

LiveInfo: Adapting Web Experience by Customization and Annotation

Paul P. Maglio and Stephen Farrell

IBM Almaden Research Center
650 Harry Road
San Jose, California 95120
{pmaglio,sfarrell}@almaden.ibm.com
<http://www.almaden.ibm.com/cs/people/pmaglio>

Abstract. Intermediaries are perfectly suited to customizing and annotating web pages, as they can stand in the flow of data between web browser and web server, monitoring user behavior and modifying page markup. In this paper, we present LiveInfo, an intermediary-based framework for customizing and annotating web pages. LiveInfo breaks customization/annotation into four steps: (a) splitting streaming data into useful chunks, (b) identifying meaningful patterns of chunks, (c) merging together overlapping patterns, and (d) adding markup to customize and annotate. Each of these steps is easily replaced or configured, making it simple to adapt web experience by customization and annotation.

1 Introduction

The World Wide Web (WWW) presents users with a particular view of information — the view that the information authors built into it. Systems for adapting the web to individual users or to groups of users can present different views of the available information. Many approaches for adapting the web are targeted at applications such as education, modifying what links appear on pages based on a user’s skill [8, 6]. Other approaches adapt web pages to a user’s history of interaction with the web by adding links to pages that have been repeatedly visited [4, 15]. A system for adapting web pages is a special case of an *attentive system* [18], as adapting web pages involves monitoring what web users see and modifying what web users see, effectively paying attention to what users are doing to attend to their information needs. *Intermediaries* are particularly well-suited to adapting web pages, as they can sit between an information consumer and an information producer, monitoring and modifying data that flow between them [2, 3].

In this paper, we outline an intermediary-based approach for adapting a web user’s experience through customization, a process that takes the user into account when modifying pages, and annotation, a process that takes facts about the world outside the user into account when modifying pages. In what follows, we first discuss intermediary computation, highlighting its role in customization and annotation, and then describe our intermediary-based scheme for web page adaptation by customization and annotation.

2 Intermediaries

Intermediaries are a general class of computational entities that act on data flowing along an information stream [2, 3]. In general, intermediaries sit along a data flow, possibly affecting the data that flow from some origin to some destination. In the case of computer networks, the abstract concept “intermediary” might be instantiated as hypertext transfer protocol (HTTP) proxy server, which provides a specific point on the communication stream for adding function. Many protocols support the addition of intermediary computation, including simple mail transfer protocol (SMTP), which can incorporate mail exchangers at specific points along the flow, and network news transfer protocol (NNTP), which involves peer-to-peer information transfer and so servers themselves can act as intermediaries.

At a low level, there are a variety of ways in which an intermediary might affect the flow of data (see Figure 1). In the simplest case, an intermediary might tunnel data through without having any effect on the data at all. Another basic intermediary function is filtering, which preserves the essential structure of the origin data, but removes or adds information to the stream. An intermediary can replace the origin data with data from a different source, or it can merge data from several different sources.

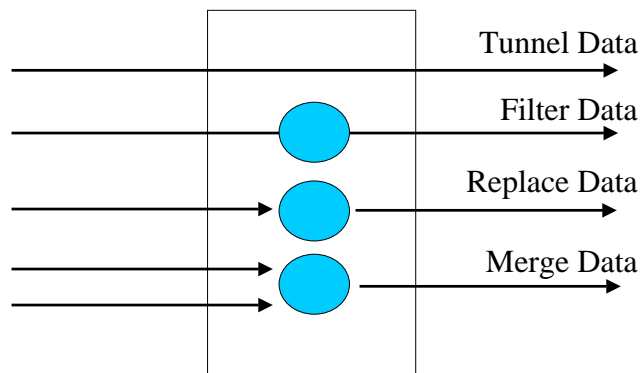


Fig. 1. An intermediary can perform many operations on data flowing through it.

