WebVigil: An approach to Just-In-Time Information Propagation In Large Network-Centric Environments [1]

Sharma Chakravarthy, Jyoti Jacob, Naveen Pandrangi, Anoop Sanka
Computer Science and Engineering Department
The University of Texas at Arlington, Arlington, TX 76019
sharma@cse.uta.edu

ABSTRACT

Efficient and effective change detection and notification is becoming increasingly important for environments such as WWW and distributed heterogeneous systems. Change detection for structured data has been studied extensively. Change detection and notification for unstructured data in the form of html and XML documents is the goal of this work. The objectives of this work are to investigate the specification, management, and propagation of changes as requested by a user in a timely manner while meeting the quality of service requirements.

In this paper, we elaborate on the problem, issues that need to be addressed, and our preliminary approach. We present an architecture and discuss the functionality that needs to be supported by various modules in the architecture. We plan on using the active capability in the form of Event-Condition-Action (or ECA) rules developed so far, and a combination of push/pull paradigm for this problem.

1 Introduction

Active rules have been proposed as a paradigm to satisfy the needs of many database and other applications that require a timely response to situations. Event–Condition–Action (or ECA) rules are used to capture the active capability in a system. The utility and functionality of active capability (ECA rules) has been well established in the context of databases. In order for the active capability to be useful for a large class of advanced applications, it is necessary to go beyond what has been proposed/developed in the context of databases. Specifically, extensions beyond the current state of the art in active capability are needed along several dimensions:

1. Make the active capability available for non-database applications, in addition to database applications;
2. Make the active capability available in distributed environments
3. Make the active capability available for heterogeneous sources of events (whether they are databases are not).

In this paper, we address 2) and 3) based on our preliminary architecture.

There are a number of situations where one needs to know when changes are made to one or more documents that are stored in a distributed (typically heterogeneous) environment. The numbers of documents that need to be monitored for changes are large and are spread over multiple information repositories. The emphasis here is on selective notification; that is, changes

[1]This work was supported, in part, by the Office of Naval Research & the SPAWAR System Center–San Diego & by the Rome Laboratory (grant F30602-01-2-05430), and by NSF (grant IIS-0123730).

are notified to appropriate persons/groups based upon interest (or profile/policy) that has been established earlier. Also, there should be a mechanism for establishing the interests/profiles/policies. Currently, change detection is done either manually or by using queries to check whether any document of interest has changed (since the last check). This entails wasted resources and at the same time does not meet the intended timeliness (where important) of change detection and associated notification. Also, quality of service issues cannot be accommodated in this approach.

As an example, the above situation is very common in a large software development project where there are a number of documents, such as requirements analysis, design specification, detailed design document, and implementation documents. The life cycle of such projects are in years (and some in decades) and changes to various documents of the project take place throughout the life cycle. Typically, a large number of people are working on the project and managers need to be aware of the changes to any one of the documents to make sure the changes are propagated properly to other relevant documents and appropriate actions are taken. Large software developments happen in distributed environments. Information retrieval in the context of the web is another example that has similar characteristics. Different users may be interested in knowing changes to specific web pages (or even combinations there-of), and want to know when those changes take place. The approach proposed in this paper will avoid periodic polling of the web to see whether the information has changed or not. Some examples are: students want to know when the web contents of the courses they have registered for change; users may want to know when news items are posted with some specific context they are interested in. In general, the ability to specify changes to arbitrary documents and get notified in different ways will be useful for reducing the wasteful navigation of web in this information age. The proposed approach also provides a powerful way to disseminate information efficiently without sending unnecessary or irrelevant information. It also frees the user from having to constantly monitor for changes using the pull paradigm.

Today, information retrieval is mostly done using the pull technology where the user is responsible for posing the appropriate query (or queries) to retrieve needed information. The burden of knowing changes to contents of pages in interested web sites is on the user, rather than on the system. Although there are a number of systems that send information to interested users selectively (periodically by airlines, for example), the approach commonly used is to use a mailing list to send compiled information. Other tools that provide real-time updates in the web context (e.g., stock updates) are custom systems that still use the pull technology underneath to refresh the screen periodically.

