Manuscript Number:

Title: Automating Content Extraction of HTML Documents

Article Type: Manuscript

Section/Category:

Keywords: DOM trees, content extraction, reformatting, HTML documents, accessibility, speech rendering, text summarization.

Corresponding Author: Suhit Gupta Columbia University

First Author: Suhit Gupta

Order of Authors: Suhit Gupta; Gail E Kaiser, B.S., M.S., Ph.D; Peter Grimm, B.E.; Michael F Chiang, M.D.; Justin Starren, M.D.

Abstract:

Automating Content Extraction of HTML Documents

Suhit Gupta
Columbia University
Dept. of Comp. Sci.
New York, NY 10027, US
001-212-939-7184
suhit@cs.columbia.edu

Gail E. Kaiser
Columbia University
Dept. of Comp. Sci.
New York, NY 10027, US
001-212-939-7000
kaiser@cs.columbia.edu

Peter Grimm
Columbia University
Dept. of Elec. Eng.
New York, NY 10027, US
001-212-939-7000
pmg23@columbia.edu

Michael F. Chiang
Columbia University
Depts. of Ophthalmology
and Biomedical Informatics
New York, NY 10032, US
001-212-305-9535
chiang@dbmi.columbia.edu

Justin Starren
Columbia University
Depts. of Biomedical
Informatics and Radiology
New York, NY 10032, US
001-212-305-3443
starren@dbmi.columbia.edu

Abstract. Web pages often contain clutter (such as unnecessary images and extraneous links) around the body of an article that distracts a user from actual content. Extraction of "useful and relevant" content from web pages has many applications, including cell phone and PDA browsing, speech rendering for the visually impaired, and text summarization. Most approaches to making content more readable involve changing font size or removing HTML and data components such as images, which takes away from a webpage's inherent look and feel. Unlike "Content Reformatting", which aims to reproduce the entire webpage in a more convenient form, our solution directly addresses "Content Extraction". We have developed a framework that employs an easily extensible set of techniques. It incorporates advantages of previous work on content extraction. Our key insight is to work with DOM trees, a W3C specified interface that allows programs to dynamically access document structure, rather than with raw HTML markup. We have implemented our approach in a publicly available Web proxy to extract content from HTML web pages. This proxy can be used both centrally, administered for groups of users, as well as by individuals for personal browsers. We have also, after receiving feedback from users about the proxy, created a revised version with improved performance and accessibility in mind.

Categories and Subject Descriptors. I.7.4 [**Document and Text Processing**]: Electronic Publishing; H.3.5 [**Information Storage and Retrieval**]: Online Information Services – *Web-based Services*

General Terms. Human Factors, Algorithms, Standardization.

Keywords. DOM trees, content extraction, reformatting, HTML documents, accessibility, speech rendering, text summarization.

1. Introduction

Web pages are often cluttered with distracting features around the body of an article that distract the user from the actual content they're interested in. These "features" may include pop-up ads, flashy banner advertisements, unnecessary images, or links scattered around the screen. Automatic extraction of useful and relevant content from web pages has many applications, ranging from enabling end users to accessing the web more easily over constrained devices like PDAs and cellular phones to providing better access to the web for the visually impaired.

Most traditional approaches to removing clutter or making content more readable involve increasing font size, removing images, disabling JavaScript, etc., or a combination of these methods, all of which eliminate the webpage's inherent look-and-feel. Examples include WPAR [18], Webwiper [19] and JunkBusters [20]. All of these products involve hardcoded techniques for certain common web page designs as well as fixed "blacklists" of advertisers. This can produce inaccurate results if the software encounters a layout that it hasn't been programmed to handle. Another approach has been content reformatting which reorganizes the data so that it fits on a PDA; however, this does not eliminate clutter but merely reorganizes it. Opera [21], for example, utilizes their proprietary Small Screen Rendering technology that reformats web pages to fit inside the screen width. We propose a "Content Extraction" technique that can remove clutter without destroying webpage layout, making more of a page's content viewable at once. These techniques should also work on web pages made up of multiple content bodies, even if they are separated by the distracting features or with them interspersed within the different sections of content.

Content extraction is particularly useful for the visually impaired and blind [48]. A common practice for improving web page accessibility for the visually impaired is to increase font size and decrease screen resolution; however, this also increases the size of the clutter, reducing effectiveness. Screen readers for the blind, like Hal Screen Reader by Dolphin Computer Access [46] or Microsoft's Narrator [47], don't usually automatically remove such clutter either and often read out full raw HTML. Webaim Screen Reader [49] and IBM Homepage Reader [50] do attempt to enhance usability by pruning out duplicate pieces of information however they tend to be slow and do not give enough control to the user in directly selecting what a user may be interested in [48]. Therefore, both groups benefit from extraction, as less material must be read to obtain the desired results.

Natural Language Processing (NLP) and information retrieval (IR) algorithms can also benefit from content extraction, as they rely on the relevance of content and the reduction of "standard word error rate" to produce accurate results [13], where the error rate is number of words incorrectly processed from the original format. Content extraction allows the algorithms to process only the extracted content as input as opposed to cluttered data coming directly from the web [14]. Currently, most NLP-based algorithms require writing specialized extractors for each web domain [14][15]. While generalized content extraction is less accurate than hand-tailored extractors, they are often sufficient [22] and reduce labor involved in adopting information retrieval systems.

While many algorithms for content extraction already exist, it appears that few working implementations can be applied in a general manner. Our solution employs a series of techniques that address the aforementioned problems, and makes it easy to implement and experiment with additional algorithms.

In order to analyze a web page for content extraction, we pass web pages through an open source HTML parser, which creates a Document Object Model (DOM) tree, an approach also adopted by Chen et al. [56]. The Document Object Model (www.w3.org/DOM) is a standard for creating and manipulating in-memory representations of HTML (and XML) content. By parsing a webpage's HTML into a DOM tree, we can not only extract information from large logical units similar to Buyukkokten's "Semantic Textual Units" (STUs, see [3][4]), but can also manipulate smaller units such as specific links within the structure of the DOM tree. In addition, DOM trees are highly transformable and can be easily used to reconstruct a complete webpage. Finally, increasing support for the Document Object Model makes our solution widely portable.

One caveat is important to note: Determining the specific content that an arbitrary author intended to portray or, more significantly from our perspective, which an arbitrary user prefers to read, is very hard. Crunch extracts the "content" heuristically, with heuristics customizable by an administrator and/or by a savvy user; there is probably no precise "one size fits all" algorithm that could achieve this goal. In particular, we do not attempt to model either author or user tasks, nor their corresponding context or intentions, but any non-intrusive approach to doing so would also likely be heuristic and thus also imprecise. Therefore, one of the limitations of our framework is that Crunch may remove items from the web page that the user may be interested in, and may present content that the user is not particularly interested in. One way to ameliorate this restriction may be to summarize all removed materials in meaningful chunks, and produce this information in another pane or at the bottom of the page; another approach may be to "learn" on a per-user and/or per website basis, e.g., from data gathered via user studies like the one we report.

In section 2, we discuss the existing solutions out there. In sections 3 and 4, we describe our approach at an abstract level and addressing system implementation issues, respectively. Section 5 presents the initial findings from our ongoing user study. We consider potential future work in section 6, finally concluding in section 7. The appendices present additional materials for interested readers.

2. Related Work

There is a large body of related work in content identification and information retrieval that attempts to solve similar problems using various other techniques. Finn et al. [1] discuss methods for content extraction from "single-article" sources, where content is presumed to be in a single body. The algorithm tokenizes a page into either words or tags; the page is then sectioned into 3 contiguous regions, placing boundaries to partition the document such that most tags are placed into outside regions and word tokens into the center region. This approach works well for single-body documents, but destroys the structure of the HTML and doesn't produce good results for multi-body documents, i.e., where content is segmented into multiple smaller pieces, common on Web logs ("blogs") like Slashdot (http://slashdot.org). In order for content of multi-body documents to be successfully extracted, the running time of the algorithm would become polynomial time with a degree equal to the number of separate bodies, i.e., extraction of a document containing 8 different bodies would run in $O(N^8)$, N being the number of tokens in the document.

McKeown et al. [8][15] similarly use semantic boundaries to detect the largest body of text on a webpage (by counting the number of words) and classify that as content. This method works well with simple pages. However, this algorithm produces noisy or inaccurate results handling multi-body documents, especially with random advertisement and image placement.

Rahman et al. [2] propose another technique that uses structural analysis, contextual analysis, and summarization. The structure of an HTML document is first analyzed and then decomposed into smaller subsections. The content of the individual sections can then be extracted and summarized. Contextual analysis is performed with proximity and HTML structure analysis in addition to "natural language processing involving contextual grammar and vector modeling" [2]. However, this proposal has yet to be implemented. Furthermore, while the paper lays out prerequisites for content extraction, it doesn't propose methods to do so.

Many approaches have been suggested for formatting web pages to fit on the small screens of cellular phones and PDAs. For instance, the Opera browser [16] uses the handheld CSS media type. Bitstream ThunderHawk [17] uses intelligent font resizing: "[It] renders the text using the Kaasila family of fonts, fine tunes images using ThunderHawk's graphic scaling, compacts the data, and sends the page to the ThunderHawk client on the wireless device" [27]. The Skweezer Proxy [28] simply reorganizes the physical layout of the webpage retaining all original content. In general, the reformatting for small screens approaches basically end up only reorganizing the content of the webpage to better fit on the constrained device but still require a user to scroll and hunt for content.

Buyukkokten et al. [3][10] define "accordion summarization" as a strategy where a page can be shrunk or expanded much like the instrument. They also discuss a method to transform a web page into a hierarchy of individual content units called Semantic Textual Units, or STUs. First, STUs are built by analyzing syntactic features of an HTML document, such as text contained within paragraph (<P>), table cell (<TD>), and frame component (<FRAME>) tags. These features are then arranged into a hierarchy based on the HTML formatting of each STU. STUs that contain HTML header tags (<H1>, <H2>, and <H3>) or bold text (<B>) are given a higher level in the hierarchy than plain text. This hierarchical structure is finally displayed on PDAs and cellular phones, but typically showing different content than the original work. In particular, once the STU has been identified, Buyukkokten, et al. [3][4] perform summarization on the STUs to produce the content that is then displayed on PDAs and cell phones. While Buyukkokten's hierarchy is similar to our DOM tree-based model, DOM trees remain highly editable because they abstract the tags away from the content, unlike the STUs, but can easily be reconstructed back into a complete webpage – although summarization filters could similarly be applied to select subtrees. Further, DOM trees are a widely-adopted W3C standard, easing support and integration of our technology.

Kaasinen et al. [5] discuss methods to divide a web page into individual units likened to cards in a deck. Like STUs, a web page is divided into a series of hierarchical "cards" that are placed into a "deck". This deck of cards is presented to the user one card at a time for easy browsing. The paper also suggests a simple conversion of HTML content to WML (Wireless Markup Language), resulting in the removal of simple information such as images and bitmaps from the web page so that scrolling is minimized for small displays. The cards are created by this HTML to WML conversion proxy [5]. While this reduction has advantages, the method proposed in that paper shares problems with STUs. The problem with the deck-of-cards model is that it relies on splitting a page into tiny sections that can then be browsed as windows. But this means that it is up to the user to determine on which cards the actual contents are located, and since this system was used primarily on cell phones, scrolling through the different cards in the entire deck soon became tedious.

Chen et al. [56] propose a similar approach to the deck of cards method, except in their case using the DOM tree for organizing and dividing up the document. They propose showing an overview of the desired page so the user can select the portion of the page he/she is truly interested in. When selected, that portion of the page is zoomed into full view. One of their key insights is that their overview page is actually a collection of semantic blocks that the original page has been broken up into, each one color coded to show the different blocks to the user. This, very nicely, provides the user with a table of contents from which to select the desired section. While this is an excellent idea, it still involves the user clicking on the block of choice, and then going back and forth between the overview and the full view.

None of these concepts solve the problem of automatically extracting just the content, although they do provide simpler means in which the content can be found. These approaches perform limited analysis of web pages themselves and in some cases information is lost in the analysis process. By parsing a webpage into a DOM tree, we have found that one not only gets better results but has more control over the exact pieces of information that can be manipulated while extracting content.

3. Our Approach

Our solution employs multiple extensible techniques that incorporate the advantages of the previous work on content extraction like accordion summarization and content discovery, and attempts to avoid the common pitfalls like noisy results and slow performance. Since a content extraction algorithm can be applied to many different applications, for example in the fields of NLP and IR, as well as assistive technologies like those that help the visually impaired, we implemented it so that it can be easily used in this variety of cases. Through an extensive set of preferences, the extraction algorithm can be highly customized for different uses. These settings are easily editable through the GUI, through method calls that have been exposed through a simple API, or direct manipulation of the settings file on disk. The GUI itself can also easily be easily integrated (as a Swing JPanel for Crunch 1.0 or as a standard widget for Crunch 2.0) into any Java project, or one can customize it directly. The content extraction algorithm is also implemented as an interface for easy incorporation into other programs. The content extractor's broad set of features and customizability allow others to easily add their own version of the algorithm to any product. Further discussion on Crunch as a framework can be found in Section 4.2.

In order to analyze a web page for content extraction, the page is first passed through an HTML parser that creates a DOM tree representation of the web page. We use OpenXML [21] as our HTML parser in Crunch 1.0 and NekoHTML [23] in Crunch 2.0. One of the advantages of using these HTML parsers is that they take care of correcting the HTML. HTML on the Internet can be extremely malformed and most popular browsers like Internet Explorer and Mozilla are able to handle incorrect HTML by making the closest guess to what the HTML should be; however, DOM parsers are highly susceptible as parsing is usually done on a per tag basis. Since the parsers that we chose correct the HTML before forming the DOM tree, we do not have to deal with error resiliency. Once parsed, the resulting DOM document can be seamlessly shown as a webpage to the end-user by flattening the tree and producing back the HTML.

This process accomplishes the steps of structural analysis and structural decomposition analogous to those done by several other techniques (see Section 2). The DOM tree is hierarchically arranged and can be analyzed in sections or as a whole, providing a wide range of flexibility for our extraction algorithm. Just as the approach mentioned by Kaasinen et al. modifies the HTML to restructure the content of the page, our content extractor navigates the DOM tree recursively, using a series of different filtering techniques to remove and adjust specific nodes and leave only the content behind. In our first attempt, Crunch 1.0, we designed a one-pass system that extracted content by running a series on filters one after the other, i.e., the selected filters just ran sequentially on the output produced by the previous filters. This caused problems at times when parts of a webpage that the user wanted were removed. In Crunch 2.0, we amended this by making it a multi-pass system. Here we keep multiple copies of a webpage in memory and a filter checks for the optimal copy to work on. A large number of examples demonstrating the results of different filter settings are shown in Appendix A.

Crunch as a framework handles the webpage, but the filters that are plugged into the framework make it dynamic and customizable. The framework defines a standard API, shown in Appendix B, which a programmer implements when creating a plug-in. The programmer also decides the order in which the filters are run in order to maximize the benefit of each one. An example construction of a Crunch 2.0 plug-in is given in Section 4.2. Each of the filters can be easily turned on and off either by the user, the administrator or the programmer, and can potentially be customized to a certain degree through a GUI if provided by the programmer.

There are two sets of filters that we have implemented, with different levels of granularity, in both Crunch 1.0 and 2.0. The first set of filters simply ignores tags or specific attributes within tags but keep track of them in memory. With these filters, images, links, scripts, styles, and many other elements can be quickly removed from the web page. This process of filtering is similar to Kaasinen's conversion of HTML to WML. However, the second set of filters is more complex and algorithmic, providing a higher level of content extraction. This set, which can be extended, currently consists of the advertisement remover, the link list remover, the removed link retainer and the empty table remover. In Crunch 2.0, we also added filters that allow the user to control the font size and word wrapping of the output, and heuristic functions guiding the multi-pass processor, to evaluate the acceptability of the output as each filter pass edits the DOM tree. This ensures that we don't suffer from some of the pitfalls of version 1.0 where occasionally pages returned null outputs after passing through Crunch, e.g., link heavy pages like www.msn.com, as shown later in Figures 11 and 12. Finally, in the newer version, we have attempted to allow for greater control on most of the filters by adding supplementary options. For example, users now have the ability of controlling, at a finer granularity, complex web pages where certain HTML structures are embedded within others, e.g., within table cells.

The advertisement remover uses a common and efficient technique to remove advertisements. As the DOM tree is parsed, the values of the "src" and "href" attributes throughout the page are surveyed to determine the servers to which the links refer. If an address matches against a list of common advertisement servers, the node of the DOM tree that contained the link is removed. This process is similar to the use of an operating systems-level "hosts" file to prevent a computer from connecting to advertiser hosts. Hanzlik [6] examines this technique and cites a list of hosts, which we use for our advertisement remover. In order to avoid the common pitfall of deploying a fixed blacklist of advertisers, our software also periodically updates the list from http://accs-net.com, a site that specializes in creating such blacklists. This is a technique employed by most ad blocking software.

The link list remover employs a filtering technique that removes all "link lists", which are bodies of content either in the page or within table cells for which the ratio of the number of links to the number of non-linked words is greater than a specific threshold (known as the link/text removal ratio). When the DOM parser encounters a table cell, the Link List Remover tallies the number of links and non-linked words. The number of non-linked words is determined by taking the number of letters not contained in a link and dividing it by the average number of characters per word, which we preset as 5 (although it may be overridden by the user and could, in principle, be derived from the specific web page or web domain). If the ratio is greater than the user-determined link/text removal ratio (default ratio is set to 0.35), the content of the table cell (and, optionally, the cell itself) is removed. This algorithm succeeds in removing most long link lists that tend to reside along the sides of web pages while leaving the text-intensive portions of the page intact.

After these steps, we have found that numerous tables that are either completely empty or have several empty cells take up large swaths of space remain on the webpage. The empty table remover removes tables that are empty of any “substantive” information. The user determines, through settings, which HTML tags should be considered to be substance and how many characters within a table are needed to be viewed as substantive, set much like the word size or link-to-text ratio settings set earlier. This does not require much prior knowledge of HTML since the syntax of the markup language is simple and matches words from the English language closely, e.g., table, form, etc. The table remover checks a table for substance after it has been parsed through the filter. If a table has either no substance or less than some user defined threshold, it is removed from the tree. This algorithm effectively removes any tables left over from previous filters that contain small amounts of unimportant information. This filter is typically run towards the end to maximize its benefit.

While the above filters remove non-content from the page, the removed link retainer adds link information back at the end of the document to keep the page browsable. The removed link retainer keeps track of all the text links that are removed throughout the filtering process. After the DOM tree is completely parsed, the list of removed links is added to the bottom of the page. In this way, any important navigational links that were previously removed remain accessible, and since the parser had parsed them initially as separate units, each menu or navigational link is kept intact and they can all be viewed without any loss of original setup or style.

After the entire page is parsed and modified appropriately, it can be output in either HTML or as plain text (filters could be added to translate to another output format such as WML). The plain text output removes all the tags and retains only the text of the site, while eliminating most white space. The result is a text document that contains the main content of the page in a format suitable for summarization, speech rendering or storage. This technique is significantly different from Rahman et al. [2], which states that a decomposed webpage should be *analyzed* using NLP techniques to find the content. It is true that NLP techniques may produce better results, but at the cost of far more time consuming processing. Our algorithm doesn’t technically find the content but instead eliminates likely non-content. In this manner, we can still process and return results for sites that don’t have an explicit “main body”.

Crunch, however, does have some limitations:

1) Crunch cannot filter non-HTML content like Flash. It allows a boolean choice of whether to keep or remove such structures but it can't help edit or filter within the animation itself.

2) Dynamically generated pages often aren't filtered so nicely for the same reason as above. The script, whether it be javascript, ASP or JSP is either left completely disabled, causing dynamic pages to not load correctly, or left on which leaves all respective scripts active on the page.

3) Crunch does not distinguish between different users. There is only one set of options, whether an individual is using the proxy or whether it is set up as groupware.

4) There are no artificially intelligent heuristics or machine learning algorithms implemented yet, e.g., to learn a user's browsing patterns and change user (or group) settings dynamically.

