International Journal of Software Engineering and Knowledge Engineering
© World Scientific Publishing Company

COST SIMULATION AND PERFORMANCE OPTIMIZATION OF WEB-BASED APPLICATIONS ON MOBILE CHANNELS*

MATTHIAS BOOK, VOLKER GRUHN, MALTE HÜLDER, ANDRÉ KÖHLER

Deutsche Telekom Chair of Applied Telematics/e-Business, Dept. of Computer Science, University of Leipzig
Klostergasse 3, 04109 Leipzig, Germany
{*book, gruhn, huelder, koehler*}*@ebus.informatik.uni-leipzig.de*

ANDREAS KRIEGEL

Commerz Business Consulting, Commerzbank AG†
Mainzer Landstr. 185, 60327 Frankfurt am Main, Germany
andreas.kriegel@commerzbank.com

Received (20 January 2006)
Revised (6 June 2006)
Accepted (Day Month Year)

When considering the addition of a mobile presentation channel to an existing web-based application, a key question that has to be answered even before development begins is how the mobile channel's characteristics will impact the user experience and the cost of using the application. If either of these factors is outside acceptable limits, economical considerations may forbid adding the channels, even if it would be feasible from a purely technical perspective. Both of these factors depend considerably on two metrics: The time required to transmit data over the mobile network, and the volume transmitted.

The PETTICOAT method presented in this paper uses the dialog flow model and web server log files of an existing application to identify typical interaction sequences and to compile volume statistics, which are then run through a tool that simulates the volume and time that would be incurred by executing the interaction sequences on a mobile channel. From the simulated volume and time data, we can then calculate the cost of accessing the application on a mobile channel, and derive suitable approaches for optimizing cost and response times.

Keywords: Web engineering; mobile communications; cost estimation.

1. Introduction

As thin client applications, web-based applications have the advantage of independence from the user and his preferred device. Just the existence of a browser and a suitable network connection are needed. Thus, web-based applications seem to be convenient for mobile use. But in hands-on trials of such scenarios, the response time of the application is often notably worse compared to its use in a LAN environment. Furthermore, the communication costs are hard to predict. An organization that plans to provide mobile access to its

*A preliminary version of this paper was presented at the 5^{th} Intl. Conference on Quality Software (QSIC 2005)[1].
†The work described in this paper was performed at the University of Leipzig.

existing web-based applications for a large group of mobile workers needs detailed information about the usability and estimated cost of the application in a mobile environment *before* investing any effort in building it. Therefore, the expected performance (in terms of response time) as well as the expected cost of the application on different mobile networks need to be quantified at an early stage. With PETTICOAT (**Pe**rformance **T**uning and cos**t** d**isc**overy of m**o**bile web-based **A**pplica**t**ions), we present a method that can be used to address this situation [1].

The PETTICOAT method can be used by software developers as well as software project managers. After compiling all necessary information, a tool calculates indicators that reveal the application's response time and communication costs in the mobile environment. This way, decisions about the development of a mobile channel for an application can be based on quantitative arguments. If the application is classified as not immediately suitable for mobile use, decision makers can use the detailed results to consider whether it is reasonable to address particular deficits in the application's design revealed by the simulation. This optimization can be conducted for single features or the whole application. Note that since the PETTICOAT approach extrapolates characteristics from existing to additional channels, it cannot be used when the whole application is still in the conceptual stage, but is intended to assess the feasibility of later channel additions to existing applications.

In this paper, we describe how the PETTICOAT method was employed in a case study that we performed in cooperation with an insurance company. The following section presents each step of the method in detail: After an overview of the related work in Sect. 2, we use examples from a case study to show how to model the application structure as a dialog flow (Sect. 3.1), identify typical interaction sequences within the application (Sect. 3.2), measure the time and data volume in the existing application (Sect. 3.3), specify the mobile channels' characteristics (Sect. 3.4), simulate the application's interaction sequences on different mobile channels and evaluate the usability and cost implications of the observed time and data volume (Sect. 3.5). In Sect. 4, we then present different approaches for optimizing the underlying factors affecting cost and usability. Section 5 concludes by outlining our ongoing and future work in this area.

2. Related Work

Dutta et al. show how frequent and thus critical user paths can be identified in e-commerce applications [2]. They provide a model of user behavior in the form of session graphs and conduct analyses regarding the most frequently used user paths, as well as critical edge sequences. This technique could be useful for our approach, since the identification of the most frequently used subset of all possible user paths in the application model is needed.

Furthermore, there are lots of approaches for web log analysis aimed at classifying user paths, e.g. the work of Spiliopoulou [3], Berkhin et al. [4], Kim et al. [5], Heer and Chi [6], Chi et al. [7], and Gillenson et al. [8]. Especially the identification of long sequences by Pitkow and Pirolli [9] seems to be an important topic for the PETTICOAT concept. The identification of actually chosen user paths vs. all possible user paths in the application model is needed in order to obtain meaningful results from the following simulation. In this context, the work of Mao et al. [10] is of specific interest. They present an idea for a cluster-based online monitoring system for web traffic. The target-oriented analysis of web traffic is a task to be solved within the PETTICOAT approach.

As PETTICOAT particularly addresses the analysis of dynamic web applications instead of static web pages, the analysis of web traffic is even more difficult. This problem is addressed e.g. by Berendt and Spiliopoulou [11], who deal with dynamic web content generation and website analysis.

Other approaches to improving the performance of web-based applications have focused on using thin clients to transmit just the image of the application (e.g. the work of Lai et al. [12]). The findings of this work are of relevance for the deduction of consequences (application design, bandwidth restriction) based on the simulation results. In this context, Bent et al. [13] as well as Krishnamurthy and Wills [14] report interesting results from an analysis of large websites regarding performance, cache and cookie issues. These results could be used for the creation of a package of measures in order to modify the analyzed website regarding performance issues in the mobile environment.

3. The PETTICOAT Method

The PETTICOAT method provides decision makers with indicators on the economical feasibility of mobile channel development. In a nutshell, it involves identifying interaction sequences in a dialog flow model of the existing application, measuring the time and data volume incurred in their execution (either by analyzing web server log files or observing real-time traffic), and then simulating how the same interaction sequences would perform when subjected to the frame conditions of a mobile channel. As a result of the simulation, we gain time and volume projections for the interaction sequences that allow us to estimate the cost and response times incurred by working with the application on different mobile channels, and to optimize these values by modifying the application's contents and infrastructure (Fig. 1).

The following subsections present these steps in more detail and illustrate them with excerpts from a case study we performed for an insurance company. In that project, we applied the PETTICOAT method to the prototype of a new web-based offer management system in order to estimate the cost that will be incurred each month by insurance agents accessing the system over mobile networks such as GSM, GPRS and UMTS.

