# Annotating Virtual Web Documents with DynamicMarks

Zoltán Fiala, Klaus Meißner

Dresden University of Technology
Heinz-Nixdorf Endowed Chair for Multimedia Technology
D-01062 Dresden, Mommsenstr. 13
[zoltan.fiala, kmeiss]@inf.tu-dresden.de

**Abstract:** Annotating Web pages facilitates document management and collaboration on the WWW, and is also a key factor of user-driven metadata creation on the Semantic Web. However, though there exists a broad number of annotation systems for static HTML, none of them supports the pervasive annotation of dynamically generated Web content. The *DynamicMarks* annotation system introduced in this paper relies on a component-based document format for personalized adaptive Web sites. It enables the fine-granular, ubiquitous annotation of dynamically assembled pages by reversely mapping notes to reusable adaptive content components. Moreover, annotating existing static HTML documents is provided by automatically wrapping them to components. Beside locally stored private remarks, shared annotations are also supported, allowing for pervasive asynchronous communication on the Web.

## 1 Introduction

Creating Web annotations is an important aspect of document management and collaboration on the WWW. Authors and readers mark Web pages in order to remember things better, to communicate or to manage information assets more effectively. Popular scenarios are e-learning applications supporting asynchronous communication between students and tutors, distributed authoring systems allowing concurrent editing, but also online product presentations enabling to send feedback to the system via personal remarks. Within the scope of the Semantic Web, annotation technology gains even more significance. Users' notes attached to Web documents act as additional external metadata that can be used to create individual navigation structures over existing resources, to support search engines with personalized information, to easily locate items that have been marked as relevant by other users etc.

Recently, a number of solutions for annotating static HTML pages (e.g. *ComMentor* [RMW95], *YaWaS* [DV00], *Annotator* [OAM00], *Annotea* [KKP01] etc.) have been developed. For such content an annotation is unambiguously identified by the URL of the affected Web page and some offset parameters describing its correct position [DV00].

However, the intensive change of the WWW to a ubiquitous personalized medium indicates the development of interactive Web sites that are automatically adapted to different client devices and semantic user preferences. A fundamental way to meet this requirement is the dynamic generation of virtual Web pages from separately stored configurable, reusable pieces of content. *A virtual document is a document for which no* *persistent state exists and for which some or all of each instance is generated and inserted at run time* [WS99]. Typical virtual Web documents are individual product offerings, Web-based learning environments with personalized didactical paths, columns of electronic newspapers showing always up-to-date news for various end devices etc. Watters et al. [WS99] mention important research scenarios for virtual documents, such as generation, search, revisiting, versioning, authentication, reference and *annotation*.

Though there is a growing need to create annotations on different presentation views of virtual documents using different devices, no existing annotation system manages this challenge. Firstly, they have been exclusively developed for desktop Web browsers. Secondly, since notes are not attached to the original content pieces, but to the volatile Web pages presenting the final document view, they go lost, if those pages are updated or if the same content is shown in a different context or layout on another Web page for another user. Thus, annotating virtual Web documents necessitates significant changes to today's annotation technology.

Attaching annotations to arbitrary dynamic Web pages is impossible, as they are neither persistent, nor do they contain adequate information on the origin, structure and volatility of the presented data. However, if the content pieces to be inserted are described, stored and composed in a structured way and sufficient information on this structure is provided during page generation, the resulting pages can be annotated, too. The main idea is to *reversely map* annotations to the inserted content.

To present this idea, this paper introduces an approach called *DynamicMarks*. It relies on a component-based XML document format for device-independent adaptive Web applications. Its key concept is the reverse mapping of annotations to fine-granular adaptable content components using different clients. Moreover, by automatically wrapping static HTML to monolithic components, the functionality of existing annotation systems is provided, too. After a short introduction to the XML-based document format the basic concepts and processes of *DynamicMarks* are described.

