GREEN WEB ENGINEERING

A Set of Principles to Support the Development and Operation of "Green" Websites and their Utilization during a Website's Life Cycle

Markus Dick, Stefan Naumann and Alexandra Held
University of Applied Sciences Trier, Umwelt-Campus Birkenfeld, Campusallee, 55768 Hoppstädten-Weiersbach, Germany

Keywords: Green Web Engineering, Green IT, Sustainable Development, Caching, Compression, Power Indicator.

Abstract: The power consumption of ICT and Internet is still increasing. To date, it is not clear if the energy savings through ICT overbalance the energy consumption by ICT, or not. In either case, it is suggestive to enforce the energy efficiency of the Web. In our paper, we present a set of 12 principles, which help e.g. to reduce the net load by caching or compressing. In order to classify our suggestions we group them by three main roles in Web Engineering. Additionally, we recommend using data centres which utilize "classic" Green IT.

1 INTRODUCTION

Global warming, greenhouse gas (GHG) effects, climate change, and sustainable development (SD) are key challenges of the 21st century (World Commission, 1991; Gore, 2006). Information and Communication Technology (ICT) takes a double-edged part within these challenges. On the one hand, ICT can optimize material flows and therefore reduce energy consumption (Hilty, 2005). On the other hand, ICT itself is consuming more and more energy: the energy consumption of IT and especially of the Web is still increasing (Erdmann et al. 2004). E.g. the estimated power consumption of data centres in the U.S. was increasing from 28 in 2000 to 61 billion kw/h in the year 2006 (U.S. EPA, 2007) and in the world from 58 billion kw/h in 2000 to 123 billion kw/h in 2005 (Koomey, 2007). Consequently, considering problems like climate change, reducing energy consumption of the Web is necessary.

Our paper discusses some possibilities to reduce especially the data transfer of the net from a technical point of view and suggests a classification through a conceptual framework for existing and prospective solutions. From a theoretical point of view, our approach can be classified into the new research field of "Sustainability Informatics" (Naumann, 2008a).

2 BACKGROUND

Up to now, several publications examine the relationship between the field of sustainability and ICT. They discuss the impact of IT on the environment (Göhring, 2004; Hilty, 2008) or consider the balance between energy savings and energy consumptions by ICT (Coroama & Hilty, 2009). Especially, to date it is not clear whether energy consumption by ICT is greater or smaller than energy savings by ICT, e.g. because of more efficient processes or simulations of scenarios.

Regarding "Green Web Engineering" the field of "Green IT" with suggestions of virtualization, green data centres etc. is covered amongst others by Hird (2008) and Velte (2008). Especially a tool to visualize, whether a website is hosted with renewable energies is described in Naumann (2008b). An approach, which tries to calculate the environmental impact of a website, is the CO2-Stats-Project (http://www.co2stats.com). However, since this approach is valuable, it seems not clear to the public how the measurement is calculated in detail.

Hilty (2008) gives some hints for software developers which can be partly applied to Web developers. Abenius (2009) introduces a model for green software which comprises aspects like speed, energy modelling and existing tools.

Dick M., Naumann S. and Held A.
GREEN WEB ENGINEERING - A Set of Principles to Support the Development and Operation of âĂIJGreenâĂİ Websites and their Utilization during a WebsiteâĂŹs Life Cycle.
DOI: 10.5220/0002805200480055
In *Proceedings of the 6th International Conference on Web Information Systems and Technology (WEBIST 2010)*, page
ISBN: 978-989-674-025-2

3 GREEN WEB ENGINEERING AND THE LIFE CYCLE OF WEBSITES

Firstly, we underlay our conceptual framework with a definition: "*Green Web Engineering* describes the art of developing, designing, maintaining, administrating, and using a website in such a manner, that direct and indirect energy consumption within the complete life cycle of a website is reduced."

The aim of our framework is to reinforce Green Web Engineering. In order to classify our suggestions we distinguish different roles within a website life cycle.

Designers are developers, web designers, or authors. Here, we understand *developers* as "classic" software developers, who program server-side applications, JavaScript, AJAX etc., or fulfil other more technical oriented tasks of website developing. *Web designers* are responsible for the screen design, decide on layout and colours, and develop screen flows. *Authors* maintain a website by generating content, deciding about the content structure, giving permissions to other authors etc.