We believe that some of the techniques developed for active databases, when extended appropriately along with new research extensions will provide a solution to the above class of problems. In addition, there is the theoretical foundation for event specification, and its detection in centralized and distributed environments. The main objective of this project is to develop the theory, architecture, and prototype implementation of a selective propagation approach that can be applied to web and other large-scale network-centric environments. We will draw upon the techniques developed for Sentinel and re-examine them from a broader, general-purpose context. Some of the issues that will be investigated in this project are:

- Development of an approach (both language and constraints) to specify (primitive) changes to a hierarchical (XML) document at different level of granularity. Develop a GUI, if needed.
- Ability to specify combinations of primitive changes using a language such as Snoop which will allow one to specify higher levels of abstractions of changes (such as combinations of changes, sequences of changes, aggregate changes, etc.)
- Develop techniques for selective propagation between a web server and its browsing clients
- Extend the above to propagate selective changes from one or more web server to another web server (distributed case)
- Develop propagation techniques that take into account QoS and other constraints
- Developing solutions to the above issues will enable us to develop a general-purpose solution to selective information propagation for a large network-centric environment.

The remainder of the paper is organized as follows. In section two we give an overview of related work. In section three we discuss the push/pull paradigms and their relevance to the change detection problem on structured documents. In section four, we present architecture and discuss the functionality of the components. Finally, we discuss future work and draw some conclusions in section 5.

2 Related Work

Many tools have been developed and are currently available for tracking changes to web pages. **AIDE** (AT&T Internet Difference Engine) developed by AT&T [1] shows the difference between two html pages. The granularity of change detection is restricted to a page in AIDE. It is not possible to view changes at a finer level of granularity, such as links within a page, keywords, images, table, lists or phrases.

Changedetection.com [2] allows users to register their request and notifies them when there is a change. We believe that polling (or timestamp information) is used for detecting changes to a page of interest. When a change is detected, the user is notified. The notification does not include what has changed in the page. The user is not given a choice of specifying the type of changes to be tracked on a particular page. Again, the granularity is a page.

Mind-it [3] and **WebCQ** [4] both support customized change detection and notification. Mind-it formerly known as URL-Minder is commercially available. Both these systems track changes to a finer level of granularity in a page. They do not support change specification on multiple pages and combinations of changes within a page (e.g., phrase change and a link change). They also do not use active capability for either detecting changes or propagating changes.

In Xyleme [5, 6], the idea of active paradigm is being used for detecting changes by evaluation of continuous/monitoring queries on XML/HTML documents. The focus is on the subscription language and continuous queries.

3 Push/Pull Paradigms

Traditional approach to information management has been through the use of a Database Management System (or a DBMS). Early DBMSs were developed to satisfy the needs of certain classes of business applications (mainly airline and banking industries). The requirements of these industries were to store, retrieve, and manipulate large amounts of data concurrently, and in a consistent manner (plus allow for failure recovery etc.). Data was stored in databases and the user had to perform

operations explicitly to retrieve data from the system. The burden of retrieving relevant information was on the user. This is the traditional "pull" paradigm where the user retrieves information by performing an explicit action in the form of a query, application, or transaction execution.

Even for traditional business applications, such as inventory control, the pull approach poses certain limitations. For example, in order to keep track of an inventory item (to order additional supplies when the number of widgets falls below a threshold), one has to periodically check (by executing a query) to find out how many widgets are currently present. The traditional DBMS is not capable of automatically informing the user that widgets have fallen below a pre-specified threshold. Not surprisingly, this approach is still heavily used in web navigation, search, and retrieval.

