知 能 と 複 雑 系 116−10
（1999. 5. 24）

Push Mode of Change and Difference Information on the Web Based on Agent Interaction

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— **Information sources on the Web are hugely dynamic. The changes made upon them occur at different rates and in unpredictable ways. Besides the target information itself, the changes upon the previously released information are irrefutably significant and worth being notified to those who perceived the out of date information in less than no time. Unfortunately, stock type information source has no means to inform its prospective users of 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 is provided in push mode without elaborate effort of information providers. The service also includes shared resource management to make the system fit for a large-scale service.**

I. 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. The large percentage 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 push 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 created a system that tracks and provides review of changes on the Web called *WebBeholder*. We incorporate shared resource management with the WebBeholder 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 rest of this paper is organized as follows. Section 2 describes the architecture of the system, section 3 shared resource management, and section 4 the difference and display issues. Section 5 explains the induced push mode. Section 6 presents our results. Section 7 discusses related work and section 8 concludes.

II. System Architecture

A WebBeholder community is the community that consists of a service provider agent, a number of mediators, and a number of mobile agents that represent their users. The users customize their own agents to meet their preferences before dispatching them into the community. These agents are called personal agent. The WebBeholder community is designed to provide an environment in

which various kinds of agents can interact with one another to achieve change detection and presentation on the Web.

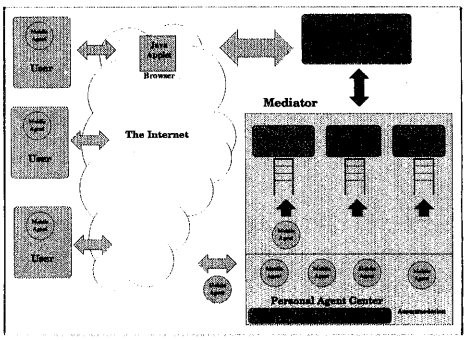

Fig. 1. *The WebBeholder Community with both transaction agent and Java applet accesses to the service.*

The environment of overall system for the WebBeholder community is shown in [Fig.1]. The users of the community dispatch their own agents to the Mediator via the Internet. At the Mediator site, all personal agents are bound in a provided platform which the agents can execute their codes under a restricted control. There are three service modules within the Mediator. All service modules run independently. Each service module serves the personal agent in its own queue. The Request Broker is the module that negotiates with and posts the queries to the Service Provider Agent for the personal agents.

The *Navigator* module tells the personal agents about locations of other WebBeholder communities. This service is provided in the case that personal agents could not find any information on the pages assigned by their users. The personal agent can query the Navigator to look further for some communities that have the desired information.

The *Facilities* module provides facilities for incoming personal agents. Since the mobile agents in the provided platform of the Mediator site have restricted access to the Internet and resource usage, the Facilities module offers these facilities under limited operations. The details of facilities are described in the topic *Facilities in the Community.*

The main agent that offers services to the community is the *Service Provider Agent.* The architecture of the service provider agent is shown in [Fig.2]. Its main modules can be listed as following:

• **Agent**: The heart of the service provider agent. It interacts with other modules in order to retrieve and compare HTML documents.

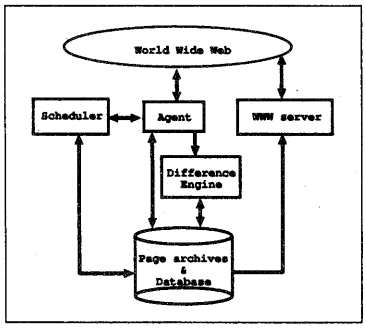

Fig. 2. *The service provider agent.*

• **Scheduler**: The scheduler will look up the pages registered for each user then makes a schedule of checking for the user. It constructs a timetable for the agent to make sure that each user will be served right in time.

• **Difference Engine**: The agent 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 innovative algorithm proposed in this research. The detail on Difference Engine is given in the *Difference and Display* section.

• **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 users are notified by their personal agents, they can view the changes with their browsers via the WWW server.