Administrators maintain a website from a technical point of view, allocate space and supply virtual machines, configure ports etc. *Website administrators* maintain a dedicated website and are responsible (in conjunction with the authors) for access permissions.

Users of a website are all people, who have access to the site and obtain information, communicate with other users etc.

4 A SET OF 12 GREEN WEB ENGINEERING PRINCIPLES

In this subsection we present 12 principles how to develop, maintain, and use a green website regarding our definition of "Green Web Engineering". Our suggestions are structured through our classification of typical user roles within the website life cycle.

4.1 Suggestions for Administrators

4.1.1 Configure HTTPs Caching Support

Since the configuration of the user's web browser cannot be affected by web designers or administrators, they have to focus on the server-side configuration aspects of caching. Where approx. 80% of the web users have their browsers configured for caching, however 20% always have an empty cache (Theurer, 2007). The inclusion of caching metadata by the web server will significantly decrease the amount of HTTP requests and HTTP responses. Caching in HTTP/1.1 is designed to reduce

- the need to send requests to servers ("expiration" mechanism) and
- the need to send full responses back to the clients ("validation" mechanism).

The validation mechanism does not reduce the amount of HTTP-requests but it reduces the payload of the HTTP responses that are sent back to the client and thus addresses network bandwidth reduction (Fielding et al., 1999:74).

In order to facilitate the expiration mechanism of HTTP servers, administrators can specify an Expires or Cache-Control header in their response. The Expires header as described with HTTP/1.0 (Berners-Lee, 1996:41) defines the absolute date after which the response is expected to be stale. One minor problem with the Expires header is that it uses explicit date and time strings and thus requires server and client clocks to be synchronized (Crocker, 1982:26; Souders, 2007:22). Beginning with HTTP/1.1 that limitation has been overcome with the Cache-Control header. It uses the max-age directive to define the seconds, which the requested resource may remain in the cache. To stay compatible with older HTTP clients that do not support the HTTP/1.1 protocol, one can define the Expires header alongside with the Cache-Control header. In that case the Cache-Control header overrides the Expires header.

The validation mechanism is utilized by HTTP clients that have a resource entry in their cache that has already expired. In that case the client may send a conditional HTTP request to the server. This conditional HTTP request looks exactly like a normal HTTP request but in addition it carries a so called "validator" that may be used by the server to decide whether the resource requested by the client is still up to date or needs to be refreshed. In case that it needs a refresh the new data is sent to the client, otherwise the server responds with the HTTP status code "304 Not modified". There are two validators that may be used: Last-Modified dates or Entity Tag cache validators. In case of the Last-Modified date a cache entry is considered to be valid if the requested resource has not been modified since the given Last-Modified date. An Entity Tag is a unique identifier for a specific version (entity) of a resource (Fielding et al., 1999:85). The calculation of the Entity Tags depends on the implementation of the web server. The HTTP/1.1 specification states that servers should send both, Entity Tag and Last-Modified values in their responses. HTTP/1.1 clients are forced by the specification to use Entity Tags in

cache-conditional requests if provided by the server. In addition clients should also apply a Last-Modified date if one was set (Fielding et al., 1999:86).

In order to reduce the total amount of HTTP requests or HTTP payload sizes we suggest configuring the client cache support properly. This means:

1. setting far future expiration dates and cache-control headers for resources that infrequently change
2. setting Last-Modified headers and Entity Tags for all resources that do not need recalculation on subsequent requests (mainly static content)

A simple example configuration fragment for the popular Apache web server may look like this:

```
ExpiresActive On
<FilesMatch "\.(html|jpg|png|js|css)$">
   ExpiresDefault "access plus 355
   days"
   FileETag MTime Size
</FilesMatch>
```

The Apache configuration directive ExpiresDefault handles both, the generation of an Expires header and the generation of a Cache-Control header for the given resource types.

4.1.2 Use Compression

Today, many modern web browsers support some kind of compression. Compression reduces not only response size and thus transfer time, but also power consumption as a result of the smaller size and shorter transfer times.

