Chapter 4

Automated Analysis Methodology and Tools

4.1 Introduction

This chapter presents an automated methodology for analyzing Web interfaces that leverages the benefits of both performance evaluation and usability evaluation as discussed in the previous chapters. In the spirit of measurement techniques in the performance evaluation domain, benchmarking in particular, the methodology entails computing an extensive set of quantitative page-level and site-level measures. These measures can be viewed as a benchmark because they enable objective comparison of Web interfaces. Specifically, these measures are used to derive statistical models of highly-rated interfaces. As is done with guideline review methods in the usability evaluation domain, the models are then used in the automated analysis of Web pages and sites. Unlike other guideline review methods, the guidelines are in essence derived from empirical data. Hence, it is also possible to use the models to validate or invalidate many guidelines proposed in the Web design literature.

This chapter and the subsequent two chapters detail the steps followed in this dissertation towards developing an automated analysis method for Web interfaces. First, this chapter presents two analysis scenarios as motivation for the methodology. Then, the methodology and tools are discussed. Chapter 5 summarizes an extensive survey of Web design literature culminating in the development of 157 page-level and site-level quantitative measures; these measures are computed by the Metrics Computation Tool discussed in this chapter. These measures are then used to develop statistical models of highly-rated Web interfaces in Chapter 6; these statistical models are incorporated into the Analysis Tool discussed in this chapter.

4.2 Analysis Scenarios

4.2.1 Web Interface Evaluation

As background for the methodology presented in this chapter, Figure 4.1 depicts an analysis scenario: a Web designer seeking to determine the quality of an interface design. If the site has already been designed and implemented, the designer could use the site as input to an analysis tool. The analysis tool (or benchmark program) would then sample pages within the site and generate a number of quantitative measures pertaining to all aspects of the interface. As discussed in Chapter 2, a key component of benchmarking is the ability to determine how well benchmark results com-