At a higher level, an intermediary can perform a variety of functions. For HTTP streams, for instance, intermediary computation (e.g., by an HTTP proxy server) might be used to (a) customize content based on history of interaction of individual users [4, 15], (b) annotate content with traffic lights to indicate network delay [4, 7], (c) add awareness and interaction to enable collaboration with other users [16, 17], (d) transcode data from one image type to another [11, 13], (e) cache content, and (f) aggregate content from several sources. Thus, we

distinguish six broad applications of intermediary computation: customization, annotation, collaboration, transcoding, caching, and aggregation. More precisely, we distinguish these intermediary functions by the kinds of information they take into account when performing their transformations. Customization takes into account information about the user or the user's environment when modifying data on the stream, for instance, adding links the user often visits. Annotation takes into account information about the world outside the user or user's environment, for instance, by determining link speed. Collaboration takes into account information about other users, for instance, what web page they are currently visiting. Transcoding takes into account information about the data's input and desired output format, for instance, transforming a JPEG to a GIF image. Aggregation takes into account a second data stream, for instance, merging results from several engines into a single page. Caching takes into account when data were last stored or last changed.

2.1 Customization and Annotation

Here, we focus on customization and annotation. As noted, these are distinguished from other sorts of intermediary functions by what the intermediary process takes into account. Customization uses information about the user to tailor the data flowing along the stream. For instance, the way many web-based shopping sites automatically fill out forms based on stored user profiles is a kind of customization (such as, Yahoo! Shopping¹). In the adaptive hypertext field, one common customization is to incrementally add links to pages based on a student's familiarity with the course, for example, allowing the student access to more advanced material only after the student has a certain amount of experience with less advanced material (e.g., [8]). In the case of automatic form-filling, the web is customized based on an explicit profile provided by the user. In the case of adaptive courseware, pages are customized based on a model of the user gathered implicitly from user activities (see also [6]).

Annotation relies on information about the world outside the user to tailor data flowing along the stream. For instance, many web systems have been constructed to annotate hyperlinks on web pages with visual indications of how popular or well-traveled the link is (e.g., [10, 19]). Others have used auditory annotations to convey a link's popularity [9] or how long it takes to download [1]. Still others have annotated links to network news articles with ratings [14] or links to search results with visual indications of relevance along certain dimensions [12]. In all cases, annotations are derived from information about the document, its history of use, its server, and so on.

Of course, customization and annotation can be combined by an intermediary process that takes account of both information about the user and information about the world outside the user when tailoring the data stream. Taken together, customization and annotation can do more than the two taken independently,

¹ <http://shopping.yahoo.com/>

as annotations can be customized and customizations can be annotated. For instance, adaptive courseware might customize pages to users' abilities, removing and adding text and links as appropriate, and then added links might be annotated with information about the difficulty of the linked-to material (cf. [6]), resulting in annotated customization.

2.2 Web Intermediaries (WBI)

The Web Intermediaries (WBI) Development Kit² is an implemented framework for adding intermediary functions to the WWW [2–4]. WBI is a programmable proxy server designed for easy development and deployment of intermediary applications. Using WBI, intermediary applications are constructed from four basic building blocks: request editors, generators, document editors, and monitors. We refer to these collectively as MEGs, for Monitors, Editors, Generators. Monitors observe transactions without affecting them. Editors modify outgoing requests or incoming documents. Generators produce documents in response to requests. WBI dynamically constructs a data path through the various MEGs for each transaction. To configure the route for a particular request, WBI has a rule associated with each MEG that specifies a boolean condition indicating whether the MEG should be involved in a transaction based on header information about the request or response. An application (WBI plugin) is usually composed of a number of MEGs that operate in concert to produce a new function.

One way WBI can customize web pages is by adding links to pages frequently visited by a particular user [15]. Because web users rely on routines and standard behaviors to access information, WBI's "Short Cuts" plugin adds links to pages that a user routinely visits shortly after visiting the current page. This plugin uses a monitor to build up a database of pages that a user visits, and a document editor to add links when there are pages in the database that the user habitually visits within some radius (such as 5 clicks) from the current page.