3.1. *Modelling the Dialog Flow*

As a basis for our analysis, we need a model of the application's complete dialog structure. We use the Dialog Flow Notation (DFN) [15] for this purpose. This graphical notation models an application's dialog flow as a directed graph of states that are connected by transitions. We call the transitions "events" and the states "dialog elements", distinguishing "masks" (web pages rendered on the client) and "actions" (business logic executed on the server). Events can carry parameters that transport business data such as form input.

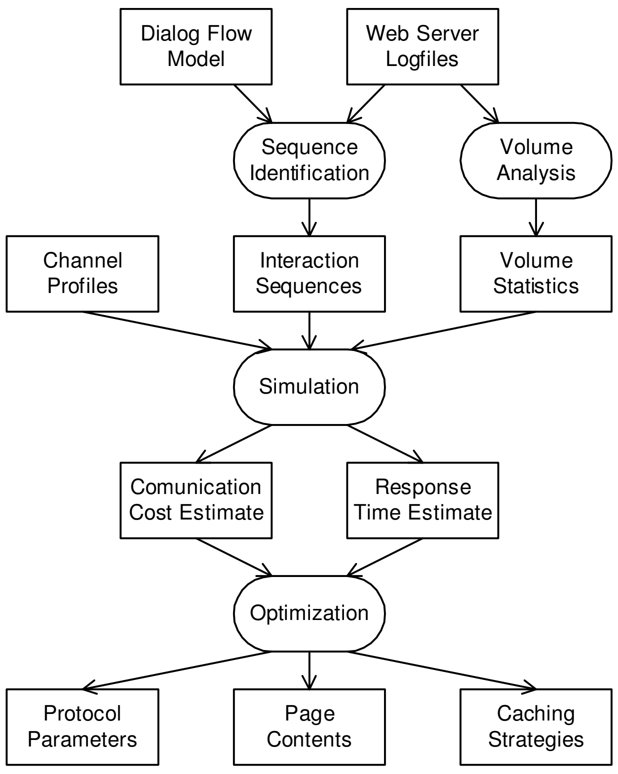


Fig. 1. The PETTICOAT method

By building such dialog graphs from masks, actions and events, the developer can specify all possible user interactions with the application. To increase the expressive power of the specification, dialog graphs can be encapsulated in "dialog modules" that can be reused in different contexts within the same application by nesting them into the dialog flow at arbitrary levels. This allows the developer to build complex dialog structures that closely mirror the users' mental model of the complex business processes supported by large-scale web applications. While tool support for automatically deriving the dialog graphs of complex applications would be desirable, re-engineering methods like path analysis in web server log files cannot capture the semantics of the encountered links and produce as clean dialog graphs as a manual reconstruction can yield.

As an example, Fig. 2 shows the dialog flow of the offer management system analyzed in the case study. Since we were looking at a rather simple prototype, the model does not make use of the DFN's dialog modularization capabilities and comprises only seven dialog masks connected through a number of actions that implement various business operations, an exemplary selection of which is shown in Table 1. Field staff users enter the application through the *initialize system* action *(0)*, which leads to the *Search Transaction Form* mask *(A)* where they can look up, create or edit transactions. Using the other masks and actions,

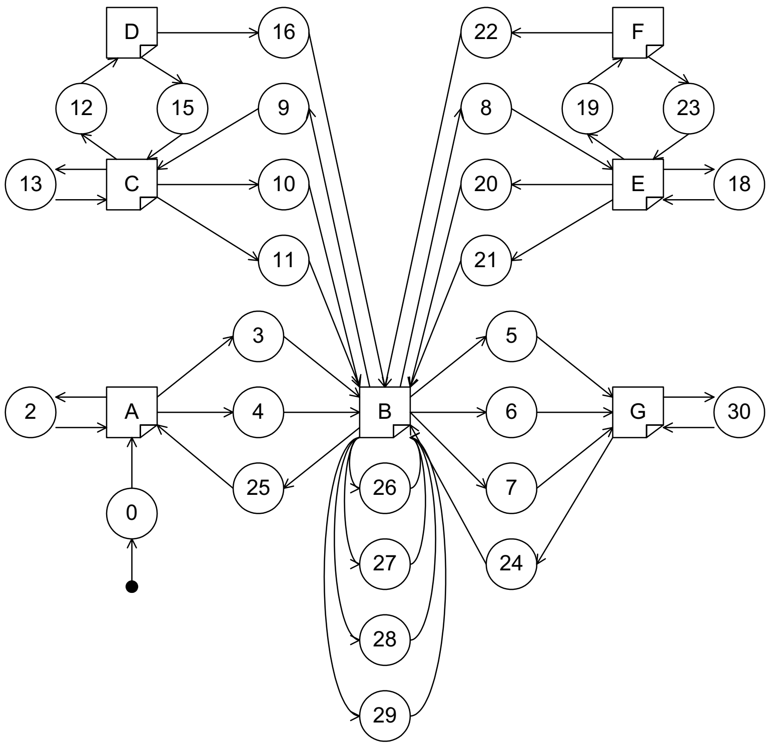


Fig. 2. Dialog graph of the offer management system

Mask ID	Content
A	Search Transaction Form
B	Edit Transaction Form
C	Associate Agent Form
D	Create Agent Form
E	Associate Insurance Holder Form
F	Create Insurance Holder Form
G	Edit Offer Form

Action ID	Function
0	initialize system
2	search transactions
4	prep transaction for editing
6	prep offer for editing
...	...
26	expand transaction elements
27	collapse transaction elements
28	load documents
29	browse transaction elements
30	process offer modifications

Table 1. Masks and actions of the offer management system (excerpt)

transactions can be associated with insurance agents, insurance holders and policy offers.

In order to use these graphical specifications as input for the following steps, they can be automatically translated into the XML-based Dialog Flow Specification Language (DFSL) [15]. However, we will skip this straightforward conversion step here.

3.2. *Identifying Interaction Sequences*

The dialog graphs of an application specify all possible ways of interaction that the user interface allows. Since the same business process may be accomplished in a number of similar, but still different ways, there will typically be some more and some less frequently traversed paths through the dialog graph (called “interaction sequences” from now on). To arrive at a representative cost projection for the business processes performed with the application, we therefore need to analyze the actual interaction sequences that occur in the application. By identifying the sequences that the users traverse most frequently, we can later weigh the cost they incurred accordingly.

