WebVigiL: An approach to Just-In-Time Information Propagation In Large Network-Centric Environments

Jyoti Jacob, Anoop Sanka, Naveen Pandrangi, and Sharma Chakravarthy

The University of Texas at Arlington, Arlington, TX 76019
{jacob, asanka, pandrang, sharma}@cse.uta.edu

Abstract

In this chapter, we argue that push technology is critical for a large number of applications – traditional as well as network-centric. Push technology can be implemented in various ways depending upon the "openness" of the underlying systems into which the push technology is being inserted. We have experimented with several approaches: integrated, mediated (using intelligent agents),and wrapper-based.

Intelligent agents are useful for supporting push technology in situations where the underlying system/software provides some hooks but is not open to user modification. In this chapter, we first overview the need for push and pull technologies. We then provide an overview of active capability for detecting changes in databases and other structured environments.

In addition to traditional database applications, a number of newer applications can benefit from the concepts that came out of active capability. For example, efficient and effective change detection and notification of web pages is becoming increasingly important for environments such as the Web and distributed heterogeneous systems. The objectives of the WebVigiL project 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 chapter, we elaborate on the 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 use the active capability in the form of Event-Condition-Action (or ECA) rules 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.

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 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 themselves. Currently, change detection is done either manually or by using queries (pulling information by polling) 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 timeliness (where important) of change detection and associated notification.

As an example, the above situation is very common in large software development projects where there exists a large 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 of the documents to make sure the changes are propagated properly to other relevant documents and appropriate actions are taken. More over, 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 chapterwill free the user from having to constantly monitor for changes using the pull paradigm. 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.

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 (air-

lines, for example), the approach typically uses a mailing list to send compiled information. Other tools that provide real-time updates in the web context (e.g., stock updates) are custom-built 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 effort 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 (an object-oriented active DBMS) and re-examine them from a broader, general-purpose context.

The remainder of the chapteris organized as follows. In section 2, we give an overview of related work. In section 3 we discuss the push/pull paradigms and their relevance to the change detection problem. In section 4, 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

Considerable research has been done in active database [1] [12] [2]. In WebVigiL, the active capability is used for the run time management of sentinels, as without an event-based approach, asynchronous monitoring is not possible. Several tools have been developed, and are currently available for some of the problems addressed in WebVigiL. AIDE developed by AT &T [13] displays the difference between two HTML pages. Changedetection.com [26] allows users to register their request and notifies them when there is a change. The change is always detected at the page level and not at the granularity chosen by the user, as in WebVigiL. Mind-it [27] and WebCQ [16] both support customized change detection and notification. Both of these systems track changes to a finer level of granularity in a page. Mind-it does not support change specification on combinations of change types within a page (e.g., phrase change and a link change) as permitted in WebVigiL. In WebCQ, changes are detected only between subsequent versions of the page unlike WebVigiL, which provides flexibility in specifying the reference page. In Xyleme [28] [17] [20], 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. Lifespan and dependent sentinels are not supported. Change detection algorithms are simpler in WebVigiL detects changes on the information seen by the user (i.e. on the contents and not on the structure). Finally, unlike Xyleme, WebVigiL supports both user-specified frequency and on change (for page fetches). WebVigiL offers user-centric specifications in terms of on change (for fetching) and best ef-

fort (for notification) that are important from users' viewpoint. The adaptive push/pull [18] approach evaluates the effect of various approaches (push, pull, and combinations there-of) from the point of view of propagating changes from server to the client. WebVigiL addresses a different problem - that of change detection based on an expressive user specification. Finally, the use of ECA rules and the optimisations that accompany its usage are novel to WebVigiL.

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.