Figure 1 indicates a different approach to information retrieval and management. In this *push* paradigm, the user *does not* have to query or retrieve information as it changes. The system is responsible for accepting user needs (in the form of situations to monitor, business rules, constraints, profiles, continuous search queries,) and informs the user (or a set of users) when something of interest happens. For the widget example above, the user indicates the threshold and the notification mechanism. The system monitors the quantity on hand every time a widget is sold or returned (only when a change takes place; not periodically) and informs the user in a timely manner. This paradigm relieves the user from frequently querying the data sources, and shifts the responsibility of situation monitoring from the user to the system. Of course, in order to accomplish this, the system needs to have additional functionality that is not part of traditional DBMSs. Although this mode of operation is recognized as beneficial and results in significantly less data transfers, accomplishing this for various architectures (such as distributed, federated and network-

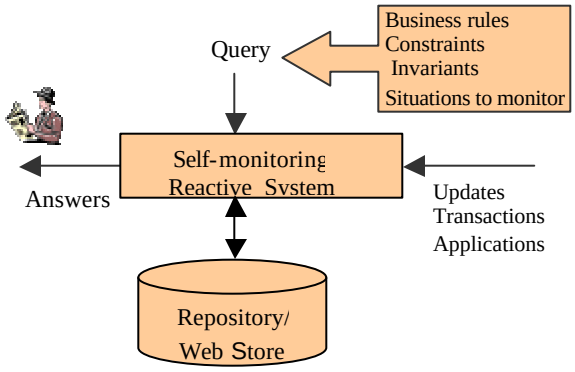

Figure 1 Information Retrieval using the Push Paradigm

centric) requires enhancements to the underlying system or incorporate agents or mediators that can carry this out in a non-intrusive manner. In other words, the system needs to have the capability to *selectively push* information. This is a paradigm shift from how traditional information systems are architected and implemented. It is also a paradigm shift from the users' viewpoint as well.

3.1 Push-Based Architectures

Push technology can be introduced into a system in a number of ways. The approach primarily depends on the characteristics of the underlying system in terms of its openness. The following options can be inferred based on the underlying system characteristics:

3.1.1 Integrated

In this approach the underlying system is actually modified to incorporate the push technology in the form of ECA (event-condition-action) rules. This approach assumes that the source code for the underlying software

is available and the developers have sufficient understanding of the system to make changes at the kernel level. For example, the Sentinel object-oriented active system [7-9] used this approach on the OpenOODB system from Texas Instruments [10]. The sentry mechanism of the underlying system was extended to introduce notifications inside the wrapper for each method to detect primitive events. Once primitive events were detected, more complex composite events were detected and rules executed outside of the underlying system.

The primary advantage of the integrated approach is its flexibility to add minimum amount of code and incorporate many kinds of optimisation that results in good performance. The footprint for primitive event detection is small. Some of the functionality needed for selective push technology (such as deferred action execution) can be easily incorporated using the integrated approach.

So far, a number of research prototypes of active database systems have been developed, such as HiPAC [11], Ariel[12], Sentinel [7, 13], Starburst [14], Exact [15], Postgres [16], PEARD [17], SAMOS [18, 19] etc. Most of them are developed from scratch or integrated directly into the kernel of the DBMS. The integrated approach provides the following advantages [7]:

- Do not require any changes to existing applications.
- DBMS is responsible for optimizing ECA rules.
- DBMS functionality is extended.
- Modularity/maintenance of applications is better and maintenance is easier.

However, the implementation of an integrated approach requires access to the internals of a DBMS into which the active capability is being integrated. This requirement of access to source code makes the cost of integrated approach very high and requires a long integration time as well. Hence, most integrated systems are research prototypes.

3.1.2 Agent-Based/Mediated

The assumption for this approach is that one does not have access to the source code of the underlying system. In fact, this is true in many real-life scenarios where a commercial-of-the-shelf (or COTS) system is being used (relational DBMS is an example). However, the underlying system may provide some hooks using which one can incorporate push capability effectively. We have experimented with this approach in a number of ways and have developed mediators/agents [20] to add full active capability to a relational DBMS. Intelligent agents are introduced between the end user (client) and the system (of course transparently to the user) and the agent provides additional capabilities that are not provided by the underlying system.