## 2 Basis: A Component-based Document Format

The document format introduced by the AMACONT project [Fi03] aims at composing Web sites of configurable content components. These are documents or document fragments, instances of a specific XML-grammar representing adaptable Web content on different abstraction levels – see Figure 1. Web sites are constructed by aggregating and linking components to complex document structures. During generation these document structures are translated into Web pages in a specific output format adapted to a specific user model or client. The document format was defined by XML Schema.

On the lowest level there are media components encapsulating media assets (text, structured text, image, video etc.) by describing them with technical (MPEG-7) and non-technical metadata. Note that even whole documents (such as HTML-pages) can be handled as monolithic media assets, by automatically extending them with appropriate metadata.

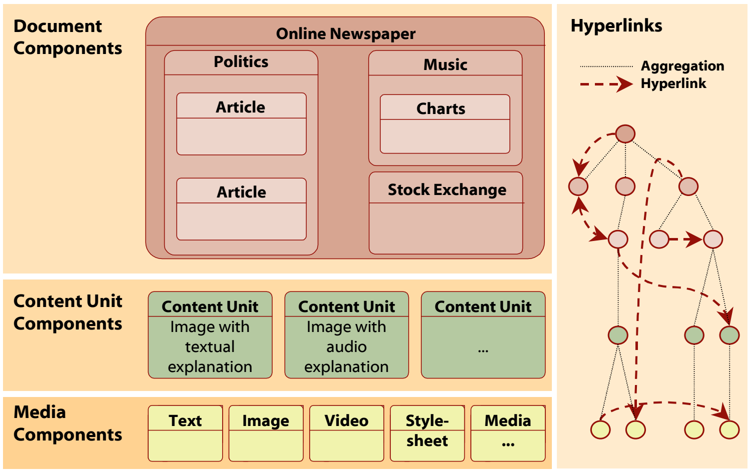


Figure 1: A Component-based Document Format

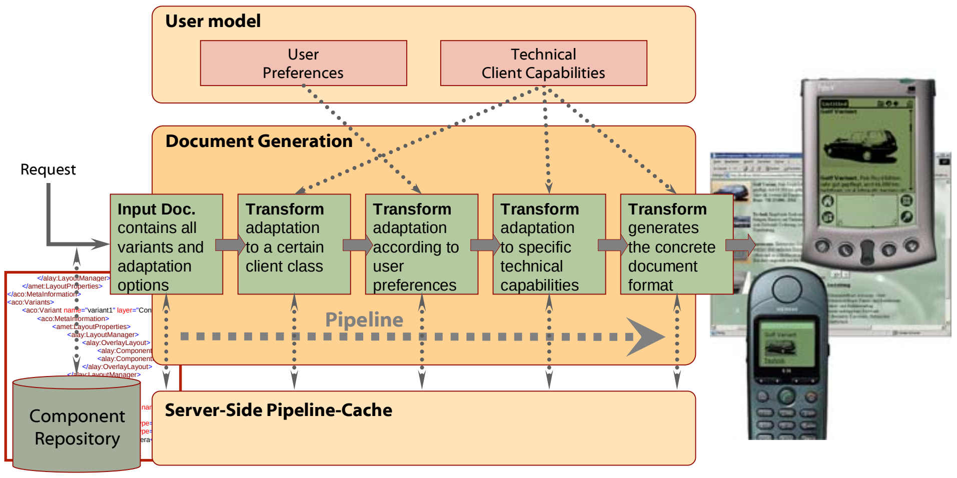
The second level combines media components belonging together semantically (e.g. an image with textual description) to so called *content units*. The spatial adjustment of contained media components is described by client-independent layout properties abstracting from the exact resolution and presentation style of the current display. On the next level, *document components* are specified as parts of Web presentations playing a well defined semantic role (e.g. a news column, a product presentation or even a Web site). They can either reference content units, or aggregate other document components. The resulting hierarchy describing the logical structure of a Web site is strongly dependent from the application context. Again, the spatial adjustment of subcomponents is described in a client-independent way. Finally, the orthogonal *hyperlink view* defines links spanned over all component levels. Uni- and bidirectional typed hyperlinks based on the standards XLink, XPath and XPointer are supported.