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 etc.) and informs the user (or a set of users) when something of interest happens. This paradigm relieves the user from frequently querying the data sources, and shifts the responsibility of 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-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 designed 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

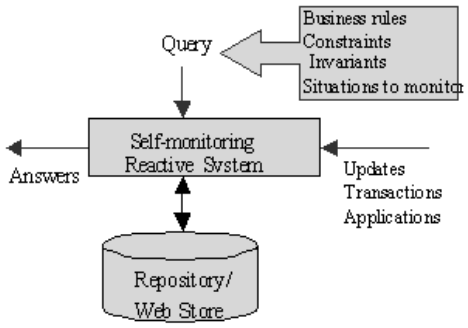

Fig. 1. Information Retrieval using the Push Paradigm

underlying system characteristics:
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] [8] used this approach on the OpenOODB system from Texas Instruments. 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, the composite events were detected and rules were 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 optimization 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 [1], Ariel [11], Sentinel [8] [5], Starburst [12], Exact [2], Postgres [3], PEARD [10], SAMOS [6] [4] etc. Most of them are developed from scratch or integrated directly into the kernel of the DBMS. The integrated approach provides the following advantages [8]: i)Do not require any changes to existing applications, ii) DBMS is responsible for optimizing ECA rules, iii)DBMS functionality is extended, and iv) 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.
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 [15] 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.

Wrapper-Based: For this approach, the assumption is that the underlying system is a legacy system and hence does not support appropriate hooks. Hence it is extremely difficult (and impossible in most cases) to modify the underlying source code. Typically, a wrapper 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 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 it is the only alternative to bring the system on par with other systems and/or 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 and semi-structured 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.

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

4.1 User Specification

There is a need to define an expressive specification language using which the user can specify his/her policy for monitoring and notification of changes to web pages. WebVigiL provides an expressive language with well-defined semantics for specifying the monitoring requirements of a user, pertaining to the Web. Each monitoring request is termed a Sentinel. The syntax of the

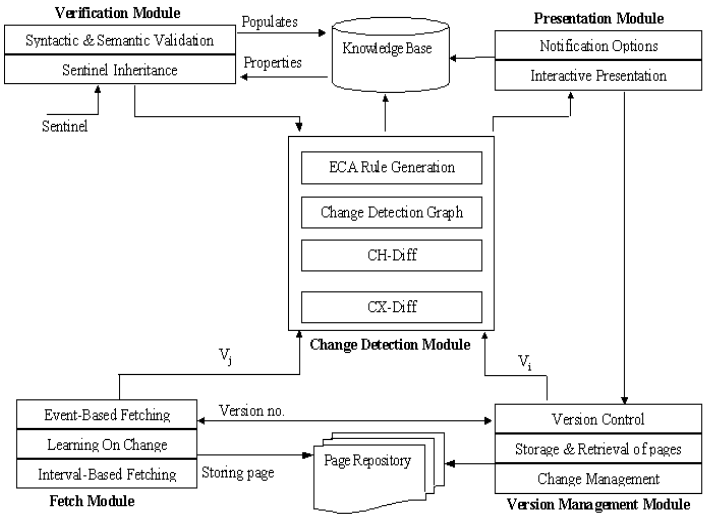

Fig. 2. WebVigiL Architecture

specification language is shown in Figure 3. The semantics of the change specification language are briefly summarized below:

The language allows the specification of a sentinel in terms of previously defined sentinels. This provides a mechanism for tracking correlated changes. Hence, the sentinel target can be a URL of a web page or a previously defined sentinel. Also, start and end of a sentinel may be based on other sentinels. The user can specify his/her change type of interest from a suite of change types at appropriate levels of granularity. Changes only at page level may be overkill in many cases. Hence, WebVigiL supports customized changes like keywords, phrases etc pertaining to portions of a page. In addition, WebVigiL also supports detection of more than one change (composite changes) on a given page using binary operators AND and OR. Change detection using unary operator NOT is also supported.

The specification language also supports activation and deactivation of a sentinel. WebVigiL does not currently support continuous queries but only interval-based monitoring request. Hence, the user has to specify the start and end time of a sentinel. Users can specify the duration as one of the following: (a) Now (b) Absolute time (c) Relative time and (d) Event based time. 'Now' keeps track of the current time, and hence the current time (time of sentinel specification) is used. Absolute time T can be specified as a fixed point on the time line. Relative time is defined as an offset from a time point (either absolute or event-based). We assume that events, such as the start and end of a sentinel can be mapped to specific time points and can be used to trigger the start or end of a new sentinel. Start of a sentinel can also be dependent on the active state of another sentinel and is specified by the event 'during'.