Web browsers usually support the GZIP compression format or DEFLATE. Both formats are especially named in the HTTP/1.1 specification. The Accept-Encoding header is used by web browsers to indicate which content encodings they support. A web server may compress the content using one of the compression methods listed by the browser and must notify the browser within the Content-Encoding response header which compression method is used (Fielding et al., 1999).

Compression is not as simple as it seems, because there are some older browser versions that claim that they support compression but actually do not, because of incompatibilities or bugs. On the other hand more than 95% (ADTECH, 2008) of all installed and used browsers in Europe are known to support GZIP compression. Therefore regarding our goal of “Green Web Engineering” it is reasonable to enable compression on the server side.

However, not all content types are suitable for compression, e.g. compressed image file formats, compressed music and video files or PDF documents. Compressing these file types is sometimes even counterproductive. Hence, compression should be used for files that are well compressible like text based files. With a simple example website that was compressed via GZIP compression with an Apache web server, we achieved traffic savings as shown in table 1. The average saving for the whole example site is approx. 60% (assuming PNG-images are not compressed).

Table 1: Total vs. compressed filesize example.

Example content	Total size (KB)	Compressed size (KB)	Savings
index.html	5.45	2.44	55.3%
style.css	2.73	0.68	75.1%
prototype.js	126.00	29.51	76.6%
ida-logo.png	24.80	24.86	-0.2%
ucb-logo.png	9.27	9.28	-0.1%

Here we see a simple example configuration fragment for the Apache web server using the compression module for text based document types:

```
AddOutputFilterByType DEFLATE text/html
AddOutputFilterByType DEFLATE text/css
AddOutputFilterByType DEFLATE
   application/javascript
```

4.1.3 Apply “Green IT” Concepts

So far, several web hosters exist that offer web hosting with renewable energy. Additionally, administrators can apply the newest techniques regarding Green IT like virtualization strategies.

4.2 Suggestions for Developers, Designers and Authors

4.2.1 Support Caching on the Server Side

As described earlier in section 4.1.1 caching capabilities of HTTP are designed to reduce the amount of HTTP requests and transferred data volume significantly. Up to now we examined only web server configurations that handle static content. Content that is dynamically generated by scripts or programs on the server side is usually not treated to be cacheable and therefore not tagged with caching metadata by default.

However, there may be cases where dynamic content changes less frequently than once for each request. Therefore caching it may be possible without the risk of presenting obsolete data to visitors. In our opinion the following tasks are important:

- retrieving presented content dynamically out of content repositories
- presenting dynamically generated charts as images

Concerning the first point, well-engineered content management systems may be configured to use some kind of caching strategy and to support Expires and Last-Modified headers. Hence, we focus on websites or web applications that are not run via a Content Management System (CMS), but are implemented directly in scripts and programs. For instance, assuming a web based application that collects power meter readings of a family home, calculates its power consumption daily and presents it in charts on a website, the Last-Modified date can be set to the date and time where the last meter reading has been inserted into the database.

We recommend that web developers implement support for Expires and Last-Modified headers in their web programs and scripts whenever possible, to decrease the transferred data volume and thus power consumption. The following example code shows how this may be realized in principle, with the PHP scripting language. A more sophisticated example is given by Nottingham (2009).

```
header('Expires: ' . toGMT($expires));
header('Last-Modified:
'.toGMT($lastMod));
if(!isModifiedSince($lastMod)) {
  header('HTTP/1.1 304 Not Modified');
  exit();
}
echo 'ANY CONTENT';
```

At first, the Expires header is set to a date that is acceptable for the given content type, to assume its freshness. The Last-Modified header is set with a value that indicates when the underlying content was modified last. After setting the headers, the Last-Modified-Since header of the HTTP request is compared against the Last-Modified date of the content. If the content of the browser cache is still valid, the HTTP Not-Modified status is sent. Otherwise the content is sent with an HTTP Ok status.

4.2.2 Minimize JavaScript

JavaScript is used widely on websites to enhance interactivity. JavaScript code fragments are often embedded directly in HTML or in specific JavaScript files. The former increases the response size, whereas the latter increases the amount of HTTP requests especially when more than one JavaScript file is used in a website.