The concrete structure shown in Figure 1 is just an example. Generally, arbitrary component structures according to the level-based document format can be created. Furthermore, beside static components (i.e. components with fixed content), component templates can be used, too. These are component skeletons declaring the structural, behavioral and layout aspects of components in a generic way that can be instantiated by being filled with dynamically queried data. E.g. a component author might create such a generic template for a news column that is filled with different content each day.

### 2.1 Document Generation

Document generation is based on a stepwise pipeline concept in order to achieve code reuse and higher performance through caching mechanisms – see Figure 2. In the beginning, a complex document is dynamically assembled from a component repository according to a user's request. It contains information about subcomponents to be included and metadata describing adaptation rules. All possible presentation forms of the component concerning content, layout, and structure are encapsulated.

Depending on the current user model it is subdued to a series of transformations. Each step considers a certain aspect of adaptation by performing conversions to the document (selection and configuration of components). Finally, a Web document in a specific output format (HTML, cHTML, WML etc.) is generated. Figure 2 depicts a possible pipeline with three steps, namely adaptation to a client class, to semantic user preferences and lastly to specific technical capabilities. In Section 3.4 annotation merging as a possible additional step of personalized document generation will be introduced. The document generator was realized on the basis of the pipeline-based publishing framework Cocoon [LZ02].


User model
User Preferences
Technical Client Capabilities
Document Generation
Request
Input Doc. contains all variants and adaptation options
Transform adaptation to a certain client class
Transform adaptation according to user preferences
Transform adaptation to specific technical capabilities
Transform generates the concrete document format
Pipeline
Component Repository
Server-Side Pipeline-Cache

```
            </alay:LayoutManager>
        </amet:LayoutProperties>
</aco:MetaInformation>
<aco:Variants>
        <aco:Variant name="variant1" layer="Con
            <aco:MetaInformation>
                <amet:LayoutProperties>
                    <alay:LayoutManager>
                        <alay:OverlayLayout>
                            <alay:Component
                            <alay:Component
                        </alay:OverlayLayout>
                    </alay:LayoutManager>
```

Figure 2: Pipeline-based Document Generation

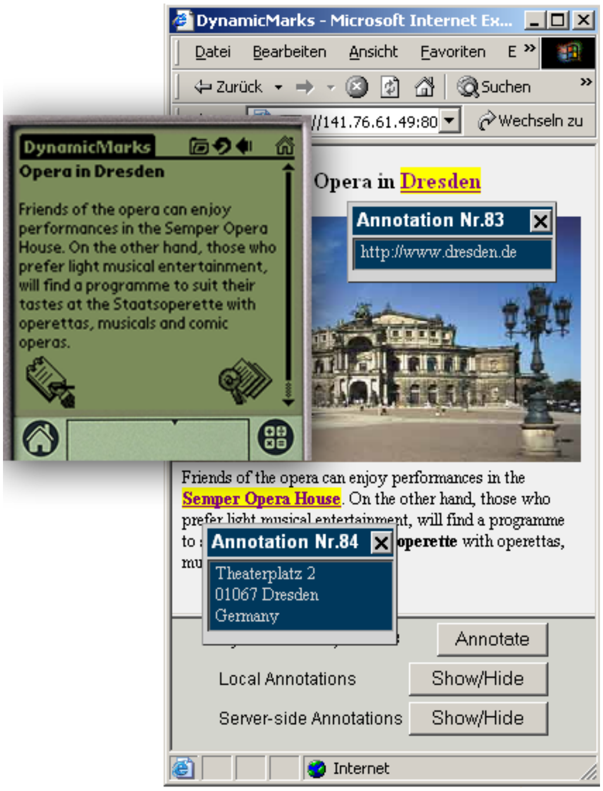
## 3 The *DynamicMarks* Annotation System

*DynamicMarks* was developed to ubiquitously annotate Web pages basing on the mentioned document format. Its underlying idea is to reversely map annotations to components using different clients. Reuse of adaptable components in other scenarios implies reuse of both content and annotations. Moreover, automatically wrapping static HTML to monolithic media components provides existing annotation systems' functionality. The basic processes of *DynamicMarks* are: annotation creation, location, storage and insertion.