In the case study, we identified and analyzed 15 interaction sequences (i.e. subsets of the whole dialog graph) of the offer management system. As an example, Fig. 3 shows the sequence for finding a transaction, browsing its elements and editing the associated offer. The events in this sequence are annotated with probabilities to reflect the different possibilities of executing this business process. Since a user’s interaction steps are not isolated from each other, but depend on the history of his interactions, these probabilities are conditional: In the notation, we first note the probability of a user following this event, and then (after a vertical bar) note which action the user must have executed before as a condition for this probability. For example, from mask *A*, there is a 1.0 probability that the user will execute action *2* under the condition that he executed action *0* before, but a 0.5 probability that he will execute action *2* if he already executed that action before. In other words, if the user just entered the application, he will definitely use the search feature, but if he already searched for a transaction, there’s only a 50% probability that he will use the search feature again. Rather, there’s also a 50% probability that he will proceed to edit the transaction he found (denoted by the $0.5|2$ probability for the event leading to action *4*). For events without annotation, the implicit probability of traversal is 100%, regardless of the previously executed action.

One might argue that the events’ probabilities may depend on a longer history than just the last executed action. Indeed, we are currently investigating the level of history that should be incorporated into this model in order to achieve sufficiently accurate approximations of the users’ behavior. In our case study, the probabilities were estimated based on practical considerations. Alternatively, a more realistic probability model can be reached by evaluating user tracking information that is routinely collected in web server log files [9]. In complex web applications, however, the logged URLs may not always indicate unambiguously which page was ultimately presented to the user. To increase the quality of the path identification, it may be necessary to log interaction data directly in the dialog control logic instead at the web server level. We are currently investigating ways of accomplishing this, ideally with non-invasive methods that do not require changes to the application logic.

While it is helpful to visualize the interaction sequences graphically in the conceptualization phase of the study, they need to be converted to a machine-readable format in order to be processed by the simulation tool. We use a variation of the DFSL for this purpose. The resulting sequence specification also contains estimates on how often each sequence will be executed by each user each month, which will be used towards the end of the simulation in order to calculate the approximate monthly cost of executing all sequences.

3.3. *Measuring Data Volume and Time*

As mentioned in the introduction, the two main factors influencing the cost of interaction with an application over a mobile channel are the time spent online and the data volume transmitted. To project these metrics for mobile channels, we measure them on the existing stationary channel and then input them into the simulation.

There are a few challenges in the details of this measurement process, however: Most importantly, for the volume measurement, we need to distinguish between static and dynamic content. While static content (such as images) always incurs the same volume (apart from caching effects, which can be accounted for in the simulation), dynamic content (such as search result pages) can produce a different volume for each request. To obtain accurate estimates, we need to deduce a probability distribution or an average value from the accumulated volume data. Since the distribution for each page depends heavily on its contents and context, we cannot use a generic formula for this purpose, but must rely on individual measurements. Also, web server log files only log the net volume of the content, but not any overhead introduced on lower levels of the protocol stack that nevertheless does count for billing purposes. This overhead can either be ascertained by observing the data flow directly on a sufficiently low protocol level instead of relying on server logfiles, or by factoring it into the simulation in accordance with the respective protocol specifications.

In our case study, we used HttpWatch 3.2, a simple HTTP traffic listener [16] to obtain the necessary data. The characteristics of each web page, image etc. were described in an XML-based format where each of those "web elements" is represented by a `WebElement`

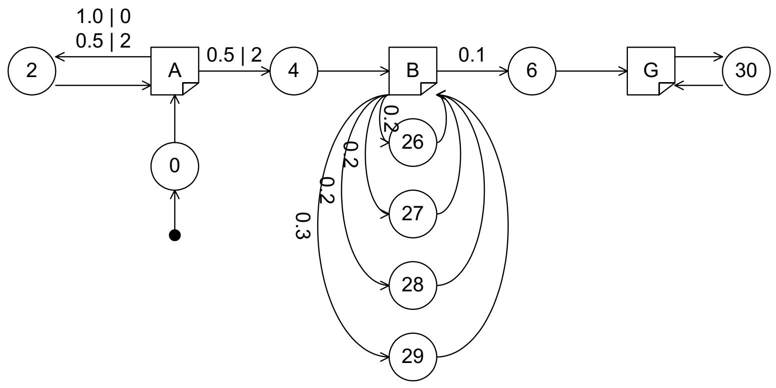


Fig. 3. Interaction sequence for finding and editing an offer associated with a transaction

tag that contains tags for its various attributes: Tags starting with `Request` or `Response`, for example, contain the data volume incurred for the request and the response of the web element in bytes, depending on whether the web server configuration allows HTTP compression or not. The `Inlines` tag contains references to web elements such as images included in a page. For each of these, we can specify the offset of their include point on the page (i.e. the number of bytes of the parent web element that need to be loaded before the inline web element is requested by the browser).

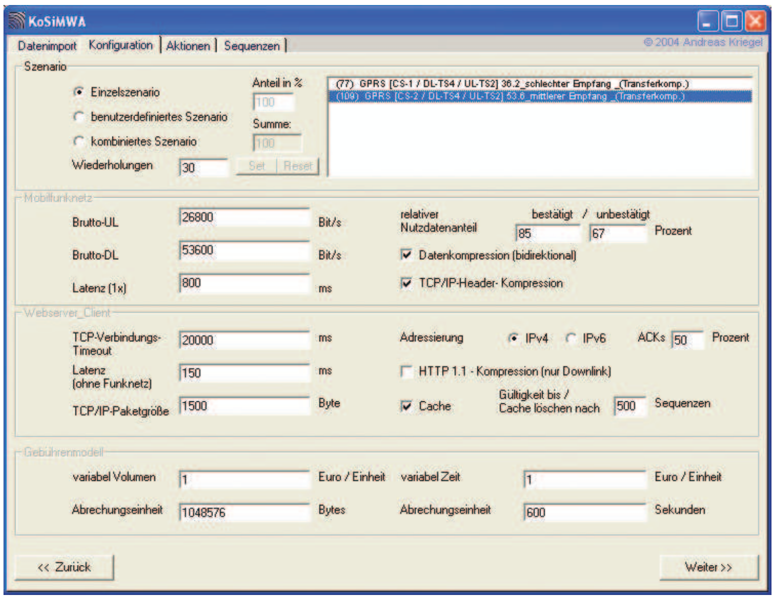
To measure the time it takes to complete an interaction sequence, a number of contributing factors need to be considered. This total time a user spends online is the sum of user activity (e.g. filling in forms), upstream and downstream transmission time, channel latency and server processing time. To accurately distinguish all these contributing factors, we would need synchronized timing on both the server and the client. Fortunately, however, only the user and server activity matter for the subsequent simulation, since the observed transmission time and latency already depend on the stationary channel that we measured on. We can thus deduct them from the overall time during the simulation based on our knowledge of the stationary channel characteristics and volume transmitted. This way, we are left with the user and server activity time, to which we can add the newly calculated transmission time and latency based on the mobile channel's characteristics. These times are specified for each action, i.e. each transition between masks, in an XML-based format.