To minimize HTML content size we suggest externalizing any JavaScript code fragments embedded in HTML into special JavaScript files. On the one hand, this results in an extra HTTP request, but on the other hand it

1. enables browsers to cache these files and to provide them in subsequent requests without calling the origin server
2. reduces the HTML download size especially when the HTML content can not be cached because it must be recalculated for each request.

If JavaScript code resides in dedicated files, we recommend defining distinct Expires header rules in web server configurations, so that these files expire further in the future than the HTML content files do (assuming that HTML content changes more frequent than the technical framework).

If more than one JavaScript file is used in a website, it may save some HTTP requests if these files are merged into one file. Despite increasing the JavaScript filesize, the number of transferred bytes is a little bit smaller, because only one HTTP request must be sent and probably revalidated when users are pressing the reload button in their web browser.

The JavaScript filesize can be significantly minimized by removing all unnecessary whitespace characters or comments. It can be further reduced by removing all functions, methods or code blocks that are not needed within a particular website. This is true for websites that use only a few functions or methods of huge JavaScript libraries.

Another technique is obfuscation. Most available tools first apply the minimization technique and than shrink the code by substituting variable, method or function identifiers and also method and function parameter names with shorter ones. Self defined API functions referenced within HTML code must remain untouched, of course.

Table 2: Minimization vs. obfuscation vs. compression.

prototype-1.6.0.3.js	Uncompressed (KB)	GZIP compressed (KB)
Total size	126.70	28.49
Minimized size JSMin	93.09	22.71
Obfuscated size Dojo ShrinkSafe	87.66	24.94

Table 2 compares the effectiveness of the minimization tool JSMin (http://javascript.crockford. com/) against that of the obfuscation tool ShrinkSafe (known from the popular Dojo Toolkit

http://www.dojotoolkit.org/) while minimizing the well known JavaScript library Prototype (http://www.prototypejs.org/). As one may notice, obfuscation is more effective than minimization in the case of special tools used here. This results from the fact that minimization does not substitute any function identifiers with shorter names as the default obfuscation configuration does. In contrast to GZIP compression the overall savings of minimization or obfuscation are less effective (and in that special case of obfuscation even counterproductive, because there are fewer repetitions left in the JavaScript code that may possibly lead to better compression results). Further comparisons with JavaScript libraries lead to similar results (Dojo Toolkit, script.aculo.us).

Against this background we recommend to minimize JavaScript with a tool like JSMin and to compress the HTTP response with GZIP to achieve lower data transfer volumes. We recommend using obfuscation in scenarios where compression can not be used (e.g. if HTTP clients are not fully compatible with compression algorithms).

Additionally, we recommend minimizing or obfuscating the files before they are deployed on the server instead of minimizing or obfuscating them dynamically for each HTTP request to save processor time and thus power consumption.

4.2.3 Minimize and Optimize CSS

Minimizing and optimizing CSS files offers many possibilities to reduce HTTP requests. Savings from 40% to 60% are possible, if HTML or JavaScript operation is implemented with CSS (King, 2008: 177). Through a layout, which is designed with CSS, it is possible to save 25% to 50% of HTML filesize (King, 2008: 180).

Table 3 shows the filesize of an exemplary CSS stylesheet after optimization and minimization with different strengths.

Table 3: Comparison of CSS filesizes after minimization.

Without minimization	2,812 kb
With standard minimization	2,181 kb
With high minimization	2,040 kb
With highest minimization	2,021 kb

These results were obtained with CSSTidy (http://csstidy.sourceforge.net/). This tool uses some techniques to optimize CSS code in different ways and strengths. In this case, the standard, high and highest minimizations were tested.

Standard minimization means that the compression goal is to reach a balance between size and readability. With this minimization, the file is 631 kb smaller and the compression ratio is 22.4%. One technique the tool uses are abbreviations, for example with colour definitions. The long definition `{color:#ffcc00;}` can be written as `{color:#fc0;}`.

The second minimization is “higher”; the readability should be moderate and the size small. The example file was reduced with this compression by 27.5%. In addition to the other techniques unnecessary whitespaces, strings and blank lines are deleted. This may constrain the readability but reduce filesize.

The third minimization is the “highest”. Its main goal is to achieve the smallest filesize. The readability is not considered. The compression ratio is the highest: 28.1%. In this case, additionally word-wraps are deleted. The whole code therefore is in one line.