A. Facilities in the Community

The facilities in the community consist of:

• **Post Office**: The personal agents may have messages for their owners when they find something interesting or just for emergency cases. The message can be sent via the post office of the community.

• **Accommodation**: This provides accommodation for some personal agents that wait for some predictable events or could not go back to its user for a while.

• **Broadcasting service**: This facility allows broadcasting to all agents in the Personal Agent Center. This facility is also used to establish communication among personal agents.

B. Communication in the WebBeholder

When personal agents receive assignments from their users, they go to some Mediator sites. Right in each Mediator site, there will be many personal agents. Each personal agent has capability to deal with components in the Mediator site.The personal agents are independent of one another. The communication layers in the WebBeholder is shown in [Fig.3]. The communication in the com-

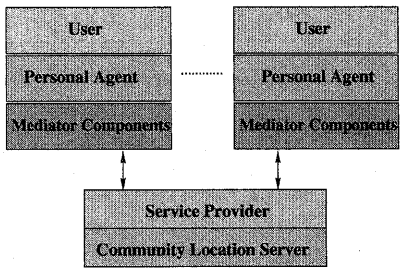

Fig. 3. *Communication layers in the WebBeholder.*

munity can be categorized as shown below:

• **User vs. Personal Agent**: User can define the goal of a personal agent via the dialog provided by the personal agent. The user is required to provide the page in interest together with a location of Mediator site, the frequency of checking, the threshold weight that the service provider agent uses to determine whether the changes are significant enough to notify the user, the depth of checking, and the email address of the user. The service provider regards the email address as the ID of each service.

• **Personal Agent vs. Components**: Each personal agent knows how to deal with various components in each mediator site, such as Request Broker, Navigator, and Facilities.

• **Components vs. Service Provider Agent**: The components each Mediator site deals with the Service Provider Agent in order to carry requests gathered from various personal agent to it.

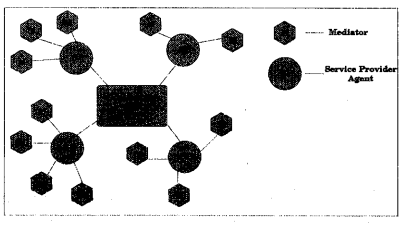

Fig. 4. *A number of WebBeholder Communities are linked together by a central Community Location Server.*

• **Service Provider Agent vs. Community Location Server**: This level provides the door to another community via the Community Location Server. The WebBeholder communities are linked together as shown in [Fig. 4]. The Community Location Server is the center of all communities. Its holds the information about location of service provider agents, the Web pages they are responsible for, and their Mediator sites. The information may be asked from the Navigator modules in Mediator sites in order to dispatch some personal agents to where the desired information is already provided.

Finally, some users may be comfortable to deal with the service provider directly. The system provides a Java applet that makes the request directly to the service provider. By this way of interface, the users have to do most of things manually and have no access to the benefit of the mobile agents.

III. Shared Resource Management

When serving a large number of users, we expect to have some identical or nearly identical requests. These requests can share the resource. Unlike the service for individual usage, we at-

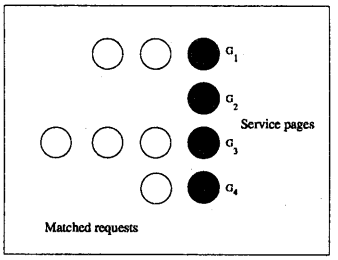

Fig. 5. *Resource sharing of matched requests.*

tempt 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. 5] shown that among different service pages, there are some identical requests. Moreover, the amount of 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. We can express the utilization of resource as following equations.

$$N = \sum_{i=1}^{M} G_i \quad (1)$$

$$M = P_{diff} N \quad (2)$$

$$\psi = \frac{N - M}{M} = \frac{1 - P_{diff}}{P_{diff}} \quad (3)$$

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

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 P_{diff} which ranges from $\frac{1}{N}$ to 1. The range tells us that our utilization factor ranges from 0 to $N - 1$.