### 3.1 Annotation Creation

After a document was generated and presented, users can select parts of it and attach personal remarks. However, in some scenarios it is meaningful to restrict annotations only to selected areas of Web presentations. Take the case of dynamically generated electronic shops, where it is unnecessary to annotate the page header or the navigation pane. In *DynamicMarks* the author of a Web presentation can select certain components to be annotatable. By default all components are annotatable.

Annotation creation features depend on the client, too. Whereas today's popular desktop HTML-browsers allow selecting and manipulating arbitrary parts of Web pages (or components), most handheld browsers do not offer this capability. Moreover, text input via handhelds is still a big challenge for users. Thus, the annotation creation interface has to be adapted to client capabilities. In *DynamicMarks* two annotation scenarios, namely desktop browsers and PDAs were examined.



Opera in Dresden

Friends of the opera can enjoy performances in the Semper Opera House. On the other hand, those who prefer light musical entertainment, will find a programme to suit their tastes at the Staatsoperette with operettas, musicals and comic operas.

Opera in Dresden

Annotation Nr.83

http://www.dresden.de

Friends of the opera can enjoy performances in the Semper Opera House. On the other hand, those who prefer light musical entertainment, will find a programme to operette with operettas,

Annotation Nr.84

Theaterplatz 2
01067 Dresden
Germany

Figure 3: Adaptation of the content and the annotation interface

The Web-based tool for annotation creation in dynamically assembled (X)HTML documents was developed for today's popular desktop browsers, e.g. Mozilla 1.0+, Netscape 6.0+ and Internet Explorer 5.0+. The only requirement was the ability to display frames and to support the DOM2 Range-functionality of JavaScript. As depicted in Figure 3, arbitrary parts of documents might be selected. By clicking the "Annotate" button an annotation to the selection can be created.

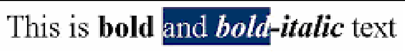
At present annotation types like text highlighting, *emphasizing* and underlining are supported, but richer features, e.g. margin notes, drawings and multimedia annotations will be examined in the future. Notes are shown by correctly positioned popup windows realized in JavaScript.

PDA annotations require a different approach, since most handheld browsers do not support the selection and manipulation of arbitrary documents parts. This problem is solved by marking specific points of the presented Web content where annotations are invited. During document generation, annotatable document components being inserted in the output are automatically provided with small icons anchoring to separate pages where annotations can be entered or viewed. Note Figure 3 presenting the same content adapted to both presentation and annotation capabilities of different clients.

## 3.2 Annotation Location

Annotations created by users have to be attached not to volatile Web pages, but to components serving as the main building blocks of those pages. For the sake of reversibility during annotation location, information about source components must be preserved across document generation. Therefore, <SPAN> tags parameterized by the source components' IDs are inserted in the generated output. While not influencing the presentation view, they enable to unambiguously map annotations to components.

The following example demonstrates this idea. The table below depicts a media component representing formatted text (on the left), the result of transforming it to HTML (on the right), as well as a possible selection made in the WYSIWYG browser view (in the right-bottom cell). Aided by the <SPAN>-tag representing structure information in the generated output, the selection can be unambiguously mapped to the component by the XPointer expression xpointer(string-range(//*[@**id='ft1'**],"",14,8)) independent of any certain document format [De02]. (The *id* attribute contains the component's unique ID, the numbers 14 and 8 locate the first character and the length of the selection.) For HTML pages these expressions are calculated by using the Range-functionality of the DOM2 API provided by JavaScript. The corresponding functions are automatically inserted in the generated output during document generation.