There are further techniques, which optimize CSS. Examples are to replace inline styles with type selectors, to remove unnecessary code or to use descendant or contextual selectors instead of inline classes. CSS provides grouping multiple selectors with the same declaration and grouping multiple declarations with the same selector in one rule set. Duplicate element declarations can be avoided through the use of inheritance. Some separated property declarations can be combined in one single declaration. `Margin`, `border`, `font`, `background`, `list-style` and `outline` are special shorthand notations which combine separate single properties. Names of classes or IDs should be short or abbreviated. Long names are only important for developers, not for users. On the other hand, short names should be meaningful for developers in the context, too.

The CSS declarations should generally be saved in an external file and then included in the website documents. This minimizes the filesize of the web document and the CSS declarations will be loaded once and not with every request of the website.

CSS can also be used to replace less efficient code. One example is the table definition in HTML, which can be replaced by CSS. Another point is the usage of embedded JavaScript, as 84.4% of web pages use it. These methods can often be replaced with more efficient CSS code. Examples are drop-down menus and the rollover effect (King, 2008: 177).

4.2.4 Optimize Graphical Elements and Logos

54% of an average website is graphics (King, 2008: 157). Because of this, optimizing images is a promising way to minimize HTTP responses.

A logo, unlike a picture is a graphical object, which symbolizes mostly a company or an organisation. The logo is normally included many times in a website. Thus, optimizing the logo can be more efficient than optimizing a picture, which is mostly shown once on a website.

Typically a logo is not a photograph or chart and in comparison to these smaller, so it can be saved as a GIF or PNG file. PNG has a better compression algorithm and especially flat-colour images are 10% to 30% smaller than in GIF (King, 2008:169).

There are some possibilities to optimize a logo. The most efficient ones are: changing the colour mode from RGB to an indexed palette, compressing, saving with special web settings and replacing graphical text with CSS formatted text.

The following example shows how a logo can be optimized. The logo consists of a graphical object, a background with colour gradient and graphical text (see Figure 1). The first step is to reduce the colours from RGB to an indexed palette. In this special case the colours can be reduced to six. Table 4 shows the filesize after optimization.

Table 4: Comparison of PNG filesizes after optimization.

Original logo with RGB colours	16,959 bytes
Logo with 16 colours	2,956 bytes
Logo with 6 colours	1,984 bytes
Logo with 4 colours without text	604 bytes

Figure 1: Original logo with RGB colours.

The next step is the transformation from graphical text to CSS text. First, the graphical text will be deleted from the image. Afterwards, the text will be included in the image with CSS code, which defines a layer that has the logo as a background image. The dimensions are equal to the image dimensions. The CSS code for the example is as follows:

```
logo { margin: 0px; width: 370px;
  height: 64px; padding: 10px 70px;
  font:bold 19px Arial, sans-serif;
  background-image:url(logo.png);
  background-repeat:no-repeat; }
```

The final logo with CSS formatted text is shown in Figure 2. Looking at this example the filesize can be reduced from 16,959 bytes to 604 bytes (96% reduction).

Figure 2: Final logo with 4 colours and CSS font.

Single-pixel, plain-coloured images, e.g. GIFs, are often used to achieve a certain stretching distance. These images can be replaced with CSS-code, where spacing cells have the same function as the images. This causes the layout to be the same, but through the usage of CSS the download of the images is not necessary anymore.

Images, which are lying relatively close to each other, can be combined into one picture. Additionally, links on the single pictures can be set with an image-map (King, 2008:165). Another possibility to combine images is to use CSS sprites. Sprites can be very effective, if many small images like icons are on a website. The required part of the whole image will be shown with the background-position rule at the designated position on the website (Souders, 2007).

4.2.5 Optimize Photographs

The filesize of a photograph is mostly greater than that of a logo. Consequently, there are more options to reduce the filesize.

Generally spoken, images should be stored with special web settings and the dimensions should be equal to the desired dimensions on the website.

A special technique to reduce the filesize is blurring the background. The blur technique can be used for images, where some parts are not as important as others (King, 2008:169).

Figure 3: File with 100% quality.