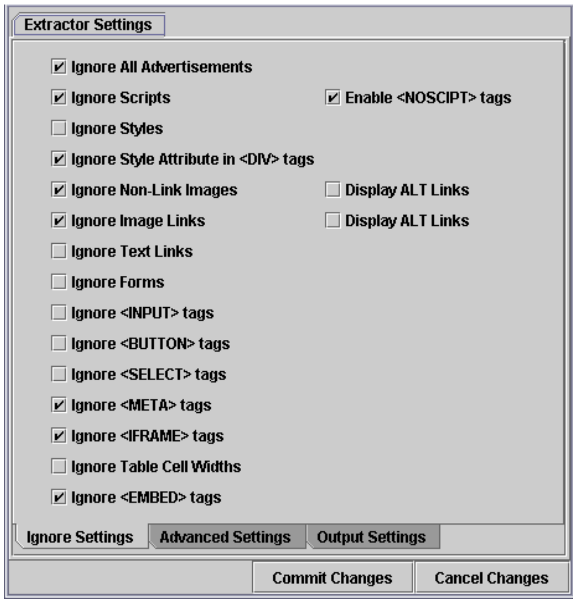
4. Implementation

4.1 CRUNCH 1.0

4.1.1. *Overview*

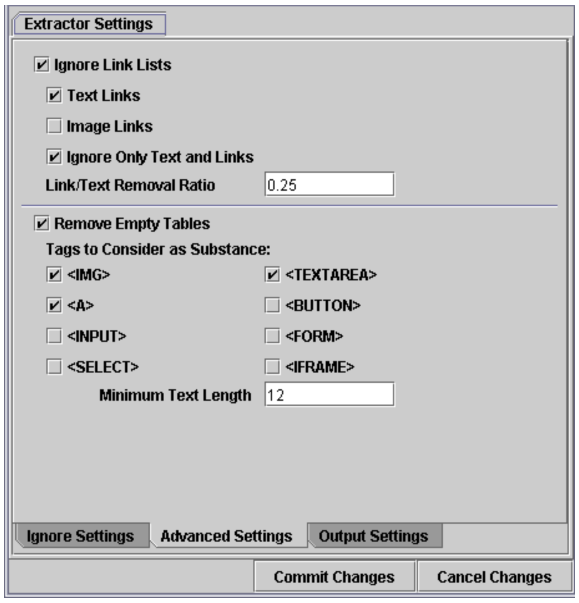
In order to make our extractor easy to use, we implemented it as a web proxy (program and instructions are accessible at http://www.psl.cs.columbia.edu/proxy). The proxy can be used as a personal filter by individual users as well as a central system for groups of people. In the case where Crunch is set up as groupware, users can access the proxy by simply setting their browser to do so, as most modern browsers can now point to external proxies for filtering content. This allows an administrator to set up the extractor and provide content extraction services for a group. The proxy is coupled with a graphical user interface (GUI) to customize its behavior. The separate screens of the GUI are shown in Figures 1-3. Figure 1 shows the very broad options that can be turned off or on that ignore certain tags completely. Figure 2 has more advanced options that give more granular control, while Figure 3 show controls on output. The current implementation of the proxy is in Java for cross-platform support, and has been successfully tested on Windows, MacOS, Linux and Solaris.

The Content Extraction framework itself has a complexity of O(N + P), where N is the number of nodes in the DOM tree after the HTML page is parsed and P is the sum of the complexities of the plug-ins; therefore the overall complexity is O(N) without plug-ins. Crunch 1.0 is implemented as a one-pass system, so it is the plug-ins that truly determine the running time of the system. For example, the plug-in that edits tables has an algorithm whose worst case running time is $O(M^2)$ for complex nested tables; without such nesting, the typical running time is O(M), where M is the number of elements composing the table; so the overall running time of the system works out to be $O(N + M^2)$ with the table plug-in. During tests, the algorithm performs quickly and efficiently following proxy customization. The proxy can handle most web pages, including those with badly formatted HTML, because of the corrections automatically applied while the page is parsed into a DOM tree. However, sites that are extremely link heavy produce bad results; when the link to text ratio approaches 100%, we experienced anomalous behavior.



☑ Ignore All Advertisements
☑ Ignore Scripts ☑ Enable <NOSCIPT> tags
☐ Ignore Styles
☑ Ignore Style Attribute in <DIV> tags
☑ Ignore Non-Link Images ☐ Display ALT Links
☑ Ignore Image Links ☐ Display ALT Links
☐ Ignore Text Links
☐ Ignore Forms
☐ Ignore <INPUT> tags
☐ Ignore <BUTTON> tags
☐ Ignore <SELECT> tags
☑ Ignore <META> tags
☑ Ignore <IFRAME> tags
☐ Ignore Table Cell Widths
☑ Ignore <EMBED> tags

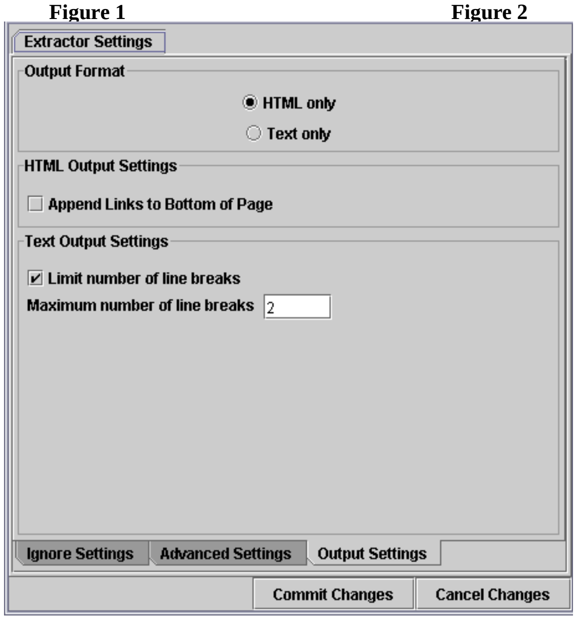
Figure 1



☑ Ignore Link Lists
☑ Text Links
☐ Image Links
☑ Ignore Only Text and Links
Link/Text Removal Ratio 0.25

☑ Remove Empty Tables
Tags to Consider as Substance:
☑ <IMG> ☑ <TEXTAREA>
☑ <A> ☐ <BUTTON>
☐ <INPUT> ☐ <FORM>
☐ <SELECT> ☐ <IFRAME>
Minimum Text Length 12

Figure 2



Output Format
● HTML only
○ Text only

HTML Output Settings
☐ Append Links to Bottom of Page

Text Output Settings
☑ Limit number of line breaks
Maximum number of line breaks 2

Figure 3

Depending on the type and complexity of the web page, the content extraction suite can produce a wide variety of output. The algorithm performs well on pages with large blocks of text such as news articles and mid-size to long informational passages. Most navigational bars and extraneous elements of web pages such as advertisements and side panels are removed or reduced in size. Figures 4 and 5 show an example.

When printed out in text format, most of the resulting text is directly related to the content of the page, making it possible to use summarization and keyword extraction algorithms efficiently and accurately. An example of text format extraction performed on the webpage in Figure 5 is shown in Figure 6.

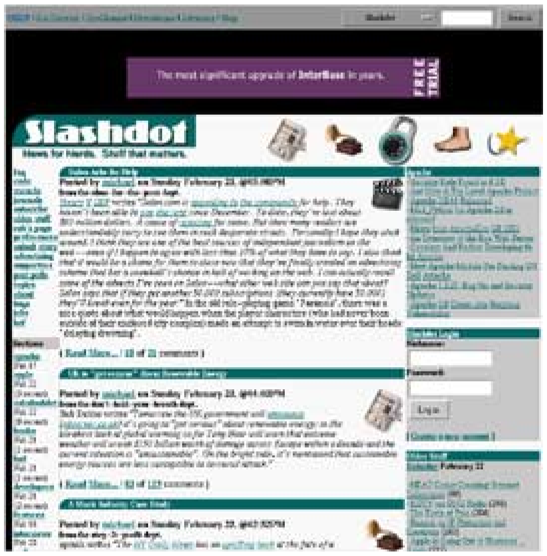


Figure 4 – Before

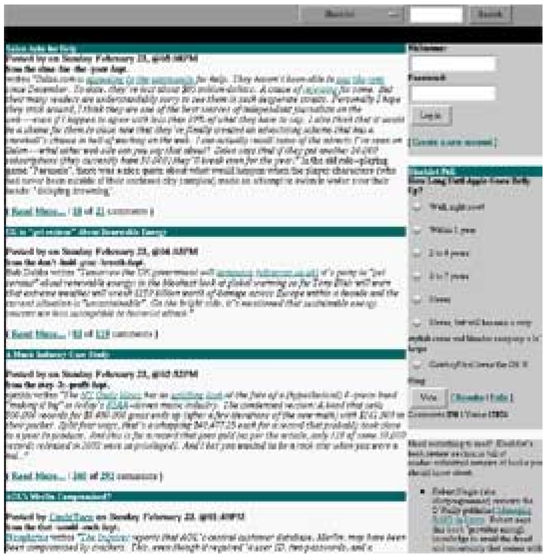
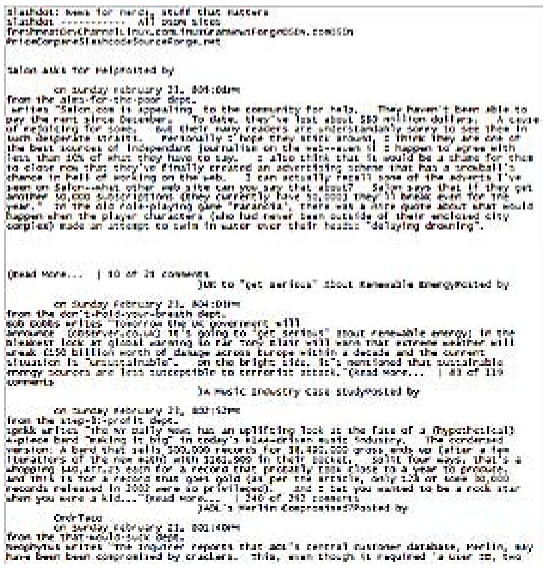


Figure 5 - After



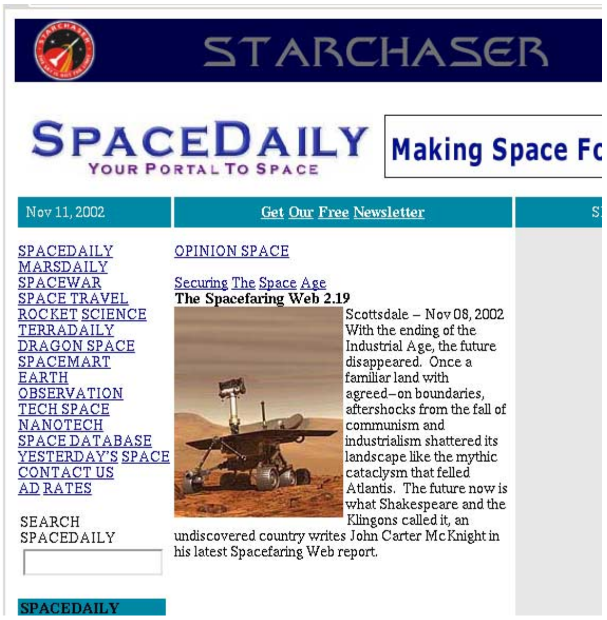
```
Slashdot: News for nerds, stuff that matters
Slashdot ----------- All OSDN sites
FreshmeatDevChannelLinux.comLinuxGramNewsForgeOSDN.comSSIn
PriceCompareSlashcodeSourceForge.net

Salon Asks for HelpPosted by

        on Sunday February 23, @04:08PM
from the alms-for-the-poor dept.
 writes "Salon.com is appealing  to the community for help.   They haven't been able to
pay the rent since December.   To date, they've lost about $80 million dollars.   A cause
of rejoicing for some.   But their many readers are understandably sorry to see them in
such desperate straits.   Personally I hope they stick around, I think they are one of
the best sources of independant journalism on the net--even if I happen to agree with
less than 10% of what they have to say.   I also think that it would be a shame for them
to close now that they've finally created an advertising scheme that has a snowball's
chance in hell of working on the web.   I can actually recall some of the adverts I've
seen on Salon--what other web site can you say that about?   Salon says that if they get
another 50,000 subscriptions (they currently have 50,000) they'll break even for the
year."  In the old role-playing game "Paranoia", there was a nice quote about what would
happen when the player characters (who had never been outside of their enclosed city
complex) made an attempt to swim in water over their heads: "delaying drowning".



(Read More...  | 10 of 21 comments
                                  )UK to "get serious" about Renewable EnergyPosted by

        on Sunday February 23, @04:01PM
from the don't-hold-your-breath dept.
Bob Dobbs writes "Tomorrow the UK government will
announce  (observer.co.uk) it's going to "get serious" about renewable energy; in the
bleakest look at global warming so far Tony Blair will warn that extreme weather will
wreak £150 billion worth of damage across Europe within a decade and the current
situation is "unsustainable".   On the bright side, it's mentioned that sustainable
energy sources are less susceptible to terrorist attack."(Read More...  | 83 of 119
comments
                                  )A Music Industry Case StudyPosted by

        on Sunday February 23, @02:52PM
from the step-3:-profit dept.
spekk writes "The NY Daily News has an uplifting look at the fate of a (hypothetical)
4-piece band "making it big" in today's label-driven music industry.   The condensed
version: A band that sells 500,000 records for $8,492,000 gross ends up (after a few
iterations of the new math) with $161,909 in their pocket.   Split four ways, that's a
whopping $40,477.25 each for a record that probably took close to a year to produce,
and this is for a record that goes gold (as per the article, only 11% or some 30,000
records released in 2002 were so privileged).   And I bet you wanted to be a rock star
when you were a kid..."(Read More...  | 240 of 292 comments
                                  )AOL's Merlin Compromised?Posted by
        CmdrTaco
        on Sunday February 23, @01:40PM
from the that-would-suck dept.
Neophytus writes "The Inquirer reports that AOL's central customer database, Merlin, may
have been been compromised by crackers.  This, even though it required "a user ID, two
```

Figure 6 – Text Only

The initial implementation of the proxy was designed for simplicity in order to test and design content extraction algorithms. It spawns a new thread to handle each new connection, limiting its scalability. Most of the performance drop from using the proxy originates from the proxy's need to download the entire page before sending it to the client.

4.1.2. *Illustrative examples*

Figures 7 and 8 show an example of a typical page from www.spacer.com and a filtered version of that page, respectively. This is another good example of a site that is presented in a content-rich format. On the other hand, Figures 9 and 10 show the front page of www.planetunreal.com, a site dedicated to the Unreal Tournament 2003 first-person shooter game (www.epicgames.com), before and after content extraction. Despite producing results that are rich in text, screenshots of the game are also removed, which the user might deem relevant content.



Nov 11, 2002

Get Our Free Newsletter

SPACEDAILY
MARSDAILY
SPACEWAR
SPACE TRAVEL
ROCKET SCIENCE
TERRADAILY
DRAGON SPACE
SPACEMART
EARTH OBSERVATION
TECH SPACE
NANOTECH
SPACE DATABASE
YESTERDAY'S SPACE
CONTACT US
AD RATES

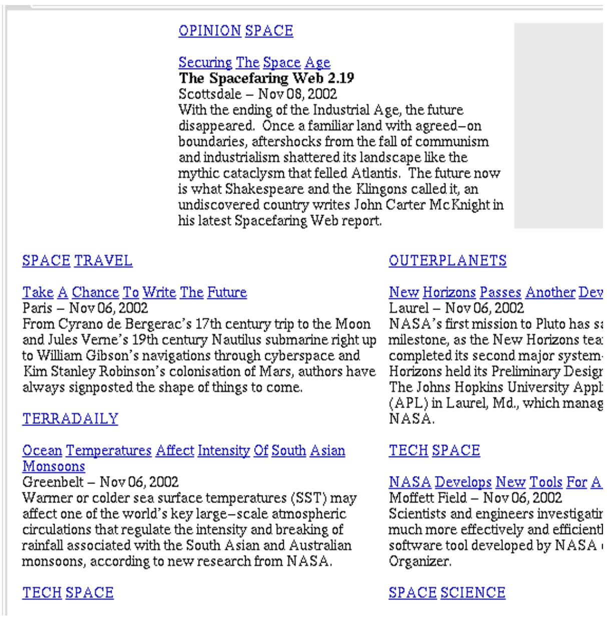
SEARCH SPACEDAILY

SPACEDAILY

OPINION SPACE

Securing The Space Age
The Spacefaring Web 2.19
Scottsdale – Nov 08, 2002
With the ending of the Industrial Age, the future disappeared. Once a familiar land with agreed–on boundaries, aftershocks from the fall of communism and industrialism shattered its landscape like the mythic cataclysm that felled Atlantis. The future now is what Shakespeare and the Klingons called it, an undiscovered country writes John Carter McKnight in his latest Spacefaring Web report.

Figure 7 – Before


OPINION SPACE

Securing The Space Age
The Spacefaring Web 2.19
Scottsdale – Nov 08, 2002
With the ending of the Industrial Age, the future disappeared. Once a familiar land with agreed–on boundaries, aftershocks from the fall of communism and industrialism shattered its landscape like the mythic cataclysm that felled Atlantis. The future now is what Shakespeare and the Klingons called it, an undiscovered country writes John Carter McKnight in his latest Spacefaring Web report.

SPACE TRAVEL

Take A Chance To Write The Future
Paris – Nov 06, 2002
From Cyrano de Bergerac's 17th century trip to the Moon and Jules Verne's 19th century Nautilus submarine right up to William Gibson's nerconauts through cyberspace and Kim Stanley Robinson's colonisation of Mars, authors have always signposted the shape of things to come.

TERRADAILY

Ocean Temperatures Affect Intensity Of South Asian Monsoons
Greenbelt – Nov 06, 2002
Warmer or colder sea surface temperatures (SST) may affect one of the world's key large–scale atmospheric circulations that regulate the intensity and breaking of rainfall associated with the South Asian and Australian monsoons, according to new research from NASA.

TECH SPACE

OUTERPLANETS

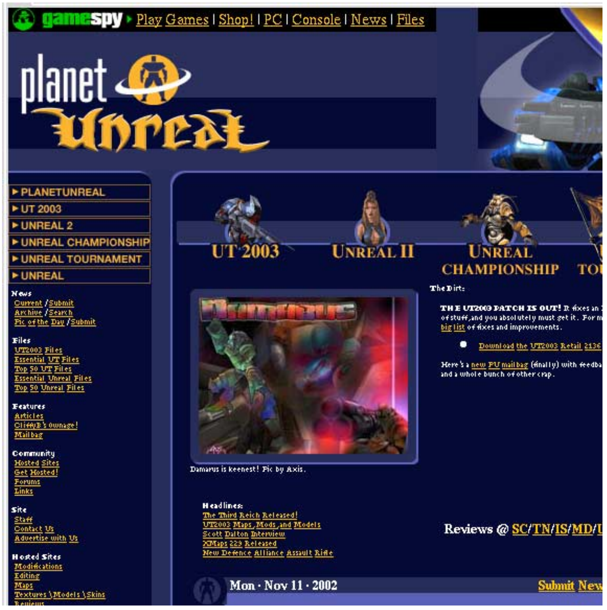
New Horizons Passes Another Dev
Laurel – Nov 06, 2002
NASA's first mission to Pluto has s
milestone, as the New Horizons tea
completed its second major system-
Horizons held its Preliminary Desig
The Johns Hopkins University Appl
(APL) in Laurel, Md., which manag
NASA.

TECH SPACE

NASA Develops New Tools For A
Moffett Field – Nov 06, 2002
Scientists and engineers investigatin
much more effectively and efficientl
software tool developed by NASA
Organizer.

SPACE SCIENCE

Figure 8 - After



▶ PLANETUNREAL
▶ UT 2003
▶ UNREAL 2
▶ UNREAL CHAMPIONSHIP
▶ UNREAL TOURNAMENT
▶ UNREAL

News
Current / Submit
Archive / Search
Pic of the Day / Submit

Files
UT2003 Files
Essential UT Files
Top 50 UT Files
Essential Unreal Files
Top 50 Unreal Files

Features
Articles
CliffyB's ownage!
Mailbag

Community
Hosted Sites
Get Hosted!
Forums
Links

Site
Staff
Contact Us
Advertise with Us

Hosted Sites
Modifications
Editing
Maps
Textures \ Models \ Skins
Reviews

The Dirt:

THE UT2003 PATCH IS OUT! It fixes an
of stuff, and you absolutely must get it. For m
big list of fixes and improvements.

- Download the UT2003 Retail 2136

Here's a new PU mailbag (finally) with feedba
and a whole bunch of other crap.

Damarus is keenest! Pic by Axis.