IV. 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). 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 sequence of tags and context as shown in [Fig. 6]. We use LOCTAGS

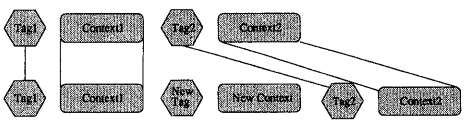

Fig. 6. *Comparison of context at the right place.*

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 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. When comparing context, we apply the traditional LCS.

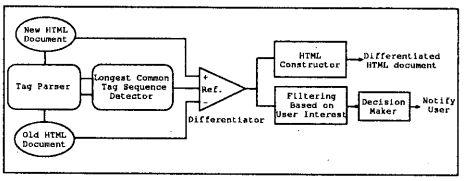

Fig. 7. *HTML Difference Engine with User's interest based Filter*

[Fig. 7] shows the diagram of our HTML Difference Engine. The tool takes both the old and new HTML document as its inputs. Each document is parsed by the tag parser. 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 line 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.

V. 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 send 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 following subsections.

A. User Interests

Users of the framework is served by personal agents based on their interests. The personal agents represent their users in the WWW and work as if the users perform the tasks themselves. In order to deliver such performance, the personal agents need to know their users well enough to give the result that satisfy their needs. A personal agent places the orders received from its

user to the service provider agent together with the preferences and interests of the user. This subsection discusses about the information that we incorporate into our filtering process. This research divides the interests of the users into following categories:

- **Existence of contents:** Many users are curious to know whether the contents they interested in are exist in 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 applet, etc. that appear in 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 notify 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 in the appearance of the documents. Unfortunately, the presentation of change in appearance is not straightforward. The appearance of HTML document must be either the old appearance or the new one. This issue is worth taking into account. We deal with this issue by selecting the new appearance for our summary document. Meanwhile we count the change of appearance in the filtering process. The information of change of appearance participates in the decision making process which results in whether the overall changes are significant enough to notify the user.
- **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 need natural language processing techniques. We compromised the correctness with computational costs. The filter deals with this issue by the hint of words that relevant to the user's area of interests.

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.

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

B. HTML Document Stream Filtering

In Fig.[7], 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 is there any new content inserted to the document. In the same manner, the filter check whether is there 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 those changes.

$$Score_{content} = \sum_{i=1}^{N_c} w_i(\phi) \quad (4)$$

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

$$Score_{total} = Score_{content} + Score_{topic} \quad (6)$$

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, it likely to be the closer to the topic in user interest. Finally the total score is summed up and then determined by the decision maker. If the score is above the specified threshold, the

personal agent will be informed to notify its user of the changes.

VI. Implementation

The prototype of the system has been implemented locally in our laboratory. We run 2 service provider agents and 2 mediator sites. The mobile agent environment is developed on mobile agent package provided by IBM Tokyo Research Lab. Currently, we are serving up to 20 users and monitoring over 200 pages. The followings are some of our results.

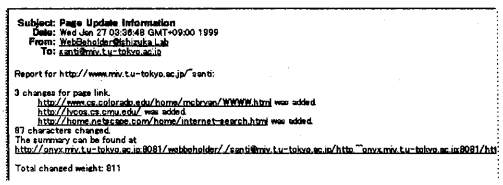

Fig. 8. *Changes information pushed from the service provider via Email.*

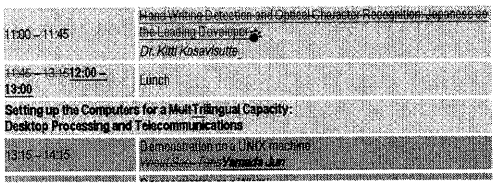

Fig. 9. *A look of presentation of changes.*

VII. 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 of 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. The service is made available in both transaction agent and Java applet which deal with the service provider agent directly. 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 group 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 more simple 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.

VIII. Summary

In this paper, we presented the evolution of the mechanism that induces push mode of changes and differences on the Web. The HTML Difference Engine that implements LOCTAGS algorithm is capable of providing comprehensive presentation of changes. The shared resource management helps us in providing information pushing service to multiple users. The number of identical or nearly identical requests have a significant impact on resource utilization, and overall performance. 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.