Adaptive Changes Monitoring Service in Web Repositories Based on Agent Games

Santi Saeyor Mitsuru Ishizuka
Dept. of Information and Communication Engineering, Faculty of Engineering,
University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, JAPAN,
{santi,ishizuka}@miv.t.u-tokyo.ac.jp

Abstract: Changes made in Web repositories occur at unpredictable rates. Besides the target information itself, the changes upon the previously released information are significant and worth being notified to those who perceived the out of date information as soon as possible. Unfortunately, stock type information source has no means to inform its prospective users about the changes. While the stock type information source occupies a large percentage of sources on the Web, it is necessary to have a system that monitors changes on the Web, and provides comprehensive presentation to the prospective users. This paper proposes a mechanism that incorporates change monitoring and presentation service for a user community. The service keeps improving its utilization factor by several schemes based on the decision made by game analysis. Thus, the service can be provided to a relatively large group of users.

1. Introduction

The information in Web repositories is changed dynamically without any prior notification. At present, a large number of information sources are stock type. The users access this type of information in pull mode, mostly by Web Browsers. These information sources have no mechanism to bring the changed information to prospective users. The users have to deal with the matter by themselves. Browsing through the sites for new updates is not only time consuming task but also vain in case that there is no change made on the sites once visited. This puts a significant load to the users besides exploring brand new information.

This paper considers the evolution of mechanism that detects and evaluates changes on the Web, as well as provides it in comprehensive form for prospective users. With this system, The ubiquitous stock type information sources on the Web have no need to provide any effort to convey their updates to the users. As a result, the information is seemingly transferred in push mode.

We incorporate shared resource management in our system in order to enable a larger scale of service. The shared resource management plays an important role to make the push mode transfer of changes and differences practical. The system would not be practical if the available resources are used to provide service to a large group of users without an effective resource management processing. The service attempts to increase the utilization factor of the system by several schemes. Each scheme tries to increase identical services in the monitoring process. These identical or virtually identical requests give a significant impact on the utilization of our service. As long as we can implement the service with reasonable resource allocation, more users can have access to our service. At this point, we let the service interacts with user. However, the interaction must be made minimal in order to maintain the level of automatism of the service.

We consider the schemes that deal with user as games. The games are played by users and the service agent. The service agent decides its moves based on experience in the past and game analysis. A useful tool we use here is the game theory. We apply the tool from the viewpoint of our service. Overall, the service agent improves the utilization factor of the service and gets the most out of limited interaction with the users.

2. Architecture of the service

The monitoring service consists of the service provider and user interface. The service is designed so that the service can be accessed from the Internet. Each user accesses the service via the WWW using any browser with

Java Virtual Machine. Requests can be made directly to the service provider. The service provider consists of several modules as follows:

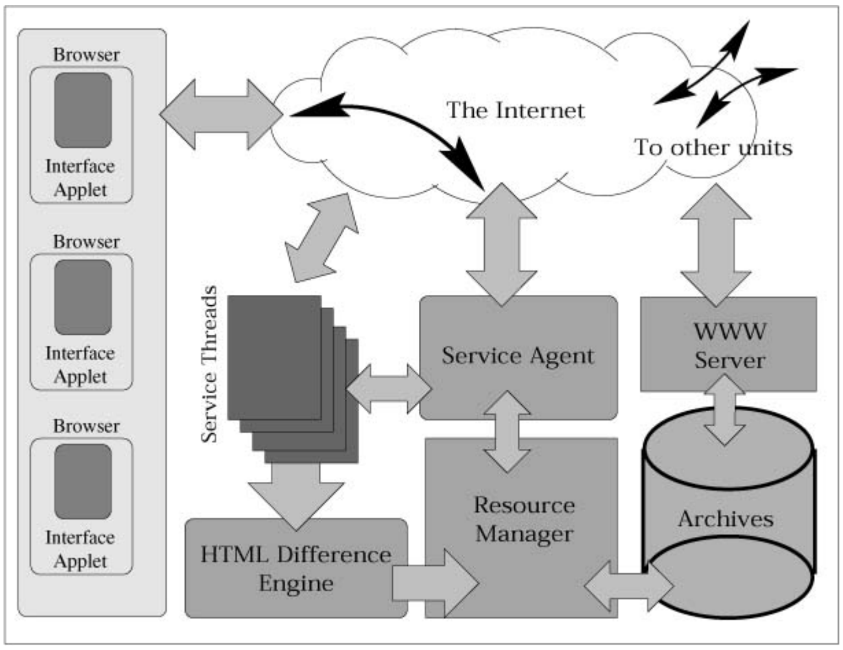

Fig. 1 The architecture of change monitoring service.

- **Resource Manager**: This module handles all resources for the service. Resource Manager will keep the results from the HTML Difference Engine in the archives. It keeps improving the utilization factor of the system by matching all identical or virtually identical requests.
- **Service Threads**: Each Web page monitoring request will be handled by a service thread. The thread keep monitoring and comparing revisions of that Web page.
- **Service Agent**: This is the heart of the service that interacts with other modules in order to retrieve and compare Web pages. It takes requests from the users and consults the Resource Manager to allocate resource for incoming requests. It activates the service threads to start services. When the users query their service profiles, it works as a service broker that retrieves the profiles for the users. The users can edit their service profiles via the Service Agent. Moreover, the Service Agent handles all responses made by users when the service needs to know opinions of the users. The responses from the users are proceeded to the Resource Manager in order to decide how to improve the service.
- **HTML Difference Engine**: The service threads implement the Difference Engine in order to compare the content of updated pages and see whether there are significant changes in them. Running the Difference Engine compares the old and new versions of HTML documents. At the same time, it will summarize the updated information into another HTML document by an innovative algorithm described "Difference and Display" subsection.
- **WWW server**: The page archives contain the old and new version of Web pages together with summary pages constructed by the HTML Difference Engine. When the users are notified by their Service Agent, they can view the summary pages with their browsers via the WWW server.

3. Monitoring the changes

The monitoring service keep monitoring the changes in Web repositories and making comprehensive presentation of the changes available to the users. Once the Service Agent found significant changes, it notifies the users. The consideration for each task is described in detail in following topics:

3.1. Difference and Display

Our HTML Difference Engine implement the algorithm called "Longest Common Tag Sequence"

(LOCTAGS) which is developed from the basic idea of Longest Common Subsequence (LCS) algorithm. The well known LCS has been widely used in comparison of text document revisions for long. When comparing 2 revisions of text with the capability of common subsequence extraction, we can tell the actions made on the new document easily. Unfortunately, the traditional LCS is not applicable to hypertext document. In order to cope with the structured text like HTML, we regard the HTML document as a sequence of tags and context. We use LOCTAGS to find the common subsequence of the new and old version of tag streams. Once we have the common subsequence of tags, we can point exactly which tags were deleted or added. This information helps us in comparing the context pairs at the right place. In other words, the LOCTAGS provides information about which context in the older revision should be compare to which context in the new revision.

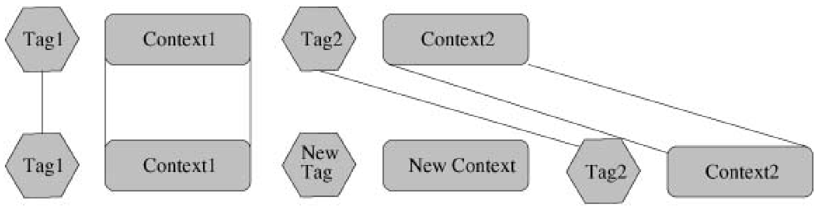

Fig. 2 The streams of HTML document.

When comparing context, we apply the traditional LCS. Difference Engine. The tool takes both the old and new HTML documents as its inputs. Both tags stream are fed to the LOCTAGS detector. The result, common tag subsequence, is used as a reference information at the differentiator. Right here, one of the output lines is fed to the HTML Constructor in order to produce a summary page in HTML format. On the other hand, the output is fed to the filtering process that evaluates the changes based on user interests. The result of evaluation is used by the decision-maker. The decision-maker decides whether the changes should be pushed to the users.