3.1.3 Wrapper-Based

For this approach, the assumption is that the underlying system is a legacy system and as a result does not support appropriate hooks and hence it is extremely difficult (and impossible in most cases) to modify the underlying source code. Typically, a wrapper (or a whopper) is built which interfaces to the outside world and push capabilities are added to this wrapper. The wrapper in turn uses the API of the underlying legacy system and may add some additional functionality, not provided by the underlying system (sorting, for example). This approach needs a good understanding of the underlying system and the wrapper has to be developed for each legacy system separately. This approach is not preferred unless this is the only alternative to bring the system on par with other systems to bring the legacy system into a federation or a distributed environment.

4 WebVigil Architecture

WebVigil is a change detection and notification system, which can monitor and detect changes to unstructured documents in general. The current work addresses HTML/XML documents that are part of a web repository. WebVigil aims at investigating the specification, management, and propagation of changes as requested by the user in a timely manner while meeting the quality of service requirements. Figure 2 summarizes the high level architecture of WebVigil. Users specify their interest in the form of a sentinel that is used for change detection and presentation. Information from the sentinel is extracted and stored in a data/knowledge base and is used by the other modules in the system.

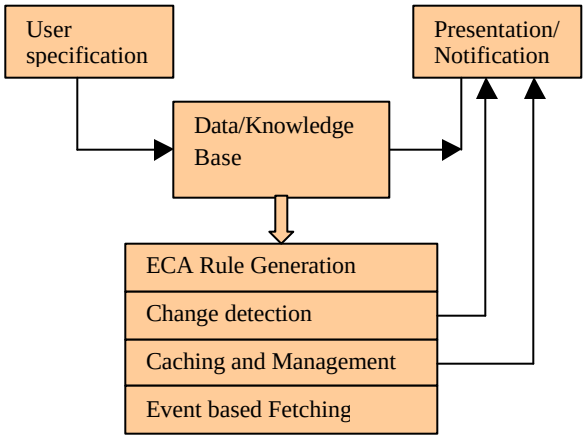

Figure 2. Web Vigil Architecture

The functionality of each module in the architecture is described briefly in the following sections.

4.1 User specification

Users may wish to track changes to a given page with respect to links, words, keywords, phrase, images, table(s), list(s), or any change. We define such a request from the user as a sentinel. The user creates a sentinel to define the changes of interest with respect to a page. A partial syntax of the sentinel is shown in Figure 3. The system generates a unique identifier for every sentinel. The sentinel-target specifies the Url to be monitored for change detection. Sentinel type can be a primitive change (links, images…) or a composite change (combination of primitive changes using options such as AND, OR and NOT). The lifespan of the sentinel can be periodic (from a fixed point of time to another fixed point of time) or aperiodic (from and to activation/termination of other sentinels set by the same user). Once the sentinel is initialised, it becomes active when the condition associated with it becomes true.

```
<Sentinel>::=
Create Sentinel <sentinel-name> Using < sentinel-target>
        Monitor < sentinel-type>
        [From <time point> | <event>]
        [TO <time point>| <to-event>]
        [Notify every <time interval>
        [By <notify options>]

<sentinel-name>::=Identifier
<sentinel-type>::=<change type> [<options> <change type>]
<change type>::=any change| links | images | words |
                    table id | list id | phrase |
                    regular expression | keywords
<sentinel-target>::=<sentinel name>|<url> [<binary op>
                     <sentinel_name>|<url>]
<time interval>::=<integer>{<minute>| <hour>| <day> |
                    <week> }
<time point>::={<month>|<day>|<year>} [+ <time interval>]
<options>::=AND | OR | SEQ
<event>::=start(<sentinel name>) [+ <time interval>] |
            During (<sentinel name>)
<to-event>::=start(<sentinel name>) [+<time interval>] |
              end(<sentinel name>) [+ time interval]
<notify options>::=email <email address> |
                    fax <fax no> | PDA <details> |
                    interactive <details>
<binary op>::=AND|OR|SEQ
```

Figure 3. User Specification

The Notify of a sentinel specifies the frequency with which the user wishes to be informed of changes. The "notify options" gives the users a set of methods for change notification. The sentinel is set with default settings unless stated otherwise by the user. The default settings being:

- FROM: time at which sentinel is initiated.
- NOTIFY: Immediate.