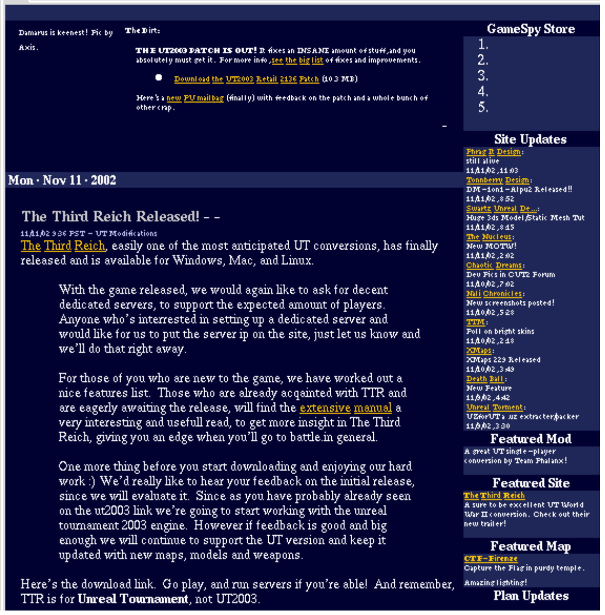
Headlines:
The Third Reich Released!
UT2003 Maps, Mods, and Models
Scott Dalton Interview
XMaps 229 Released
New Defence Alliance Assault Rifle

Reviews @ SC/TN/IS/MD/U

Mon · Nov 11 · 2002

Submit New

Figure 9 – Before


Damarus is keenest! Pic by Axis.

The Dirt:

THE UT2003 PATCH IS OUT! It fixes an INSANE amount of stuff, and you absolutely must get it. For more info, see the big list of fixes and improvements.

- Download the UT2003 Retail 2136 Patch (10.3 MB)

Here's a new PU mailbag (finally) with feedback on the patch and a whole bunch of other crap.

Mon · Nov 11 · 2002

The Third Reich Released! - -
11/11/02 9:36 PST – UT Modifications
The Third Reich, easily one of the most anticipated UT conversions, has finally released and is available for Windows, Mac, and Linux.

With the game released, we would again like to ask for decent dedicated servers, to support the expected amount of players. Anyone who's interrested in setting up a dedicated server and would like for us to put the server ip on the site, just let us know and we'll do that right away.

For those of you who are new to the game, we have worked out a nice features list. Those who are already acqainted with TTR and are eagerly awaiting the release, will find the extensive manual a very interesting and usefull read, to get more insight in The Third Reich, giving you an edge when you'll go to battle in general.

One more thing before you start downloading and enjoying our hard work :) We'd really like to hear your feedback on the initial release, since we will evaluate it. Since as you have probably already seen on the ut2003 link we're going to start working with the unreal tournament 2003 engine. However if feedback is good and big enough we will continue to support the UT version and keep it updated with new maps, models and weapons.

Here's the download link. Go play, and run servers if you're able! And remember, TTR is for **Unreal Tournament**, not UT2003.

GameSpy Store
1.
2.
3.
4.
5.

Site Updates
Phrag It Design:
still alive
11/11/02, 11:03
Tonnberry Design:
DM-1on1-Alpu2 Released!!
11/11/02, 8:52
Swartz Unreal De...:
Huge 3ds Model/Static Mesh Tut
11/11/02, 8:15
The Nucleus:
New MOTW!
11/11/02, 2:02
Chaotic Dreams:
Dev Pics in CUT2 Forum
11/10/02, 7:02
Nali Chronicles:
New screenshots posted!
11/10/02, 5:28
TTM:
Poll on bright skins
11/10/02, 2:18
XMaps:
XMaps 229 Released
11/10/02, 3:49
Death Ball:
New Feature
11/9/02, 4:42
Unreal Torment:
UZ4orUT: a .uz extracter/packer
11/9/02, 3:30

Featured Mod
A great UT single-player conversion by Team Phalanx!

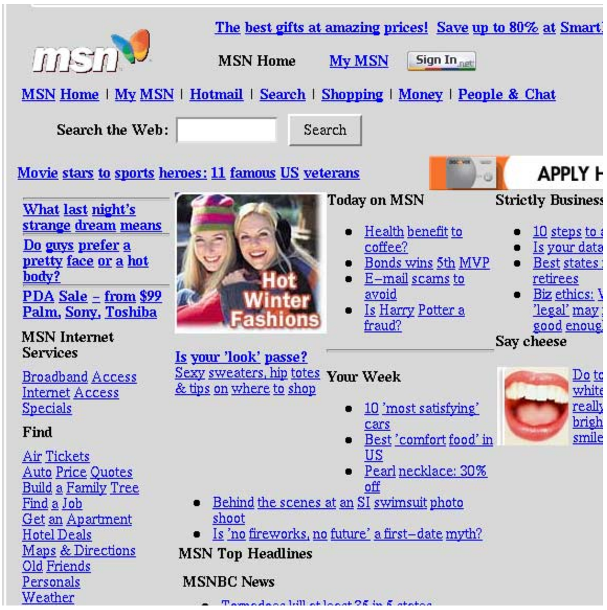
Featured Site
The Third Reich
A sure to be excellent UT World War II conversion. Check out their new trailer!

Featured Map
CTF-Firenze
Capture the Flag in purdy temple.
Amazing lighting!

Plan Updates

Figure 10 - After

Figures 11 and 12 show www.msn.com in its pre- and post-filtered state. Since the site is a portal which contains links and little else, the proxy does not find any coherent content to keep. We investigated heuristics that would leave such pages either untouched, or alternatively employ only the most basic filters that only remove advertisements and banners, and implemented such techniques in Crunch 2.0 (see Section 4.2).


The best gifts at amazing prices! Save up to 80% at Smart

MSN Home My MSN Sign In

MSN Home | My MSN | Hotmail | Search | Shopping | Money | People & Chat

Search the Web: Search

Movie stars to sports heroes: 11 famous US veterans

APPLY

What last night's strange dream means
Do guys prefer a pretty face or a hot body?
PDA Sale – from $99 Palm, Sony, Toshiba

MSN Internet Services
Broadband Access
Internet Access
Specials

Find
Air Tickets
Auto Price Quotes
Build a Family Tree
Find a Job
Get an Apartment
Hotel Deals
Maps & Directions
Old Friends
Personals
Weather

Hot Winter Fashions

Is your 'look' passe?
Sexy sweaters, hip totes & tips on where to shop

Today on MSN
- Health benefit to coffee?
- Bonds wins 5th MVP
- E–mail scams to avoid
- Is Harry Potter a fraud?

Strictly Business
- 10 steps to
- Is your data
- Best states retirees
- Biz ethics: 'legal' may good enoug

Say cheese
Do to white really brigh smile

Your Week
- 10 'most satisfying' cars
- Best 'comfort food' in US
- Pearl necklace: 30% off

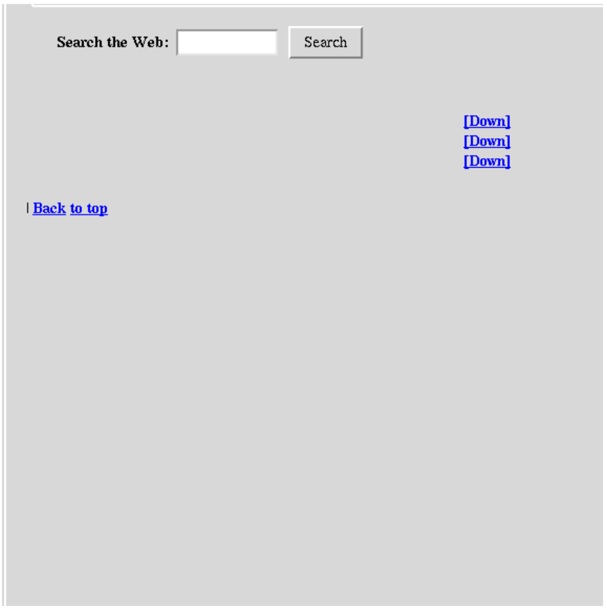
- Behind the scenes at an SI swimsuit photo shoot
- Is 'no fireworks, no future' a first–date myth?

MSN Top Headlines

MSNBC News

- Tornadoes kill at least 35 in 5 states

Figure 11 – Before


Search the Web: Search

[Down]
[Down]
[Down]

| Back to top

Figure 12 - After

From these examples one may get the impression that input fields are affected irregularly by our proxy; this is because the run-time decision of leaving them in or removing them from the page is dependent on the tables or frames they are contained in. Forms are handled as one semantic unit, where either a form is displayed or not based on the user setting. Additionally, we should mention that there isn't any sort of preservation of objects that may be lost after the HTML is passed through our parser, except links can be retained as explained above. The user would have to change the settings of the proxy and reload the page to see the previously removed content. However, a different set of filters could be developed to move rather than just remove content, for forms or other identifiable HTML elements or data.

4.1.3. *Implementation details*

The life cycle of the process that gets a page to the client's browser through the proxy from a very high level is - the client passes a request for the webpage to the proxy which opens a socket, fetches the original content of the page, and parses the page to create a DOM tree representation. It is then passed through the different filters based on the settings set by the user. The edited DOM tree is then either flattened into the HTML form, and sent back to the client's browser, or stripped of all HTML tags and only the text content is sent to the client for rendering. The architectural diagram of Crunch 1.0 is shown in Figure 13.

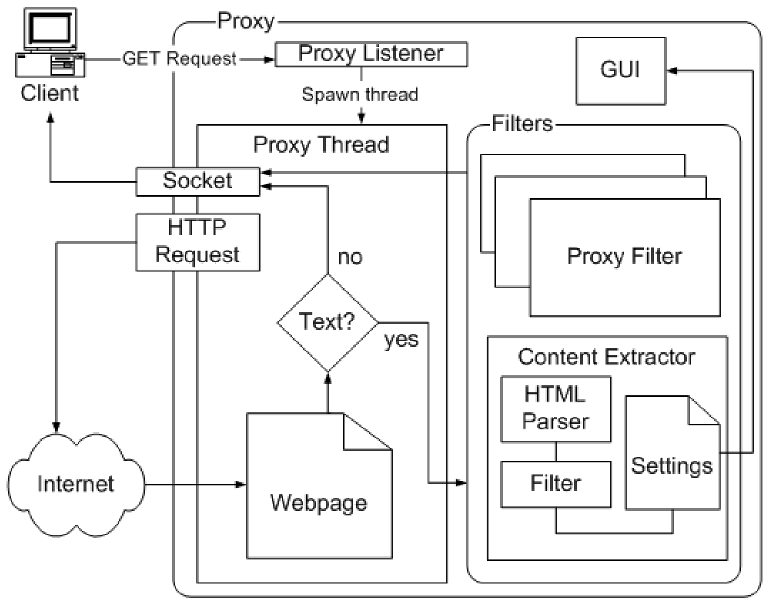


Figure 13. Architectural diagram of the system

In more detail, in order to analyze a web page for content extraction, the page is passed through an HTML parser that creates a Document Object Model tree. The algorithm begins by starting at the root node of the DOM tree (the <HTML> tag), and proceeds by parsing through its children using a recursive depth first search function called *filterNode()*. The function initializes a Boolean variable (*mCheckChildren*) to true to allow *filterNode()* to check the children. The currently selected node is then passed through a filter method called *passThroughFilters()* that analyzes and modifies the node based on a series of user-selected preferences. At any time within *passThroughFilters()*, the *mCheckChildren* variable can be set to false, which allows the individual filter to prevent specific subtrees from being filtered. That is, certain filters can elect to produce the final result at a given node and not allow any other filters to edit the content after that. After the node is filtered accordingly, *filterNode()* is recursively called using the children if the *mCheckChildren* variable is still true.

The filtering method, *passThroughFilters()*, performs the majority of the content extraction. It begins by examining the node it is passed to see if it is a “text node” (data) or an “element node” (HTML tag). Element nodes are examined and modified in a series of passes. First, any filters that edit an element node but do not delete it are applied. For example, the user can enable a preference that will remove all table cell widths, and it would be applied in the first phase because it modifies the attributes of table cell nodes without deleting them.

The second phase in examining element nodes is to apply all filters that delete nodes from the DOM tree. Most of these filters prevent the *filterNode()* method from recursively checking the children by setting *mCheckChildren* to false. A few of the filters in this subset set *mCheckChildren* to true so as to continue with a modified version of the original *filterNode()* method. For example, the empty table remover filter sets *mCheckChildren* to false so that it can itself recursively search through the <TABLE> tag using a bottom-up depth first search while *filterNode()* uses a top-down depth first search. Finally, if the node is a text node, any text filters are applied (there are currently none, but there may be in the future).

4.2 CRUNCH 2.0

4.2.1. Overview

Crunch 1.0 nicely demonstrated the proof-of-concept design of the system as a framework, but certain problems needed to be addressed in order for Crunch to be widely used. Most notably, the HTML parser we were using, OpenXML, had serious performance problems and memory leaks, and was no longer under active development. We also wanted to move to a staged event architecture and asynchronous callbacks to avoid threading scalability issues. There were also unacceptable results when Crunch's initial set of filters were applied to certain classes of websites, as exemplified by the link-heavy MSN example above producing essentially a blank page. Further, the user interface for Crunch was not very friendly for administrators and probably inaccessible to typical end users.

After releasing Crunch 1.0 in September 2002, we also received several suggestions from early users for additions and improvements. An informal user study of blindfolded students was conducted in May 2003 followed by a formal user study with blind and visually impaired users begun in December 2003; the first results from the latter are discussed in Section 5. The NLP group at Columbia University tried using Crunch briefly for their Newsblaster [8][9] project, which is a system that automatically tracks, clusters and summarizes each day's news programmatically. They used Crunch as their input mechanism in order to run their natural language processing algorithms on content extracted by Crunch rather than noisy data streams coming straight from the web.

As indicated in section 3, Crunch 2.0 is similar to its predecessor. However, we spent time on improving its performance and user interface, and several changes were made in the supplied set of heuristic filters, e.g., to show more useful results for link-heavy pages. We optimized the content extractor filter even though function is inherently the same. Additional filters were added that allow the user to control the font size and word wrapping of the output. Perhaps most importantly, heuristic functions were added in the form of a multi-pass system that evaluates the output the DOM tree passed through each filter. This prevents link-heavy pages like www.msn.com from returning blank pages as output, as shown in Figures 11 and 12; with the new result-checking heuristics of Crunch 2.0, we instead got the better results shown in Figures 14 and 15 for a link-heavy page.

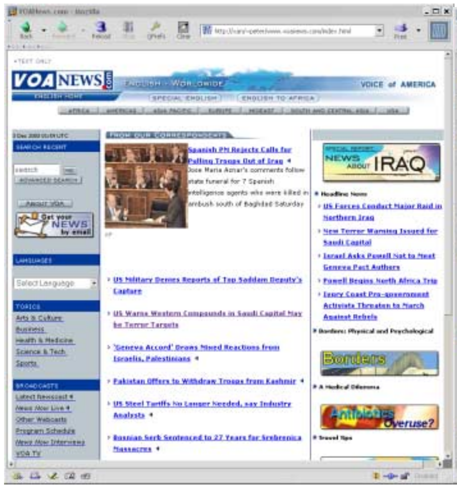


Figure 14 - Example of a link-heavy page

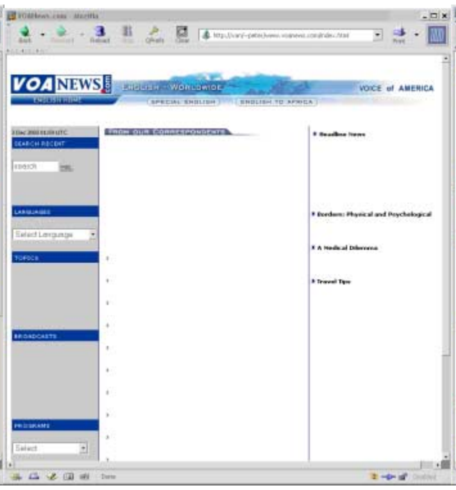


Figure 15 - Output from Crunch 1.0

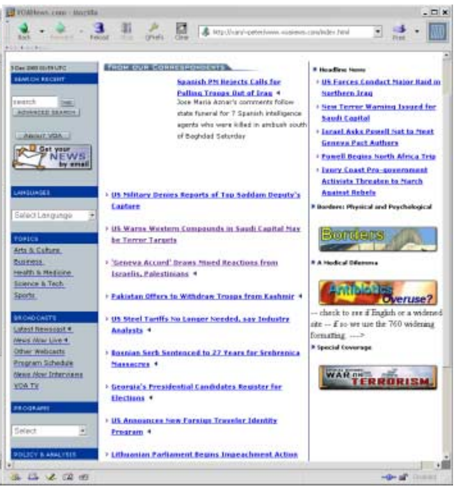


Figure 16 - Output from Crunch 2.0

Finally, in the newer version, we have attempted to allow for greater flexibility to most of the filters by adding supplementary options to each. For example, users now have the ability of controlling, at a finer granularity, complex web pages where certain HTML structures are embedded within others, like having the ability to control not only the content on the entire page but also within table cells. The filters in Crunch 2.0, we find, also work better because of the improvements in parsing, mainly due to the replacement in our HTML parser from OpenXML [21] to NekoHTML [23]. Appendix A show several suites of screenshots with different sets of Crunch 2.0 filters applied.

Like Crunch 1.0, the complexity of the newer version remains at O(N+P); however, the worst case running time increases to O(N*P), where N is the number of nodes in the DOM tree after the HTML page is parsed and P is the complexity of the plug-ins with highest running time. The increase in worst case complexity is due to the fact that we have switched to a multi-pass system. Therefore, in case of a bad result, a filtered webpage may have to revert to a previous state and re-run through the proxy with a different set of options; this may happen for any number of nodes in the DOM tree.

4.2.2. Technical improvements in version 2

Even though the basic architecture of the system is the same as shown in Figure 13, there are some notable changes. Besides those discussed below, Appendix C shows some important differences in the internal coding of the proxy and plug-ins.

1) Replaced OpenXML with NekoHTML

The original version of Crunch used OpenXML [21] as the HTML parser. OpenXML had problems with efficiency, which didn't seem likely to be fixed since OpenXML is apparently no longer an active project. So we switched the HTML parser to NekoHTML [23]. NekoHTML is an HTML scanner and tag balancer that parses HTML for Xerces, an XML implementation that is part of the Apache project [51]. It has many benefits, most notably the increased speed, but a key longer-term benefit is that we are now using a parser that is under active development. NekoHTML currently has some problems parsing some pages, most notably the output not always rendering the same as the input, e.g., with certain complex nested tables and some CSS pages. However, most of these errors are minor cosmetic ones that our proxy attempts, usually successfully, to fix in the multi-pass scheme. Additionally, the developers of NekoHTML are apparently working on this problem.

2) Performance tuning

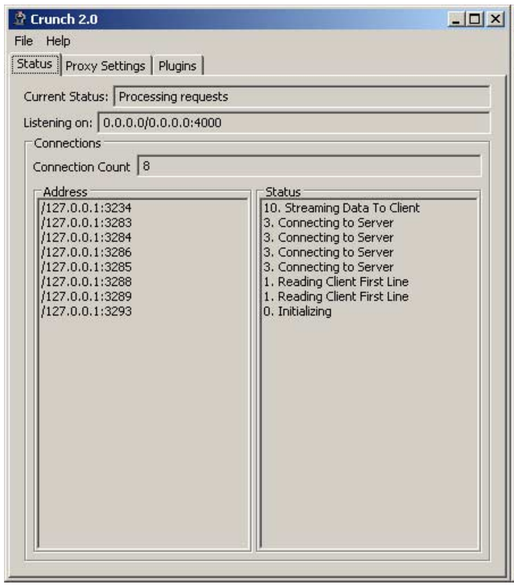
As mentioned above, a speed improvement was achieved through switching to NekoHTML. The other major contributor to increased speed was the optimization of Crunch's networking code. The code was originally written using the Java IO package. Switching to the Java NIO package was considered and we wrote a small testbed, but ran into excessive complications using NIO's asynchronous callbacks. Therefore, we instead optimized the Java IO code by collapsing multiple writes and reads into fewer writes and reads, dealing with timeouts more efficiently, and removing unnecessary or redundant calls in the transfer loops. Server performance and bandwidth utilization now seems adequate, but we have not yet conducted a performance study with large loads.

We moved to the staged event architecture and asynchronous callbacks to avoid threading scalability issues. The concept of the staged event architecture was introduced formally by Welsh [55] for performance gains in highly concurrent server applications, so that they are able to "support massive concurrency demands" [55]. We took the same concept and extended it in our framework so that Crunch can meet the demands of several parallel requests in a groupware setup.

3) Switch to SWT

The Java Swing GUI was replaced with SWT, IBM's Standard Widget Toolkit [24]. This was done to enhance the user interface for ease of administration. SWT is highly responsive, partially due to its use of JNI and native routine calls that can take advantage of the operating system's built-in optimizations. It also uses native GUI widgets to provide a look and feel consistent with that of the operating system, while remaining operating system independent. Not only does the GUI match the operating system's theme, SWT generally looks much better than Java's AWT and Swing. As an added benefit, SWT allows the program to be compiled into a binary executable, resulting in a faster startup time, a smaller distribution, less memory used, and an easier installation for novice users.