3.4. *Defining Channel Characteristics*

Besides the description of the application's interaction patterns, mask and action characteristics, we still need a detailed specification of the target (usually mobile) network environment, since different mobile networks have different characteristics regarding bandwidth, latency, pricing etc. We define these characteristics in XML-based "channel profiles" for each network that shall be considered in the simulation. In our case study, we defined 16 channels, including different compression variants for GSM, HSCSD, GPRS and UMTS networks. The tool also considers effects of fluctuating signal strength. Each profile contains the gross uplink and downlink bandwidth in bits/s, as well as several attributes for packet and compression characteristics. Furthermore, the network provider's rates for volume-based and time-based billing are contained in the profile description. The rates in the case study were based on pricing plans of a German telecommunications provider. The channel profiles are stored in XML documents and can be edited by the user in the simulation tool, as illustrated by the definition of a GPRS 53.6 channel profile without data compression under the assumption of strong reception in Fig. 4.

3.5. *Simulation of Interaction Sequences on Different Channels*

In order to perform the simulation, our tool requires the XML documents produced in the previous steps as input, i.e. the application profile that contains the web elements and their volume data, the actions and their timing data, and the sequences with their probability and frequency data; and also the channel profiles containing the bandwidth, latency and



Szenario

[✓ Einzelszenario] benutzerdefiniertes Szenario kombiniertes Szenario

Anteil in %: 100

Summe: 100

Wiederholungen: 30

(77) GPRS [CS-1 / DL-TS4 / UL-TS2] 36.2_schlechter Empfang _(Transferkomp.)
(109) GPRS [CS-2 / DL-TS4 / UL-TS2] 53.6_mittlerer Empfang _(Transferkomp.)

Mobilfunknetz

Brutto-UL	26800	Bit/s	relativer Nutzdatenanteil bestätigt / unbestätigt	85 / 67 Prozent
Brutto-DL	53600	Bit/s	✓ Datenkompression (bidirektional)	
Latenz (1x)	800	ms	✓ TCP/IP-Header- Kompression	

Webserver_Client

TCP-Verbindungs-Timeout	20000	ms	Adressierung	[✓ IPv4] IPv6; ACKs 50 Prozent
Latenz (ohne Funknetz)	150	ms	HTTP 1.1 - Kompression (nur Downlink)	
TCP/IP-Paketgröße	1500	Byte	✓ Cache	Gültigkeit bis / Cache löschen nach 500 Sequenzen

Gebührenmodell

variabel Volumen	1	Euro / Einheit	variabel Zeit	1	Euro / Einheit
Abrechungseinheit	1048576	Bytes	Abrechungseinheit	600	Sekunden

Fig. 4. Specification of channel characteristics in the simulation tool

pricing characteristics. Using this input, the simulation tool works in three steps that will be described in detail in the following sections:

(1) The simulator begins each interaction sequence at its entry point. Taking into account the branching probabilities, it then simulates the time it takes to load each mask in the sequence, considering the inline element offsets which incur latency and traffic that delay the completion of the mask. This step already yields insights into usability problems that may be caused by unacceptably high response times.
(2) The results for each interaction step of a sequence are accumulated, taking the user's idle time between interactions into account.
(3) The results for each interaction sequence are multiplied by its estimated frequency per user and month. Summing up the results finally yields the projected communication costs of the application per user and month.

3.5.1. *Simulation of Interaction Steps*

The simulation results for the response times of the actions (i.e. the transitions between the masks) in the interaction sequence shown in Fig. 3 are given in Fig. 5 for a selection of channels. The diagram clearly shows that the use of a client-side cache (actions marked "with cache (w/c)") significantly reduces the response time in contrast to executing the same sequence without a cache (marked "no cache (n/c)"). However, it may still be relevant to investigate an application's response time without a cache since not all mobile platforms provide sufficient cache memory (we will discuss the optimiziation potential of different caching strategies in more detail in Sect. 4.3).

From Fig. 5, we can also determine that only the UMTS channel with enabled browser cache supports answering times of less than three seconds for this application and thus provides adequate usability (if we follow Shneiderman's rule that response times for simple actions should not exceed one second, while four seconds are the maximum response time for standard actions [17]).

Figure 5 also indicates that in our case study, the response times on the GPRS channel are longer than those on the GSM or HSCSD channel (using the identical compression and caching mechanism). This may come as a surprise, as GPRS may provide three to four times the bandwidth of a GSM 14.4 channel. On the other hand, GPRS has a network latency of about two seconds, so any request is delayed by about two seconds before the transmission of the requested data actually begins. By that time, the requested data would already have reached the recipient on the GSM channel. Only when the cache is deactivated, the GPRS channel can make up for the latency with its higher bandwidth.

Other, more complex timing constraints than the above-mentioned three-second usability rule are also conceivable. For example, we could define the constraint that mask n has to be loaded within t seconds after leaving mask m. The results gained in the simulation may then indicate which masks are responsible for failing the constraints and need to be optimized. If no redesign seems feasible, the application cannot be used on the simulated channel with the specified constraints. For example, in our case study, the results on the