Figure 3 shows the image with 100% JPEG quality and no blur, the filesize is 126,246 bytes. After blurring the background with strength 8, the filesize is

only 18,162 bytes; this image is shown in figure 4. It shows that the person in the foreground is in a satisfying quality whereas the less important background is blurred. But still, the main image information is conveyed.

Figure 4: File with 80% quality and blurred background.

4.2.6 Optimize Videos and Animations

Multimedia files like videos account for 98.6% of the transferred bytes in the Internet (King, 2008:159). Hence, minimizing multimedia files can reduce traffic. Basic techniques are editing with special web settings, using special codecs, reducing the frame rate and using professional equipment

Other types of multimedia files are Flash animations. These can also be reduced and optimized. Here, non-optimized images and too many animation frames are the problems. The pictures should be optimized with image processing programs before they are imported in Flash. Animations should have a reduced frame rate (King, 2008:170).

4.3 Suggestions for Users

4.3.1 Configure the Web Browser

The former sections focused on the web designers' and administrators' options to reduce network traffic and thus power consumption. As was mentioned, administrators have no possibility to influence the browser configuration directly, like e.g. forcing the browser to cache delivered content according to the corresponding HTTP response header fields. Web users can also contribute a large part to the reduction of network traffic and power consumption by configuring the caching capabilities of their web browsers in a way that far future expiry dates can take effect, or by installing web browsers that are fully compatible with GZIP based HTTP compression. Furthermore, there is a lot of advertisement on web pages today that causes additional, but avoidable network traffic.

Thus, we recommend that one should

- configure large caches in Web browsers
- not clear caches during browser shutdown
- use browsers supporting GZIP compression
- use advertisement blockers to block advertisement images and Flash

4.3.2 Visualize Green Power Status of a Website

In order to assess whether a website is hosted with renewable energies, tools like the "Power Indicator" (Naumann, 2008b) can help users by visualizing the "green power state" of a website.

4.3.3 Apply "Green IT" on the Client Side

Of course, users of a website also have the possibility to obtain renewable energies within their companies or households. It is also recommended to utilize the available energy management options of client computers.

5 SUMMARY AND OUTLOOK

The power consumption of ICT and the Internet is still increasing. However, currently it is not clear whether energy savings through ICT overbalance its energy consumption, or not. In either case it is rational to enhance the energy efficiency of the Web.

In our paper we presented a set of 12 principles, which help e.g. to reduce the net load through caching or compressing. In order to classify our suggestions we grouped them by three main roles in Web Engineering: developers, administrators, and users of websites. Additionally, we recommend using data centres, which use renewable energies and follow the principles of classic Green IT.

Further evaluations have to be done that compare the overall traffic and processing time savings of a website if our principles are applied in different combinations or not. Therefore our current studies are focussing on measurement environments and approximations that allow us to draw conclusions on the resulting power savings from a local and probably also global point of view.

Our next steps are to detail and to broaden our model. More details can help to find solutions for more special problems regarding energy efficiency. A generic process model for "Green Web Engineering" is also reasonable and can be integrated in standardized models of Software Engineering. The broadening comprises e.g. aspects like barrier-free

techniques, since the idea of sustainable development contains also aspects of sociability and free access to common goods.

Additionally, it is worth to examine and evaluate our framework in detail and to suggest a ranking of "best practices".

ACKNOWLEDGEMENTS

This paper evolved from the research project "Green Software Engineering" funded by the German Federal Ministry of Education and Research.

REFERENCES

Abenius, S., 2009. Green IT & Green Software – Time and Energy Savings Using Existing Tools. In: Wohlgemuth, V., Page, B. & Voigt, C. eds. 2009. *Environmental Informatics and Industrial Environmental Protection: Concepts, Methods and Tools. 23th International Conference on Informatics for Environmental Protection*. Volume 1. Aachen: Shaker Verlag, pp. 57-66.

ADTECH, 2008. *Survey Unveils Extent of Internet Explorer Domination Across the European Browser Landscape*. [Online] ADTECH: London. Available at: http://www.adtech.com/news/pr-08-07_en.htm [Accessed 10 Oct. 2009].

Berners-Lee, T., Fielding, R. & Frystyk, H., 1996. *Hypertext Transfer Protocol -- HTTP/1.0*. Request for Comments 1945. [Online] Network Working Group. Available at: http://tools.ietf.org/html/rfc1945 [Accessed 10 Oct. 2009].