3.2. Push it to the users

Once the changes are detected, the difference engine evaluates the content of the changes. The evaluation is necessary because it is no use to push a piece of insignificant information of changes to the user. The evaluation is taken place in the filter based on user interests. If the changes are considered significant, the service provider sends notification to the user and let the user select to show the difference. The criteria used in this process focus on the filtering process, which is based on user interest as described in next topic.

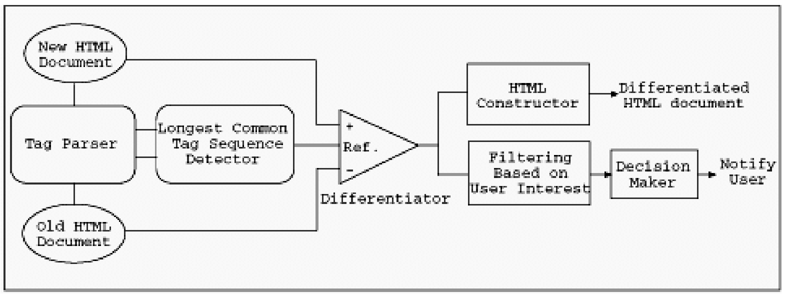

Fig. 3 The HTML Difference Engine with the Decision Maker

3.3. User Interests

We divide the interests of the users into following categories:

- Existence of contents: Many users are curious to know whether the contents they interested in are still exist in or newly added to the document. It is inevitable to check the existence of components again once the document was reviewed. The filter used in our service provider agent deals with this demand by checking the existence of links, images, Java applets, etc. that appear in the new version of document. The filter provides a report of the change in existence of component by comparing with the old version. The service provider agent notifies the user via email with this information in order to tell the user roughly how much the document has been changed.
- Appearance of document: HTML specification includes many tags that control the flow and structure of HTML documents. In some cases, many users are curious about the change in the appearance of the

documents. We take the appearance changes into account when evaluating the changes.

- Topics in interest: This category is the most important issue when we deal with the user's interests. In real world, people evaluate whether the content they are reading fall in the area of their interest by the context. If we insist to deal with this issue rigorously, we have an expensive processing to pay. The evaluation of relevance to user's interest areas is considered to be heuristic. On the other hand, the process needs natural language processing techniques. We compromised the correctness with computational costs. The filter deals with this issue by the key words that relevant to the interests of the user.
- Scoring of changes: When the filter found any changes that fall into the first two categories, it evaluates a score of those changes. The score of each category is a fix value assigned accordingly to the importance of the category. However, the third category deals with key words. We assume that the more occurrences of the key word in the document, the closer to the user's interest the document is. The score of each key word is increased as the more occurrences of the key word are detected. Finally, the total score is the summation of all categories of changes. The score is then compared to the specified threshold. It the score is higher than the threshold, the Service Agent notifies the user about the changes.

4. Adaptive Resource Management

When serving a large number of users, we expect to have some identical or virtually identical requests. These requests can share the resource. The more identical or virtually identical requests, the better utilization factor we can get from the service. The Resource Manager deals dynamically with the request matching. The conditions of virtually identical requests change dynamically upon the changes in Web repositories and the parameters of the request made by the users. If we decide to push changes and difference information, we attempt to push the information to prospective users. The fact is that each user has different degree of interest and attitude against the detected changes. In the paper, we define the utilization factor of the service as follows:

Utilization factor:

$$U = \frac{N - M}{M}$$

Where:

N = the number of all request

M = the number of different kinds of requests. M is also defined in term of probability of having different kinds of request (P_{diff}).

$$U = \frac{N - P_{diff}N}{P_{diff}N} = \frac{1 - P_{diff}}{P_{diff}}$$

We can see obviously that if we share resource among users, we are likely to get more profit than serving each user separately. The utilization factor, finally, depends on the Pdiff, which ranges from 1/N to 1. The range tells us that our utilization factor ranges from 0 to N-1.