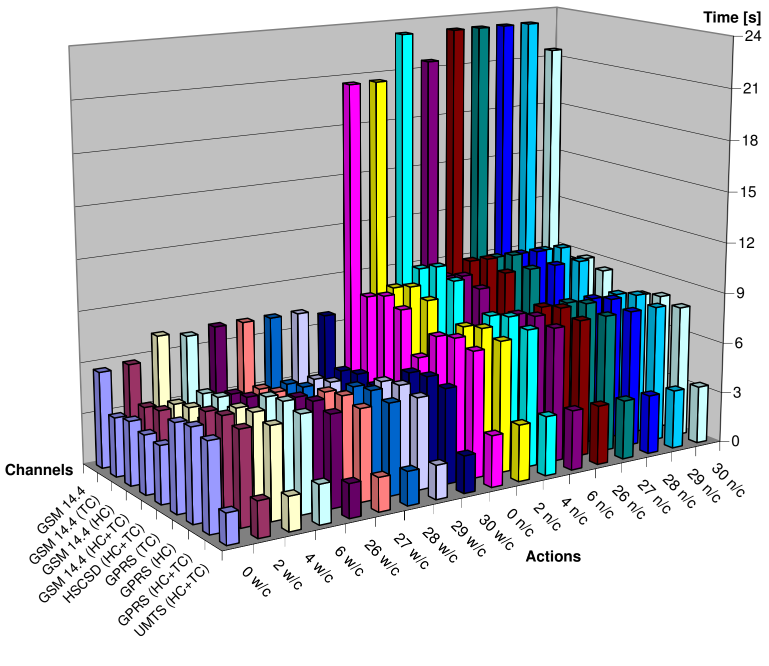


Fig. 5. Simulation results for response times

Sequence	using cache	Data	GSM 14.4	GSM 14.4 (Transfer Comp.)	GSM 14.4 (HTTP Comp.)	GSM 14.4 (HTTP Comp. + Transfer Comp.)	HSCSD (HTTP Comp. + Transfer Comp.)	GPRS (Transfer Comp.)	GPRS (HTTP Comp.)	GPRS (HTTP Comp. + Transfer Comp.)	UMTS (HTTP Comp. + Transfer Comp.)
4	w/c	Time [s]	81.4	63.9	66.2	64.0	62.9	75.2	76.1	74.6	52.0
		Volume [kB]	61	62	22	23	22	62	22	23	22
		Charge (t) [€]	0.41	0.34	0.35	0.34	0.28	0.13	0.13	0.12	0.09
		Charge (vol) [€]	–	–	–	–	–	0.06	0.02	0.02	0.02
4	n/c	Time [s]	206.2	105.5	108.9	105.3	76.5	89.2	90.6	88.6	58.7
		Volume [kB]	292	292	109	109	109	293	109	109	109
		Charge (t) [€]	1.03	0.56	0.57	0.55	0.34	0.15	0.15	0.15	0.10
		Charge (vol) [€]	–	–	–	–	–	0.29	0.11	0.11	0.11
11	w/c	Time [s]	185.4	121.1	130.9	144.3	129.0	156.2	160.5	154.4	110.2
		Volume [kB]	143	126	40	45	41	137	42	41	43
		Charge (t) [€]	0.93	0.64	0.69	0.76	0.58	0.26	0.27	0.26	0.18
		Charge (vol) [€]	–	–	–	–	–	0.13	0.04	0.04	0.04
11	n/c	Time [s]	420.8	219.0	230.2	212.2	165.3	191.4	203.4	199.5	124.7
		Volume [kB]	580	577	222	209	220	566	221	222	212
		Charge (t) [€]	2.10	1.16	1.21	1.12	0.74	0.32	0.34	0.33	0.21
		Charge (vol) [€]	–	–	–	–	–	0.55	0.22	0.22	0.21
12	w/c	Time [s]	210.1	157.3	154.6	162.5	149.1	191.0	184.7	182.9	129.9
		Volume [kB]	150	150	43	47	43	152	45	45	45
		Charge (t) [€]	1.05	0.83	0.81	0.86	0.66	0.32	0.31	0.31	0.22
		Charge (vol) [€]	–	–	–	–	–	0.15	0.04	0.04	0.04
12	n/c	Time [s]	467.5	250.9	257.0	248.5	193.8	222.4	223.8	217.3	150.8
		Volume [kB]	621	622	231	230	236	620	226	223	230
		Charge (t) [€]	2.34	1.32	1.35	1.30	0.87	0.37	0.37	0.36	0.25
		Charge (vol) [€]	–	–	–	–	–	0.61	0.22	0.22	0.23

Table 2. Simulation results for performing interaction sequences on different channels (excerpt)

GPRS channel indicate that a redesign should particularly focus on embedding inline elements at the top of a mask (as described in Sect. 4.2), so that requests for these elements can be sent earlier by the browser and the network latency is mitigated by the initial page still being loaded.

3.5.2. *Simulation of Interaction Sequences*

In the second step, the simulation tool sums up the results for the individual steps in a sequence gained in the first step. It also adds the user's estimated idle time to simulate how long a user works with a mask on average before the next mask is requested. This way, the tool determines how long it typically takes to execute a whole interaction sequence on a channel (taking the different probabilities for the sequence variants into account), and how many bytes are transferred in the process. Using the providers' rates specified in the channel profiles, the tool can then calculate the cost of performing each interaction sequence on each channel.

In Table 2, an excerpt of the results gained for a selection of channels and sequences from our case study is given. For each of the available channels (GSM, HSCSD, GPRS and UMTS), four simulations were carried out using no compression, HTTP compression (by the web server), transfer compression (by the carrier), and both HTTP and transfer compression. Sequence 4 denotes the process of a user creating a new policy offer, which has to be associated with an existing insurance agent and a new insurance holder. Sequence 11 represents the process of creating a new policy offer by copying an existing one. Finally, sequence 12 contains the results for finding and editing a policy offer, as shown in Fig. 3 and Fig. 5. For each channel/sequence combination, the table contains the time taken to execute the whole sequence, the total amount of kilobytes transferred in the process, and the cost incurred under a time- and volume-based pricing plan on the respective channel. Since volume-based billing is not available for GSM and HSCSD channels, the respective fields remain blank.

The results indicate that the use of data compression reduces the data volume to roughly a third of the uncompressed volume, resulting in lower transmission times. It is important to note, however, that when using transfer compression, the carrier will charge for the uncompressed data volume. HTTP compression (discussed in Sect. 4.1) thus seems to be the better choice for volume-based pricing plans, as only the reduced data volume is billed. This effect can be observed e.g. when comparing the results for scenario 11 (with and without cache) on the GPRS channel with transfer vs. HTTP compression. A volume-based plan also allows for more flexibility regarding idle times, since longer client-side activities before requesting the next mask are not billed. On the other hand, transfer compression seems to be the best choice for time-based plans, because it yields shortest transfer times resulting in lower charges. Combining both transfer and HTTP compression may combine their advantages, but due to a greater overhead and slightly longer execution time on the web server, this combination may not yield the lowest cost regarding time and/or volume.

3.5.3. *Simulation of Monthly Usage*

In the final simulation step, the tool uses the results gained so far to project the total cost that will be incurred when one user works with all interaction sequences in the application over the course of one month. This enables project managers to estimate the total communication costs that can be expected on all channels, and decide if the addition of a mobile channel will pay off.

For our case study, the final results indicated that a UMTS channel with combined transfer and HTTP compression and a volume-based pricing plan is the best option. This scenario would incur an estimated monthly cost of € 55.11 per user. A volume-based plan on a GPRS channel with transfer and HTTP compression would cost only € 54.94 per user and month, but exhibits worse usability due to the high network latency, as Fig. 5 illustrated. Since UMTS is currently not available all over the country, GPRS can still be recommended as a suitable backup solution with limited usability. The time-based plans for the HSCSD and GSM channel would result in monthly costs of € 298.35 and € 421.19 per user, respectively, with both using only transfer compression, since the combination of transfer and HTTP compression would be even more expensive in total.

4. Cost and Response Time Optimization

Whilte the simulation results presented above cannot be generalized, they serve as an example illustrating that due to the typical bandwidth and pricing schemes of today's mobile channels, the transmitted data volume is a decisive factor in the cost and response time of web-based applications. To keep cost and reponse time below certain thresholds that determine the economical and ergonomical feasibility of the application, one or more optimization strategies may have to be employed. We can distinguish protocol-based, content-based and cache-based optimization approaches, each of which will be discussed in more detail in the following sections.