Coroama, V. & Hilty, L.M., 2009: Energy Consumed vs. Energy Saved by ICT – A Closer Look. In: Wohlgemuth, V., Page, B. & Voigt, C. eds., 2009. *Environmental Informatics and Industrial Environmental Protection: Concepts, Methods and Tools. 23th International Conference on Informatics for Environmental Protection*. Volume 2. Aachen: Shaker Verlag, pp. 353-361.

Crocker, David H., 1982. *Standard for the Format of ARPA Internet Text Messages*. Request for Comments 822. [Online] University of Delaware. Available at: http://tools.ietf.org/ html/rfc822 [Accessed 10 Oct. 2009].

Erdmann, L., Hilty, L.M., Goodman, J. & Arnfalk, P., 2004. *The Future Impact of ICTs on Environmental Sustainability. Technical Report Series EUR 21384 EN*. [Online] Brussels: IPTS. Available at: http://ftp.jrc.es/EURdoc/eur21384en.pdf [Accessed 3. Oct. 2009].

Fielding, R.; Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. & Berners-Lee, T., 1999. *Hypertext Transfer Protocol -- HTTP/1.1*. Request for Comments 2616. [Online] The Internet Society. Available at: http://tools.ietf.org/html/ rfc2616 [Accessed 10 Oct. 2009].

Göhring, W., 2004. The Memorandum "Sustainable Information Society". In: *18th International Conference on Informatics for Environmental Protection, Geneva 2004*.

Gore, Al, 2006. *An Inconvenient Truth*. Emmaus: Rodale.

Hilty, L.M., 2008. *Information Technology and Sustainability*. Books on Demand GmbH: Norderstedt.

Hilty, L.M., Seifert, E.K. & Treibert; R. eds., 2005. *Information Systems for Sustainable Development*. Hershey: Idea Group Publishing.

Hird, G., 2008. *Green IT in Practice: How One Company Is Approaching the Greening of Its IT*. Cambridge: IT Governance Publishing.

King, A., 2008. *Website Optimization*. 1st ed. Sebastopol: O'Reilly Media.

Koomey, J.G., 2007. *Estimating total Power Consumption by Servers in the U.S. and the World. Final report, February 15, 2007*. [Online] Analytics Press: Oakland. Available: http://enterprise.amd.com/ Downloads/svrpwrusecompletefinal.pdf [Accessed: 4 Oct. 2009].

Naumann, S., 2008a. Sustainability Informatics – A new Subfield of Applied Informatics? In: Möller, A., Page, B. & Schreiber, M. eds., 2008. *EnviroInfo 2008. Environmental Informatics and Industrial Ecology, 22nd International Conference on Environmental Informatics*. Aachen: Shaker Verlag, pp. 384-389.

Naumann, S., Gresk, S. & Schäfer, K., 2008b. How Green is the Web? Visualizing the Power Quality of Websites. In: Möller, A., Page, B. & Schreiber, M. eds., 2008. *EnviroInfo 2008. Environmental Informatics and Industrial Ecology, 22nd International Conference on Environmental Informatics*. Aachen: Shaker Verlag, pp. 62-65.

Nottingham, M., 2009. *Caching Tutorial*. [Online] Available at: http://www.mnot.net/cache_docs/ [Accessed 10 Oct. 2009].

Souders, S., 2007. *High Performance Web Sites*. Sebastopol: O'Reilly Media.

Theurer, T., 2007. *Performance Research, Part 2: Browser Cache Usage - Exposed!* [Online] Available at: http://www.yuiblog.com/blog/2007/01/04/ performance-research-part-2/ [Accessed 10 Oct. 2009].

U.S. EPA (U.S. Environmental Protection Agency, Energy Star Program), 2007. *Report to Congress on Server and Data Center Energy Efficiency, Public Law 109-431, August 2, 2007*.

Velte, T. , Velte A. & Elsenpeter, R., 2008. *Green IT: reduce your information system's environmental impact while adding to the bottom line*. New York: McGraw-Hill.

World Commission on Environment and Development ed., 1991. *Our common future*. Oxford: Oxford University Press, 13th edition.