Adaptive Service to Monitor Changes in Web Repositories Based on Game Analysis

Santi Saeyor
and
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changes upon the previously released information are significant and worth being notified to those who perceived the out of date information as soon as possible.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.

Keywords
Adaptive Service, Change Monitoring, Web repository, Game Analysis

1. INTRODUCTION

The explosive growth of the World Wide Web (WWW) brings about overwhelming information, in addition, it is supposed to be changed dynamically without any prior notification. 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 information of the changes 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. We need some representatives to do such burdensome and tedious jobs for us. Furthermore, we would like to know when the changes occurred and how they look. That means not only tracking tools but notification and presentation issues are also taken into account.

This paper considers the evolution of mechanism that detects and evaluates changes on the Web, provides it in comprehensive form, and pushes the information to prospective users. At overall level, these operations induce the flow of information, change and difference instances, from the information sources to the 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 the framework 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 view point 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 architecture of the change monitoring service is shown in [Fig.1]. The service is provided openly on the Web. The user access the service via the WWW by using any browser with Java Virtual Machine. Requests can be made directly to the *Service Agent* which is the front end of the service.

The functions of main modules can be listed as follows:

- **Resource Manager**: All resources for the service are handled by this module. The results from the HTML Difference Engine will be kept in the archives by Resource Manager. 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 retrieve 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 need 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. The old and new versions of HTML documents are compared by running the Difference Engine. The results from Difference Engine are very important for the agent to classify the changes. At the same time, it will summarize the updated information into another HTML document by an innovative algorithm described in the *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 the 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 considerations for each task is described in detail in following topics.

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 as shown in [Fig. 2]. 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 help us in comparing the context pairs at the right places. In other words, the LOCTAGS provides information about which context in the older revision should be compare to which context in the new revision. When comparing context, we apply the traditional LCS.

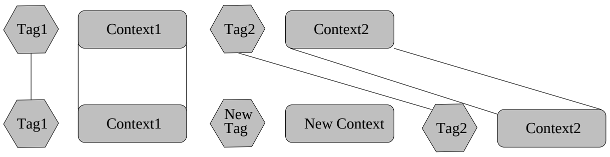


Fig. 2. *Comparison of context at the right place.*

[Fig. 3] shows the diagram of our HTML Difference Engine. The tool takes both the old and new HTML documents as its inputs. Each document is parsed by the tag parser. Both tag streams are fed to the LOCTAGS detector. The result, common tag subsequence, is used as a reference information by the differentiator. Right here, one of the output line is