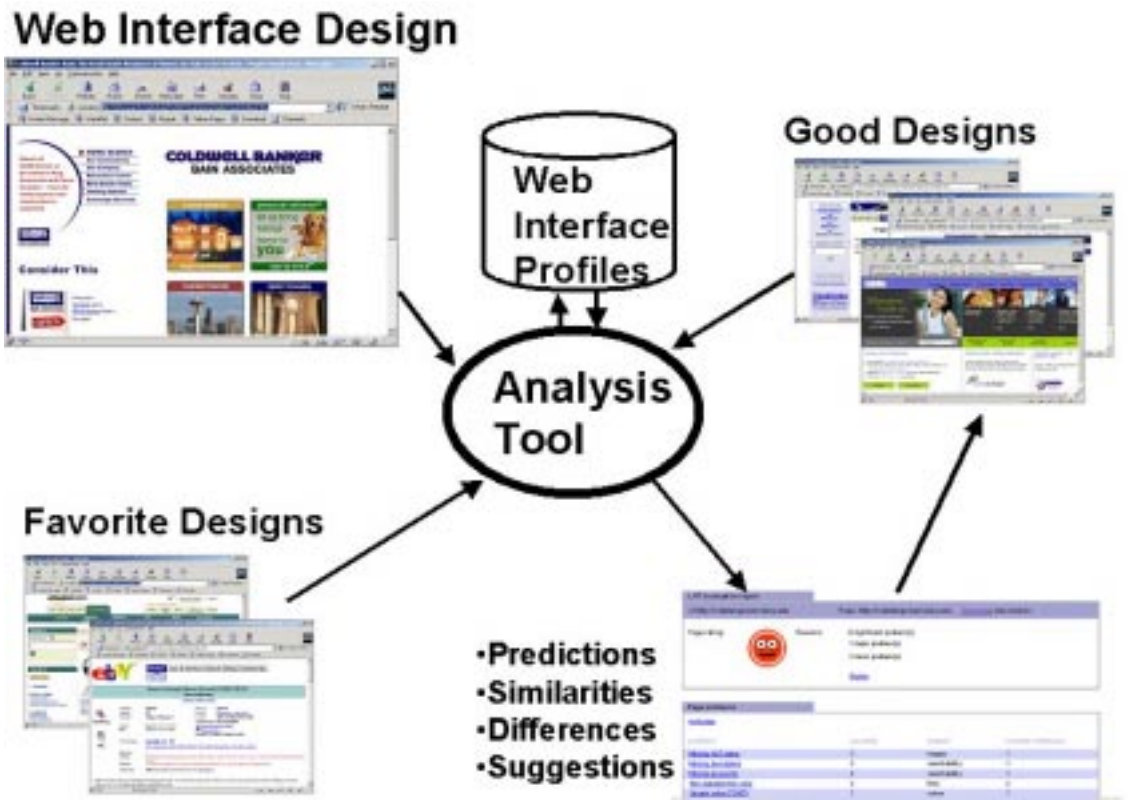

Figure 4.1: A Web interface evaluation scenario.

pare to other systems or a best case. For this scenario, designs that have been rated favorably by expert reviewers or users could be used for comparison purposes. Hence, the analysis tool could compare the input design's quantitative measures to those for highly-rated designs. Ideally, the analysis tool goes beyond traditional benchmarking and generates a report containing an interface quality or usability prediction, links to similar highly-rated designs from the comparison sample, differences and similarities to the highly-rated designs, and specific suggestions for improvements. The designer could use these results to choose between alternative designs as well as to inform design improvements. This analysis process could be iterated as necessary.

Similarly, the designer could use the analysis tool to explore results for other interface designs, such as favorite sites. This process may help to educate the designer on subtle aspects of design that may not be apparent from simply inspecting interfaces. The methodology and tools developed in this dissertation were designed to support many aspects of this Web interface evaluation scenario. Currently, recommending design improvements and presenting comparable designs are not supported; future work will focus on these aspects. However, all other aspects of the scenario are fully supported.

4.2.2 Web Interface Design

One can also imagine that a designer would want to obtain feedback on interface designs earlier during the design phase as opposed to after implementation. If the tool supported analysis of designs represented as images or templates, then it would be possible to support this evaluation. In particular, the tool needs to use image processing during analysis. One could also imagine supporting the designer even earlier in the design process, such as during the design exploration

and refinement phases [Newman and Landay 2000]. Given a large collection of favorably-rated sites, the designer could explore this collection to stimulate design ideas similarly to practices employed in the architecture domain [Elliott 2001]. During the design exploration phase, the designer could look for ways to organize content within health sites as well as navigation schemes, for example. During the design refinement phase, the designer may look for good page layouts, color palettes, site maps, navigation bars, form designs, etc.

Ideally, characteristics of Web pages and sites can be represented in a way to facilitate easily identifying pages and sites that satisfy queries similar to the ones above. Task-based search techniques that exploit metadata [Elliott 2001; English *et al.* 2001; Hearst 2000] should be helpful; Hearst [2000] proposes an approach wherein search interfaces present users with metadata facets for refining search results. Elliott [2001] presents a similar approach for exploring large online collections of architecture images; metadata describes the content of images, including location, architect, style, and kind of building. Metadata for Web interfaces could consist of the quantitative measures developed in this dissertation (discussed in Chapter 5) as well as others that describe for instance the size of the site, the type of site, page size, a page's functional type, elements on a page (e.g., navigation bars), as well as site ratings. Many of these measures could be computed automatically by the analysis methodology presented in this chapter. These measures could be organized into metadata facets. Scapin *et al.* [2000] presents a useful and relevant framework for organizing Web guidelines that includes a taxonomy of index keys (e.g., alignment, buttons, downloading, headings, language, scrolling, navigation structure, and so on); this taxonomy could be used to organize the measures into metadata facets.

4.3 Methodology

Figure 4.2 depicts the methodology developed to support the interface evaluation scenario. Earlier work on this methodology was published in [Ivory *et al.* 2000; Ivory *et al.* 2001]. The approach was developed mainly for information-centric Web interfaces (sites whose primary tasks entail locating specific information) as opposed to functionally-oriented interfaces (sites wherein users follow explicit task sequences). However, this approach could be used to some degree for functionally-oriented interfaces, since these interfaces typically present information as well.