<Sentinel>	:: =	Create Sentinel <sentinel-name> Using < sentinel-target> [Monitor < sentinel-type>] [Fetch <time interval>| on change] [From <time point>	<from event>] [To <time point>| <to event>] [Notify By <contact options>] [Every <time interval>	interactive| best effort	immediate] [Compare <compare options>]
<sentinel-name>	::=	Identifier
<sentinel-type>	::=	[<unary op>]<change type> [<binary op> <change type>]
<change type>	::=	any change| all links| all images | all words [except {<word1>,..<word*n*>}] | table :{<table id> }	list : {<list id>} | phrase : {<phrase1>[,<phrase2>, ..<phrase*n*>]} | regular expression : {<exp>} | keywords : {<word1> [, word2 ,..word*n*]}
<sentinel-target>	::=	sentinel<sentinel name>|<url> [<binary op> sentinel < sentinel name>|<url>]
<time interval>	::=	<integer>{second | minute| hour| day| week }
<time point>	::=	<month>/<day>/<year>[+ <time interval>] | Now [+ <time interval>]
<unary op>	::=	NOT
<binary op>	::=	AND | OR
<from event>	::=	start(<sentinel name>)[+ <time interval>] | during(<sentinel name>) | end(<sentinel name>)[+ time interval]
<to event>	::=	start(<sentinel name>)[+<time interval>] | end(<sentinel name>)[+ time interval]
<contact options>	::=	email <email address>| fax <fax no>| PDA <details>
<compare options>	::=	pairwise | moving<n> | every<n>
<n>	::=	integer

Fig. 3. Change Specification Language Syntax

'during s' defines that a sentinel should be started in the closed interval of s and the start should be mapped to 'Now'.

To detect changes between two given versions of the same page, the versions of the page needs to be fetched. Hence, the fetch frequency of a page has a bearing on the monitoring of the page. The user can monitor a page based on the actual change frequency, or at a user-specified frequency. The specification of the actual change frequency relieves the user of knowing when the page changes and requests the system to do its best effort to fetch the page on modification. In this case, the system uses the heuristics-based Best Effort Algorithm (BEA) as explained in [21] to fetch the pages.

Notification and propagation of changes is an important aspect of change monitoring. The mechanism selected for notification is important especially when multiple types of devices with varying capabilities are involved. The "contact options" allows the users to select the appropriate mechanism for notification from the following set of options: email, fax, or PDA. In addition, when to notify the detected changes is also very important. The user can specify a user-defined time interval or may want to be notified as soon as a change is detected. The user may specify a notification frequency of immediate or best effort for this purpose. immediate indicates notification immediately on change detection. Best effort is defined as notify ASAP (as soon as possible) after change detection. Hence, best effort is equivalent to immediate but will have lesser priority than immediate for notification. WebVigiL uses the active capability to notify the users of the changes detected for immediate

and best effort options. Interactive is a notification approach where the user interactively explores the detected changes as and when needed. A WebVigiL dashboard will be provided to the user to view and query the changes generated by his sentinels. The change detection module in WebVigiL creates a change detection graph (CDG) based on the user properties for detecting changes and notifies the user, when a change is detected. The CDG will be explained in detail in section 4.3.

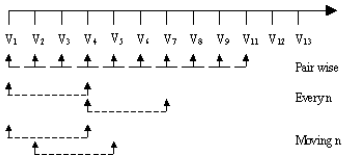

Fig. 4. Compare Methods

To allow flexible monitoring of the pages, the user should have the privilege of choosing the page for reference. Given a sequence of versions $V_1, V_2 ..... V_n$, of the same page, the user may be interested in knowing changes with respect to different references. Hence, the specification language provides multiple ways to compare changes. The various compare options are shown in Figure 4. The default is pairwise, which will allow change detection between two chronologically adjacent versions. Every n allows a user to detect changes between versions V and V+n. For the next comparison, the nth page becomes the reference page. moving n is a moving window concept for tracking changes. For moving n, the first reference page is compared with the nth page. For the next comparison, the subsequent version is taken as the reference page and is compared with nth page, starting from the reference page. For example, if a user specifies the compare option of moving n where n=4. V_1 will be the reference page for V_4. The next comparison will be between V_2 and V_5.