One way WBI can annotate web pages is by adding extra information about the hyperlinks on a page, such as the speed of downloading information from a particular site [4, 7]. WBI's "Traffic Lights" plugin uses a document editor to add little green, yellow, and red images next to hyperlinks to inform the user that responses from the server on the other end of that link is fast, slower, or slowest.

3 LiveInfo Framework

LiveInfo is a WBI plugin for customizing and annotating hypertext markup language (HTML) data. It provides a mechanism for transforming "interesting" words or phrases into a new kind of visually distinct hyperlink. When followed, these LiveInfo links pop up a non-disruptive, floating window that contains more information about the hyperlinked word or phrase. In one example, an IBM employee's name appearing in a web page from the `ibm.com` domain is transformed

² <http://www.almaden.ibm.com/cs/wbi/>

into a hyperlink with a pale blue background, which when clicked, will pop up summary data for that person from the IBM-internal directory, including phone number, email address, and so on (see Figure 2).

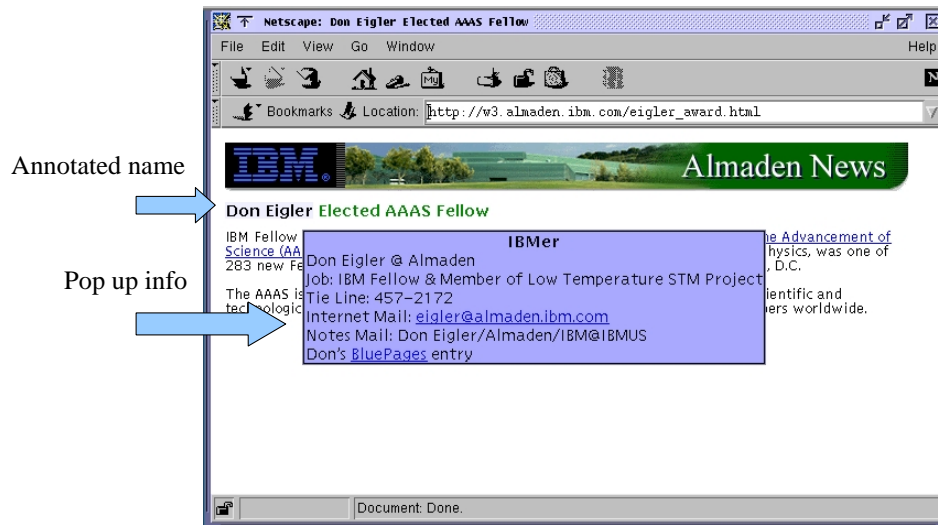


Fig. 2. Web page showing annotation of IBM employee name, and a pop up window showing employee data.

Annotating web pages with hyperlinks is not new (see GuruNet³, flyswat⁴ and Wordbot⁵), but such previous systems were inflexible. For example, GuruNet and flyswat are tied to browser-specific plugins as well as proprietary and non-extendible content sources. By contrast, LiveInfo was designed to be client-neutral and to support extendible content. This flexibility is achieved with an open, pluggable architecture built on the WBI application programming interface (API)⁶. In particular, the logic for recognizing items in a document can be loaded and unloaded dynamically, allowing a user or organization to select and create custom data analyzers.

3.1 Architecture and Implementation

LiveInfo's annotation process is broken down into four steps. Each of these steps constitutes a pluggable component of the LiveInfo framework (see Figure 3):

³ <http://www.gurunet.com/>

⁴ <http://www.flyswat.com/>

⁵ <http://www.wordbot.com/>

⁶ <http://www.almaden.ibm.com/cs/wbi/doc/>

1. **Chunker**: stream \rightarrow chunks
2. **Analyzer**: chunks \rightarrow Items
3. **Coalescer**: Items \rightarrow CItems
4. **Markupper**: CItems, chunks \rightarrow stream

This design was motivated by interests of flexibility and performance.

Because document editors modify the data stream in real-time, performance is essential. To this end, LiveInfo uses streaming rather than buffering of the data source; communicates with components through simple method calls; and analyzes data in chunks rather than term-by-term.