The analysis methodology consists of two distinct but related phases: 1. establishing an interface quality baseline; and 2. analyzing interface quality. Both phases share common activities: crawling Web sites to download pages and associated elements (Site Crawling); and computing page-level and site-level quantitative measures (Metrics Computation). During the first phase, the page-level and site-level measures coupled with expert ratings of sites are analyzed to determine profiles of highly-rated interfaces. These profiles encapsulate key quantitative measures, thresholds for these measures, and effective relationships among key measures; thus, representing an interface quality baseline. During the second phase, the page-level and site-level measures are compared to the developed profiles and are used to assess interface quality. Sites analyzed in the latter phase are usually not the same as the sites used to develop profiles. Once profiles are developed, the analysis phase can be used on an ongoing basis. However, the interface quality baseline phase needs to be repeated periodically (annually or semi-annually) to ensure that profiles reflect current Web design practices.

This analysis methodology is consistent with other guideline review methods discussed in Chapter 2. It is also consistent with benchmarking methods discussed in Chapter 3. Specifically, it includes a synthetic benchmark that mimics a Web browser loading Web pages and uses an internal driver (i.e., one program for loading and running the workload). It also includes an automated

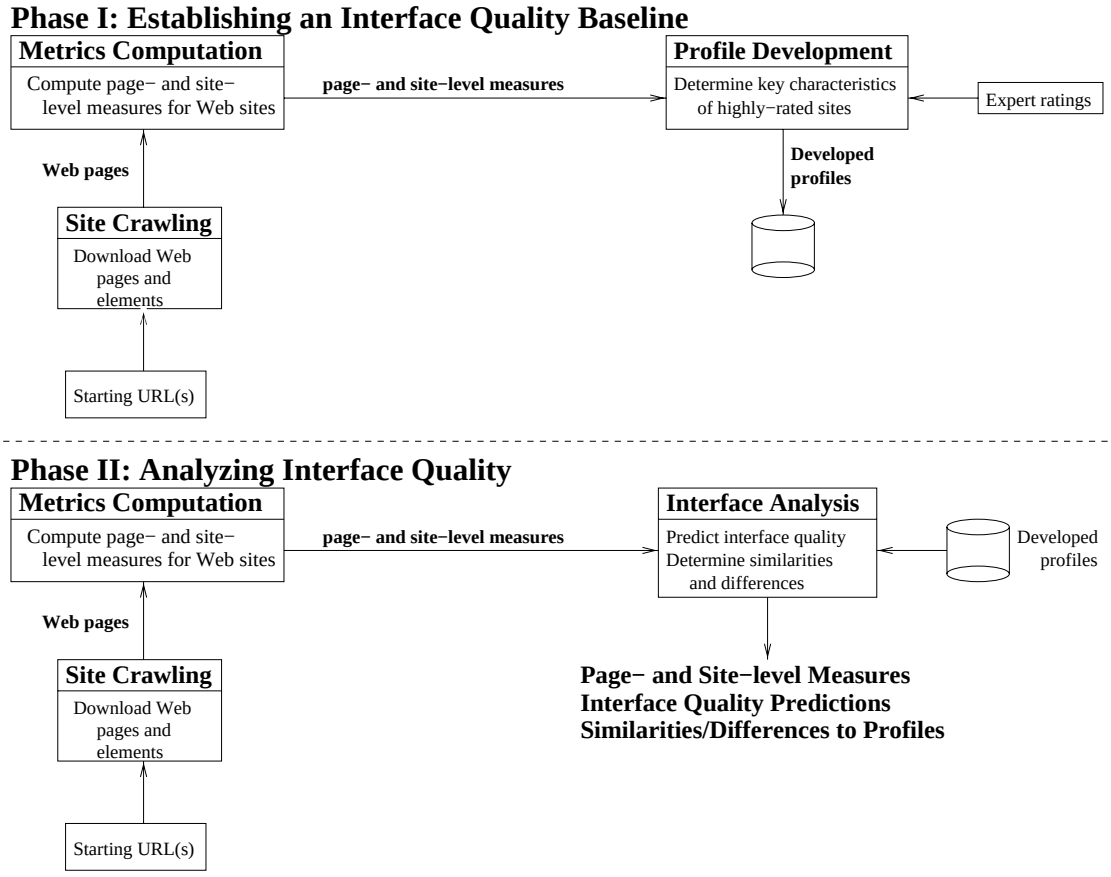

Figure 4.2: Overview of the analysis methodology.