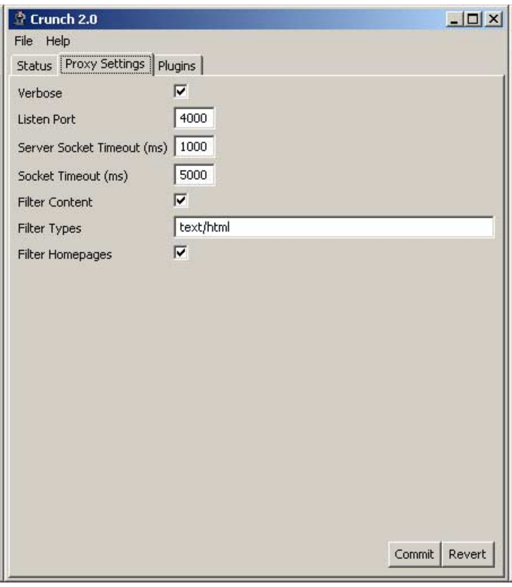
The resulting user interface was redesigned to be more responsive, perform faster and generally be more user friendly, whether for the administrator or for the end user. Screenshots of the proxy are shown in Figures 17-22, where we see the basic settings and the available plug-ins in Figures 17-19 and the actual plug-in setting controls in Figures 20-22.



Current Status: Processing requests
Listening on: 0.0.0.0/0.0.0.0:4000
Connections
Connection Count 8

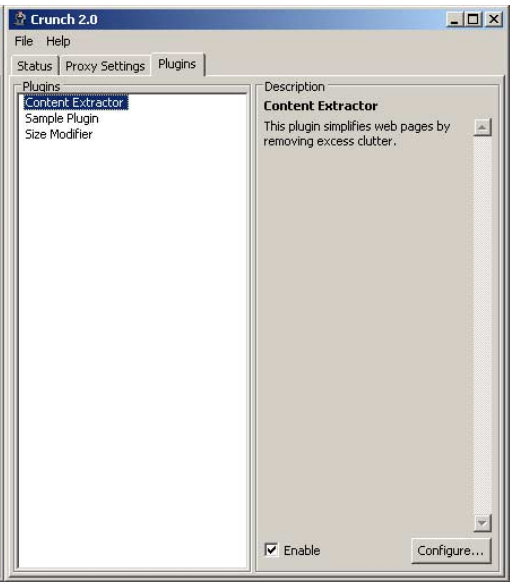
Address	Status
/127.0.0.1:3234	10. Streaming Data To Client
/127.0.0.1:3283	3. Connecting to Server
/127.0.0.1:3284	3. Connecting to Server
/127.0.0.1:3286	3. Connecting to Server
/127.0.0.1:3285	3. Connecting to Server
/127.0.0.1:3288	1. Reading Client First Line
/127.0.0.1:3289	1. Reading Client First Line
/127.0.0.1:3293	0. Initializing

Figure 147



Verbose ☑
Listen Port 4000
Server Socket Timeout (ms) 1000
Socket Timeout (ms) 5000
Filter Content ☑
Filter Types text/html
Filter Homepages ☑

Figure 18



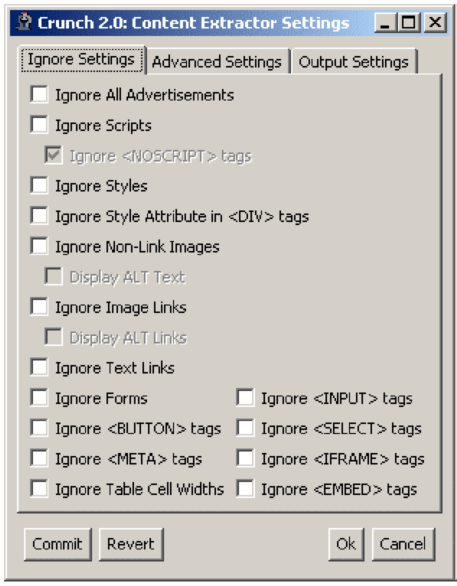
Plugins
Content Extractor
Sample Plugin
Size Modifier

Description
Content Extractor
This plugin simplifies web pages by removing excess clutter.

☑ Enable

Figure 19

The User Interface for Crunch 2.0: The three basic tabs viewing activity, changing basic settings, and viewing the plug-ins available



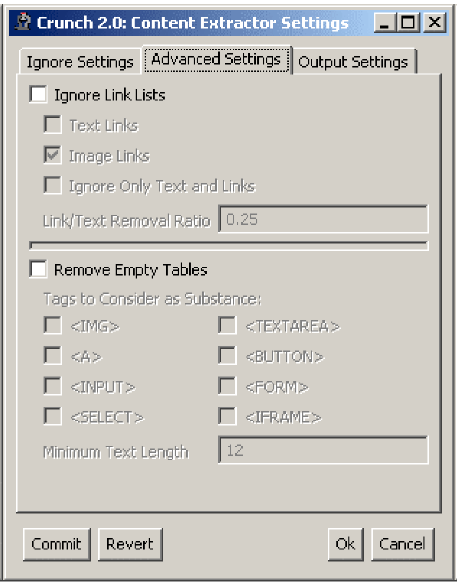
Crunch 2.0: Content Extractor Settings

Ignore Settings | Advanced Settings | Output Settings

- ☐ Ignore All Advertisements
- ☐ Ignore Scripts
  - ☑ Ignore <NOSCRIPT> tags
- ☐ Ignore Styles
- ☐ Ignore Style Attribute in <DIV> tags
- ☐ Ignore Non-Link Images
  - ☐ Display ALT Text
- ☐ Ignore Image Links
  - ☐ Display ALT Links
- ☐ Ignore Text Links
- ☐ Ignore Forms
- ☐ Ignore <INPUT> tags
- ☐ Ignore <BUTTON> tags
- ☐ Ignore <SELECT> tags
- ☐ Ignore <META> tags
- ☐ Ignore <IFRAME> tags
- ☐ Ignore Table Cell Widths
- ☐ Ignore <EMBED> tags

Commit | Revert | Ok | Cancel

Figure 20



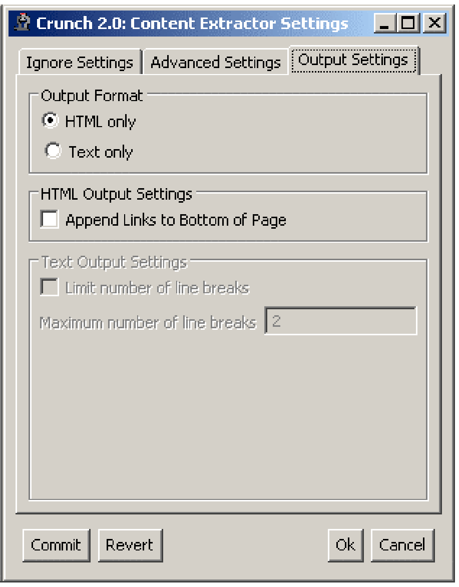
Crunch 2.0: Content Extractor Settings

Ignore Settings | Advanced Settings | Output Settings

- ☐ Ignore Link Lists
  - ☐ Text Links
  - ☑ Image Links
  - ☐ Ignore Only Text and Links
- Link/Text Removal Ratio: 0.25
- ☐ Remove Empty Tables
- Tags to Consider as Substance:
  - ☐ <IMG>
  - ☐ <TEXTAREA>
  - ☐ <A>
  - ☐ <BUTTON>
  - ☐ <INPUT>
  - ☐ <FORM>
  - ☐ <SELECT>
  - ☐ <IFRAME>
- Minimum Text Length: 12

Commit | Revert | Ok | Cancel

Figure 21



Crunch 2.0: Content Extractor Settings

Ignore Settings | Advanced Settings | Output Settings

Output Format
- ● HTML only
- ○ Text only

HTML Output Settings
- ☐ Append Links to Bottom of Page

Text Output Settings
- ☐ Limit number of line breaks
- Maximum number of line breaks: 2

Commit | Revert | Ok | Cancel

Figure 22

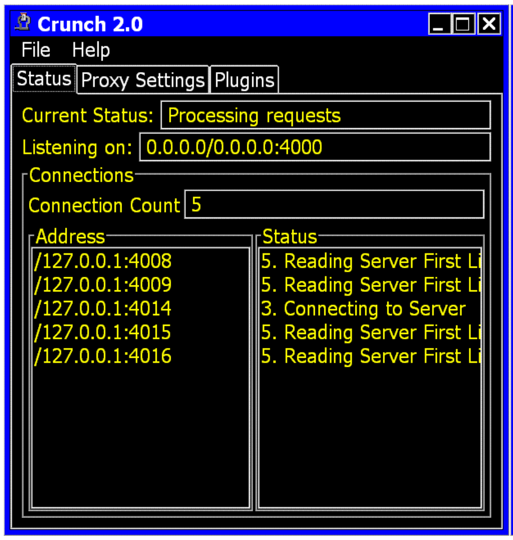
The Advanced User Interface for Crunch 2.0: The options available to customize the current plug-ins

4) Accessibility

Of the plethora of benefits to using SWT, many of which are mentioned above, the most important is accessibility. One of Crunch's purposes is to assist disabled persons in browsing the web, yet the previous version of Crunch itself, i.e., the actual application and the administrative user interface, were highly inaccessible.

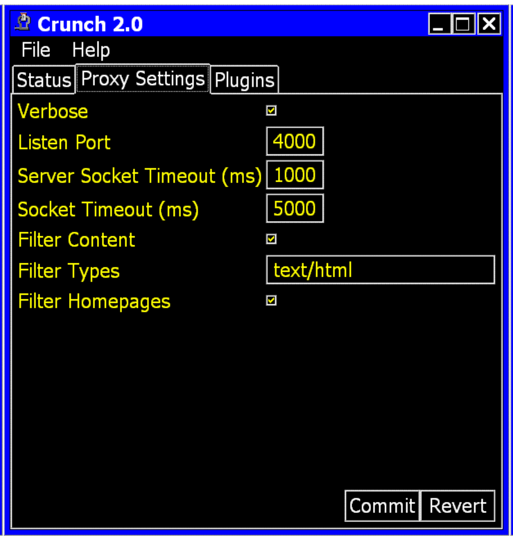
There are three basic categories of accessibility support: mobility enablement, visual enhancement, and screen readers [25]. Mobility enablement is provided in that all settings can be easily accessed through the keyboard without any assistance through the mouse. SWT provides keyboard accelerators in the API, as well as intelligently supporting tabbing through GUI components. SWT uses the operating system's theme for its look and feel, which means that the operating system is allowed to handle usability and visual enhancements. The best example of this is Window's accessibility features [25], such as large fonts and high contrast themes, being incorporated into the GUI. SWT also supports Microsoft Active Accessibility Support (MAAS) [26], so by default there is support for screen readers that read content from the window with focus and its associated widgets.

Usually a person requiring a screen reader will not be able to position a mouse pointer finely enough to successfully use a mouse [25], so it is important that mobility enhancement features coincide nicely with screen readers. Since SWT uses native APIs, screen readers and other accessibility options are able to work nicely together to provide the disabled with a viable way of configuring Crunch 2. As an added benefit to Windows users, SWT can use Windows themes in the same way that it uses accessibility feature of the operating system [25]. Examples screenshots of the proxy in the high contrast scheme are shown in Figures 23-29. Note that Figure 29 shows how the user can also adjust the font size of the website text from within the proxy.

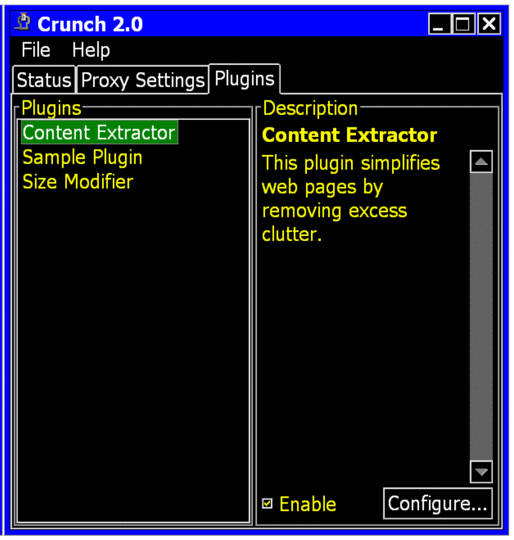


Current Status: Processing requests
Listening on: 0.0.0.0/0.0.0.0:4000
Connections
Connection Count 5

Address	Status
/127.0.0.1:4008	5. Reading Server First Li
/127.0.0.1:4009	5. Reading Server First Li
/127.0.0.1:4014	3. Connecting to Server
/127.0.0.1:4015	5. Reading Server First Li
/127.0.0.1:4016	5. Reading Server First Li



Verbose	☑
Listen Port	4000
Server Socket Timeout (ms)	1000
Socket Timeout (ms)	5000
Filter Content	☑
Filter Types	text/html
Filter Homepages	☑



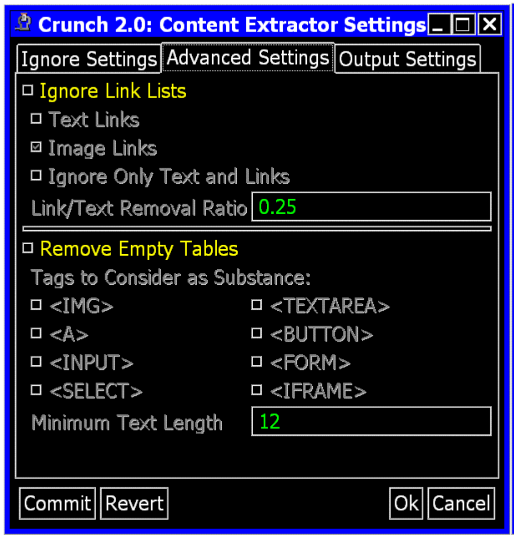
Plugins
Content Extractor
Sample Plugin
Size Modifier

Description
Content Extractor
This plugin simplifies web pages by removing excess clutter.

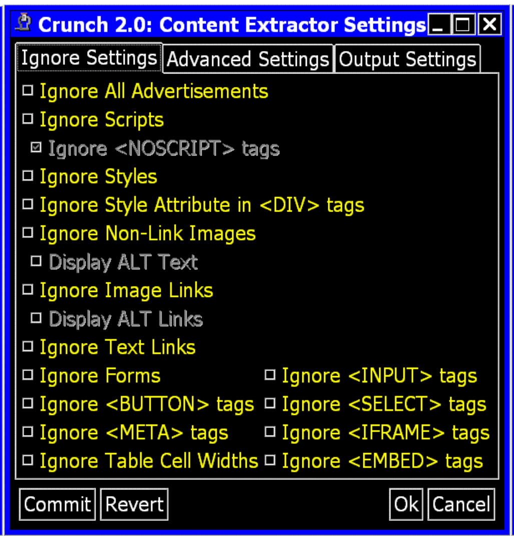
☑ Enable

Figure 23 Figure 24 Figure 25

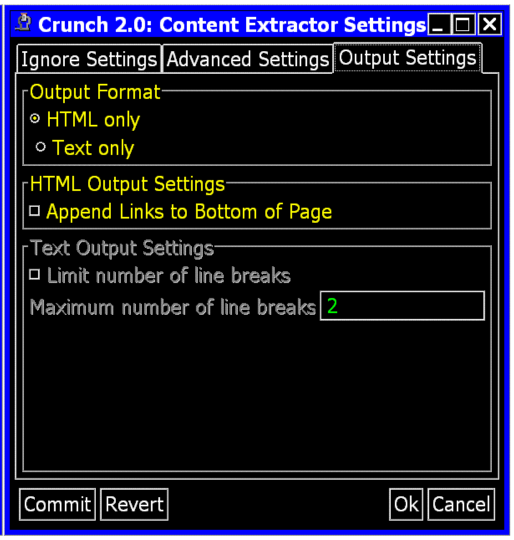
The Basic User Interface for Crunch 2.0 in high contrast format


Crunch 2.0: Content Extractor Settings
Ignore Settings Advanced Settings Output Settings
Commit Revert Ok Cancel

☐ Ignore Link Lists
☐ Text Links
☑ Image Links
☐ Ignore Only Text and Links
Link/Text Removal Ratio 0.25
☐ Remove Empty Tables
Tags to Consider as Substance:
☐ <IMG> ☐ <TEXTAREA>
☐ <A> ☐ <BUTTON>
☐ <INPUT> ☐ <FORM>
☐ <SELECT> ☐ <IFRAME>
Minimum Text Length 12

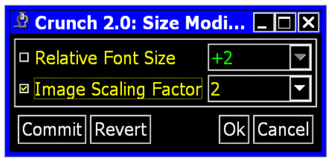

Crunch 2.0: Content Extractor Settings
Ignore Settings Advanced Settings Output Settings
Commit Revert Ok Cancel

☐ Ignore All Advertisements
☐ Ignore Scripts
☑ Ignore <NOSCRIPT> tags
☐ Ignore Styles
☐ Ignore Style Attribute in <DIV> tags
☐ Ignore Non-Link Images
☐ Display ALT Text
☐ Ignore Image Links
☐ Display ALT Links
☐ Ignore Text Links
☐ Ignore Forms ☐ Ignore <INPUT> tags
☐ Ignore <BUTTON> tags ☐ Ignore <SELECT> tags
☐ Ignore <META> tags ☐ Ignore <IFRAME> tags
☐ Ignore Table Cell Widths ☐ Ignore <EMBED> tags


Crunch 2.0: Content Extractor Settings
Ignore Settings Advanced Settings Output Settings
Commit Revert Ok Cancel

Output Format
[✓ HTML only]
Text only
HTML Output Settings
☐ Append Links to Bottom of Page
Text Output Settings
☐ Limit number of line breaks
Maximum number of line breaks 2

Figure 26 Figure 27 Figure 28



☐ Relative Font Size +2
☑ Image Scaling Factor 2

Figure 29

The Advanced User Interface for Crunch 2.0in high contrast format

4.2.3. Example plug-ins for PDAs

One very important requirement is that Crunch is able to support other people’s heuristics, following a modular approach, so more features can easily be added as plug-ins. The extension APIs for both Crunch 1.0 and 2.0 are shown in Appendix B.

One common application for content reformatting and filtering is for Personal Digital Assistants. PDA's have small screens that make viewing unmodified web pages very difficult. Most web pages are designed for resolutions upwards of 800x600 while a majority of PDA's are only 240x320. There are several applications that try to solve this problem.

Bitstream Inc. makes a browser called ThunderHawk (previously mentioned in Section 2). While it is easy to view content on ThunderHawk using Crunch, it is also possible to do very similar processing using a plug-in for Crunch. The resize algorithm was created to demonstrate this ability, showing that a plug-in that performs font replacement and element/image scaling is easy to integrate. Part of the integrated plug-in is shown in Figure 26 (there in high contrast format). Using this, Crunch is now able to give similar results as given by ThunderHawk itself.

Another solution for modifying web pages for small PDA screens is the Skweezer Proxy [28]. Skweezer is implemented very similarly to Crunch in that it acts like a proxy and modifies the content of the webpage before sending it to the client. It reformats web pages such that they wrap intelligently, which prevents unnecessary side scrolling. Crunch can be set to co-exist with Skweezer by using Skweezer as proxy between Crunch and the Internet. Since Skweezer is not open-source, we were unable to integrate it as a plug-in but instead re-implemented a similar algorithm to Skweezer as a plug-in test case.

To implement Skweezer one would create a class that extends ProxyFilter (see Appendix B). The method that should do the actual processing is the process(Document, Document, Document) method. It should be thread safe because multiple threads can be accessing it at the same time.

```
        public abstract Document process(
                Document originalDocument,
                Document previousDocument,
                Document currentDocument);
```

Crunch 2 can load the plug-in at initialization by editing the constructor of Crunch2.java to have the line

```
                proxy.registerPlugin(new SkweezerPlugin());
```

appended to the already existing plug-ins.

```
                proxy.registerPlugin(new ContentExtractor());
                proxy.registerPlugin(new SamplePlugin());
                proxy.registerPlugin(new SizeModifier());
```

The order these lines appear in is the order the plug-ins are applied to filtered content. In this manner, we can add any number of plug-ins.

5. User Study: Web accessibility by visually disabled users

5.1 INTRODUCTION: WEB ACCESSIBILITY

Direct-manipulation graphical users interfaces (GUIs) are widely considered to be a major advance in human-computer interaction because they allow users to perform tasks in a safe, effective, efficient, and enjoyable manner [29-30]. The popularization of GUIs is in part responsible for the rapid growth of computer and Internet technologies during the past decade. However, this trend threatens to create significant barriers to accessibility by visually disabled patients, who are unable to rely on the graphical cues and symbolic representations that are fundamental to GUIs [31-33].