- BY: e-mail.

The Immediate indicates that the user should be notified as soon as the page changes. Of course, there may be a small interval between the change occurrence and detection by the WebVigil. We plan on quantifying this difference more formally and validate through experiments. If an interval is specified, the user is notified using the interval even if the page changes several times during that interval.

Consider the following scenario: *Jill wants to be notified daily by e-mail for change in links and images to the page "http://www.gallery.com" starting from Feb 2,2002 to Mar 2, 2002.* The sentinel for the above scenario is as follows

```
Create Sentinel sen_1
      ON "http://www.gallery.com"
       MONITOR links AND images
       FROM Feb 2, 2002
       TO    Mar 2, 2002
      NOTIFY every day
      BY email jill@aol.com
```

4.2 Data/Knowledge Base (D/KB)

Knowledge Base is a persistent repository containing meta-data about each user, number and names of sentinels set by each user, and details of the contents of the sentinel (frequency of notification, change type etc.). User input is parsed and required information is extracted and stored for later use. For example, for each Url, it stores the following parameters: last modified date, last check time, checksum, and frequency of checks. D/KB may also store notification method and notification frequency for each <user-Url> pair. The D/KB also acts as a persistent store so that all the memory resident information can be regenerated in case of a system crash. The rest of the modules of WebVigil use the D/KB for information needed at run time. AIDE maintains a relational database containing information about each page, each user and relationship between them.

4.3 ECA Rule Generation

We plan on using ECA rules and event detection approach in two places; i) rules for retrieving pages in an intelligent manner based on the user specification (e.g., user frequency coupled with whether the page has changed in that interval) and ii) for propagating pages to detect higher level changes. ECA rules will help us to propagate changes requested by the user in a timely manner. In WebVigil ECA rule generation module uses the concepts defined in [8, 9] to provide the required active capability. This module constructs and maintains "change detection graphs" which keep track of relationships between the pages and sentinels. Each node specifies the change requested in the sentinel on that page. In a change detection graph, the leaf node represents the page of interest and non-leaf nodes represent operators for various types of changes (e.g., phrase change is an operator).

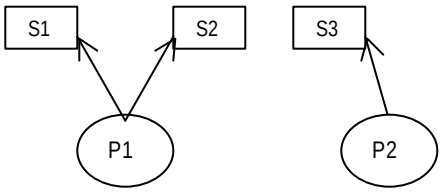

Figure 4. Change Detection Graphs

Figure 4 shows a change detection graph for a page P1 where nodes S1 and S2 represent the type of change detection requested by sentinels present on P1. For every leaf node P_i a periodic or aperiodic rule Ri is generated with the event part of the rule specifying the frequency and the action part with calls to the fetch procedure followed by a notification to the change detection graph, if necessary.

4.4 Change Detection

Detection algorithms have been developed to detect changes between two versions of a page with respect to a change type. For a change to be detected the object of interest is extracted from the given versions of the page depending upon the change type. Figure 5 shows the change types that are identified and supported in the current prototype of WebVigil. Change to links, images, words and keyword(s) is captured in terms of insertion or deletion.

Object identification, extraction and change detection is complicated for phrases. For identifying an object (phrase) in a given page we use the words surrounding it as its signature. We assume that these words are relatively stable. WebCQ [4] uses the concept of a bounding box to tackle this problem. Change to table and list is specified in terms of an update made to their contents. An insertion of a new table or list is not captured under this change type. For phrase change an insert or delete indicates appearance or disappearance of the complete phrase in the page. Currently the change detection algorithms are being reviewed for better performance and for scale up. Abiteboul et al [21] detect changes at the page level and insertions at the node level and is somewhat different from our focus.