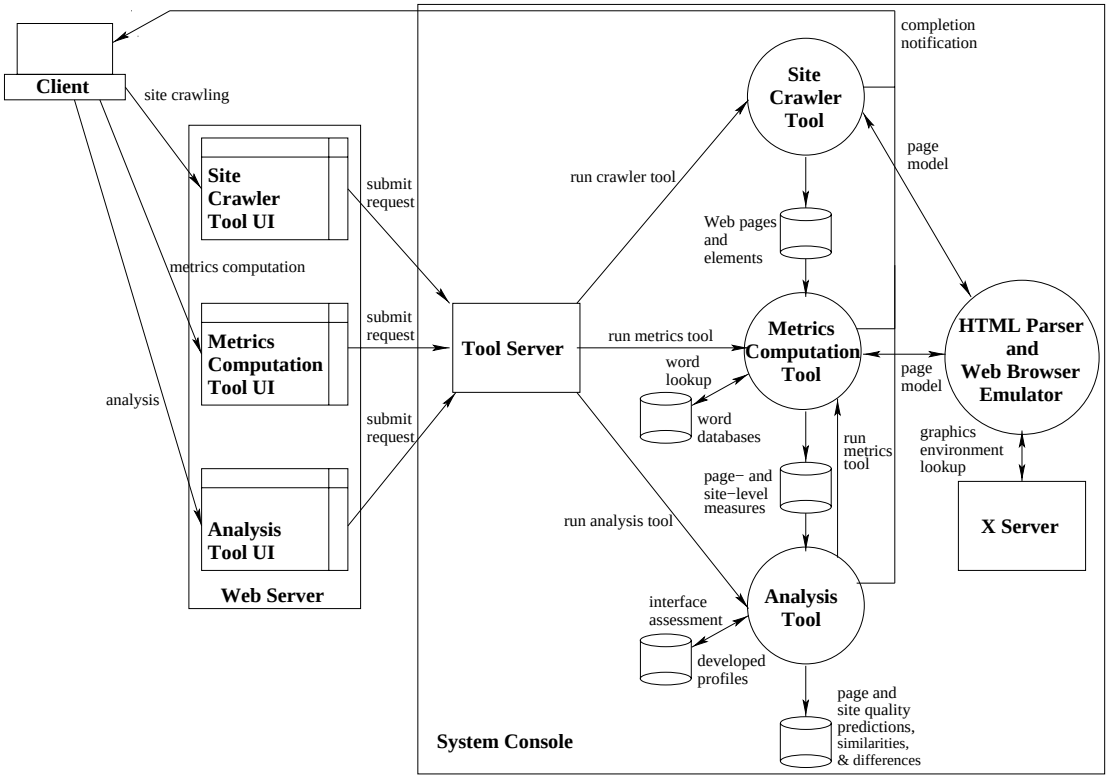

Figure 4.3: Architecture of tools developed to support the analysis methodology. All tools are available as part of the WebTango Research Project.

analysis component similar to operationalized guidelines. What distinguishes this analysis approach from other guideline review methods is: 1. the use of quantitative measures; 2. the use of empirical data to develop guidelines; and 3. the use of profiles (highly-rated interfaces as determined by expert ratings) as a comparison basis.

4.4 Tools

Figure 4.3 depicts the architecture of tools developed to support the analysis methodology. The following steps were taken to support the profile development discussed in Chapter 6.

1. Page were downloaded from numerous sites and stored on a Web server (site crawling).
2. Page-level and site-level measures were computed for each downloaded page and site (metrics computation).

Profiles developed in Chapter 6 are incorporated into the Analysis Tool. The Analysis Tool invokes the Metrics Computation Tool to generate measures; thus, eliminating the need to compute measures as a separate step.

This process is available to the public via separate interfaces for the site crawling and analysis steps; future work will integrate these interfaces into one UI to support the entire process. The interface for each tool routes requests to a server daemon (the Tool Server) for processing; the daemon in turn forks new processes to forward requests to the appropriate backend tool (Site

Crawler, Metrics Computation, or Analysis Tool). Both the Site Crawler and Metrics Computation Tools interact with the HTML Parser and Browser Emulator; this component creates a detailed representation of a Web page, including the x and y location of each page element, the width and height of each element, the font used, foreground and background color, and other attributes. The browser emulator determines many of the details, such as the height and width of text, by querying the graphics environment via the X Server running on the system console.