At the same time, the number of visually impaired users is expected to increase dramatically as the population continues to age; for example, it is estimated that the number of Americans over the age of 65 will double between 2000 and 2040 [34]. In 1997, the United States Census Bureau estimated that there were 7.7 million adults with "non-severe visual limitation," which was defined as a self-reported or proxy-reported "difficulty with seeing words and letters, even with eyeglasses." The Census Bureau similarly estimated that there were 1.8 million American adults with "severe visual limitation," which was defined as the "inability to see words and letters, even with eyeglasses" [35]. For medical or legal purposes, visual disability is generally categorized using more formal criteria: "legal blindness" is defined by having visual acuity of 20/200 or worse in the better eye, or a visual field extent of less than 20 degrees in diameter. Similarly, "visual impairment" is defined by having 20/40 or worse vision in the better eye even with eyeglasses. Patients with even minimal visual impairment are likely to encounter problems in everyday life. For example, people with vision worse than 20/40 cannot obtain an unrestricted driver's license in most states, and may require assistive devices such as magnifiers for reading [52].

The overall goal of visual assistive technology is to provide alternative, equivalent mechanisms for computer and Web accessibility. Screen readers translate text and graphical displays into auditory output, and have become a predominant assistive technology for users with severe visual disability [36]. However, the current quality of speech-based Web navigation is very limited. In particular, the large quantity of information on Web documents imposes an enormous cognitive load on visually disabled users who must rely on auditory transmission alone, compared to sighted users who are able to identify relevant information by visual scanning [37]. Content extraction from Web pages using Crunch provides an opportunity to provide filtered documents as input to screen readers. This may allow visually disabled users to understand the essential content of Web documents more quickly and effectively.

We performed a preliminary usability evaluation of Crunch 1.0 to supplement screen reading software for Web navigation by visually disabled users. The study design was based on previously established usability testing and cognitive analysis methodologies, in which subjects are asked to "think aloud" while performing representative computer-based tasks [38-40]. This process was captured with full video and audio recordings, providing a source of data rich in physical, temporal, and social context [41-42]. In particular, this usability study was intended to compare the quantitative and qualitative aspects of speech-based Web navigation by a completely blind user, both with and without Crunch.

5.2 USABILITY STUDY METHODS

5.2.1 Subject and software

Full informed consent was obtained before inclusion of volunteers in this study. The subject for this usability evaluation was a 50 year-old woman who had been completely blind since birth. She had no light perception from either eye, and required a guide dog for mobility. She learned Braille as a child, finished a graduate school degree program, and was employed as a full-time teacher. The subject described herself as "comfortable" with computers and the Web, and used these regularly for work. She was very familiar with assistive technologies such as screen readers, and was able to type over 20 words per minute using a standard QWERTY keyboard.

A popular screen reading Web browser (IBM Homepage Reader®; IBM, White Plains, NY) was selected for this study because it was easy to install and integrate with Crunch. The study subject had used this particular screen reader in the past, and was asked to perform Web navigation until she felt comfortable using all basic commands.

5.2.2 Design of Web-based tasks

Two representative Web-based tasks were developed that satisfied three criteria: (1) Each task involved a website that was among the 50 most popular sites, based on the well-known PageRank algorithm [43-44]. This was to ensure that tasks were representative of common Web browsing procedures. (2) Each task was extensively bench-tested by the authors to ensure that it met a sufficient number of World Wide Web Consortium accessibility guidelines to be completed using speech-based navigation with a screen reader alone [45]. Many popular websites failed to satisfy this criterion. (3) Each task was extensively bench-tested to ensure that it functioned properly with Crunch 1.0, and that it could be completed by speech-based navigation using Crunch 1.0 together with screen reading software.

Table 1 describes the two tasks. Each task was further bench-tested to determine the sequence and number of steps required for completion with screen reading software, both with and without Crunch. Additional testing was performed to determine the optimal Crunch system configuration settings that would allow all tasks to be completed.

Table 1. Web-based tasks to be completed by subjects using screen reading software, with or without content extraction using CRUNCH proxy.

Task (website)	Description
A (www.usatoday.com)	Identify and read top story under “Sports” section
B (www.cnn.com)	Identify and read top headline story

5.2.3 Test protocol

Approval for the study protocol was obtained by the Institutional Review Board at Columbia University Medical Center. The subject was asked to perform Task A using the screen reader alone, and then to perform Task B using Crunch 1.0 and the screen reader. During this process, the subject was instructed to “think aloud” and verbalize impressions while performing speech-based navigation. After completing the two tasks, the subject was asked to provide specific qualitative feedback about the testing procedure. A survey was used to rate various aspects of Web navigation on a five-point Likert scale, both with and without Crunch: (a) Usefulness of technology for performing the task. (b) Ease of deciding next step in navigation using technology. (c) Ease of understanding Web document layout with technology. (d) Ease of locating desired information of Web document using technology. (e) Overall satisfaction with technology.

While performing tasks, the study subject was videotaped and audiotaped using a portable usability engineering system [29, 41]. A video converter (Communication Specialties, Haupaugue, NY) converted the monitor display to a video signal (S-video) for capture on videotape using a digital video camera (Toshiba; New York, NY). A microphone provided audio input to the video camera, in order to record statements and questions, as well as the screen reader sounds. A cassette recorder was used to capture additional sounds. Finally, a standard 8 mm video camera was used to record keystrokes while the user interacted with the system.

5.2.4 Data analysis

Contents of videotapes and cassette tapes were transcribed verbatim, and annotated with time-stamps. Tapes were then coded using a standard method adapted from previous studies, in order to note particular aspects of system usability [41]. User actions were described as an overall task, which was divided into goals and subgoals. Each subject action was coded either as a correct response, an error, or a correct response to an error. Errors were categorized into one of three groups: (1) Errors in understanding of the interface. This included selection of unintended links, incorrect interpretation or hearing of speech, and confusion with manipulation of GUI widgets or browser commands. (2) Errors in understanding of document layout or navigation. This included any confusion caused by incorrect mental representation of documents, such as misunderstanding of navigation bars, or becoming "lost" while navigating within or between pages. (3) Errors in understanding caused by Web design or browser malfunctions. This included failure to comply with standard Web accessibility guidelines [45].

The total time required to complete each task was measured. This was used to calculate the time required to complete each step of the task, based on results from bench-testing. The causes of Web browsing errors were determined from detailed analysis of audiotapes and videotapes. Numerical ratings of Web browsing surveys were tabulated.

5.3 RESULTS

5.3.1 Bench-testing of tasks

Each task was carefully reviewed to determine the sequence and number of steps required for completion, both without and with Crunch 1.0. Figure 30 demonstrates the results of this analysis for Task B, which required more steps without Crunch (65 steps) than with it (38 steps). Similarly, Task A required more steps without Crunch (73 steps) than with it (23 steps). This reduction of steps required for each task was because the content extraction process simplified direct access to the Web document contents by removing all navigation bar links.

Task B: Go to www.cnn.com. Identify and begin reading top headline story.

Using screen reader without CRUNCH 1.0:

(A) Open text box and go to website → CNN homepage opens.
(B) Pass 2 lines. Select "Skip to main content" link.
(C) Pass 9 lines. Select "Full Story" link → Full Story page opens.
(D) Pass 2 lines. Select "Skip to main content" link.
(E) Pass 52 lines. Begin reading story text.

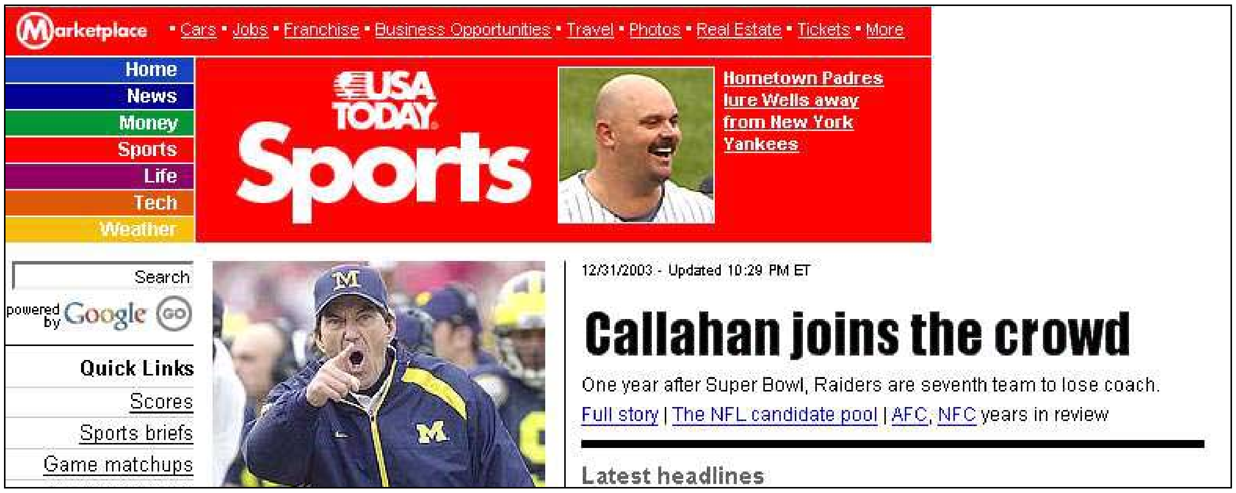
Using screen reader with CRUNCH 1.0:

(A) Open text box and go to website → CNN homepage opens.
(B) Pass 2 lines. Select "Skip to main content" link.
(C) Pass 8 lines. Select "Full Story" link → Full Story page opens.
(D) Pass 2 lines. Select "Skip to main content" link.
(E) Pass 26 lines. Begin reading story text.

Figure 30. Example of bench-testing to determine the sequence and number of steps required to complete tasks with screen reading software, both without and with CRUNCH 1.0.

5.3.2 Features of navigation

Using a screen reader without Crunch, the subject did not complete Task A ("Go to www.usatoday.com and read the top Sports story") successfully. After 21 minutes and 15 seconds, she began reading an incorrect story. Based on the fact that this task should have taken 73 steps to complete successfully, the subject required an average of 17.5 seconds per step without Crunch. Transcription and subsequent analysis of tapes revealed that the subject made a total of 31 cognitive errors during the navigation process for Task A. Based on the taxonomy described above, these errors were classified into three categories: (1) 11 errors in understanding or using the speech-based interface. For example, the subject attempted to use a "search" function, but was unable to properly enter the desired term into the text box. (2) 14 errors in document layout or navigation. For example, the web page layout caused the screen reader to announce the full navigation bar on every page (Figure 31). Even when the subject had already reached the correct "Sports" page, she became disoriented by hearing the navigation link lists. As a result, she mistakenly re-selected the "Sports" link nine additional times. When the subject finally reached the top sports story, she failed to recognize it as a story, apparently because the document made no announcement before beginning to read the story title. Therefore, she continued past the top story and eventually selected an incorrect link as the story to read. (3) 6 errors caused by Web design or browser malfunctions. For example, the subject mistakenly attempted to select a link to an advertisement banner, believing that it contained relevant information.


Marketplace
• Cars • Jobs • Franchise • Business Opportunities • Travel • Photos • Real Estate • Tickets • More
Home
News
Money
Sports
Life
Tech
Weather
USA TODAY Sports
Hometown Padres lure Wells away from New York Yankees
Search
powered by Google GO
Quick Links
Scores
Sports briefs
Game matchups
Latest headlines

12/31/2003 - Updated 10:29 PM ET

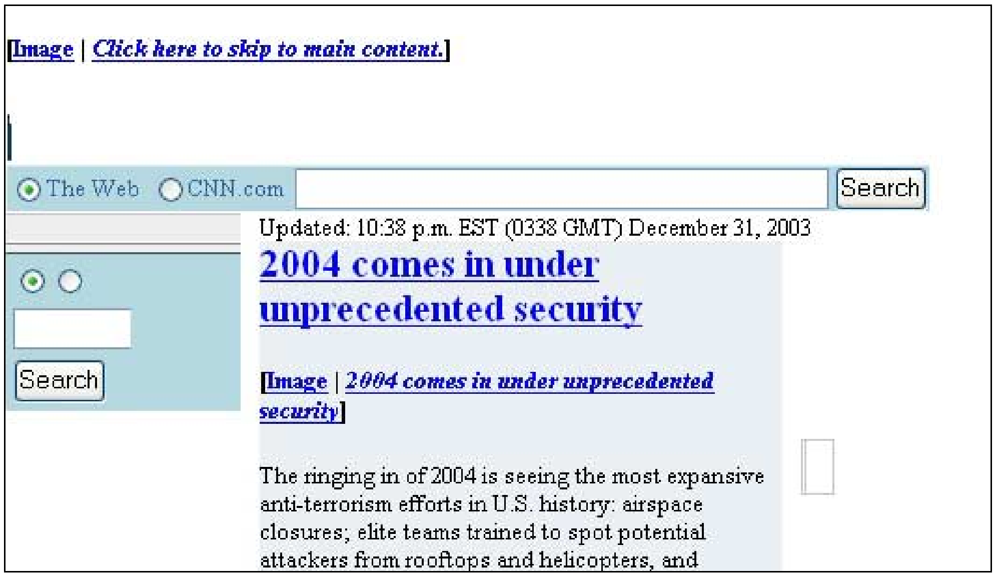
Callahan joins the crowd

One year after Super Bowl, Raiders are seventh team to lose coach.

Full story | The NFL candidate pool | AFC, NFC years in review

Figure 31. Navigation of Task A using screen reader without CRUNCH. Document layout forced announcement of navigation bar, even when subject was already on correct "Sports" page. This caused subject to become disoriented and select "Sports" navigation link nine additional times.

Using the screen reader with Crunch, the subject completed Task B ("Go to www.cnn.com and read the top headline story") successfully. After 2 minutes, she began reading the correct story. Based on the fact that this task should have taken 38 steps to complete successfully, the subject required an average of 3.2 seconds per step with Crunch. Transcription and subsequent analysis of tapes revealed that the subject did not make any cognitive errors during the navigation process. This was apparently because the main headline story was placed very near the beginning of the filtered document, without extraneous navigation bar or other link lists (Figure 32).



Updated: 10:38 p.m. EST (0338 GMT) December 31, 2003

2004 comes in under unprecedented security

[Image | 2004 comes in under unprecedented security]

The ringing in of 2004 is seeing the most expansive anti-terrorism efforts in U.S. history: airspace closures; elite teams trained to spot potential attackers from rooftops and helicopters, and

Figure 32. Navigation of Task B using screen reader with CRUNCH. Subject made no cognitive errors in navigation, apparently because headline story was placed near beginning of the filtered document.

5.3.3 Qualitative user evaluation

After completing Tasks A and B, the subject was surveyed regarding attitudes toward various aspects of speech-based Web navigation without and with Crunch 1.0. Results are summarized in Table 2.

Table 2. Survey evaluation of subject's attitudes toward speech navigation, without and with CRUNCH 1.0. Scores are based on Likert scale (1=Strongly agree, 2=Agree, 3=Neutral, 4=Disagree, 5=Strongly Disagree).

Aspect of navigation	Score	
	Without CRUNCH	With CRUNCH
Useful to read Web pages	4	2
Easy to decide next step	3	2
Easy to understand Web layout	2	2
Easy to locate information	4	3
Overall satisfied with navigation	5	3

5.4 DISCUSSION OF USABILITY STUDY

This pilot evaluation employed a usability engineering approach to analyze the application of Crunch for speech-based Web navigation by a completely blind subject. It was designed as a paired study, in which the subject was asked to perform tasks without and with Crunch. Bench-testing confirmed that Tasks A and B required a similar number of steps for completion, suggesting that they were of comparable complexity. By transcribing, time-stamping, and coding the video and audio recordings of user interactions with the system, it was possible to measure the speed and error rate of Web navigation. In addition, it was possible to categorize the cause of each navigation error.

Overall, the results of this preliminary user study suggest that Crunch has potential to provide advantages over conventional speech-based browsing in terms of speed, error rate, and qualitative satisfaction. This is primarily by removing extraneous content, and thereby simplifying the process of finding the important information on the page. Bench-testing also demonstrated that Tasks A and B both required fewer steps for completion with Crunch than without it. However, supplementation with content extraction is not clearly superior to conventional speech-based browsing. For example, by removing features such as link lists, Crunch has potential to cause new errors in understanding page layout and navigation. Similarly, Crunch inserts removed link lists at the end of the Web document, where they may be extremely difficult for users to navigate because of the lack of surrounding context. Finally, Crunch does not perform useful content extraction on all websites (e.g., see Figures 11 and 12), and it was difficult to develop a corpus of representative tasks for evaluation purposes.

This preliminary usability evaluation has two important limitations: (1) It involved only one subject, and therefore could not include meaningful analysis for statistical significance or reproducibility among various users. (2) Because it involved only two standardized tasks, conclusions may not be generalizable to other Web-based tasks. These limitations are being addressed by ongoing usability studies that involve recruitment and testing of additional visually disabled subjects. Results of evaluation studies will provide additional data for iterative design improvements to content extraction systems such as Crunch, and provide insight into the cognitive models used by visually disabled users for speech-based Web navigation.

6. Future Directions

Crunch uses a third-party HTML parser to create DOM trees from web pages. We have switched to NekoHTML to resolve the problems with OpenXML. However, we still intend to support commercial parsers, such as Microsoft's HTML parser (which is used in Internet Explorer), in the next revision. Integration will be accomplished by porting the existing Crunch proxy to C#/.NET, which will allow for easy integration with COM components (of which the MS HTML parser is one).

We are continuing work towards improving the proxy's performance; in particular, we aim to improve both latency and scalability, especially with the advent of browsers such as Avantbrowser [53] and Mozilla [54] that support tabbed browsing, i.e., treating multiple open web pages as part of the same session.

We are also investigating supporting more sophisticated statistical, information retrieval and natural language processing approaches as additional heuristics to improve the utility and accuracy of our current system.

Currently we do not do any form of learning of a user's browsing habits. It may be possible to implement artificially intelligent heuristic algorithms, such as Bayesian learning or Markov Model creation, as a browser plug-in that reads metadata from the client about how to change the settings. Such a browser plug-in might provide an interface for the user to rate pages, that is, Crunch's rendition of pages, and could update Crunch's configuration via extra HTTP metadata. The improved Crunch 2.0 plug-in interface is instrumental in allowing these kinds of heuristics because it allows programmatic changes to settings. With the addition of trainable filtering, Crunch could adapt to a particular user's or group's preferences. Even basic control from the browser, without any AI, would enhance Crunch's usability because the user wouldn't have to switch applications to change a setting or to enable or disable filtering.

The user study with blind users is in its earliest stages, and will be reported more formally when completed.

Finally, one of our main goals was to expose a simple API for programmers to extend, so that current and future natural language processing and information retrieval algorithms can easily be added to Crunch. This would allow users to truly be able to customize the content they would like to view on visited web pages. Full evaluation of the API and plug-in framework will not be possible until sufficient outside developers have worked with Crunch.

7. Conclusion

Many web pages contain excessive clutter around the bodies of one or more articles, the actual content of the page. Although much research has been done on content extraction, and there are many special-case solutions to remove advertising (particularly pop-ups) or reformat for small screens, it is still a relatively new field where few general purpose tools are available so most researchers must construct their content extractors from scratch. Our approach, working with the Document Object Model tree as opposed to raw HTML markup, enables us to apply in tandem an extensible collection of Content Extraction filters, and potentially other kinds of filters such as format translators and NLP summarizers. The heuristic filters that we have developed to date, though simple, are quite effective.

Crunch has been implemented as a freely-available web proxy that anyone can use to extract content from HTML documents for their own purposes. The second version of Crunch is fast and efficient, and allows for easy integration of third party filters as plug-ins. It also offers a simple, easy to use user interface for both administrators and end users. And perhaps most importantly, we have designed this system with accessibility in mind for the visually impaired, so as to facilitate the best possible web experience in conjunction with devices such as screen magnifiers and screen readers.