| | |
|---|---|
| `<formattedText id="ft1">`<br>`  <formattedTextContent>`<br>`    This is  <format class="bold">bold</format> and`<br>`    <format class="boldItalic"> bold-italic</format>`<br>`    text`<br>`  </formattedTextContent>`<br>`</formattedText>` | `<SPAN annotID="ft1">`<br>`  This is <B>bold</B> and`<br>`  <B><I>bold-italic</I></B>`<br>`  text`<br>`</SPAN>` |
| |  This is **bold** and ***bold-italic*** text |

Attaching annotations to components with unique IDs makes it also possible to assign them to the underlying content that was used to create those components. However, since creating or dynamically generating components from a knowledge base is a concern of the authoring process of adaptive Web applications, this assignment is not within the scope of this paper. (Current work in the authors' research group deals with the development of component-based adaptive Web presentations according to the model-driven Hera methodology [FHV02] for engineering semantic Web Information Systems.)

Note that the concept of inserting "hidden" structure information into the generated presentation might be generalized for a broad number of dynamic document generation systems, in order to support interoperability with *DynamicMarks*.

## 3.3 Annotation Storage

After being created and located, annotations are stored separately from content components in a database (*Annotation Repository)* either on a server or the client. However, the latter case is currently restricted to only desktop computers. One possible scenario could be to use server-side repositories for shared annotations and client-side repositories for private remarks. The XML-based metadata schema for annotation storage is closely related to that introduced by *Annotea* [KKP01].

The server-side *Shared Annotation Repository* was prototypically realized on the basis of the relational database MySQL, but an XML database implementation is planned for the future. After an annotation is created and located on the client, its coordinates and content are passed to a server-side JSP application, which determines annotation metadata and contacts the database through JDBC. For the sake of simplicity, the client-side *Local Annotation Repositori*es for desktop computers also rely on local databases accessed by locally running Web servers. The main difference between the two repository types is the way they handle annotation insertion into dynamically generated documents – see Section 3.4.

## 3.4 Annotation Insertion

Annotation insertion is the process of merging Web content with annotations. When browsing a Web page the user is presented his previously made remarks. The place of merging annotation in *DynamicMarks* mainly depends on where they are stored. While server-side annotations supporting ubiquitous note sharing are inserted during document generation on the server, private client-side notes are merged into completed Web pages in the browser. Moreover, a combination of both scenarios is possible, too.

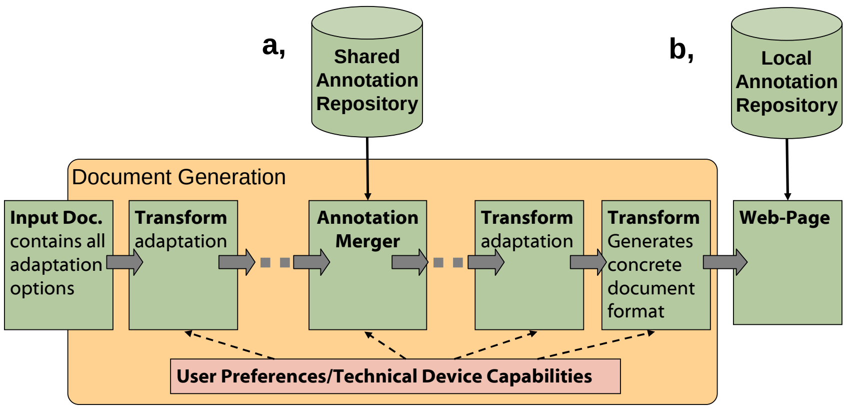

a,
Shared Annotation Repository
b,
Local Annotation Repository
Document Generation
Input Doc. contains all adaptation options
Transform adaptation
Annotation Merger
Transform adaptation
Transform Generates concrete document format
Web-Page
User Preferences/Technical Device Capabilities