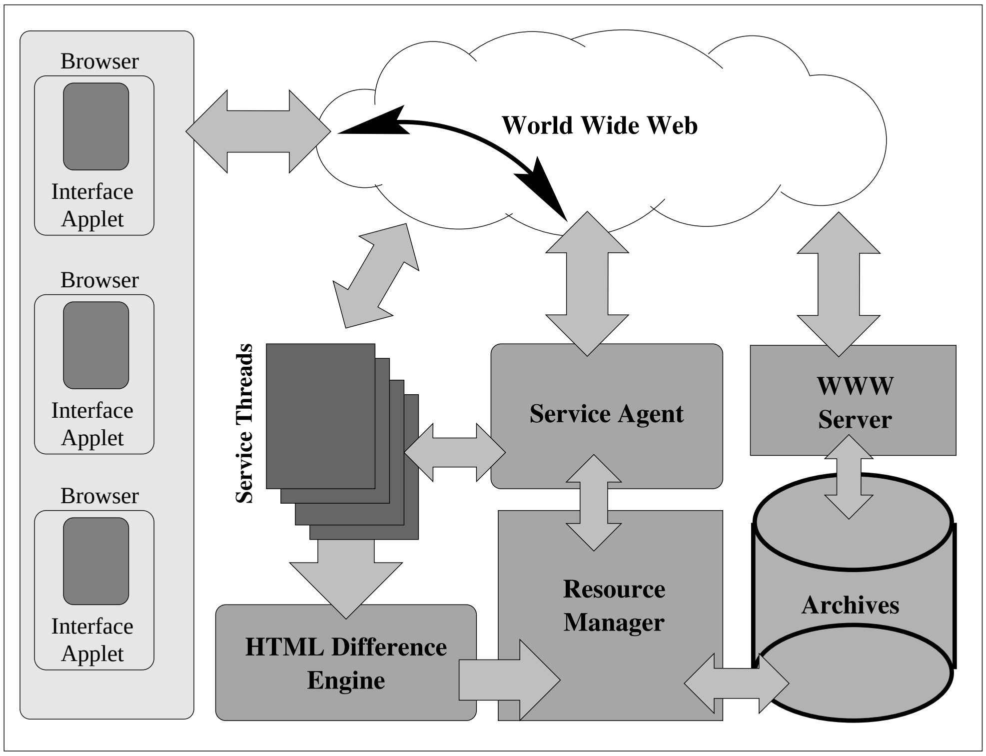


Fig. 1. *The architecture of change monitoring service.*

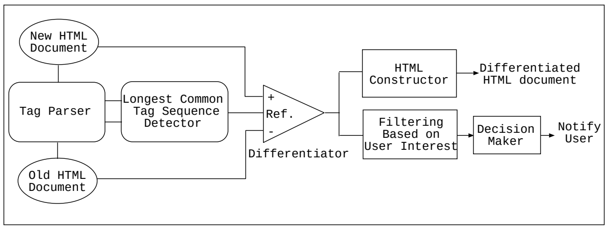


Fig. 3. *HTML Difference Engine with User's interest based Filter*

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.

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 focuses on the filtering process which is based on user interest as described in following subsections.

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 new version of document. The filter provides a report of the change in existence of components by comparing with the old version. The service provider agent notifies the user via email with this information in order to roughly tell the user 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 of the appearance in the documents. Unfortunately, the presentation of appearance changes is not straightforward. The appearance of HTML document must be either the old appearance or the new one. We deal with this by selecting the new appearance for our summary doc-

TABLE I

Change scoring for general HTML document

Category	Score
URL in <A href=...>	256
Java Applet's Bytecode	256
Image in <IMG src=...>	128
Page's Title	128
Background Image	64
Background Color	64
Header <H1>,<H2>	64
Header <H3> and smaller	32
Text (per character)	1

ument. Meanwhile, we count the appearance changes in the filtering process. The information of appearance change is used in the decision making process which tells the Service Agent whether the overall changes are significant enough to notify the users.

- **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.

It is obvious that we have many categories of user interests to deal with in our HTML document's difference stream filter. The details of incorporating these interest criteria into the filtering process will be discuss in next subsections.

HTML Document Stream Filtering In [Fig. 3], the result from the differentiator is fed into the HTML constructor and the user's interest based filter. The filtering process is perform right in this filter. The filter implements the three categories of user interest as described above. The existence of contents can be checked by finding whether the old contents are still in the document. At the same time, the filter scans the document whether any new contents are added to the document. In the same manner, the filter checks whether anything changed with the tags that control the appearance of the document. Once the filter found any changes that fall into these two categories, it evaluates a score of the changes.

$$Score_{content} = \sum_{i=1}^{N_c} w_i(\phi) \tag{1}$$

$$Score_{topic} = \sum_{i=1}^{N_k} \sum_{j=1}^{N_i} 2^{j+5} \tag{2}$$

$$Score_{total} = Score_{content} + Score_{topic} \tag{3}$$

where:
N_c = number of content and appearance changes
N_k = number of key words
N_i = number of occurrences of the i^{th} key word.
ϕ = category of the change
$w_i(\phi)$ = weight of the ϕ category

The topics in user interests can be found by checking the key words that user specified. The filter checks the existence of those key words in the context of difference in the stream then evaluates another score for this kind of user interest by the equation above. The more the occurrences of a key word, the closer to the topic in user interest we can expect. Finally the total score is summed up and then determined by the decision maker. If the score exceeds the specified threshold, the user will be notified of the changes.