Appendix A – Example Screenshots

We show some examples of typical websites with different Crunch 2.0 options turned on. The point is to give the reader an idea of the degree of control a user can have over what he/she wants to see on a webpage. The websites we chose are:

1) A typical article from www.spacedaily.com

2) The front page of Voice of America - www.voanews.com

3) The front page of a typical gaming news site - www.planetunreal.com

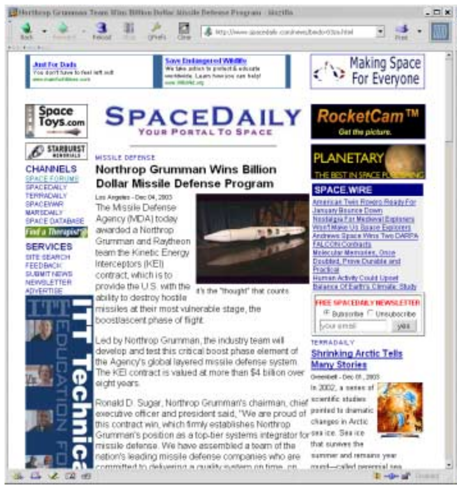
4) An article from a link and script heavy site - www.msnbc.com

5) An article from CNN – www.cnn.com

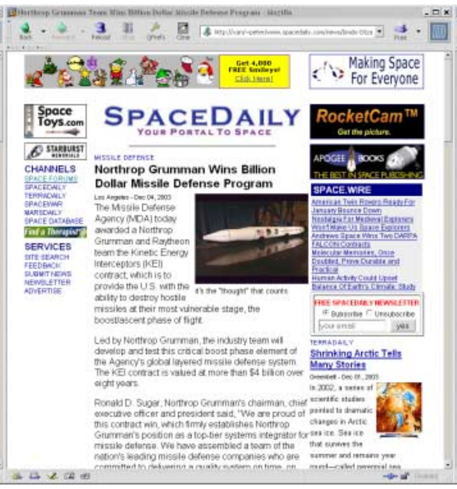
6) The front page of the geek news site – www.slashdot.org

7) The WWW2004 website frontpage – www2004.org

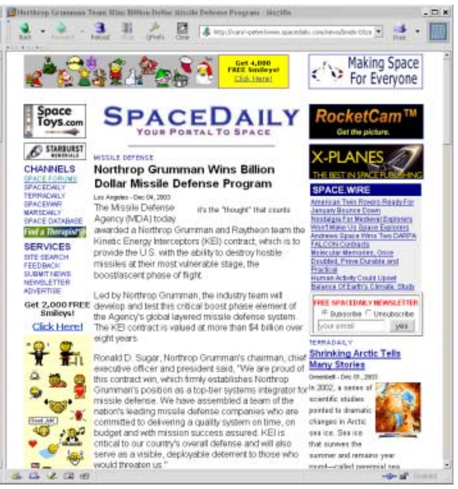
Each following page has a set of images. The images start from a screenshot of the original site, followed by a gradual increase in the number of filters used, continuing to the screenshot that was taken of the site in text-only mode. We have created this anthology of images to help the user get an idea of how Crunch and its filters work on a given webpage.



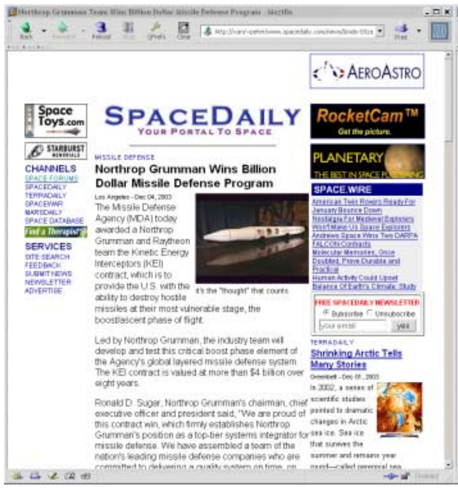
Original



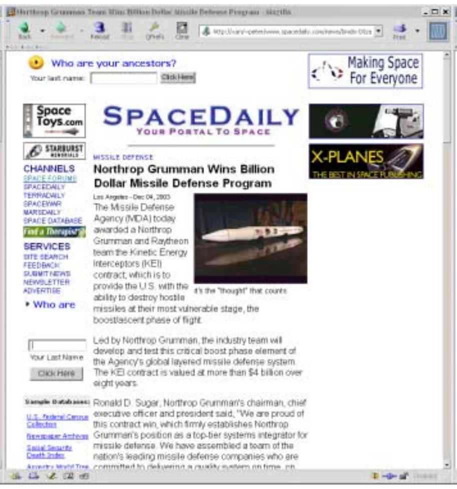
Remove link lists



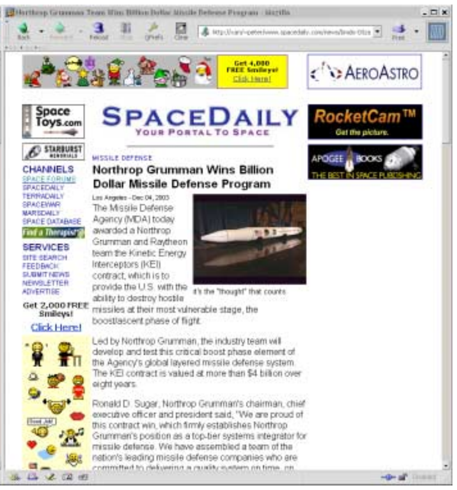
Remove non-link-list images



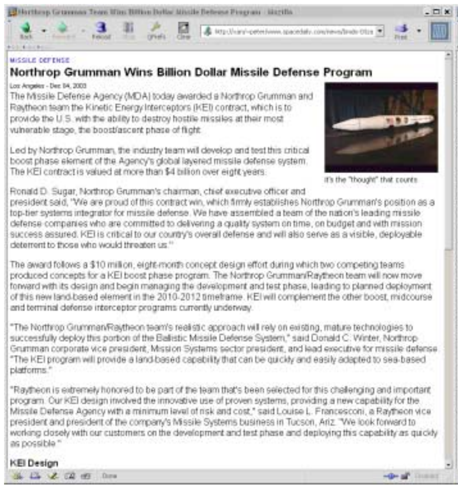
Remove Advertisements



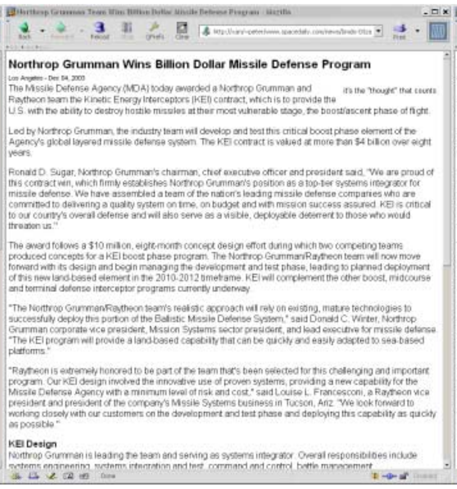
All advanced filters on



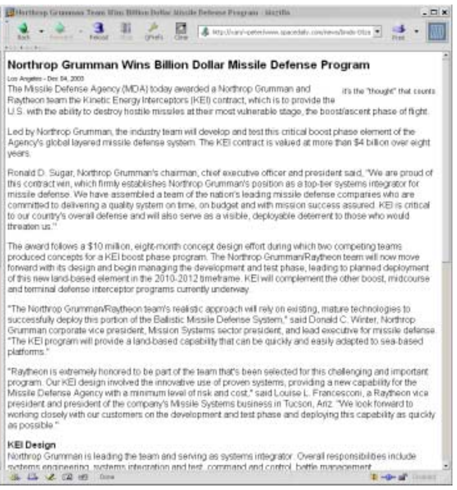
Remove non-link-list tables



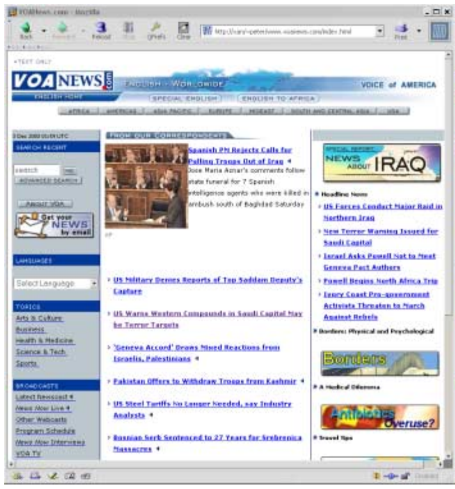
Ignore Scripts

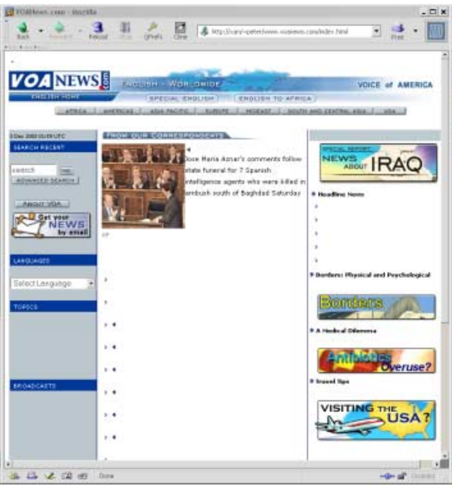


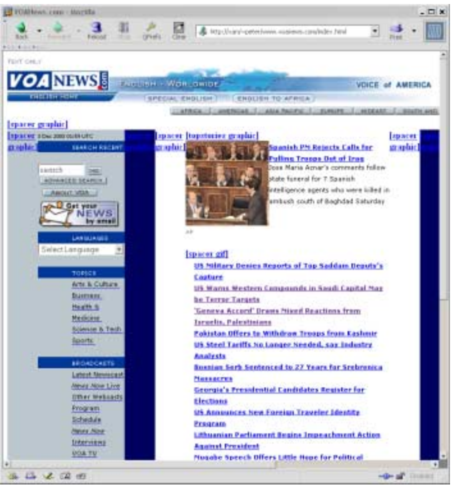
Reduce link-to-text ratio

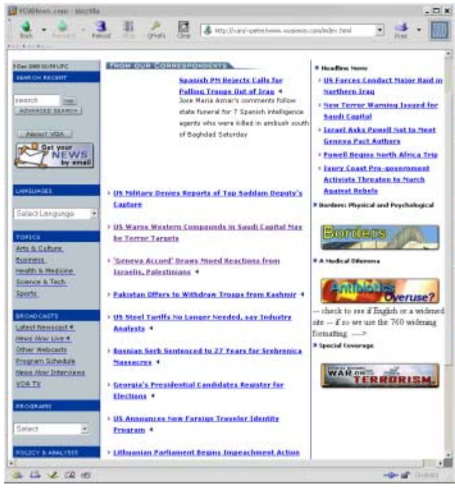


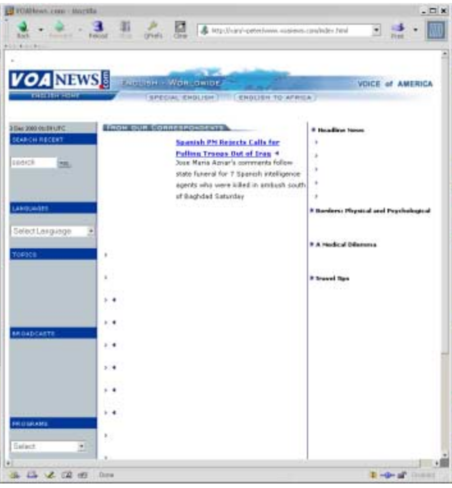
Ignore everything, text only

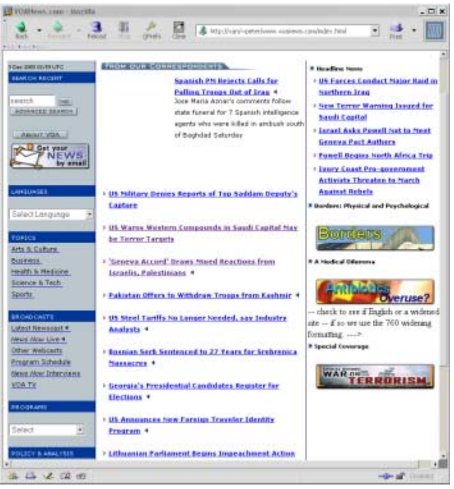

Original

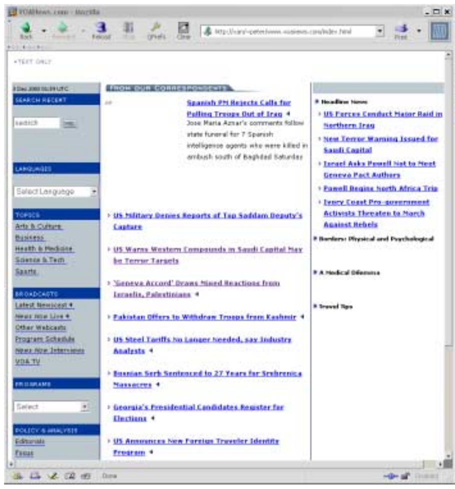

Remove link lists

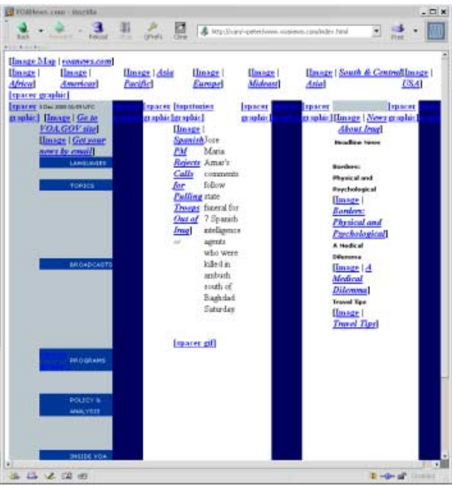

Remove non-link-list images

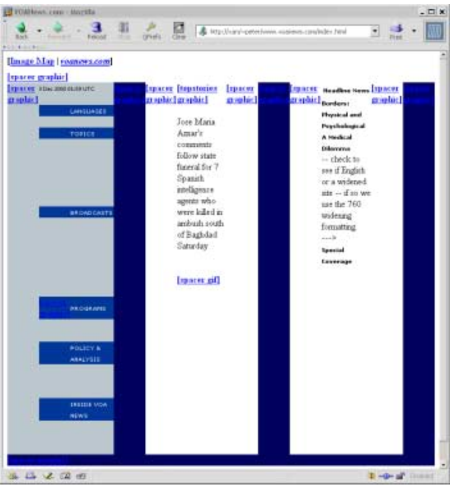

Ignore scripts

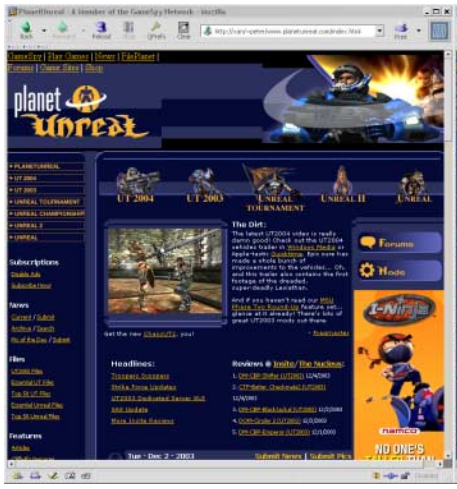

Remove Advertisements

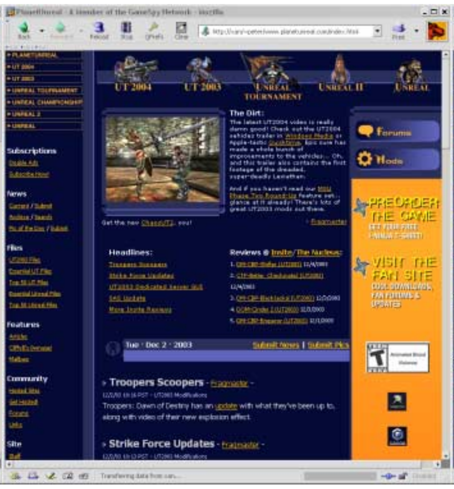

Remove empty non-link-list tables

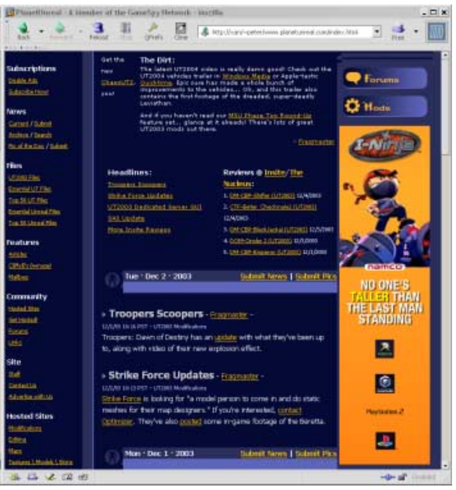

Ignore text links

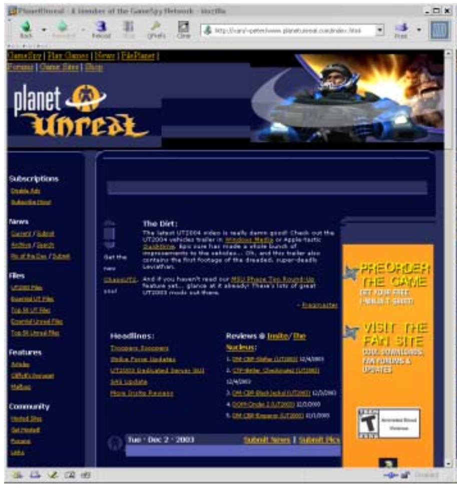

Reduce link-to-text ratio

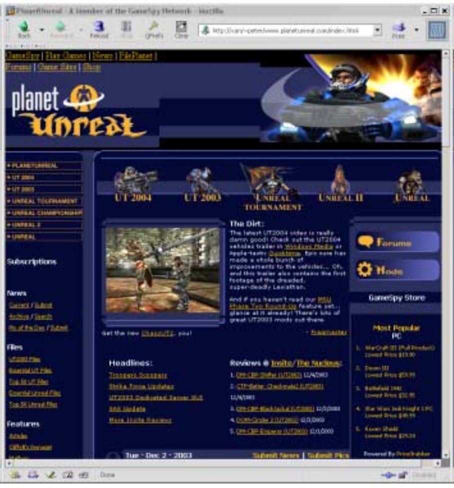

Ignore everything, text only

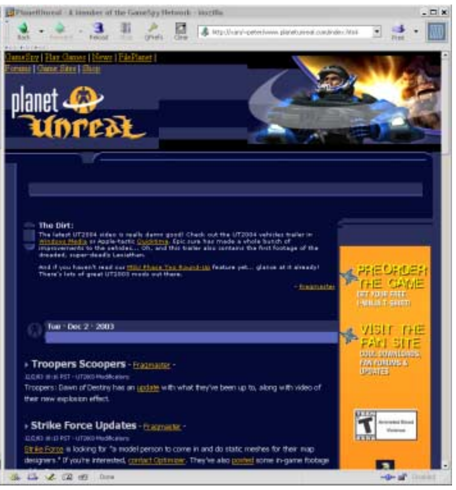

Original

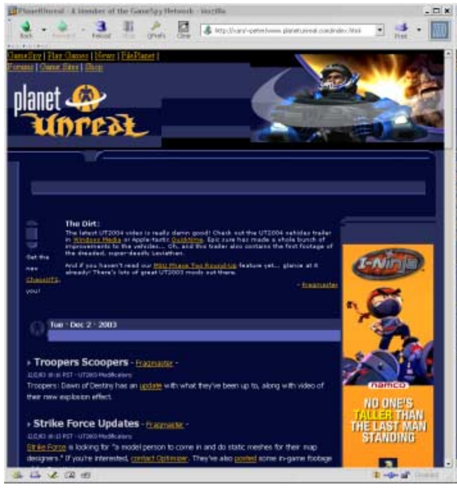

Remove empty cells

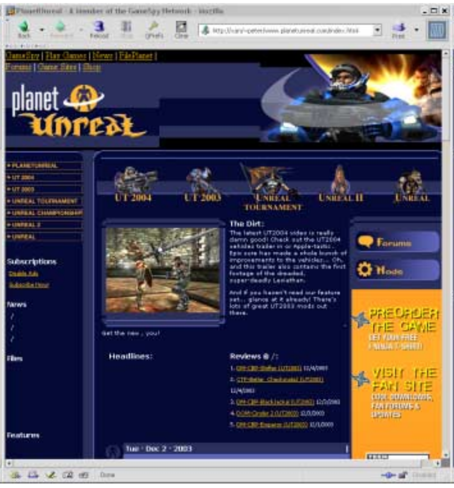

Remove empty tables

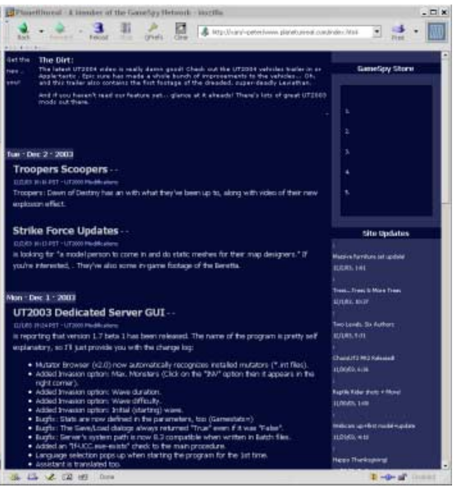

Ignore image links

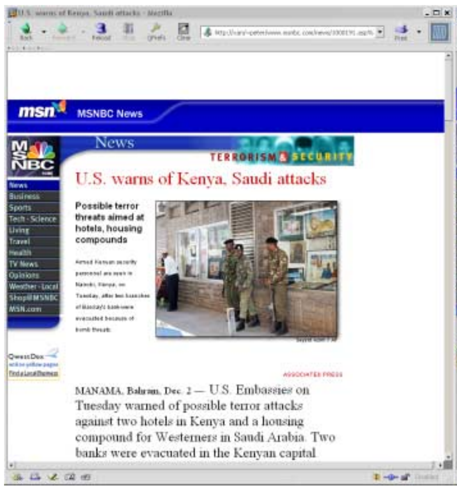

Remove Advertisements


Reduce text-to-link ratio to 0.125


All advanced filters on


Ignore scripts


Ignore everything, text only

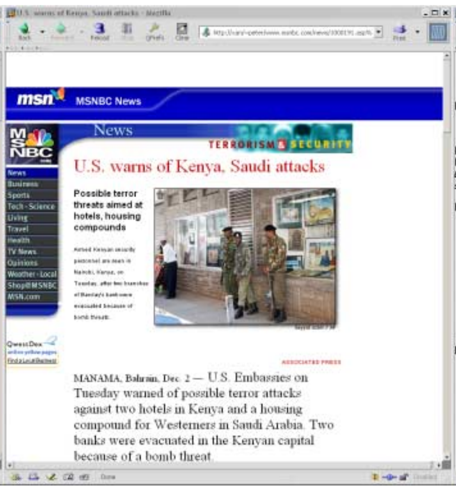


U.S. warns of Kenya, Saudi attacks

Possible terror threats aimed at hotels, housing compounds

MANAMA, Bahrain, Dec. 2 — U.S. Embassies on Tuesday warned of possible terror attacks against two hotels in Kenya and a housing compound for Westerners in Saudi Arabia. Two banks were evacuated in the Kenyan capital