Currently, each tool sends an email notification to the client when the request completes; this notification includes a link to a tarred and gzipped file containing the output of running the tool. The output of running each tool is used as input to the subsequent step. For example, pages downloaded by the Site Crawler Tool are then processed by the Metrics Computation Tool to output page-level and site-level measures. Future work will focus on developing an interactive, integrated tool.

The components depicted in Figure 4.3 comprise over 33,000 lines of Java, HTML, and PERL code; they are discussed in detail in the remainder of this section. Appendix C provides information about running the tools.

4.4.1 HTML Parser and Browser Emulator

The Multivalent Browser (developed as part of the Berkeley Digital Library Project) [Phelps 1998; Phelps 2001] was used as a starting point for the HTML Parser and Browser Emulator depicted in Figure 4.3[1]. The Multivalent Browser enables creators of digital documents to represent their documents at multiple layers of abstraction and to annotate them with behaviors (e.g., actions), highlighting, and edit marks. The browser provides a means for these documents to be distributed and viewed by others. A variety of document formats are supported – OCR output, zip, ASCII, XML, and TeX – in addition to HTML. The browser parses HTML similarly to the Netscape Navigator 4.75 Browser's method and supports stylesheets. It does not support framesets, scripts, and other objects, such as applets.

The Multivalent Browser was revised extensively (~60% of code changed) to generate a more detailed page model for use by the Site Crawler and Metrics Computation Tools. Most of the revisions focused on enumerating frames, images, links, and objects that appear in a Web page and annotating each node in the page's tree structure with information about how the element is formatted (e.g., bolded, colored, italicized, etc.), whether it is a link (and if so whether it is an internal or external link), and downloading page elements to determine their sizes. The new parser also performs numerous corrections of HTML errors (e.g., out of order tags and tables without <tr> or <td> tags) while processing Web pages.

The new parser and browser emulator was configured to simulate the Netscape Navigator 4.75 browser with default settings (fonts, link colors, menu bar, address bar, toolbar, and status line). Currently, most monitors are 800 x 600 pixels [DreamInk 2000; Nielsen 2000]; thus, the window size was fixed at 800 x 600 pixels. The current implementation does not support pages that use framesets, although it does support pages with inline frames; given that Netscape 4.75 does not support inline frames, alternative HTML is typically provided in Web pages by designers and can be processed. Future work may entail using the Mozilla or Opera parser and browser to support more accurate page rendering, framesets, and scripts as well as to address performance problems with the current tool.

The HTML Parser and Browser Emulator comprises approximately 20,000 lines of Java code. The tool requires an average of 90 seconds to generate a model of a Web page. Performance should be substantially improved by using a more robust parser and browser.

[1] The Multivalent Browser code was provided courtesy of Tom Phelps.

4.4.2 Site Crawler Tool

A special Web site crawler was developed to address limitations of existing crawlers, such as Wget [GNU Software Foundation 1999] and HTTrack [Roche 2001]. Existing crawlers rely on path information to determine the depth of pages, which can be somewhat misleading. They are also not selective in determining pages to download; they download advertisements and Macromedia Flash pages, for instance. Finally, they typically attempt to mirror the directory structure of the remote site or place all images, stylesheets, etc. into one directory; this makes it challenging to relocate an individual page and its associated elements.

Like existing crawlers, the Site Crawler Tool is multi-threaded. In addition, the tool has the following key features.

- Pages at multiple levels of the site are accessed, where level zero is the home page, level one refers to pages one link away from the home page, level two refers to pages one link away from the level one sites, and so on. The standard settings are: download the home page, up to 15 level-one pages and 45 level-two pages (3 from each of the level-one pages).
- Links are selected for crawling such that: they are not advertisements, guestbooks, Flash pages, login pages, chatrooms, documents, or shopping carts; and they are internal to the site.
- Each downloaded page is stored in a directory with all images, stylesheets, frames, and objects (e.g., scripts and applets) aggregated and stored in subdirectories. This makes it easy to relocate pages and associated elements.

The Site Crawler Tool replaces links to images, stylesheets, and other page elements on the remote server with the corresponding locally-stored files. It also creates input files for the Metrics Computation Tool.