4. SHARED 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 the changes in Web repositories and the parameters of the request made by the users.

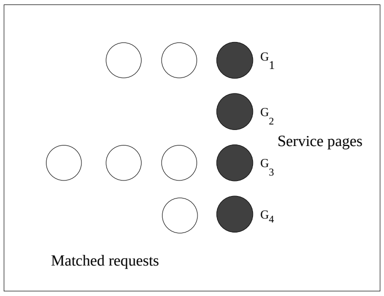


Fig. 4. *Resource sharing of matched requests.*

Unlike the service for individual usage, we attempt to provide personal service while maintaining the efficiency of resource usage. In the case of pushing changes and difference information, we push the information to prospective users. This implies that each user has different degree of interest and attitude against the detected changes. [Fig. 4] shows that among different service pages, there are some identical requests. We can express the utilization of resource as following equations.

$$N = \sum_{i=1}^{M} G_i \tag{4}$$

$$M = P_{diff} N \tag{5}$$

$$\psi = \frac{N - M}{M} = \frac{1 - P_{diff}}{P_{diff}} \tag{6}$$

where:
N = number of all requests
M = number of different kinds of requests
G_i = number of matched requests for the i^{th} group
P_{diff} = Probability of having different kinds of requests
ψ = Utilization factor

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

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 an identical page at 1500 and 1000 points respectively. Both requests are considered identical if the change score is 2000 points. However, if the change score falls between 1000 and 1500 points, the requests are no longer identical. The Resource Manager analyzes the characteristic of changes on the 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 as shown in [Fig. 5]. 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 other users as shown in [Fig. 6]. If some users accept the recommendation, the utility factor of the service trends to be increased according to the increasing matched requests.

TABLE II

Expected pay-off matrix for the games of the service. Pay-off $< x_1, x_2 >$ indicates that the Service Agent has an expected payoff of x_1 (where an improvement is worth 1, and no improvement is worth 0 for the Service Agent) and the user has an expected payoff of x_2. In the case of our service $x_1 + x_2$ must be 1.

			User
		No Adj.	Adj.
	No Adj.	$< p_0, q_0 >$	$< p_1, q_1 >$
Service Agent	Adj.	$< p_2, q_2 >$	$< p_3, q_3 >$

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 choosing the most profitable choice from the pay-off matrix of a game. We will consider how to make decision based on our service games as follows:
We define the utility:

$$\begin{gathered} \mathit{utility}(\mathit{ServiceAgent}, 1) \longleftarrow \mathit{improvement} \\ \mathit{utility}(\mathit{ServiceAgent}, 0) \longleftarrow \neg\mathit{improvement} \\ \mathit{utility}(\mathit{User}, 1) \longleftarrow \mathit{improvement} \\ \mathit{utility}(\mathit{User}, 0) \longleftarrow \neg\mathit{improvement} \end{gathered}$$

Improvement occurs when:

$$\begin{array}{lll} \mathit{improvement} & \longleftarrow & \mathit{ServiceAgent}(D)\wedge \\ & & \mathit{User}(D)\wedge \\ & & \mathit{improve\_if\_follow}(D) \\ \mathit{improvement} & \longleftarrow & \mathit{ServiceAgent}(\mathit{Adj.})\wedge \\ & & \mathit{User}(\mathit{NoAdj.})\wedge \\ & & \mathit{improve\_if\_sa\_un} \\ \mathit{improvement} & \longleftarrow & \mathit{ServiceAgent}(\mathit{NoAdj.})\wedge \\ & & \mathit{User}(\mathit{Adj.})\wedge \\ & & \mathit{improve\_if\_sn\_ua} \end{array}$$

Right here, the $\mathit{improve\_if\_sa\_un}$ is the p_2 and the $\mathit{improve\_if\_sn\_ua}$ is the p_1. Suppose that the Service Agent is to choose a strategy with $p_a = P_{ServiceAgent}(\mathit{Adjust})$ and the user is to choose a strategy with $p_u = P_{User}(\mathit{Adjust})$. In this setup, the probability of having improvement $P(\mathit{improve})$