With regards to flexibility, there are several open questions to which our best answer was to maximize modularity. Certainly the problem of recognizing “interesting” items is open and we want end users or third parties to be able to plug in code to handle this work. However, the question of how to break up the data into reasonably apportioned chunks for analysis and how to handle the case of overlapping items are also multifaceted.

Keeping this in mind, we discuss each of these components in turn.

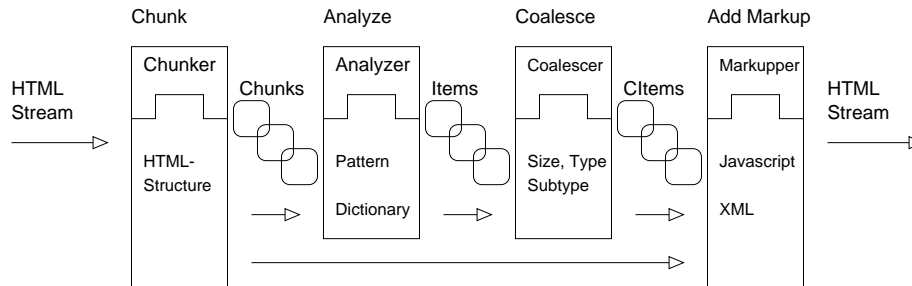


Fig. 3. Flow of processing in LiveInfo plugin.

Chunker. The **Chunker** sits in the data stream, transforming bytes that flow past it into more meaningful buffers or “chunks” of data. Constructed from an input and output stream, and the **Chunker** produces chunks (actually strings) from the input stream, and puts modified chunks back on the output stream. The default **Chunker** implementation uses HTML document structure to determine chunk boundaries: text between tags is considered a chunk. This approach is simple, prevents different structural parts of the document from being joined into a single term (e.g., different rows on a table), and avoids collisions of markup to ensure that an HTML tree is maintained. However, phrases that extend across markup (such as a phrase with a bold-faced word) are not recognized, pages that do not contain any markup are treated as one large chunk, and pages that contain a lot of markup are divided into many small chunks making processing less efficient.

Analyzer. **Analyzers** do the interesting work of finding useful terms or phrases within text. Any text analysis engine (e.g., [5]) can be plugged in to LiveInfo by writing a wrapper for it that implements the **Analyzer** interface. **Analyzers** return their result as a list of **Items** that contains the extents and meta-data for each interesting portion of text.

Two recurring specializations of the **Analyzer** interface are pattern matching and multi-term dictionary. Generic code was written to simplify the writing of analyzers of these sorts. For pattern matching, a regular expression library detects items such as phone numbers, uniform resource locators (URLs), email addresses, currency, and so on. Regular expression matching can also be used to minimize the frequency of expensive operations. For example, our name recognition algorithm for the employee directory example comprises three steps: (a) the regular expression engine searches for pairs of capitalized words, possibly separated by a single capitalized letter and a period; (b) the first name is compared against a list of known first names or the first and last name are compared against a list of known non-name words, and (c) an LDAP query is issued to confirm that the discovered pair of names is in fact in the directory.

An alternative to the pattern-matching specialization is the multi-term dictionary. In contrast to a dynamic lookup scheme that relies on regular expressions, the multi-term dictionary provides a way of building a set of key words and phrases into the **Analyzer**. As an implementation detail, this requires developers to create a Java resource that contains a list of words, and a **Scraper** class that formats the definition or link that is associated with a word.

Coalescer. The **Coalescer** implements the policy for handling the problem of overlapping items. For purposes of this discussion, “overlapping” items include strict overlaps ($a0 < b0 < a1 < b1$) as well as containment ($a0 < b0 < b1 < a1$), where $a0$ and $a1$ are the extents of item a . Overlapping items might come from the same module or from different modules. For example, the following string contains overlapping items:

```
...send a note to Stephen Farrell/Almaden/IBM and...
```

This snippet might be seen as containing many items, including the name “Stephen Farrell”, the Lotus Notes address “Stephen Farrell/Almaden/IBM”, and the stock symbol “IBM”. Note that the first two are the same kind of thing: both identify a person. Given a choice, the user would prefer the more specific match. However, the “IBM” symbol is a different kind of thing, and the fact that it is located within another recognized item (the Notes address) should not prevent the user from learning more about it.