The Site Crawler Tool comprises approximately 1,500 lines of Java, PERL, and HTML code; the Web interface for submitting crawling requests is implemented in HTML and PERL, while the actual crawler is implemented in Java. The tool spends at most twelve minutes downloading pages on a site before aborting.

4.4.3 Metrics Computation Tool

The Metrics Computation Tool is consistent with other benchmarks discussed in Chapter 3: it is portable to other platforms due to its implementation in Java; it produces quantitative measures for comparison; and its results are reproducible (i.e., successive runs of the tool on the same page produce the exact same results). The tool computes 141 page-level and 16 site-level measures; these measures assess many facets of Web interfaces as discussed in Chapter 5. Numerous heuristics were developed for computing these measures, such as detecting headings, good color usage, internal links, and graphical ads. The tool also uses a number of auxiliary databases, such as the MRC psycholinguistic database [Coltheart 2001] for determining spelling errors and the number of syllables in words, and other lists of acronyms, abbreviations, medical terms, and common (stop) words. Currently, the tool only processes English text.

The Metrics Computation Tool comprises approximately 10,500 lines of Java, PERL, and HTML code. The Web interface for submitting requests for metrics computation is implemented in HTML and PERL. The page-level measures are implemented in Java, while the between-page and site-level measures are implemented in Java and PERL. The tool requires on average two minutes to compute page-level measures per page, 30 seconds to compute between-page measures per page,

and one minute to compute site-level measures per site. The performance bottleneck with page-level measures is the need to traverse the page model at least three times to annotate nodes and compute measures. During the first phase, the HTML Parser and Browser Emulator constructs an initial page model. Then, the Metrics Computation Tool traverses the page model to annotate headings, sentences, links, and to store link text. Page-level measures are computed during the final pass through the revised page model.

4.4.4 Analysis Tool

The Analysis Tool encompasses several statistical models for assessing Web page and site quality. Statistical models include decision trees, discriminant classification functions, and K-means cluster models as discussed in Chapter 6. For cluster and discriminant classification models, the top ten measures that are similar and different from highly-rated interfaces are reported; acceptable measure values are also provided. The cluster models also report the distance between measure values on a page and measure values at the cluster centroid; the distance reflects the total standard deviation units of difference across all measures. Currently, the Analysis Tool does not provide the designer with example good sites or pages; future work will focus on developing an algorithm to provide examples. Future work will also focus on supporting automated critique or providing explicit suggestions for improvements as well as interactive analysis.

The Analysis Tool comprises about 1,500 lines of PERL and HTML code. Each of the profiles discussed in Chapter 6 is implemented in PERL, while the interface for submitting analysis requests is implemented in HTML and PERL. The tool requires one minute on average to compute page and site quality predictions for a site.

4.5 Summary

This chapter presented an analysis methodology consistent with measurement approaches used in the performance evaluation domain and guideline review approaches used in the usability evaluation domain. Unlike other Web assessment techniques, this approach uses quantitative measures and empirical data to develop guidelines for comparison purposes.

Although the tools are very robust, they suffer from several limitations. Currently, the crawling and analysis tools are separate processes and could benefit from being consolidated. Another limitation is that metrics computation is extremely compute intensive. Timings on a Sun Enterprise 450 server revealed that it requires about four minutes on average to process each Web page to compute page-level measures, between-page measures, and page quality assessments. An additional minute is needed to compute site-level measures and the quality assessments. It could take an hour to analyze the quality of a site, including all of the steps from crawling 10 pages on the site to generating site quality assessments; this depends largely on the complexity of pages in terms of page sizes and the number of associated elements, including images and stylesheets. Hence, extensive optimization is needed to enable this process to occur in real time. One possibility is to reimplement the crawling, metrics computation, and analysis processes using a robust, open source browser, such as Mozilla or Opera. This will also enable the tools to support framesets and script processing, since robust browsers support these elements.

The major limitation of this approach is that the quantitative measures do not capture users' subjective preferences. For example, one study has shown that perceived download speed is more important than actual download speed [Scanlon and Schroeder 2000a]. Although it is possible to measure actual download speed, it may not be possible to assess perceived speed. Nonetheless,

the methodology can be viewed as a reverse engineering of design decisions that were presumably informed by user input.