Change Type	Insert	Delete	Update
Link	✓	✓	-
Image	✓	✓	-
Keyword(s)	✓	✓	-
Words	✓	✓	-
Phrase	✓	✓	✓
Table	-	-	✓
List	-	-	✓

Figure 5. List of Change Types

4.5 Caching and Management of pages

An important feature of WebVigil architecture is its centralized server based repository service that archives and manages versions of pages. WebVigil retrieves and stores only those pages needed by a sentinel. The primary purpose of the repository service is to reduce the number of network connections to the remote web server, there by reducing network traffic. When a remote page fetch is initiated, the repository service checks for the existence of the remote page in its cache and if present, the latest version of the page in the cache is returned. In cases of cache miss, the repository service requests that the page be fetched from the appropriate remote server. Subsequent requests for the web page can access the page from the cache instead of repeatedly invoking a fetch procedure.

The repository service reduces network traffic and latency for obtaining the web page because WebVigil can obtain the "Target Web Pages" from the cache instead of having to request the page directly from the remote server. The quality of service for the repository service includes managing multiple versions of pages with out excessive storage overhead. WebGUIDE [22] manages versions of pages by storing pages in RCS [23] format.

4.6 Page Retrieval

WebVigil uses a wrapper for the task of retrieving the pages registered with it. The wrapper is responsible for informing WebVigil about changes in the properties of the pages. By properties, we mean the size of the page and last modified time stamp. When there is change in time stamp of the page with an increase or decrease in page size, the wrapper notifies WebVigil of the change, which then fetches and caches the page. In cases where time stamp is modified, but the page size remains the same, the wrapper informs this as a change. WebVigil

fetches and calculates the checksum of the page. The page is cached only if the calculated checksum differs from the checksum of the cached copy of this page.

For dynamically generated pages, WebVigil directly fetches the page without using the wrapper, as page properties are not available. It then checks for change by calculating the checksum of the page. The wrapper may, depending on the paradigm (Push/Pull) be either located at the web server or be a part of WebVigil. Irrespective of its location the primary function of the wrapper is to retrieve metadata and inform WebVigil of the change in page properties. WebVigil in turn fetches and caches pages of interest.

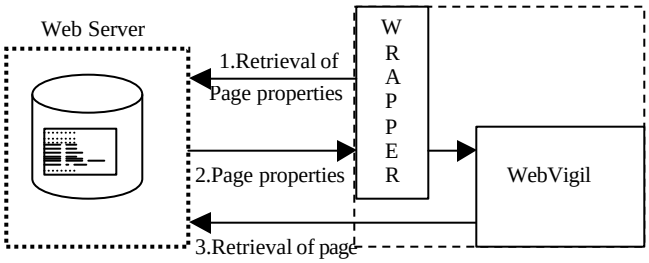

Figure 6. Local Wrapper

In the pull approach the wrapper is located at the WebVigil. It polls and pulls the properties of the pages from the remote web server Figure 6 illustrates this approach.

In the push approach the wrapper is located at the remote web server. The wrapper is assumed to know all those pages that are registered with WebVigil and belong to the web server on which it resides. It informs (pushes) the change information to WebVigil. Figure 7 illustrates this approach. The localization of the wrapper is a trade off between communication, processing and storage. At first glance it may seem obvious that the localization of wrapper should be used, but the cost of polling and network cost may be crucial in which case remote wrapper will be preferable. We intend to develop both local and remote wrappers, evaluate their performance, and use them appropriately.

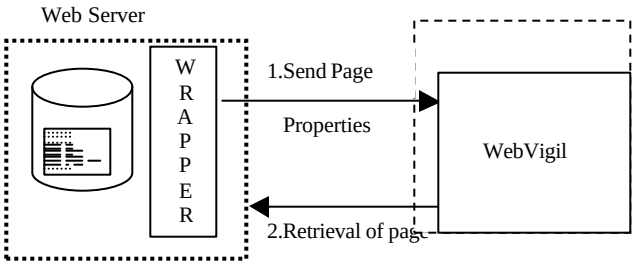

Figure 7. Remote Wrapper

4.7 Presentation and Notification

The presentation method selected should clearly state the detected differences between two web pages to the user. Therefore, computing and displaying the detected differences is very important. In this section, issues related to displaying and notifying the detected changes are discussed.