4.1. ***Protocol-based Approaches***

Users of web applications interact with pages written in the Hypertext Markup Language (HTML). These pages are transmitted between client and server using the Hypertext Transfer Protocol (HTTP) at the OSI application layer, which in turn is encapsulated by the Transmission Control Protocol (TCP) at the transport layer. The understanding of certain aspects of these underlying protocols is necessary in order to address performance issues.

4.1.1. *Persistent Connections*

When opening a new connection, TCP uses a “slow start” algorithm [18]: The server first transmits only one packet and waits for an acknowledgment from the receiver. After this, it transmits some more packets and again waits for the acknowledgment. The number of packets sent at a time without waiting for acknowledgements is then successively increased. This algorithm is useful for detecting congestion in the network before excessive packet loss has occured, but it can cause performance problems for web applications [19]. In HTTP versions prior to 1.1, each request established a new TCP connection, where precious time was spent during the slow start waiting for acknowledgements. The impact of this effect becomes especially crucial when a web page contains many inline elements such images, style sheets etc., and when the communications network introduces a high latency for transmitting packets, as is the case with GPRS.

In contrast to previous versions, HTTP 1.1 uses persistent connections [20], so client and server can send multiple messages through one persistent connection without having to wait for acknowledgement messages. This pipelining mechanism avoids the major problems caused by TCP behaviour. Comparisons of the communication behaviour between HTTP 1.1 and earlier protocol versions show bandwidth savings of up to 40% [19]. Enabling HTTP 1.1 on the server and all clients should therefore be a mandantory optimization step for increasing the performance of a web application, especially when it is accessed over a mobile network.

4.1.2. *HTTP Compression*

In addition to employing persistent connections, HTTP 1.1 can be used to deliver content in a compressed format. If the compression algorithms that server and client announce in their HTTP headers are compatible (e.g. with both using *gzip*), the browser will transparently decompress any compressed content received from the server prior to rendering it [20]. If the formats are not compatible, the server will just fall back to uncompressed transmission.

For HTML pages, developers can either choose to provide compressed versions explicitly, or configure the server to compress pages on-the-fly when they are requested. While the latter requires more processing power, it is the only feasible solution when pages are generated dynamically for each user request. The potential savings that can be achieved this way are considerable; comparisons show file size differences of typically 60% (even higher compression rates may be achieved by using lowercase letters for HTML tags, since the compression algorithms will more likely find matching patterns in the page text)[19].

Note that HTTP compression typically only makes sense for HTML pages and other ASCII content – images and other multi-media content are typically already stored in formats with intrinsic compression (such as GIF, JPG or even better PNG), so the server-side compression will not yield significant savings in file size.

4.2. *Content-Based Approaches*

The time it takes to completely load a page depends not only on the overall data volume of its HTML code and inline elements such as images, style sheets, client-side scripts etc. Rather, the page loading time is affected by the internal structure of the page, i.e. by the position of references to those inline elements, as Fig. 6 illustrates.

When a browser requests a page from a web server, some network-dependent latency time t_l passes before the request is actually transmitted to the server. The server then sends the response to the browser. Depending on the size of the requested page, this transmission takes a certain transmission time t_t. While the structure of this request-response-cycle [21] is the same on any network, the time the cycle takes to complete depends mostly on the network's bandwidth and latency (obviously the computation time required by the server to provide the requested page also plays a minor role; however, it is typically only a small fraction of the delay introduced by a mobile network and thus not considered here).

As Fig. 6 shows, a new request-response cycle is triggered for every inline element referenced in the page. Today's browsers are typically multi-threaded and will therefore

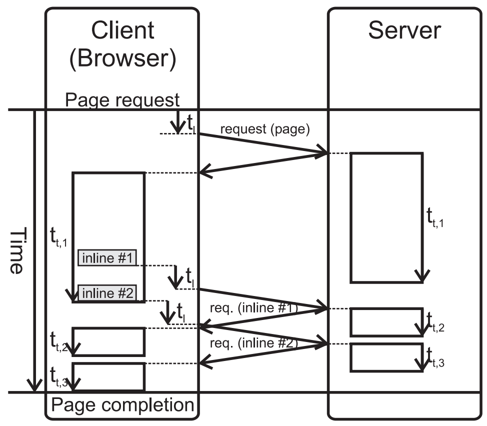


Fig. 6. Request-response timing with inline elements placed at bottom of page

send requests for inline elements as soon as the respective references are encountered, even if the responses to previous requests have not been received completely yet. However, since the browser cannot request elements that he does not yet know about, and loading the underlying page itself also takes some time, an inline element that is referenced at the end of the page will not be requested until the browser has received the whole page source code. With latency time (t_l) and transmission time (t_t) added to the late request, the user will have to wait quite a while before all content has been received and rendered completely.

In order to use the network as efficiently as possible and increase the application's usability, developers should try to reduce the waiting times, which are mainly caused by network latency. Regarding inline elements, optimizations can be achieved by smart placement of their references on the page, or by reducing the number of requests for them altogether. Both approaches will be discussed in the following subsections.

4.2.1. *Placing of Inline Elements*

While latency times of the network (t_l) cannot by reduced, Fig. 7 shows how they can be "hidden" by letting them occur during the transmission time (t_t) of an earlier response. This way, most of the waiting time experienced by the user is actually spent for transmitting data, not waiting for the network, and the parallelization results in faster page completion.

A theoretical optimization strategy would be to put all inline element references at the top of the page. For some resources such as style sheets and scripting code fragments, this is not a problem, since they are typically placed in the `<head>` section of an HTML-page anyway. However, image references are usually located wherever they are needed in the page body, and cannot always be moved without affecting the page layout.

When deciding which image references can be moved within the page source code, we need to distinguish between layout images and content images, where the former contribute to the overall appearance of the page (and are typically present on every page), while the

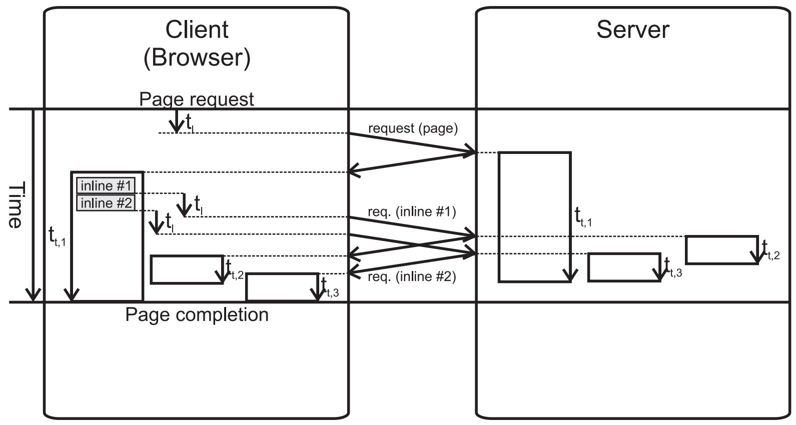


Fig. 7. Request-response timing with inline elements placed at top of page

latter provide actual information that has close ties with its immediate textual context. By authoring pages according to the guidelines of separation of content and layout, it should be possible to reference all layout images within the style sheet that is referenced at the beginning of the page. This should already shift the bulk of the inline requests towards the beginning of the page transmission time.