Original

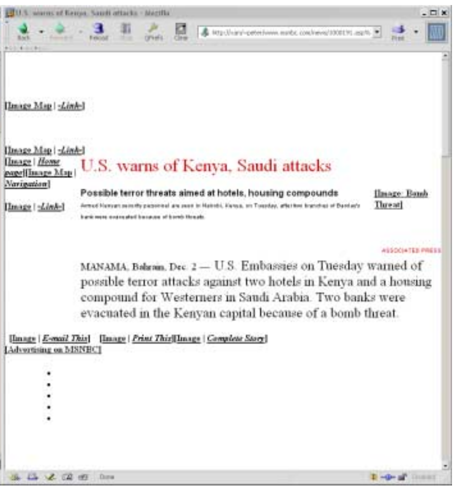


U.S. warns of Kenya, Saudi attacks

Possible terror threats aimed at hotels, housing compounds

MANAMA, Bahrain, Dec. 2 — U.S. Embassies on Tuesday warned of possible terror attacks against two hotels in Kenya and a housing compound for Westerners in Saudi Arabia. Two banks were evacuated in the Kenyan capital because of a bomb threat.

All advanced filters on



[Image Map | -Link-]

[Image Map | -Link-]
[Image | Home page][Image Map | Navigation]

[Image | -Link-]

U.S. warns of Kenya, Saudi attacks

Possible terror threats aimed at hotels, housing compounds

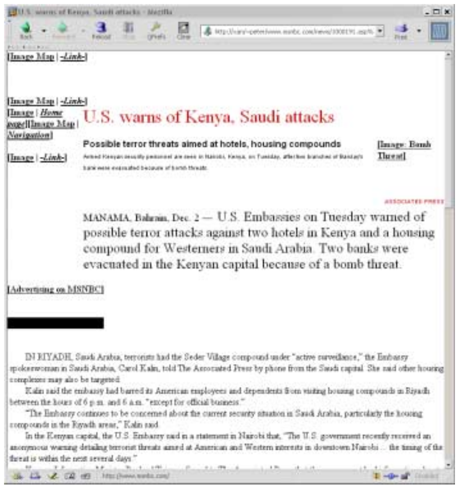
[Image: Bomb Threat]

ASSOCIATED PRESS

MANAMA, Bahrain, Dec. 2 — U.S. Embassies on Tuesday warned of possible terror attacks against two hotels in Kenya and a housing compound for Westerners in Saudi Arabia. Two banks were evacuated in the Kenyan capital because of a bomb threat.

[Image | E-mail This] [Image | Print This][Image | Complete Story]
[Advertising on MSNBC]

Ignore scripts



[Image Map | -Link-]

[Image Map | -Link-]
[Image | Home page][Image Map | Navigation]

[Image | -Link-]

U.S. warns of Kenya, Saudi attacks

Possible terror threats aimed at hotels, housing compounds

[Image: Bomb Threat]

ASSOCIATED PRESS

MANAMA, Bahrain, Dec. 2 — U.S. Embassies on Tuesday warned of possible terror attacks against two hotels in Kenya and a housing compound for Westerners in Saudi Arabia. Two banks were evacuated in the Kenyan capital because of a bomb threat.

[Advertising on MSNBC]

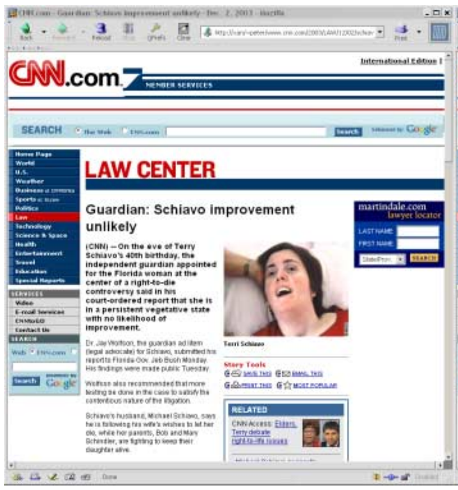
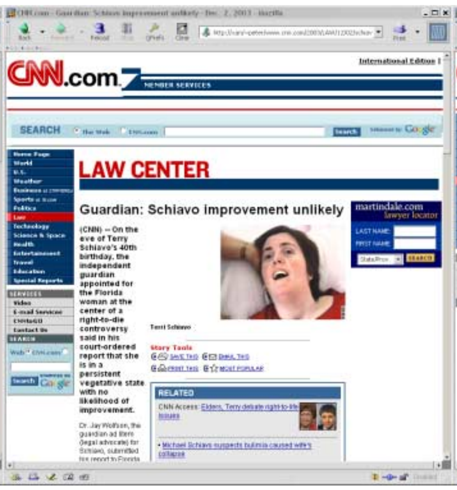
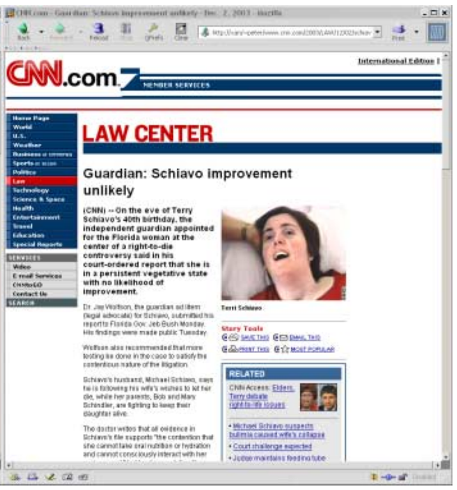
IN RIYADH, Saudi Arabia, terrorists had the Seder Village compound under "active surveillance," the Embassy spokeswoman in Saudi Arabia, Carol Kalin, told The Associated Press by phone from the Saudi capital. She said other housing complexes may also be targeted.

Kalin said the embassy had barred its American employees and dependents from visiting housing compounds in Riyadh between the hours of 6 p.m. and 6 a.m. "except for official business."

"The Embassy continues to be concerned about the current security situation in Saudi Arabia, particularly the housing compounds in the Riyadh area," Kalin said.

In the Kenyan capital, the U.S. Embassy said in a statement in Nairobi that, "The U.S. government recently received an anonymous warning detailing terrorist threats aimed at American and Western interests in downtown Nairobi ... the timing of the threat is within the next several days."

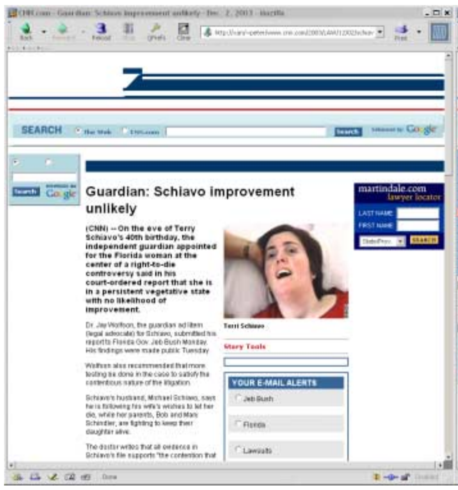
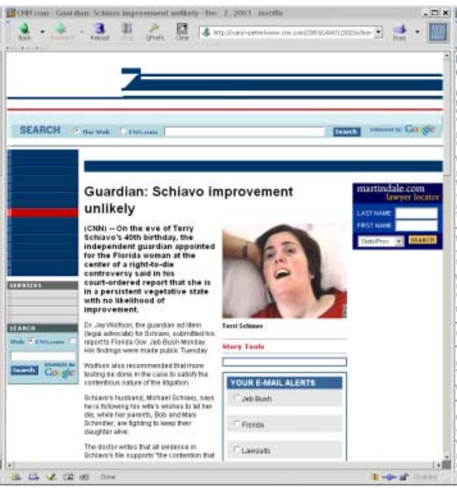
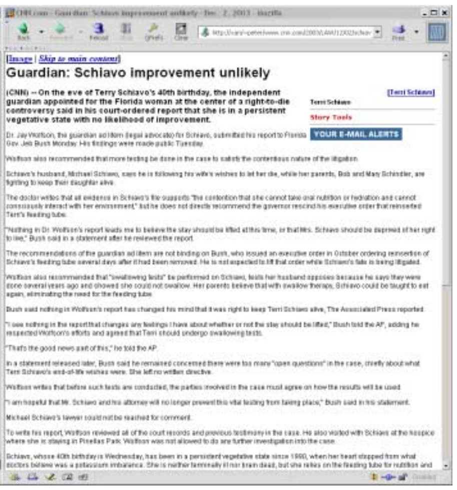
Ignore everything, text only

Original

Ignore table widths

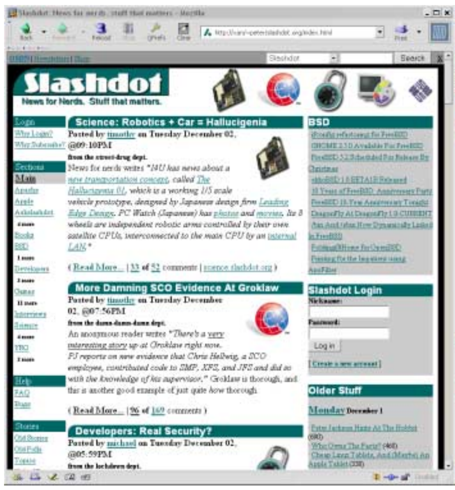
Remove forms

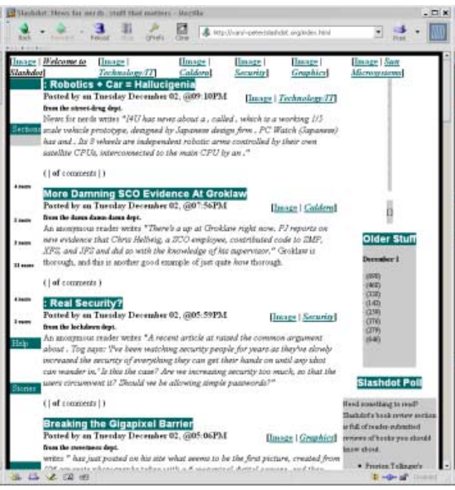




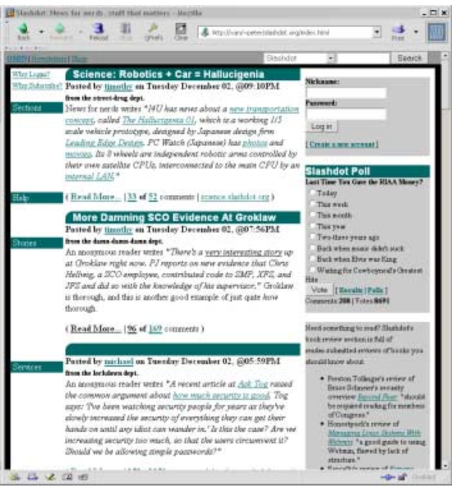
All advanced filters on

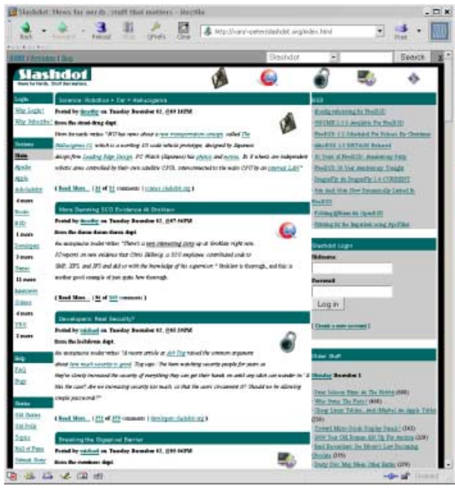
Reduce text-to-link ratio to 0.125

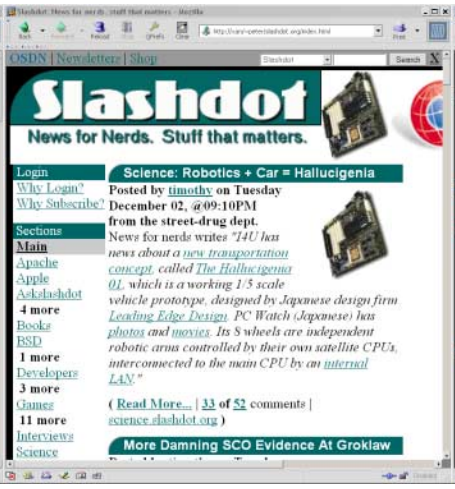
Ignore everything, text only

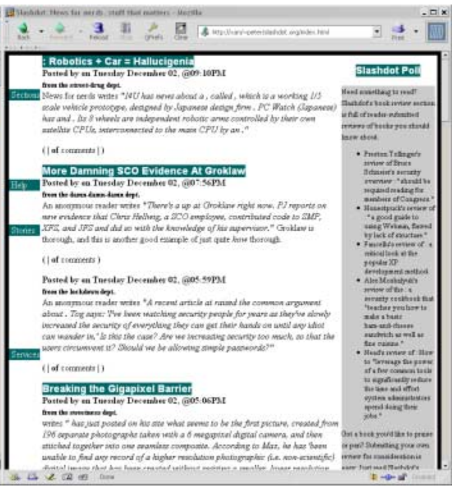

Original

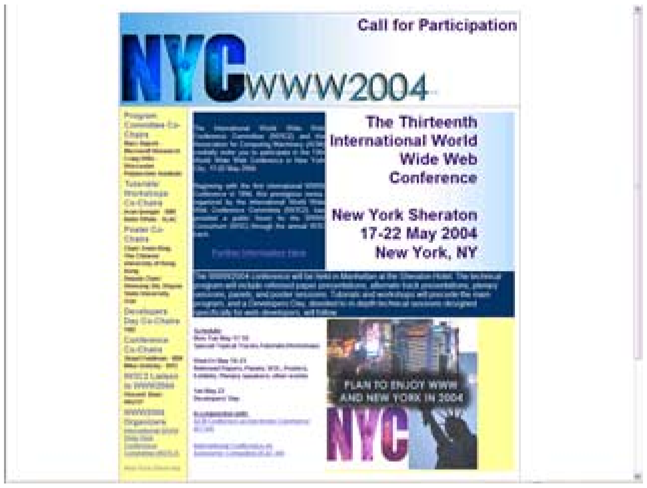

Ignore scripts


All advanced filters turned on

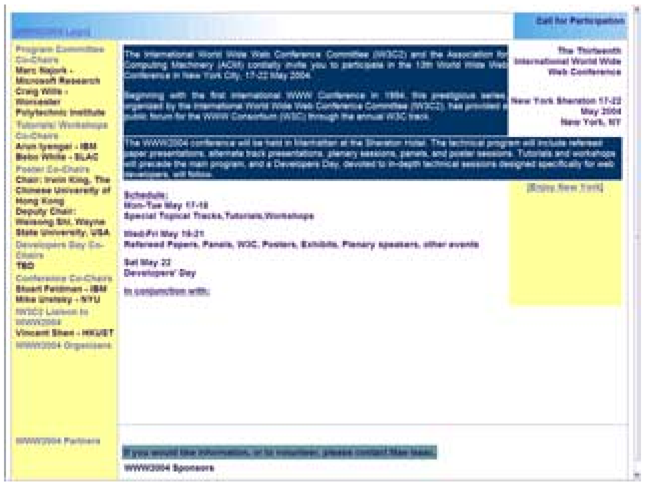

Font size -2 from original


Font size +2 from original

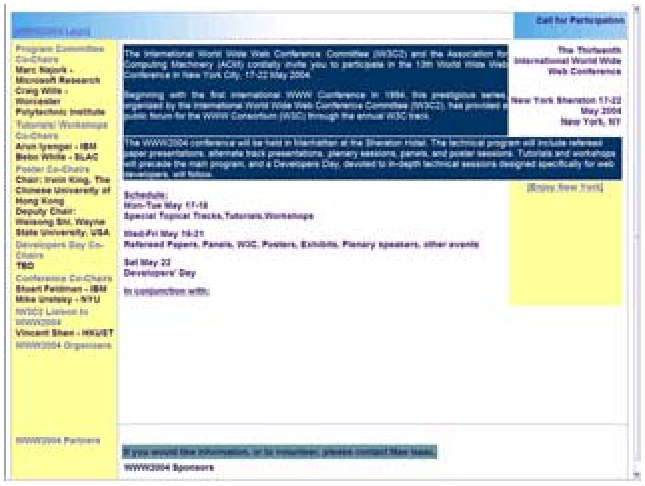

Ignore everything, text only


Call for Participation
NYC WWW2004
The Thirteenth International World Wide Web Conference
New York Sheraton 17-22 May 2004 New York, NY
PLAN TO ENJOY WWW AND NEW YORK IN 2004
NYC

Original



All advanced filters turned on



Ignore everything, text only

Appendix B – Crunch 1.0 vs. Crunch 2.0 plug-in APIs

ProxyFilter.java

```java
package psl.memento.pervasive.crunch;
import java.io.*;
public interface ProxyFilter {
		public File process(File in) throws IOException;
		public ProxyFilterSettings getSettingsGUI();
		public String getContentType();
}
```

```java
package psl.memento.pervasive.crunch2.plugins;
import org.w3c.dom.Document;
public abstract class ProxyFilter {
		private boolean enabled = true;

		public void getSettingsGUI() {
				// no settings GUI is required
		}

		public boolean hasSettingsGUI() {
				return false;
		}

		public abstract String getName();
		public abstract String getDescription();

		public void setEnabled(boolean b) {
				enabled = b;
		}

		public boolean isEnabled() {
				return enabled;
		}

		public abstract Document process(
				Document originalDocument,
				Document previousDocument,
				Document currentDocument);
}
```

ProxyFilterSettings.java

```java
package psl.memento.pervasive.crunch;

import javax.swing.JPanel;

public abstract class ProxyFilterSettings extends JPanel {
		public abstract void commitSettings();
		public abstract void revertSettings();
		public abstract String getTabName();
}
```

```java
package psl.memento.pervasive.crunch2.plugins;

public interface ProxyFilterSettings {
		public void set(String key, String value);
		public String get(String key);
		public void commitSettings();
		public void revertSettings();
}
```

Appendix C – Code differences between Crunch 1.0 and 2.0

Typical Code Differences

The original Crunch plug-in was required to implement the ProxyFilter interface. This interface consists of 3 methods. The first and most important method is the process method. It takes a file and returns a file. It does all the processing on html content that passes through the proxy. The second method is getSettingsGUI. It returns the settings GUI so that the settings for the plug-in can be changed. The third method is getContentType. It returns the content type of the output of the plug-in.