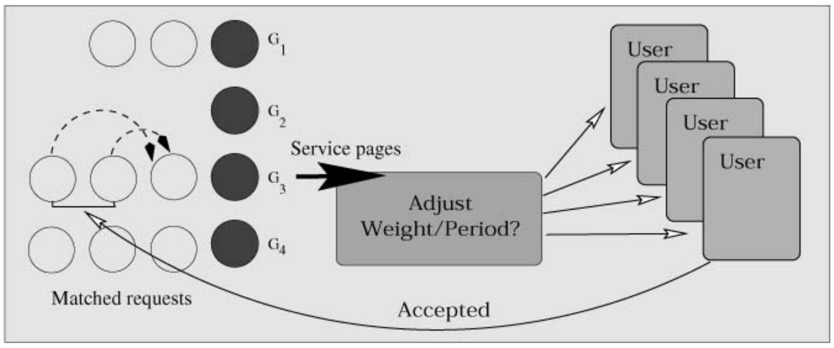

Fig. 4 The Service Agent requests the users to adjust the parameters of services

The amount of identical or virtually identical requests can vary dynamically. This is the case when some requests among currently identical requests are satisfied by the changed conditions but some are not. For examples, we decide to push changes information to the user if we found that the change score is higher than specific threshold points. Suppose we have 2 users who specified the score threshold for identical page at 1500 and 1000 points. Both requests are considered identical if the change score is 2000 points. However, if the change score fall between 1000 and 1500 points, the requests are no longer identical. The Resource Manager analyzes the characteristic of changes on Web pages. A parameter to check is the notification threshold. In some cases, some users specified high thresholds with high frequency of monitoring. If the Resource Manager found that the change rates of those pages are relatively slow, it may ask the users to adjust threshold weight of notification or monitoring frequency.

Adjusting these parameters has probability to increase more matched requests. Meanwhile, the Resource Manager detects hot requests shared by a large number of users. The hot requests trend to be interesting pages. The Resource Manager may recommend these requests to the users. If some users accept the recommendation, the utility factor of the service trends to be increased according to the increasing matched requests.

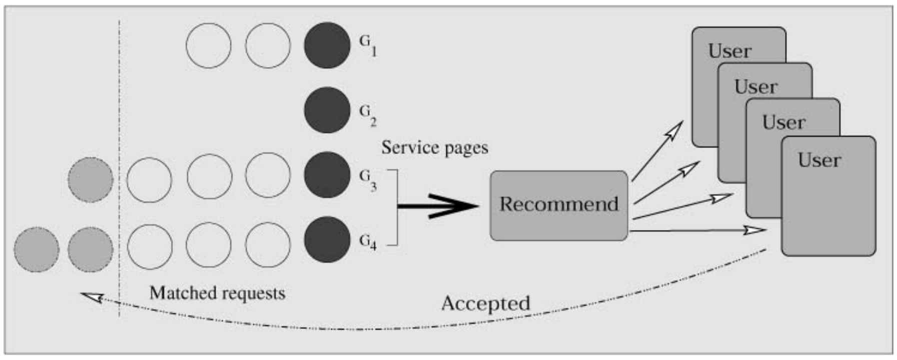

Fig. 5 The Service Agent recommends some hot pages to the users.

However, the Resource Manager has to make decision based on facts and experience in the past whether it should ask the users to adjust some parameters or recommend some hot requests to the users. The Resource Manager makes a decision by investigating the probability of improvement from the pay-off matrix of the game. We will consider how to make decision based on our service games from the following expected pay-off matrix.