If necessary, requests for content images can also be moved closer to the top through the use of various CSS or JavaScript "tricks" that trigger the pre-fetching of an image that will be displayed much lower on the page. However, we won't go into detail on the technicalities here for the sake of brevity.

4.2.2. *Reducing the Number of Requests*

A more drastic way of eliminating the latency times associated with each request is to reduce the number of requests sent by the server. Since some elements such as style sheets and scripting code can also be included directly into the HTML page instead of being referenced from a separate file, it would seem like a straightforward optimization approach to just incorporate as much content as possible directly into the HTML page. With the individual requests for these elements eliminated, we could save not only their latency times, but also the data overhead introduced by the HTTP and TCP communication.

However, this approach precludes any savings that could be achieved by caches (as explained in Sect. 4.3), especially since style sheets and scripting code are typically the identical on every page and thus ideal candidates for caching. Thus, the time and volume savings that can be achieved using either approach should be examined (possibly with the help of the simulator described in Sect. 3) before a concrete strategy is implemented.

4.3. ***Cache-Based Approaches***

4.3.1. *Browser Cache*

In the previous sections, we already explored the idea of not just reducing the amount of data transferred in every request, but also to avoid sending requests across the network in the first place: Whenever possible, information that has already been downloaded once should be stored in a client-side cache from which it can be retrieved immediately if it is needed again, rather than being requested over the network another time. Two problems routinely occur when dealing with caching approaches: Firstly, since clients' memory is limited, we cannot cache everything forever, but need a strategy for clearing up space in a full cache in order to store new content. Secondly, when the data in the original source is updated, the cached copy becomes outdated, so we need a strategy for avoiding delivery of stale data to the client. Both problems have been discussed extensively already – Podlipnig and Böszörmenyi give an overview of the literature with a focus on web caches [22].

In web engineering, the first problem is usually beyond control of application developers and users, since the cache replacement strategy is hard-coded into the browser. It is also the lesser of the two problems, since the cache memory of today's clients is typically much bigger than the amounts of data transferred in a user's session with a web application.

To address the second problem, HTTP 1.1 supports two alternatives [20]: Using the *validation strategy*, the client asks the server for confirmation that a cached element is still up-to-date by requesting just the header of the required element. The server answers with a “last modified” header that contains the modification date. Using this information, the client can determine if the cached copy is still current or if a new version must be requested in a second step. The advantage of this approach is that the client will never deliver stale information from the cache since it always confirms its validity with the authoritative source first. However, in an environment where latency times can be comparatively large, such as on a GPRS connection, the necessary validation requests will take an unacceptably long time, even though the amount of data actually transferred is very small.

For such environments, the *expiration strategy* of HTTP 1.1 is more efficient: When the server delivers a requested element to the client, it also provides an “expires” header that contains a timestamp indicating when this information will go out of date. Until that time, the client may serve the data directly from its cache without having to reconfirm the validity with the server. And obvious advantage of this approach is not only the reduced data volume, but also the reduced request frequency, so no additional latency times are introduced by caching. However, the usefulness of the strategy relies on the accuracy of the expiry predictions. Since these are typically determined by factors beyond the server’s control (such as other users modifying the same data, developers updating the application, etc.), they can only be estimated in a way that tries to balance the risk of serving stale data from the cache vs. the possibility of wasting time and bandwidth on requesting redundant versions from the server.

One may argue that in web applications, most pages are dynamically generated according to the user’s request and thus not cacheable. While this is true, caching can still be applied to supplemental elements such as images, style sheets and scripting code, which typically incur considerable transfer volume.

4.3.2. *Local Proxy*

Cache implementations of today’s web browsers are optimized for general purpose “surfing”, where the structure and contents of the sites that users will visit, as well as their navigation patterns, are entirely unknown. When trying to optimize the mobile performance of a specific web application, however, developers typically have much better knowledge of the actual site structure and typical user behaviour. This knowledge can be used to build an application-specific client-side proxy [23]. Running on the same machine as the web browser, this local proxy intercepts all requests sent by the browser and transparently optimizes them for communication with a particular application over a particular network.

Most simply, the proxy can serve as an additional, application-specific caching level: Equipped with the knowledge that certain elements of a web application (e.g. layout graphics, style sheets) virtually never change, the proxy can provide “permanent” local copies of these elements to the browser, as well as answer validation requests that the browser may generate unnecessarily. Instead of relying on the HTTP validation/expiration strategies that treat each resource separately, the local proxy can also employ known relationships between elements in order to update its cache more efficiently: For example, if the first page requested from the server contains an encoded version number that is incremented with every deployment of the application, a change in the version number could serve as an indicator to the cache handler that the cached elements have become stale and need to be re-requested.

A prototypic implementation of these strategies in a local proxy used to access the intranet application of a German insurance company already yielded notable page load time reductions: The average time for performing the login process was reduced from 62 seconds down to 15 seconds on a GPRS network and from 18 seconds down to 8 seconds on a UMTS network. In the rest of the application, the time for performing business processes was reduced by 25% on a GPRS network and by 7% on a UMTS network, on average. The difference in time savings between the login process and the other business processes is due to the fact that the login process comprises mostly static resources, while most other pages in the application are generated dynamically. The observed differences between the GPRS and UMTS networks stem from the latter's higher bandwidth and lower latency times, which reduce the impact of caching.

While these measurements are application-specific and obviously cannot be generalized, they can serve as an indicator of the scale of delay reduction that is possible using this approach. Our ongoing research focuses on further optimization strategies that may be implemented in a local proxy. For example, in applications where a manual re-design of the page layout for parallelization of latency times (as suggested in Sect. 4.2) would be infeasible, a server- and client-side proxy could work in tandem to rewrite the pages' source code dynamically in order to optimize the page load time.

4.4. *Summary of Optimization Approaches*

Due to different structure and usage characteristics of different web applications, there cannot be one optimization strategy that works best in all cases. Rather, when optimizing a web application for mobile use, developers need to examine the approaches presented in the previous sections and weigh the effort required to implement them against the estimated time and bandwidth savings that they promise.