4.2 Knowledge Base

The details of a sentinel need to be stored (in a persistent and recoverable manner) as several modules use this information at run time. For example, the change detection module detects changes based on sentinel information such as the URL to be monitored, the change and compare specifications, and the start and end of a sentinel. The fetch module fetches the pages based on the user specified fetch policy. The notification module requires appropriate contact information and notification mechanism to notify the changes. User information, such as the sentinel installation date, and the page versions for change detection and storage path of detected changes also need to be stored to allow a user to keep track of his/her sentinels.

To satisfy all the above requirements, the metadata (the WebVigiL Knowledge Base) generated and used by different modules is stored in a relational

DBMS. The monitoring request is parsed and sentinel properties are extracted, validated and stored in the Knowledge base (KB).

4.3 ECA Rule Generation

Operations associated with a sentinel are: creation of a sentinel (based on start time), monitoring the requested page, detecting changes of interest, notifying the user(s) of the change, and deactivation of sentinel. In WebVigiL, for every sentinel, the ECA rule generation module generates ECA rules [7] [9] to perform some of these operations. Briefly, an event-condition-action rule has three components: an event (occurrence of an event), a condition (checked when the associated event occurs), and an action (operations to be carried out when the condition evaluates to true). The ECA paradigm has been used for monitoring the database state in active databases and as a stand-alone system for monitoring objects in applications (both centralized and distributed [19]). As part of the Active Object-oriented system [8] [5], a local event detector (LED) [14] has been developed as a library that can be used to declare events and associate rules to be executed when events occur. Primitive events (as method executions) and temporal events (both absolute and relative time), as well as composite events (And, or, seq, and periodic used in WebVigiL) are supported in LED. ECA rules provide an elegant mechanism for supporting asynchronous executions based on events (temporal or otherwise).

Activation/Deactivation: During its lifespan, a sentinel is active and participates in change detection. A sentinel can be disabled (does not detect changes during that period) or enabled (detects changes) by the user during its lifespan. The start/end of a sentinel can be time points or events (as explained in 4.1). When a sentinel's start time is now, it is enabled immediately. But in cases where the start is at a later time point or depends on another event that has not occurred, we need to enable the sentinel only when the start time is reached or when the event of interest occurs. In WebVigiL it is the responsibility of ECA rule generation module to create appropriate events and rules to enable/disable sentinels. We achieve this as follows. Consider the scenario where sentinel s1 is defined in the interval [12/02/02, 01/02/03]. At time 12/02/02 s1 has to be enabled. The events and rules that are generated to enable s1 at compile time are shown in Figure 5. Fetch_s1 is a periodic

```
Event Temp1  = createTemporalEvent(12/02/02)
Rule T1      = createRule (Temp1)
Event Start_s1 = createEvent("start_s1")
Event Fetch_s1= createPeriodicEvent (Start_s1, 2, End_s1)
```

Fig. 5. Events and Rules for s1

```
Event Start_s2 = createEvent("start_s2")
Rule R_start_s2 = createRule(Start_s1)
```

Fig. 6. Events and Rules for s2

event created with "start_s1" as the start event, the frequency of page fetch,

and End_s1 as the end event. The rule associated with it handles the fetching of pages for s1. A rule associated with an event is fired when the event is triggered. More than one rule can be associated with an event. When event Temp1 is triggered at the specified time point, rule T1 is executed, which in turn raises the event Start_s1. Triggering of the event Start_s1 activates the sentinel s1 by activating the periodic event used for fetching the pages of URL specified in s1. Now, if another sentinel s2 which is defined over the interval [start(s1), end(s1)] arrives, the following events and rules are generated in order to enable s2 as shown in Figure 6. Here we are associating the rule R_start_s2 to the event Start_s1, which was created at the arrival of sentinel s1. This rule actually raises the Start_s2 event to activate the periodic event associated with s2. In this manner, ECA rules are used to asynchronously activate and deactivate sentinels at run time. Once the appropriate events and rules are created, the local event detector handles the execution at run time. By enabling/disabling of sentinel we mean addition/deletion of that sentinel to the change detection graph that is detailed in the next section.