4.7.1 Presentation

Different methods of displaying changes used by the existing tools are: 1.) Merging two documents, 2.) Displaying only the changes 3.) Highlighting the differences in both the pages [1, 4]. Summarizing the common and changed data into a single merged document has the advantage of displaying the common portions only once [1]. HTMLdiff [1] and Unixdiff [24] use this style to display detected changes. The disadvantage of this approach is that it is difficult for the user to view the changes when they are large in number.

Displaying only the computed differences is a better option when the user is interested in tracking changes to multiple pages or when the number of changes is large. But, highlighting the

differences by displaying both the pages side-by-side is preferable for changes like "any change" and "phrase change". In this case, the detected differences can be perceived better if the change in the new page is shown relative to the old page.

Because WebVigil will track multiple types of changes on a web page, and eventually notify using different media (email, PDA, laptop etc.), combination of all presentation styles discussed above will be relevant, as the information to be notified will vary depending on factors like notification method, number of detected differences and type of changes.

4.7.2 Notification

What, When and How to notify are three important issues for proper notification. These issues are discussed below:

4.7.2.1 Presentation Content

Presentation content should be concise and lucid. Users should be able to clearly perceive the computed differences in the context of his/her predefined specification. The notification report could contain the following basic information:

- The change detected in the latest page relative to the reference page
- User specified type of change like "any change", "all words" etc.
- URL for which the change detection module is invoked.
- Small summary explaining the detected change. This could include statuses of changes such as Insert, Delete and Changed for certain type of user-defined types of changes like "images", "all links" and "keywords" or/and the different timestamps indicating the modification, polling, change detection and notification date.

The size of the notification report will depend upon the maximum information that can be sent to a user by satisfying the network quality of service requirements.

4.7.2.2 Notification frequency

A detected change can be notified in two ways:

- Notify immediately when the change is detected
- Notify after a fixed time interval.

The user may want to be notified immediately of changes on particular pages. In such cases, immediate notification should be sent to the user. Alternatively, frequency of change detection will be very high for web pages that are modified frequently. Since frequent notification of these detected changes will prove to be a bottleneck on the network, it is preferable to send notification periodically. Thus the user can specify the notification interval in the sentinel.

4.7.2.3 Notification methods

Different notify options like email, fax, PDA and web page can be used for notification. Notification can be initiated either by the server or by the client. In WebVigil, server based push initiation is considered. The server, based on the notification frequency can push the information to the user, thus propagating the changes "just in time"(JIT).

5 Conclusions and Future Work

5.1 Conclusions

The basic architecture of WebVigil has been designed to track and propagate changes on unstructured documents as requested by the user in a timely manner, meeting the quality of service requirements. The design accommodates specification of multiple types of changes, on multiple web pages (composite events). The

existing event specification language “SNOOP” [8, 9] will be used for specifying composite events.

The design and implementation of the system will address the issues regarding scalability and user flexibility. Implementation of WebVigil will augment the current strategy of pulling information periodically and checking for interesting changes.

5.2 Future Work

This section describes future extension to the basic functionality of WebVigil. The present method detects changes between the current and the last changed page. This method can be improved upon by giving the user the choice to select the reference page. The user can specify a fixed reference page and must have the flexibility to change the reference. The moving window concept for tracking changes in WebVigil can be improved by allowing a page to be used as reference for detecting changes for the next n pages where user will define n. After changes are detected in n pages, the nth page becomes the reference page. Consider the following scenario:

Jill wants to use the first version of the page as reference. He wants to track changes for the next five revisions to the page with this reference. After five changes, the reference page should be the fifth page and the next five changes should be tracked relative to this page.

An added feature will be to notify the user of cumulative changes. The user can be given the option of being notified of cumulative n changes where n should be specified in the sentinel. Additional feature like user’s personalized change summary page can be provided. The user can lookup this page to get the history of his installed sentinels and the changes tracked till date.