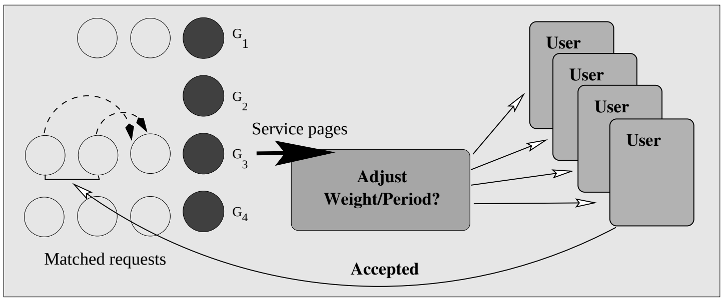


Fig. 5. *Increasing matched requests by asking users to modify appropriate parameters.*

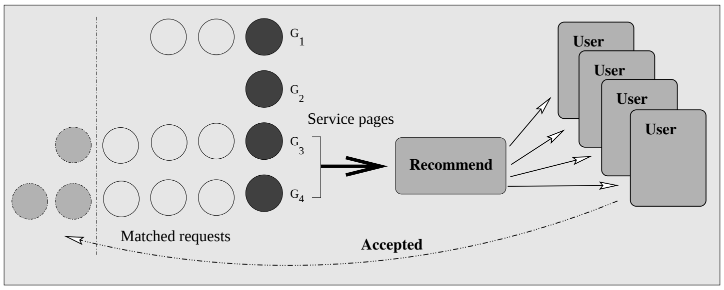


Fig. 6. *Increasing matched requests by recommending hot requests to the users.*

is defined by

$$P(improve) = p_a p_u p_3 + (1-p_a)(1-p_u)p_0 + (1-p_a)p_u p_1 + p_a(1-p_u)p_2 \tag{7}$$

In a randomized equilibrium from the Service Agent view, the payoff for *Adjust* and *NoAdjustment* must be equal. The payoff for asking for adjustment is the above formula with $p_a = 1$, the payoff for asking no adjustment is the formula with $p_a = 0$. These are equal, so we have

$$(1-p_u)p_0 + p_u p_1 = p_u p_3 + (1-p_u)p_2 \tag{8}$$

Similarly for the user

$$(1-p_a)p_0 + p_a p_2 = p_a p_3 + (1-p_a)p_1 \tag{9}$$

Solving for p_u and p_a we derive

$$p_u = \frac{p_2 - p_0}{p_1 + p_2 - p_0 - p_3} \tag{10}$$

and

$$p_a = \frac{p_1 - p_0}{p_1 + p_2 - p_0 - p_3} \tag{11}$$

Substitute in equation (7), we derive

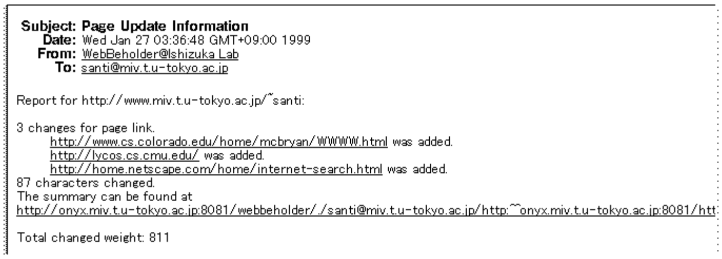
$$P(improve) = \frac{p_1 p_2 - p_0 p_3}{p_1 + p_2 - p_0 - p_3} \tag{12}$$

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 refuse the suggestion. The probability p_1 comes from the improvement made when the user adjust 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 based on this analysis. Moreover, each decision trends to be more exact as the experience of the Service Agent increases.