Figure 4: Server-side vs. Client-side Annotation Merging

### 3.4.1 Server-side Annotation Insertion

In the first case (Figure 4a) notes are inserted at configurable stages of the adaptation pipeline into partially adapted XML documents. For this purpose, custom Cocoon transformers (called *Annotation Mergers*) were developed, which perform transformations on the DOM view of XML documents based on dynamic queries to the *Shared Annotation Repository*. For the example shown in Section 3.2 the corresponding partially processed XML-document would be "retagged" like this:

```
…
<formattedTextContent>
    This is <format class="bold">bold</format>
    <ANNOT annotID="19" …>
        and <format class="boldItalic"> bold</format>
    </ANNOT>
    <format class="boldItalic">-italic</format> text
</formattedTextContent>
…
```

Note that the result is still a valid XML fragment, even though the selection addressed a non-valid range of the original XML component. (However, since annotations are stored separately from components, the original component and the existing document structure were not changed or broken.)

As the resulting code is independent of any output format, the visualization of annotations (e.g. highlighting, underlining, *emphasizing*) has to be implemented for each format separately. This can be done either by extending the stylesheets executing the last step of document generation, or – in case of Web browsers capable of processing XSLT stylesheets – also on the client side. Still, the resource intensive process of merging annotations is executed uniformly for each device.

The modularity of the pipeline architecture enables to divide server-side annotation insertion in even more steps. Consider storing both shared and personal annotations on the server. At a large number of users sharing comments, it makes sense to separate the insertion of shared and private annotations. By inserting shared comments at an earlier stage, the result can be cached and reused for merging with personal annotations at a later step. Generally, changes to the server-side annotation merging scenario can be easily achieved by using the *Annotation Merger* at variable places in the pipeline.

To sum up, server-side annotation merging has the main advantage of supporting different clients. Since notes are inserted and rendered during document generation, the clients only need to display them without modifying the requested document. This is a benefit for mobile clients with low computing capacity. Another advantage is the possibility to cache annotated, partially adapted documents for reuse for subsequent requests from different clients. Still, server-side annotation insertion causes significant server-load and additional network communication.

### 3.4.2 Client-side Annotation Merging

On the contrary, client-side annotation (Figure 4b) merging inserts notes into completed Web pages. Though supporting better privacy, it assumes the local storage of annotations and browsers with enhanced document modification capabilities. At present it is implemented for desktop computers. Again, the generated Web documents are automatically provided with specific JavaScript functions performing annotation insertion. By means of the SPAN-Tags introduced in Section 3.2 the correct place of components can be exactly relocated in the HTML output. After resolving the XPointer expressions the different kinds of annotations (underlining, emphasizing, highlighting) are visualized by inserting corresponding HTML tags via JavaScript/DOM2-functions.

## 3.5 Annotating Existing Web pages

As already mentioned, *DynamicMarks* was primarily developed for annotating documents based on the component-based format introduced in [Fi03]. However, - for the sake of “downwards compatibility” with existing systems - it was a basic requirement to annotate arbitrary static HTML pages, too.

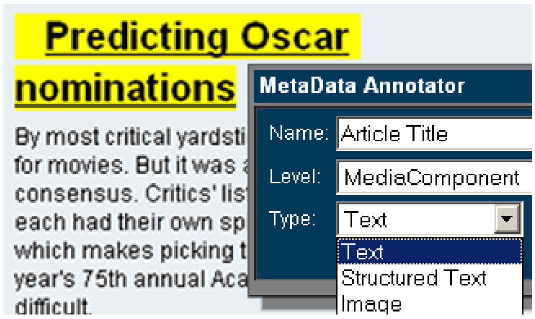
Remind Section 2, where it was explained that even whole HTML-documents can be automatically wrapped to monolithic media components (representing structured text) by attaching appropriate metadata. For this purpose, the Cocoon-based document generation pipeline can be extended with two initial "proxying" transformers. The first one cleans up the requested HTML documents against malformedness and converts them to XHTML by using *JTidy*. By a second XSLT transform specific component metadata (e.g. a unique ID built from the document's URL) as well as the mentioned JavaScript functions performing annotation location and merging are inserted.