There are several choices available to handle the case of overlapping extents. The first we call “exclude-on-overlap”. In this case, some item “wins”, presumably based upon some prioritizing scheme or by the size of the match. The problem with this approach the information gathered by the losing items is lost. The next possibility is “retain-overlaps”. This is really the null case for the coalescer: all of the items are retained and the work is passed to the presentation handler.

One major flaw of this approach is that the resulting structure will not be a tree in the case of strict overlaps, and thus the resulting document will be invalid. Another concern is that it is difficult to represent the overlaps to the user: one might imagine blending colors or hatch patterns. The final idea, “coalesce-on-overlap” is to merge all of the overlapping items and their meta-data into a super-item (CItem, or coalesced Item). This approach retains all of the information while ensuring a proper tree structure. There are some problems, however. In the case of “Stephen Farrell” and “Stephen Farrell/Almaden/IBM”, the user is handed both person identifiers, and in this case one is definitely better than the other (there are many Stephen Farrells in IBM). Another problem is the case of the largest item being much larger than the contained item. For example, if a whole paragraph of one hundred words is matched as well as several individual words within that paragraph, then the position of the words within the paragraph is lost.

A good policy might be a hybrid of exclude-on-overlap and coalesce-on-overlap: If overlapping items are from the same analyzer group (i.e., are the same kind of thing), then the one with the largest extents is chosen. If they are from separate analyzer groups, then they are coalesced. So in the current example, this means that of the two items recognized by the Bluepages analyzer group, the name “Stephen Farrell” is discarded, but the Notes address is retained. Since the stock symbol “IBM” came from a different analyzer group, it is added to the coalesced item and can be shown as another menu item in the user interface.

Markupper. The **Markupper** is responsible for generating the **begin** and **end** tags for each interesting item or coalesced set of items the system has encountered. Implementations might include an XML **Markupper** for producing machine readable content, or a Javascript **Markupper** for producing a floating-window user interface. For example, the user interface shown in Figure 2 relies on a Javascript-based interface created by a WBI generator to render the contents of the floating window. As this content is only generated on demand, this step can be more expensive than the initial annotation of the document. In this case, the contents of the corporate directory entry can be displayed.

3.2 Customization and Annotation

The LiveInfo framework can be used to customize or annotate web pages. The corporate directory plugin (shown previously in Figure 2) is an example of annotating web pages with information from the outside world, namely employee data. Another example of annotation would be a LiveInfo plugin that turns ordinary words on web pages into hyperlinks that point to their definitions as found in some web-based dictionary. In this case, the **DictionaryAnalyzer** would use a list of all terms found in some external dictionary, and the **Markupper** would add links to this external definition source to fill the pop up window with the definition.

The same approach can be used to customize web pages, for instance, by relying on a user's explicitly created profile when adding annotations to a page. In this case, the user might create a list of words or phrases that he or she is particularly interested in noticing on web pages. The **Analyzer** would use this list as its basis for discovering terms, and the **Markupper** might simply highlight these terms in a bright color. Another example of using LiveInfo to customize web experience relies on analyzing term-frequencies of words encountered on pages the user regularly visits. In this plugin, the **Analyzer** might use as input high frequency terms, and the **Markupper** would link these terms to previously encountered pages with similar term-frequency distributions.

The LiveInfo framework can also be used to customize annotations. For example, any of the annotations schemes we have described, such as the corporate directory or the dictionary, can be customized so that the user can select how the annotations are displayed. Our examples have focused on inline annotations, but a user may prefer to see these in a pop up window or in the header or footer of the page. In this way, users can customize how annotations are presented.

