. H. NELSON The Heart of Connection: Hypermedia Unified by Transclusion, *Communications of the ACM,* 8 (1995), pp. 31-33.

[23] T. H. NELSON Generalized Links, Micropayment and Transcopyright, http://www.almaden.ibm.com/almaden/npuc97/1996/tnelson.htm, (1996) Accessed May 1st, 2003.

[24] T. H. NELSON Transcopyright: Pre-Permission for Virtual Republishing, http://www.aus.xanadu.com/xanadu/transcopy.html, (1998) Accessed May 3rd, 2005.

[25] T. H. NELSON Xanalogical Structure, Needed Now More than Ever: Parallel Documents, Deep Links to Content, Deep Versioning and Deep Re-Use, *ACM Computing Surveys,* 4es (1999). See also http://xanadu.com.au/ted/XUsurvey/xuDation.html.

[26] NETSCAPE COMMUNICATIONS JavaScript Developer Central, http://developer.netscape.com/tech/javascript/, (2004) Accessed February 3rd, 2004.

[27] A. PAM Where World Wide Web Went Wrong, *Proceedings of the AUUG'95 & Asia-Pacific World Wide Web '95 Conference & Exhibition,* (1995) Sydney, Australia. See also http://www.csu.edu.au/special/conference/apwww95/papers95/apam/apam.html.

[28] A. PAM Fine-Grained Transclusion in the Hypertext Markup Language, *Internet Draft* (1997). See also http://xanadu.com.au/archive/draft-pam-html-fine-trans-00.txt.

[29] E. WILDE and D. LOWE XML Linking Language. In *XPath, XLink, XPointer, and XML: A Practical Guide to Web Hyperlinking and Transclusion* (E. WILDE and D. LOWE), (2002) pp. 169-198. Addison-Wesley Professional. See also http://searchwebservices.techtarget.com/searchWebServices/downloads/wilde-lowe_07.pdf.

[30] S. DEROSE et al. XML Linking Language (XLink) Version 1.0, http://www.w3.org/TR/xlink/, (2001) Accessed April 28th, 2005.

[31] W3C Extensible Markup Language (XML), http://www.w3.org/XML/, (2003) Accessed April 28th, 2005.

[32] S. DEROSE et al. XML Pointer Language (XPointer), http://www.w3.org/TR/xptr/, (2002) Accessed May 2nd, 2005.

[33] M. YOCOM Mac OS History, http://www.macos.utah.edu/Documentation/MacOSXClasses/macosxone/macintosh.html, (2004) Accessed May 3rd, 2005.

Contact address:

Josef Kolbitsch
Graz University of Technology
Steyrergasse 30
8010 Graz, Austria
e-mail: josef.kolbitsch@tugraz.at

Hermann Maurer
Institute for Information Systems and Computer Media
Graz University of Technology
Inffeldgasse 16c
8010 Graz, Austria
e-mail: hmaurer@iicm.edu

JOSEF KOLBITSCH is a PhD student at Graz University of Technology. His research interests include electronic encyclopaedias, digital libraries and hypermedia.

HERMANN MAURER is professor and dean of the faculty of computer science at Graz University of Technology. He is author of some twenty books and more than 600 contributions in various publications. Recently he has also published “XPERTS”, a series of science fiction novels.