After this conversion the resulting component can be annotated by the desktop annotation tool described in Section 3.1. Thus, the problem of locating annotations in existing HTML-pages is reduced to annotation location in components.

## 4 Conclusion and Future Work

This paper briefly introduced *DynamicMarks*, a pervasive annotation system for adaptive Web pages. It enables the ubiquitous annotation of dynamically generated Web content by reversely mapping notes to reusable, adaptive content components. The key concepts described in this paper have been prototypically implemented and are being verified.

At its current state, *DynamicMarks* is dependent from a specific document format. Though it enables to annotate any existing static HTML page via desktop browsers, there is no support for the pervasive annotation of arbitrary Web documents, yet. Nevertheless, the main reason for this shortcoming is the fact that most of today's Web pages have not been designed to be presented on different clients.


Predicting Oscar nominations

By most critical yardsti
for movies. But it was
consensus. Critics' lis
each had their own sp
which makes picking t
year's 75th annual Aca
difficult.

MetaData Annotator

Name: Article Title
Level: MediaComponent
Type: Text

Figure 5: Annotation-based Component Extraction

Inspired by the work of Hori et al. [Ho02] on annotation-based Web transcoding, current research deals with the semi-automatic extraction of adaptable components from existing (HTML) Web pages via annotations – see Figure 5. Fine-granular components can be identified while users select and classify HTML fragments (by attaching structural information) in an intuitive graphical way. Metadata gathered during this "annotation-based wrapping process" can be used to adapt content to various document formats, presentation and annotation interfaces.

Moreover, since text input via handhelds is still a big challenge for users, multimedia annotation capabilities of portable devices (such as TabletPCs or PDAs) will be examined. Finally, the additional server-load resulting from server-side annotation merging has to be properly quantified.

## References

[De02] DeRose, S.; Daniel, R.; Grosso, P.; Maler, E.; Marsh, J.; Walsh, N.: XML Pointer Language (Xpointer). W3C Working Draft, http://www.w3.org/TR/xptr/, 2002

[DV00] Denoue, L.; Vignollet, L.: An annotation tool for Web-browsers and its applications to information retrieval. RIAO2000, Paris, 2000

[FHV02] Frasincar, F.; Houben, G.J.; Vdovjak, R.: Specification Framework for Engineering Adaptive Web Applications. The Eleventh International Conference on the World Wide Web (WWW11) 2002

[Fi03] Fiala, Z.; Hinz, M.; Meissner, K.; Wehner, F.: A Component-based Approach for Adaptive, Dynamic Web Documents. The Twelfth International Conference on the World Wide Web (WWW2003), 20-24 May, Budapest, 2003

[Ho02] Hori, M.; Ono, K.; Koyanagi, T.; Abe, M.: Annotation by Transformation for the Automatic Generation of Content Customization Metadata. Pervasive 2002, 2002

[KKP01] Kahan, J.; Koivunen, M.; Prud’ Hommeaux, E.: Annotea: an Open RDF Infrastructure for Shared Web Annotation. Tenth International World Wide Web Conference (WWW10), Hong Kong, 2001

[LZ02] Langham, M.; Ziegeler, C.: Cocoon: Building XML Applications. New Riders, 2002

[OAM00] Ovsiannikov, I.A.; Arbib, M.; McNeill, T.: Annotation Technology. International Journal of Human-Computer Studies. 50(4). 2000

[RMW95] Röscheisen, M.; Morgensen, C.; Winograd, T.: Interaction Design for Shared World-Wide Web Annotations. In Proceedings of the CHI '95, Denver, Colorado, 1995

[WS99] Watters, C.; Shepherd, M.: Research Issues for Virtual Documents. Workshop on Virtual Documents, Hypertext Functionality and the Web at WWW8, Toronto, Canada, 1999