3.3 Analysis of LiveInfo

4 Summary

Customization and annotation are two broad methods for adapting web pages beyond the structure and information that was designed or authored into them. Intermediary-based computation provides one clear approach to adapting web pages in these ways. In this paper, we have presented the LiveInfo framework, an intermediary-based approach to web page customization and annotation.

Acknowledgments

Thanks to Rob Barrett, Steve Ihde, and Joerg Meyer, for all manner of practical and philosophical support. Thanks to Andreas Dieberger for helpful conversations on this topic. Thanks to Teenie Matlock and three anonymous reviewers for comments on a draft of this paper.

References

1. Albers, M., and Bergman, E. The audible web: Auditory enhancements for mosaic. In *CHI'95 - Conference Companion* (Denver, CO, 1995), ACM Press, pp. 318–319.
2. Barrett, R., and Maglio, P. P. Intermediaries: New places for producing and manipulating web content. In *Proceedings of the Seventh International World Wide Web Conference (WWW7)* (1998).
3. Barrett, R., and Maglio, P. P. Intermediaries: An approach to manipulating information streams. *IBM Systems Journal* 38 (1999), 629–641.
4. Barrett, R., Maglio, P. P., and Kellem, D. C. How to personalize the web. In *Proceedings of the Conference on Human Factors in Computer Systems (CHI '97)* (New York, NY, 1997), ACM Press.

5. Boguraev, B., and Kennedy, C. Applications of term identification technology: Domain description and content characterization. *Natural Language Engineering* 1 (1998), 1–28.
6. Brusilovsky, P. Methods and techniques for adaptive hypermedia. *User-Modeling and User-Adapted Interaction* 6 (1996).
7. Campbell, C. S., and Maglio, P. P. Facilitating navigation in information spaces: Road signs on the world wide web. *International Journal of Human-Computer Studies* 50 (1999), 309–327.
8. DeBra, P., and Calvi, L. Towards a generic adaptive hypermedia system. In *Proceedings of the Second Workshop on Adaptive Hypertext and Hypermedia* (1998), pp. 5–11.
9. Dieberger, A. A sonification enhanced navigation tool for history-enriched information spaces. In *International Conference on Auditory Displays (ICAD '2000)* (2000).
10. Dieberger, A., and Lonnqvist, P. Visualizing interaction history on a collaborative web server. In *Hypertext 2000* (San Anotnio, TX, 2000), ACM Press.
11. Fox, A., and Brewer, E. A. Reducing www latency and bandwidth requirements by real-time distillation. In *Proceedings of the Fifth International World Wide Web Conference (WWW5)* (1996).
12. Hearst, M. A. Tilebars: Visualization of term distribution information in full text information access. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI '95)* (Denver, CO, 1995), ACM Press, pp. 59–66.
13. Ihde, S., Maglio, P. P., Meyer, J., and Barrett, R. Intermediary-based transcoding framework. In *Poster Proceedings of the Ninth International World Wide Web Conference* (2000).
14. Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., and Riedl, J. GroupLens: Applying collaborative filtering to usenet news. *Communications of the ACM* 40 (1997), 77–87.
15. Maglio, P. P., and Barrett, R. How to build modeling agents to support web searchers. In *Proceedings of the Sixth International Conference on User Modeling* (New York, NY, 1997), Springer Wien.
16. Maglio, P. P., and Barrett, R. Adaptive communities and web places. In *Second Workshop on Adaptive Hypertext and Hypermedia* (1998), P. Brusilovsky and P. DeBra, Eds.
17. Maglio, P. P., and Barrett, R. Webplaces: Adding people to the web. In *Poster Proceedings of the Eighth International World Wide Web Conference* (1999).
18. Maglio, P. P., Barrett, R., and Campbell, C. S. and Selker, T. Suitor: An attentive information system. In *Proceedings of the International Conference on Intelligent User Interfaces 2000* (New Orleans, LA, 2000), ACM Press.
19. Wexelblat, A., and Maes, P. Footprints: History-rich tools for information foraging. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI '99)* (Pittsburgh, PA, 1999), ACM Press, pp. 270–277.