Change Detection Graph: When a page is fetched, for every sentinel that is interested in that page, change is computed and notified to the user. In situations where there are two or more sentinels interested in the same type of change on the same page we have to compute the changes more than once. We avoid this by capturing the relationship between the pages and sentinels, and grouping the sentinels on their change and page. Hence all sentinels interested in the same type of change and on the same page are grouped together. In order to represent this relationship we construct a change detection graph. Consider the scenario where sentinel S3 is specified on the web page 'www.cnn.com' for monitoring the change type 'all images'. Another sentinel S1 is also specified for the same web page but detects composite changes on 'all images' and 'all links', then the change detection graph for the sentinels S1 and S3 is as shown in Figure 7. The different types of nodes in the graph are as follows:

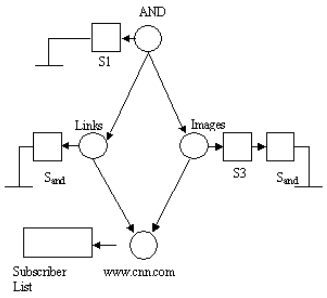

Fig. 7. Change Detection Graph

- URL node: A URL node is a leaf node that denotes the page of interest.
- Change type node: All level-1 nodes in the graph belong to this category. This node represents the type of change on a page (all words, links, images, keywords, phrases, table, list, regular expression, any change).
- Composite Node: A Composite node represents a combination of change types. All nodes that are above level-1 in the graph belong to this type.

In the graph, to facilitate the detection and propagation of changes, the relationship between nodes at different levels is captured using the subscription/notification mechanism. The higher-level nodes subscribe to the lower level nodes in the graph. This subscription information is maintained in the subscriber list at each node. At the URL node, this list contains the references to the change type nodes. At the change type nodes each sentinel will have a subscriber that will contain the references to the composite nodes. When a page is fetched, the associated URL node is notified about the page. The URL node propagates this page to all the change type nodes that have subscribed to it. Finally at the change type nodes the change is computed between the current page received and an appropriate reference page (based on the compare option) that is fetched from the page repository. If there is any change then the sentinels subscribed to it are notified. When this change type is part of a composite change, those composite nodes are also notified. Consider the scenario where a page p1 changes twice a day and sentinel S_i is interested in both the changes taking place each day whereas sentinel S_j is interested in changes once in two days. In this case both the sentinels are interested on the same type of change on the same page but with different requirements (versions of page). For this, additional information is maintained at the change type node in order to differentiate the change detection and change propagation among the sentinels. Consider the change detection graph shown in Figure 7. The arrows in the graph represent the parent-child relationship among nodes. The subscription information is maintained in the list associated with each node. Sentinel S1 specifies a composite change consisting of links AND images. Sentinel S1 should be notified only when there is a change to links and images. Since change is computed only at level-1 nodes, S1 should have a representation of itself at the constituent change type node. In order to facilitate this, a proxy sentinel S_{and} with the same properties as that of S1 is created on the links and images node. The composite node AND notifies the creator of S1 only when it receives notifications from both of its constituent sentinels. This is illustrated in Figure 7. At run time whenever a request for monitoring a page arrives, the corresponding nodes are created (if they do not already exist) and the sentinel is added to the change detection graph.