		User	
		No Adjustment	Adjust
Service Agent	No Adjustment	p_0	p_1
	Adjust	p_2	p_3

Improvement occurs when:

1. The user obeys the suggestion from the agent, we apply the p_0 and p_3.
2. When user adjusts while agent does not suggest, we apply p_1
3. The agent suggests but the user does not adjust, we apply p_2

With these policies in mind, we can implement the random equilibrium criteria and derive the probability of improvement. (If P_a is the probability that the agent suggests an adjustment, and P_u is the probability that the user adjusts the service profile)

$$P(improve) = (P_a P_u) p_3 + (1 - P_a)(1 - P_u) p_0 + P_u (1 - P_a) p_1 + P_a (1 - P_u) p_2$$

But P(improve|P_a=0) = P(improve|P_a=1) and P(improve|P_u=0) = P(improve|P_u=1)

Solve for P_a, P_u then substitute into P(improve) we derive

$$P(improve) = \frac{(p_1 p_2 - p_0 p_3)}{(p_1 + p_2 - p_0 - p_3)}$$

The service agent checks whether the P(improve) is above 0.5 which means the system has probability to improve the service more than 0.5, if it asks the user for adjustment. In our service, the probability p_3 can be derived by a function that evaluates how significant a request for adjustment is. The probability p_2 comes from the experience in the past, which is, in other words, how much the user refuses the suggestion. The probability p_1 comes from the improvement made when the user adjusts the service profile without suggestion from the Service Agent. Finally, the probability p_0 comes from the self-improvement rate occurred as the conditions of changes in Web repositories vary in time domain. We can see obviously that the variables used in the game analysis above can be evaluated at ease from the statistic of the service. The Service Agent can make decision to deal or not to deal with the user by calculating the P(improve) based on transactions in the past. Moreover, each decision trends to be more exact as the experience of the Service Agent increases.

5. Related Work

From the standpoint of tracking and viewing changes on the Web, the work that is most similar to ours is that of Douglis, et al. [Douglis 98], [Douglis 96]. They used the AT&T Internet Difference Engine to compare revisions of Web pages of some enterprises from time to time. Their work inspires a great deal in our work. We attempt to improve the service for a larger scale of users. We manage the shared resources among users in order to enable induced push mode of changes and differences. Besides, our difference engine implements the LOCTAGS algorithm, which is capable of comparing context exactly where the revisions should be compared. In addition to a target page, we assume that the child pages are likely to have relevant information. Therefore, the service provider agent can be requested to watch the target down to its child pages. If needed, the agent can also be requested to watch deep down to the grandchild pages. However, the grandchild level is limited to the pages in the same domain of each child page.

The Do-I-Care agent [Starr 96] applies social discovery and filtering to inform the users when changes are detected. Moreover, it takes advantage of training the agent by groups of users where some interesting information may be offered to the user via the effort of others. We agree with the idea but we need a simpler way for evaluation of changes. The scoring method we use is straightforward and can be carried out quickly while providing a way for users to adjust the threshold value upon their experiences. In our system, the social filtering effect occurs when the Resource Manager cooperates with the Service Agent in order to improve the utilization factor.

6. Summary

In this paper, we presented the evolution of the mechanism that notifies the users about changes in Web repositories, as well as preparing the summary of the changes. The shared resource management based on game analysis and experience in the past helps us in providing information pushing service to a large group of user. The effort to improve the utilization factor of the service brings about an adaptive monitoring service. The evaluation of the functions we used to evaluate the probability in the expected pay-off matrix of service games gives a significant impact on the accuracy of making a decision. These details should be considered in future studies.

7. References

[Douglis 98] Fred Douglis, Thomas Ball, Yih-Farn Chen and Eleftherios Koutsofios (1998). The AT&T Internet Difference Engine: Tracking and viewing changes on the web, World Wide Web, 1998 1(1), 27-44.
[Aglet 97] Aglet-Workbench - Programming Mobile Agents in Java, IBM Tokyo Research Lab., URL: http://www.trl.ibm.co.jp/aglets/.
[Bradshaw 97] Jeffrey M. Bradshaw (1997). *Software Agents*. AAAI Press/The MIT Press.
[Douglis 96] F. Douglis, T. Ball, Y. Chen, E. Koutsofios (1996). Webguide: Querying and Navigating Changes in Web Repositories, In Proceedings of the *Fifth International World Wide Web Conference*, Paris, France, May 1996, 1335-1344.
[Starr 96] Brian Starr, Mark S. Ackerman, Michael Pazzani (1996). Do-I-Care: A Collaborative Web Agent Proceeding of *ACM CHI'96*, April.