For example, benefits can quite easily be gained from enabling the compression and caching features of HTTP 1.1, provided that most clients are capable of handling them. Adapting the page layout to optimize the request-response patterns can improve response times considerably, but typically requires more effort (unless content management systems or templating engines are already being used), and has more impact on high-latency networks such as GPRS. Last but not least, the implementation of individual caching and optimization strategies in a client-side proxy allows developers to leverage technical and application domain knowledge for additional time and bandwidth savings, but requires considerable development effort. While the need for optimization is often obvious, the PETTICOAT approach presented in Sect. 3 can support this process by giving estimates on the savings that can be expected by using different strategies on different parts of the application, and thus help to focus the optimization effort on those aspects that will yield the highest benefit.

5. Conclusions and Future Work

In this paper, we have shown a method for assessing the usability implications and communication costs of adding mobile channels to an existing web-based application, and presented approaches for the optimization of web applications for mobile use.

As illustrated by the case study, the results of the simulation indicate if an existing application can be accessed efficiently on certain mobile channels, and provide clues on how the application may have to be optimized for lower response times. The simulation also provides an estimate of the cost of using the application on various mobile channels, which is a valuable factor in deciding if the introduction of a mobile channel will pay off for an organization in the future.

In our ongoing work, we are currently focusing on automated analysis of web applications to simplify the initial steps of the PETTICOAT method. This includes deriving the dialog flow model and the probabilities and frequencies of typical interaction sequences from the data contained in web server log files, rather than modeling them manually. We are also working on a refinement of the probabilistic model for the interaction sequences. Finally, we are collecting empirical evidence from industry projects to derive best practices for optimizing web applications for mobile access.

Acknowledgements

The authors would like to thank Clemens Schäfer and Daniel Trabold for their input on the local proxy optimization approach.

References

1. M. Book, V. Gruhn, M. Hülder, A. Köhler, and A. Kriegel. Cost and response time simulation for web-based applications on mobile channels. In M.F. Lau Kai-Yuan Cai, Atsushi Onishi, editor, *Proceedings of the 5th International Conference on Quality Software*, pages 83–90. Swinburne University of Technology, University of Hong Kong, IEEE Computer Society, 2005.
2. Kaushik Dutta, Debra VanderMeer, Anindya Datta, and Krithi Ramamritham. Discovering critical edge sequences in E-commerce catalogs. In *EC ’01: Proceedings of the 3rd ACM conference on Electronic Commerce*, pages 65–74. ACM Press, 2001.
3. Myra Spiliopoulou. Web Usage Mining for Web Site Evaluation. *Commun. ACM*, 43(8):127–134, 2000.
4. Pavel Berkhin, Jonathan D. Beche, and Dee Jay Randall. Interactive path analysis of web site traffic. In *KDD ’01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 414–419. ACM Press, 2001.
5. Dong-Ho Kim, Vijayalakshmi Atluri, Michael Bieber, Nabil Adam, and Yelena Yesha. A Clickstream-based Collaborative Filtering Personalization Model: Towards a Better Performance. In *WIDM ’04: Proceedings of the 6th annual ACM international workshop on Web information and data management*, pages 88–95. ACM Press, 2004.
6. Jeffrey Heer and Ed H. Chi. Separating the Swarm: Categorization Methods for User Sessions on the Web. In *CHI ’02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 243–250. ACM Press, 2002.

7. Ed H. Chi, Peter Pirolli, and James Pitkow. The scent of a site: a system for analyzing and predicting information scent, usage, and usability of a Web site. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 161–168. ACM Press, 2000.
8. Mark Gillenson, Daniel L. Sherrell, and Lei da Chen. A taxonomy of web site traversal patterns and structures. *Communications of the AIS*, 3(4es):5, 2000.
9. James Pitkow and Peter Pirolli. Mining Longest Repeating Subsequences to Predict World Wide Web Surfing. In *Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems*, 1999.
10. Yun Mao, Kang Chen, Dongsheng Wang, and Weimin Zheng. Cluster-based online monitoring system of web traffic. In *WIDM '01: Proceedings of the 3rd international workshop on Web information and data management*, pages 47–53. ACM Press, 2001.
11. Bettina Berendt and Myra Spiliopoulou. Analysis of Navigation Behaviour in Web Sites Integrating Multiple Information Systems. *The VLDB Journal*, 9(1):56–75, 2000.
12. Albert M. Lai, Jason Nieh, Bhagyashree Bohra, Vijayarka Nandikonda, Abhishek P. Surana, and Suchita Varshneya. Improving web browsing performance on wireless pdas using thin-client computing. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 143–154. ACM Press, 2004.
13. L. Bent, M. Rabinovich, G. M. Voelker, and Z. Xiao. Characterization of a Large Web Site Population with Implications for Content Delivery. In *WWW '04: Proceedings of the 13th International Conference on World Wide Web*, pages 522–533. ACM Press, 2004.
14. Balachander Krishnamurthy and Craig E. Wills. Analyzing Factors That Influence End-to-End Web Performance. In *Proceedings of the 9th International World Wide Web Conference on Computer Networks: The International Journal of Computer and Telecommunications Networking*, pages 17–32. North-Holland Publishing Co., 2000.
15. Matthias Book and Volker Gruhn. Modeling Web-Based Dialog Flows for Automatic Dialog Control. In *19th IEEE International Conference on Automated Software Engineering (ASE 2004)*, pages 100–109. IEEE Computer Society Press, 2004.
16. Simtec Limited. HttpWatch 3.2. http://www.simtec.ltd.uk, 2005.
17. Ben Shneiderman. *User Interface Design*. mitp, 2002.
18. Van Jacobson. Congestion Avoidance and Control. In *SIGCOMM '88: Symposium Proceedings on Communications, Architectures and Protocols*, pages 314–329, New York, NY, USA, 1988. ACM Press.
19. Henrik Frystyk Nielsen, James Gettys, Anselm Baird-Smith, Eric Prud'hommeaux, Håkon Wium Lie, and Chris Lilley. Network Performance Effects of HTTP/1.1, CSS1, and PNG. In *SIGCOMM '97: Proceedings of the ACM SIGCOMM '97 conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 155–166, New York, NY, USA, 1997. ACM Press.
20. W3C. RFC2616: Hypertext Transfer Protocol – HTTP 1.1, 1999.
21. David Olshefski, Jason Nieh, and Dakshi Agrawal. Using Certes to Infer Client Response Time at the Web Server. *ACM Transactions on Computer Systems (TOCS)*, 22(1):49–93, 2004.
22. Stefan Podlipnig and Laszlo Böszörmenyi. A survey of Web cache replacement strategies. *ACM Comput. Surv.*, 35(4):374–398, 2003.
23. Bruce Zenel. A general purpose proxy filtering mechanism applied to the mobile environment. *Wirel. Netw.*, 5(5):391–409, 1999.