Change Detection The change detection tool should be capable of detecting preferred change, such as the appearance/disappearance of objects of user's interest on a page. Consider the scenario: A student wants to monitor the college schedule of classes for a particular course name (keyword). In such cases, detecting changes to the complete page results in excessive com-

putation and dissemination of irrelevant information. Hence, there is a need to support detecting changes based on user's intent. We view a page as a sequence of words and certain content-defining tags while ignoring other presentation tags. Users may be interested in the appearance or disappearance of certain words or a section of contiguous words or links/images in the page. Thus the contents of a page can be classified, based on the user intent, into

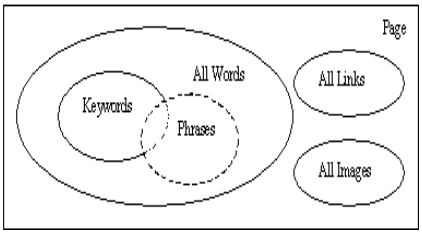

Fig. 8. Classification of page

Change Type	Synopsis
Links	Insertion of new links or deletion of old links
Images	Insertion of new images or deletion of old images
Keyword(s)	Insertion or deletion of selected words
Phrase(s)	Insertion/deletion/update to selected text phrase
All Words	Any change to words in the page
Any Change	Any change to words or links or images in page

Fig. 9. Synopsis

keywords, phrases, all-words, links and images.

- Keywords: A set of unique words from the page
- Phrase: Contiguous words from the page (no upper bound)
- All-Words: All the words in the page constitute this set
- Links: A set of hypertext references extracted from page
- Images: A set of image references extracted from the image source

The classification of the page based on user intent is shown in Figure 8. Appearance or disappearance of these content-based objects of interest can flag a change. The synopsis of the various changes detected by WebVigiL is shown in Figure 9. As the above-defined changes are content based, we do not detect changes to the structure of the document. The structure of the document is only considered for efficient change detection to the contents. WebVigiL supports change detection to HyperText Markup Language (HTML) and eXentesible Markup language, which are the standard formats for electronic publishing on the web. Over the coming years XML is likely to replace HTML as the standard web publishing language but until then both will coexist. In an XML page, combination of the content and the tags define the nature of the content whereas in HTML they define the presentation aspects of the content. Hence, as the format and representation of both HTML and XML differ, separate approaches need to be adopted for change detection for these documents. These change detection techniques are discussed in detail in [22]. In this paper, we discuss briefly the approaches used for HTML and XML.

4.4 Event-Based Fetching

WebVigiL monitors only those pages that are registered with it. For this purpose it has to fetch the pages when a change in metadata (such as the last modified time stamp or checksum) is detected. For monitoring page, after every time interval t, we check for any change in page properties, such as the last modified time stamp of the page and then actually fetch the page if there is a change. Hence we need a mechanism for triggering the monitoring requests at a given time point after the lapse of interval t. We solve this problem by using the notion of a periodic event [9]. A periodic event is an event that repeats itself with a constant and finite amount of time. The specification for a periodic event is PeriodicEvent (E1, [t], E3) where E1 and E3 are the events (or time specifications) that act as an initiator and a terminator, and t is the time interval. In WebVigiL, we use periodic events for triggering the monitoring request. Here E1 and E3 are the start and end events of a sentinel and t is the interval at which the page should be monitored. To actually fetch the page, we associate a fetch rule with this periodic event. Hence whenever a periodic event occurs, the rule associated with it is triggered. This fetch rule performs the functionality of the monitoring request, i.e., it fetches the page based on changes to the page properties. Consider the scenario where sentinel s1 specifies the fetch time as 2 days. The periodic event generated for s1 is as follows: Event Fetch_s1 = createPeriodicEvent (Start_s1, 2, End_s1), where Start_s1 and End_s1 are the start and end events of sentinel s1. The event Start_s1 initiates the periodic event. For every interval t (2 days) the periodic event is raised until event End_s1 occurs. When the periodic event is raised, the fetch rule associated with it fetches the page. For sentinels that explicitly specify the polling interval, we generate a periodic event and associate a fetch rule with it. Hence for every unique combination of interval, start event and end event we generate a unique periodic event and associate a fetch rule with it. The interesting and difficult case is when the user expects the system to notify him/her as and when the page changes. In such cases, the system is required to tune its polling interval with the change frequency of the page. For this, we generate one periodic event with a fetch rule. This rule achieves the required tuning by changing the interval of periodic event base on learning.