5. IMPLEMENTATION

The prototype of the system has been implemented locally in our laboratory. The users can access from anywhere on the Web, provided that they can access our interfacing JAVA applet. Currently, we are serving up to 20 users and monitoring over 200 pages. The followings are some of our results. Once the



Subject: **Page Update Information**
Date: Wed Jan 27 03:36:48 GMT+09:00 1999
From: WebBeholder@Ishizuka Lab
To: santi@miv.t.u-tokyo.ac.jp

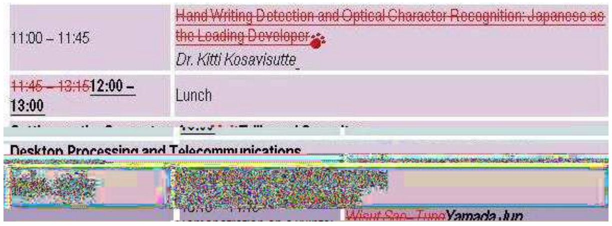
Report for http://www.miv.t.u-tokyo.ac.jp/~santi:

3 changes for page link.
    http://www.cs.colorado.edu/home/mcbryan/WWWW.html was added.
    http://lycos.cs.cmu.edu/ was added.
    http://home.netscape.com/home/internet-search.html was added.
87 characters changed.
The summary can be found at
http://onyx.miv.t.u-tokyo.ac.jp:8081/webbeholder/./santi@miv.t.u-tokyo.ac.jp/http:~~onyx.miv.t.u-tokyo.ac.jp:8081/htt

Total changed weight: 811

Fig. 7. *Changes information pushed from a Service Thread via Email.*

users are notified via Email, they can easily access the summary pages on the Service Agent side by clicking on the link attached to the notification mails. The HTML Difference Engine is able to indicate the changes precisely down to the cell level of table structure in the document. The deleted text is displayed in a stroke font and the new text is displayed in an underline font. The deleted links or images will be marked by small foot icons and the inserted links or images will be marked by peg icons. The users can jump from one change to another quickly since the changes are displayed noticeably.



11:00 – 11:45	~~Hand Writing Detection and Optical Character Recognition: Japanese as the Leading Developer~~ *Dr. Kitti Kosavisutte*
~~11:45 – 13:15~~**12:00 – 13:00**	Lunch

Desktop Processing and Telecommunications

~~Wiwat San-Tuno~~**Yamada Jun**

Fig. 8. *A look of presentation of changes.*

6. 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., [1],[2]. 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 [5] applies social discovery and filtering to inform the users when prospectively significant 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.

7. 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 payoff matrix of service games gives a significant impact on the accuracy of making a decision. These details should be considered in future studies.

REFERENCES

[1] Fred Douglis, Thomas Ball, Yih-Farn Chen and Eleftherios Koutsofios: The AT&T Internet Difference Engine: Tracking and viewing changes on the web *World Wide Web* Volume 1 Issue 1, 1998. pp. 27-44

[2] Fred Douglis: Experiences with the AT&T Internet Difference Engine 22nd International Conference for the Resource Management & Performance Evaluation of Enterprise Computing System (CMG96), December, 1996.

[3] F. Douglis, T. Ball, Y. Chen, E. Koutsofios. Webguide: Querying and Navigating Changes in Web Repositories.

In *Proceedings of the Fifth International World Wide Web Conference*, Paris, France, May 1996. pp. 1335-1344.

[4] Jeffrey M. Bradshaw: Software Agents AAAI Press/The MIT Press, 1997.

[5] Brian Starr, Mark S. Ackerman, Michael Pazzani: Do-I-Care: A Collaborative Web Agent *Proceeding of ACM CHI'96*, April, 1996.

[6] Aglet-Workbench - Programming Mobile Agents in Java, IBM Tokyo Research Lab.,URL=http://www.trl.ibm.co.jp/aglets/

[7] Kazuhiro Minami and Toshihiro Suzuki: JMT (Java-Based Moderator Templates) for Multi-Agent Planning *OOPSLA'97 Workshop*, 1997.