The newer ProxyFilter was created as an abstract class. The new version is similar to the old one, but forces the plug-ins to works on the DOM documents rather than just plain files. It includes seven methods. One method is for filtering and the other methods are for GUI integration. To have the plug-in do processing on content, there is the process method. The process method takes 3 DOM documents for input. One is the document that should be processed and the other two are for reference. CurrentDocument is the document that should be processed. PreviousDocument is the output of the previous filter and is initially just a copy of currentDocument. PreviousDocument is used for rolling back changes or other analysis after changes to currentDocument have already been made. OriginalDocument represents the document as Crunch 2 has received it from the server. This allows for more advanced heuristics, quality checking, and even rollback of the processing. The methods hasSettingsGUI and getSettingsGUI are for determining if the plug-in has a settings dialog, and if it does, displaying it. Currently there is a button that can be clicked if the plug-in has a settings GUI that will display it. The methods isEnabled and setEnabled are for changing and checking the state of the plug-in. If the plug-in is disabled, it is skipped during processing of content and is shown grayed in the plug-ins tab of the main Crunch 2 window. The next two methods, getDescription and getName are used for displaying information about the plug-in and just return strings. Code details of ProxyFilter.java can be found in Appendix B.

The original ProxyFilterSettings extends JPanel, which is inserted into the GUI. Each proxy filter had its own tab; unfortunately this forced the implementer to use Swing, which is not available in many versions of java, such as gcc-java, also known as gcj [57]. It also doesn't unify the API for easy settings modification in the software, which is important for AI algorithms. It contains three methods: commitSettings, revertSettings, and getTabName. CommitSettings and revertSettings are for committing and reverting respectively, the settings that were made in the GUI. GetTabName is for naming the tab to put the panel in. This is usually the name of the plug-in.

The new ProxyFilterSettings is not tied to a GUI at all. Its sole purpose is to programmatically allow for the editing settings. It has four functions - Get takes a string name of the setting and passes back the value as a string. Set takes a setting name and a value and sets the setting. CommitSettings and revertSettings save the settings to a file and load the settings from a file respectively. Code details of ProxyFilterSettings.java can be found in Appendix B.

Notice the differences between the Crunch 1.0 and 2.0 implementations. The Crunch 2.0 plug-in implementation is now more flexible than the original. It is no longer Swing dependent. In fact, it no longer forces the user to have any sort of settings GUI. In Crunch 2.0, while no settings implementation is forced, one is provided so that all the plug-ins can have a common method of changing settings. This will simplify the implementation of any filtering heuristics using AI algorithms that could produce better results, which may need to adaptively change the user settings based on the site and the user's reaction to a given page.

Methods that run filters over content

ProxyThread.filter(HttpStream http) in Crunch 1.0 - In the original Crunch, the filter method inside ProxyThread is what passes content through the plug-ins. It works by downloading the http content to a file, and then it runs each filter on the file and updates the content type each time. After that, it replaces the content file in the http stream with the filtered file.

```
        public void filter(HttpStream http)
                throws IOException {
                File workingFile = null;
                workingFile = http.downloadToFile();
                while (filters.hasNext()) {
                        try {
                                ProxyFilter filter =
                                        (ProxyFilter) (filters.next());
                                System.out.println("Started filtering...");
                                workingFile.deleteOnExit();
                                workingFile = filter.process(workingFile);
                                workingFile.deleteOnExit();
                                http.setAttribute(
                                        "content-type",
                                        filter.getContentType());
                                System.out.println("Done filtering.");
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                }
                http.replaceContentWithFile(workingFile);
                System.out.println("content replaced");
        }
```

PluginFilterRunner.process(File f) in Crunch 2.0 - In Crunch 2.0, the process method inside the PluginFilterRunner class is what runs all the plug-ins on the content. It takes a file as input. First it parses that file into xml, and then it gets a copy of that file and sets it as currentDocument. Next, it enters a loop that checks each plug-in for being enabled and, if so, rotates currentDocument and previousDocument, and then runs the plug-ins process method. After the loop, it writes the most current non-null document to a file.

```
        public File process(final File f) {
                // generate xml document from file
                Document originalDocument = getXML(f);
                Document previousDocument = null;
                Document currentDocument = null;

                currentDocument = copyDocument(originalDocument);

                for (int i = 0; i < plugins.length; i++) {
                        ProxyFilter plugin = plugins[i];
                        if (!plugin.isEnabled())
                                continue;

                        if (currentDocument != null)
                                previousDocument = currentDocument;

                        if (previousDocument != null)
                                currentDocument =
                                        copyDocument(previousDocument);
                        currentDocument =
                                plugin.process(
                                        originalDocument,
                                        previousDocument,
                                        currentDocument);
                }

                if (currentDocument == null)
                        currentDocument = previousDocument;
                if (currentDocument == null)
                        currentDocument = originalDocument;

                return xMLtoFile(currentDocument);
        }
```

Content Extractor Plug-in

The Content Extractor Plug-in is the main filtering plug-in for Crunch 2.0. Its implementation is very similar to how it was in the original Crunch. This is possible even though quite a few things like the parser, etc were changed since it is all compliant with the W3C standards. The main changes were optimization, bug fixing, and working it into the new interface.

When process(Document, Document, Document) is called on the content extractor plug-in, it creates a child ContentExtractor, and has that process the currentDocument. This allows the content extractor processing to be thread safe, which is important because the proxy is multithreaded. The processing begins with the filterNode(Node iNode) method being run on the document, which is the root node of the DOM tree.

Content Extraction Plug-in filterNode method

This is a typical set of recursive methods when working with DOM. Passing through every node is very simple. FilterNode(Node iNode) passes iNode through a set of filters. Then it determines whether to filter iNode's children based on the mCheckChildren variable, which the method passThroughFilters sets. The filterChildren(Node iNode) method takes a node and runs filterNode on each of its children. Running filterNode on the root of a DOM tree will result in all the nodes being filtered recursively. This process was smoothed out in Crunch 2.0.

```
private void filterNode(final Node iNode) {
        mCheckChildren = true;

        passThroughFilters(iNode);

        if (mCheckChildren)
                filterChildren(iNode);
}

private void filterChildren(final Node iNode) {
        if (iNode.hasChildNodes()) {
                Node next = iNode.getFirstChild();
                while (next != null) {
                        Node current = next;
                        next = current.getNextSibling();
                        filterNode(current);
                }
        }
}
```

Content Extraction Plug-in: Main Filtering Method - passThroughFilters method

PassThroughFilters(Node iNode) takes a node and determines what filters in the content extractor plug-in to run on it. MCheckChildren is changed to tell the recursive method not to check a given node's children. The first thing passThroughFilters(Node iNode) does is gather information about the node. Currently it gets the node's type, parent, and attributes. Then it runs filters based on the node type. Currently the only node type that it runs filters on are element nodes. Element nodes represent tags such as <BR> and <FONT>. Element nodes are filtered in several stages. The first stage is more information gathering. The node is checked for being a link and then if it is an image. This information is recorded, and then the node is passed through a second set of filters. The second set of filters only modifies element attributes. Currently the attributes that are modified are the width attributes of tables and table cells, and the style attributes of div elements. After the attributes are modified, the element is passed through filters that can delete element nodes. An example of a node to delete is an ad link. This code sequence worked well in the previous version so we stayed with it.

```
        private void passThroughFilters(final Node iNode) {
                //Check to see if the node is a Text node or an element node and
                //act accordingly
                int type = iNode.getNodeType();
                Node parent = iNode.getParentNode();

                //Get the attributes of the node
                NamedNodeMap attr = iNode.getAttributes();

                //Element node
                if (type == Node.ELEMENT_NODE) {

                        String name = iNode.getNodeName();
                        //=================================================================
                        // Set of conditions that just check the nodes without editing or
                        // deleting them
                        //=================================================================
                        //Any type of link is encountered
                        if (isLink(iNode))
                                recordLink(iNode);
                        if (isImage(iNode))
                                recordImage(iNode);

                        //=================================================================
                        // Set of conditions that edit the nodes but don't delete them
                        //=================================================================

                        //<TD|TABLE width=*> removes widths
                        if ((name.equalsIgnoreCase("TD") || name.equalsIgnoreCase("TABLE"))
                                && settings.ignoreCellWidth) {
                                if (hasAttribute(iNode, "width"))
                                        removeAttribute(iNode, "width");
                        } //if

                        //<DIV style=*> removes style
                        else if (
                                name.equalsIgnoreCase("DIV") && settings.ignoreDivStyles) {
                                if (hasAttribute(iNode, "style"))
                                        removeAttribute(iNode, "style");
                        } //if

                        //=================================================================
                        //Set of conditionals determining what to ignore and not to ignore
                        // (Conditions that DELETE nodes from the DOM tree)
                        //=================================================================
                        if (isAdLink(iNode) && settings.ignoreAds) {
                                parent.removeChild(iNode);
                                mCheckChildren = false;
                        }
                        //<TD> with Link/Text Ratio higher than threshold
                        else if (name.equalsIgnoreCase("TD") && settings.ignoreLinkCells) {
                                testRemoveCell(iNode);
                        }
                        //<A HREF> with no Images
                        else if (isTextLink(iNode) && settings.ignoreTextLinks) {
                                parent.removeChild(iNode);
                                if (settings.addLinksToBottom)
                                        enqueueLink(iNode);
                                mCheckChildren = false;
                        }
                        //<BODY>
                        else if (name.equalsIgnoreCase("BODY"))
                                mBodyNode = iNode;
                } //if (type == Node.ELEMENT_NODE)
        }
```

Example Check Methods

- *isLink(Node iNode)*: isLink checks to see if a Node is a link. First, it gets the node type and the node attributes. Then it checks to see if the node is an element and it contains an HREF attribute. If that is true, then it returns true indicating that the node is a link. Otherwise it returns false.

```
        private boolean isLink(final Node iNode) {
                int type = iNode.getNodeType();

                NamedNodeMap attr = iNode.getAttributes();

                if (type == Node.ELEMENT_NODE) {
                        String name = iNode.getNodeName();
                        if (name.equalsIgnoreCase("A")) {
                                for (int i = 0; i < attr.getLength(); i++) {
                                        if(attr.item(i).
                                          getNodeName().
                                          equalsIgnoreCase("HREF")) {
                                                return true;
                                        } //if
                                } //for
                        } //else if
                } //if

                return false;
        }
```

- *isImage(Node iNode)*: isImage checks to see if the node is an image.

```
        private boolean isImage(final Node iNode) {
                boolean image = false;

                //Check to see if the node is an image
                int type = iNode.getNodeType();
                if (type == Node.ELEMENT_NODE) {
                        if (iNode.getNodeName().equalsIgnoreCase("IMG"))
                                image = true;
                } //if

                return image;
        }
```

- *isImageLink(Node iNode)*: This method checks to see if a node is a link with an image as the link or if the node is an image, it checks if it is a link. First, it checks to see if the node is a link, and then it checks to see if any of its children are images. If that is true, then the method returns true, indicating the node is an image link. Second, it checks if the node is an image, and if its parent is a link. If this is the case, it will indicate that the node is an image link. Maps are also check for and treated as image links. Otherwise, it returns false.

```java
private boolean isImageLink(final Node iNode) {
        boolean imageLink = false;

        //Check to see if the node is a link
        if (isLink(iNode)) {

                //Check to see if the children have an image in it
                if (iNode.hasChildNodes()) {
                        Node next = iNode.getFirstChild();

                        while (next != null && !imageLink) {
                                Node current = next;
                                next = current.getNextSibling();
                                if (isImage(current))
                                        //imageLink = true;
                                        return true;

                        } //while
                } //if
        } //if
        //If the node is an image, check if its parent is a link
        else if (isImage(iNode)) {
                if (isLink(iNode.getParentNode()))
                        //imageLink = true;
                        return true;
                else {
                        // check for image maps
                        if (nodeContainsAttribute(iNode, "usemap"))
                                //imageLink = true;
                                return true;
                }

        } //else if

        return imageLink;
} //isImageLink
```

Acknowledgements

Prof. Kaiser’s Programming Systems Laboratory is funded in part by Defense Advanced Research Project Agency under DARPA Order K503 monitored by Air Force Research Laboratory F30602-00-2-0611, by National Science Foundation grants CCR-02-03876, EIA-00-71954, CCR-99-70790, and by Microsoft Research and IBM. Dr. Chiang is supported by grant LM07079 from the National Library of Medicine, and grant EY013972 from the National Eye Institute.

We would like to extend a special thanks to David L. Neistadt for support in developing Crunch 1.0, to David Kaufman, Vimla Patel, and Roy Cole for helpful discussions regarding the usability study, and to the subjects who generously volunteered their time to participate in the study.

References

[1] Aidan Finn, Nicholas Kushmerick and Barry Smyth. “Fact or fiction: Content classification for digital libraries”. In Joint DELOS-NSF Workshop on Personalisation and Recommender Systems in Digital Libraries (Dublin), 2001.

[2] A. F. R. Rahman, H. Alam and R. Hartono. “Content Extraction from HTML Documents”. In 1st Int. Workshop on Web Document Analysis (WDA2001), 2001.

[3] O. Buyukkokten, H. Garcia-Molina and A. Paepcke. “Accordion Summarization for End-Game Browsing on PDAs and Cellular Phones”. In Proc. of Conf. on Human Factors in Computing Systems (CHI'01), 2001.

[4] O. Buyukkokten, H, Garcia-Molina and A. Paepcke. “Seeing the Whole in Parts: Text Summarization for Web Browsing on Handheld Devices”. In Proc. of 10th Int. World-Wide Web Conf., 2001.

[5] E. Kaasinen, M. Aaltonen, J. Kolari, S. Melakoski and T. Laakko. “Two Approaches to Bringing Internet Services to WAP Devices”. In Proc. of 9th Int. World-Wide Web Conf., 2000.

[6] Stuart Hanzlik “Gorilla Design Studios Presents: The Hosts File”. Gorilla Design Studios. August 31, 2002. http://accs-net.com/hosts/.

[7] Marc H. Brown and Robert A. Shillner. “A New Paradigm for Browsing the Web”. In Human Factors in Computing Systems (CHI'95 Conference Companion), 1995.

[8] K.R. McKeown, R. Barzilay, D. Evans, V. Hatzivassiloglou, M.Y. Kan, B. Schiffman and S. Teufel. “Columbia Multi-document Summarization: Approach and Evaluation”, In Document Understanding Conf., 2001.

[9] N. Wacholder, D. Evans and J. Klavans. “Automatic Identification and Organization of Index Terms for Interactive Browsing”. In Joint Conf. on Digital Libraries ’01, 2001.

[10] O. Buyukkokten, H. Garcia-Molina and A. Paepcke. “Text Summarization for Web Browsing on Handheld Devices”, In Proc. of 10th Int. World-Wide Web Conf., 2001.

[11] Manuela Kunze and Dietmar Rosner. “An XML-based Approach for the Presentation and Exploitation of Extracted Information”. In 19th International Conference on Computational Linguistics, (Coling) 2002.

[12] A. F. R. Rahman, H. Alam and R. Hartono. “Understanding the Flow of Content in Summarizing HTML Documents”. In Int. Workshop on Document Layout Interpretation and its Applications, DLIA01, Sep., 2001.

[13] Wolfgang Reichl, Bob Carpenter, Jennifer Chu-Carroll and Wu Chou. “Language Modeling for Content Extraction in Human-Computer Dialogues”. In International Conference on Spoken Language Processing (ICSLP), 1998

[14] Ion Muslea, Steve Minton and Craig Knoblock. “A Hierarchal Approach to Wrapper Induction”. In Proc. of 3rd Int. Conf. on Autonomous Agents (Agents'99), 1999.

[15] Min-Yen Kan, Judith L. Klavans and Kathleen R. McKeown. “Linear Segmentation and Segment Relevance”. In Proc. of 6th Int. Workshop of Very Large Corpora (WVLC-6), 1998.

[16] http://www.opera.com

[17] http://www.bitstream.com/wireless

[18] http://sourceforge.net/projects/wpar

[19] http://www.webwiper.com

[20] http://www.junkbusters.com

[21] http://www.openxml.org

[22] Private communication, Min-Yen Kan, Columbia NLP group, 2002.

[23] http://www.apache.org/~andyc/neko/doc/html/

[24] http://www.eclipse.org/articles/Article-SWT-Design-1/SWT-Design-1.html

[25] http://www.eclipse.org/articles/Article-Accessibility/accessibility.html

[26] http://www.microsoft.com/enable/

[27] http://www.bitstream.com/wireless/server/workflow.html

[28] http://www.greenlightwireless.net/services/default.asp

[29] J. Nielsen. “Usability engineering.” New York: Academic Press, 1993.

[30] B. Schneiderman. “Designing the user interface: Strategies for effective human-computer interaction” (3rd edition). Reading, MA: Addison-Wesley, 1997.

[31] R. L. Kline and E. P. Glinert. “Improving GUI accessibility for people with low vision.” In Human Factors in Computing Systems (CHI’95 Conference Companion), 1995.

[32] W. K. Edwards, E. D. Mynatt, and K. Stockton. “Access to graphical interfaces for blind users.” Interactions 1995; 2: 54-67.

[33] I. U. Scott, W. J. Feurer, and J. A. Jacko. “Impact of graphical user interface screen features on computer task accuracy and speed in a cohort of patients with age-related macular degeneration.” Am J Ophthalmol 2002; 134: 857-862.

[34] D. P. Rice. “Chronic care in America: A 21st century challenge.” Institute for Health and Aging, University of California, San Francisco. Princeton, NJ: Robert Wood Johnson Foundation, 1996.

[35] American Foundation for the Blind. “Statistics and sources for professionals.” New York: American Foundation for the Blind, 2000.

[36] C. Brown. “Assistive technology computers and personal with disabilities.” Communications of the ACM 35: 36-45, 1992.

[37] I. J. Pitt, and A. D. N. Edwards. “Improving the usability of speech-based interfaces for blind users.” In Proceedings of the Second Annual ACM Conference on Assistive Technologies (ASSETS), 1996.

[38] C. Lewis. “Using the ‘thinking-aloud’ method in cognitive interface design.” IBM Research Report RC 9265. Yorktown Heights, NY: IBM Thomas J. Watson Research Center, 1982.

[39] K. A. Ericsson, H. A. Simon. “Protocol analysis: Verbal reports as data.” Cambridge, MA: MIT Press, 1993.

[40] A. W. Kushniruk, M. Y. Kan, K. McKeown, et al. “Usability evaluation of an experimental text summarization system and three search engines: Implications for the reengineering of health care interfaces.” Proc AMIA Symp 2002; : 420-424.

[41] A. W. Kushniruk, V. L. Patel, and J. J. Cimino. “Usability testing in medical informatics: Cognitive approaches to evaluation of information systems and user interfaces.” Proc AMIA Symp 1997; : 218-222.

[42] A. W. Kushniruk, D. R. Kaufman, V. L. Patel, et al. “Assessment of a computerized patient record system: A cognitive approach to evaluating medical technology.” MD Comput 1996; 13: 406-415.

[43] S. Brin S, and L. Page. “The anatomy of a large-scale hypertextual web search engine.” Computer Networks and ISDN Systems 1998; 30: 107-117.

[44] http://www.promotiondata.com/article.php?sid=190

[45] W. Chisolm, G. Vanderheiden, and I. Jacobs. “Web content accessibility guidelines 1.0.” Interactions 2001; 8: 35-54.

[46] http://www.dolphinuk.co.uk/products/hal.htm

[47] http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/winxppro/reader_overview.asp

[48] Chiang, Michael, “World Wide Web Accessibility by Visually Disabled Patients: Problems and Solutions” Final Report for CS6125 WHIM at Columbia University’s Computer Science Department.

[49] http://www.webaim.org/simulations/screenreader

[50] http://www-3.ibm.com/able/solution_offerings/hpr.html

[51] http://www.apache.org/

[52] Shoemaker JA: Vision problems in the US: prevalence of adult vision impairment and age-related eye diseases in America. Bethesda, MD: National Eye Institute, 2002

[53] http://www.avantbrowser.com

[54] http://www.mozilla.org

[55] Welsh, M. “The Staged Event-Driven Architecture for Highly-Concurrent Server Applications” Ph.D. Qualifying Examination Proposal, UC Berkeley, December 2000. http://www.cs.berkeley.edu/˜mdw/papers/quals-seda.pdf.

[56] Chen, Y., Ma, W.Y., and Zhang, H.J. “Detecting Web Page Structure for Adaptive Viewing on Small Form Factor Devices”. Proc. WWW’03 Budapest, Hungary, May 2003

[57] http://www.gnu.org/software/gcc/java/