6 References

1. Douglis, F., et al., The AT&T Internet Difference Engine: Tracking and Viewing Changes on the Web. in World Wide Web. 1998, Baltzer Science Publishers. p. 27-44.

2. Changedetection, http://www.changedetection.com.

3. Mind-it, http://www.netmind.com/.

4. Liu, L., C. Pu, and W. Tang. WebCQ: Detecting and Delivering Information Changes on the Web. in the Proceedings of International Conference on Information and Knowledge Management (CIKM). 2000. Washington D.C: ACM Press.

5. Xyleme, http://www.xyleme.com/.

6. Nguyen, B., et al. Monitoring XML Data on the Web. in Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data. 2001.

7. Chakravarthy, S., et al., Design of Sentinel: An Object-Oriented DBMS with Event-Based Rules. Information and Software Technology, 1994. **36**(9): p. 559--568.

8. Chakravarthy, S., et al., Composite Events for Active Databases: Semantics, Contexts and Detection, in Proc. Int'l. Conf. on Very Large Data Bases VLDB. 1994: Santiago, Chile. p. 606--617.

9. Chakravarthy, S. and D. Mishra, Snoop: An Expressive Event Specification Language for Active Databases. Data and Knowledge Engineering, 1994. **14**(10): p. 1--26.

10. Wells, D., J.A. Blakeley, and C.W. Thompson, Architecture of an Open Object-Oriented Database Management System. IEEE Computer, 1992. **25**(10): p. 74--81.

11. Chakravarthy, S., et al., HiPAC: A Research Project in Active, Time-Constrained Database Management (Final Report). 1989, Xerox Advanced Information Technology.

12. Hanson, E., The Ariel Project, in Active Database Systems - Triggers and Rules For Advanced Database Processing. 1996, Morgan Kaufman Publishers Inc. p. 63--86.

13. Anwar, E., L. Maugis, and S. Chakravarthy, A New Perspective on Rule Support for Object-Oriented Databases, in 1993 ACM SIGMOD Conf. on Management of Data. 1993: Washington D.C. p. 99-108.

14. Widom, J., The Starburst Rule System, in Active Database Systems - Triggers and Rules For Advanced Database Processing. 1996, Morgan Kaufman Publishers Inc. p. 87--110.

15. Diaz, O., N. Paton, and P. Gray, Rule Management in Object-Oriented Databases: A Unified Approach, in Proceedings 17th International Conference on Very Large Data Bases. 1991: Barcelona (Catalonia, Spain).

16. Stonebraker, M. and G. Kemnitz, The Postgres Next-Generation Database Management System. Communications of the ACM, 1991. **34**(10): p. 78--92.

17. Alexander, S.D. Urban, and S.W. Dietrich, PEARD: A Prototype Environment for Active Rule Debugging. Intelligent Information Systems : Integrating Artificial Intelligence and Database Technologies, 1996. **7**(Number 2).

18. Gatziu, S. and K.R. Dittrich, Events in an Active Object-Oriented System, in Rules in Database Systems., N. Paton and M. Williams, Editors. 1993, Springer. p. 127--142.

19. Gatziu.S and K.R.Dittrich, SAMOS: an Active, Object-Oriented Database System, in IEEE Quarterly Bulletin on Data Engineering. 1992. p. 23--26.

20. Li, L. and S. Chakravarthy. An Agent-Based Approach to Extending the Native Active Capability of Relational Database Systems. in ICDE. 1999. Australia: IEEE.

21. Cobena, G., S. Abiteboul, and A. Marian, Detecting Changes in XML Documents. Data Engineering, 2002.

22. Douglis, F., et al., WebGUIDE: Querying and Navigating Changes in Web Repositories. in Fifth International World Wide Web Conference. 1996. Paris, France.

23. Tichy, W., RCS: a system for version control, in Software-Practice & Experience. 1985. p. 637-654.

24. J. W. Hunt and M.D.Mcllroy, An algorithm for efficient file comparison. 1995, Bell Laboratories: Murray Hill, N.J.