Whenever a periodic event occurs, the corresponding rule is fired, which then checks (condition part of the rule) for change in metadata of the page and fetches the page (action part of the rule) if there is a change in metadata. Thus the periodic event controls both the polling interval and the lifespan of the fetch process. The meta-data of the page is taken into consideration for fetching. By metadata of the page, we mean page properties such as the page size, last modified time stamp and checksum of the page. A fetch cycle for a page is triggered only when there is a change between the metadata of the current version of the page to that of the previous version. Depending upon the nature (static/dynamic) of page being monitored the complete set or subset of the meta-data is used to evaluate the change.

For static pages, HTTP HEAD request is used to obtain the meta-data of the page. Change in time stamp of the page with an increase or decrease in page size, is flagged as change, and the page is fetched. In cases where time stamp is modified, but the page size is unchanged, HTTP GET request is used to retrieve the page and the checksum of the page is calculated. The page is added to the page repository only if the calculated checksum differs from the checksum of the previously cached copy of that page. For pages that are not provided with last modified timestamp such as dynamically generated pages or cases where previous attempts to retrieve page properties have failed, HTTP GET request is used to retrieve the page. Change is then flagged by calculating the checksum.

Types of Fetch Rules A fetch rule is created and used for polling the page of interest specified in the sentinel. As per the change specification, a user can specify a sentinel for fetching with two options (a) On Change (b) or Interval Based. Based on the option specified in the sentinel, Event-Based Fetch module generates a Best-Effort (BE) Rule or an Interval-Based (IB) Rule. These Rules differ in the way they handle the "t "(fetch interval) of the periodic event.

- Best Effort Rule: In situations where the user has no information about the change frequency of a page, it is necessary to tune the fetch frequency to the actual change frequency of a page. BE Rule uses a best-effort Algorithm to achieve this tuning. In the best-effort algorithm (BEA), the next fetch interval (P_{next}) is determined from the history of changes to that page. When the next polling interval is determined, the BE Rule changes the interval "t" of the periodic event. Clearly the effectiveness of the algorithm depends on the accurate estimation of the fetch interval. Event-Based Fetch Module generates a BE Rule BE_i for every unique page u_i, and maps other sentinels with fetch option "on change" on u_i,to the generated rule BE_i .
- Interval-Based Rule: The user can explicitly specify a fetch frequency. For example, Don may know that a page is changing every 4 hrs starting at 9.00 am and hence can specify a sentinel to start monitoring the page with a fetch interval of 4 hrs. For this, a periodic event whose periodicity (interval t) equal to the given interval is created and an IB rule IBi is associated with it to fetch the page. As a result there will be more than one IB rule on a given page with different or same periodicity, where each rule is associated with a unique periodic event (i.e., with different start and end times).

A BE rule and many IB rules can be set for the same page. Thus, there can be situations where both a BE rule and an IB rule fetch the same version of the page resulting in multiple copies of the same version at page repository. To avoid this situation, we synchronize the fetching with respect to the last fetch version of the page. A rule initiates the fetch process only when, there

is no version V_i of the page u, with Last-Modified-Timestamp (LMT) equal to LMT of the page it is required to fetch.

4.5 Caching and Management of pages

An important feature of WebVigiL architecture is its 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. Since all the versions of a particular page have to be stored in the cache, each URL has to be mapped to a unique directory. The complete URL cannot be used as a directory name since the length of the URL is large in many cases. Below we discuss two approaches to establish the required mapping for the directory structure. We have analyzed the approaches experimentally to determine which one scales up for tens of thousands of pages. The approaches are discussed below.

Hash-based approach: Each unique URL is inserted into a hash table with URL as the key. This unique value represents the directory where the corresponding versions are stored. Consider two URL's x/y/z/i.htm and x/y/z/j.htm, which have common path information x/y/z. Instead of hashing the whole URL twice we generate a mapping for the common path once and reuse the same mapping whenever required. We achieve this by maintaining two different mapping, one for the path and the other for file name.For example, the URL's ranger.uta.edu/`~cook/aa/lectures/l10/L10.html` and ranger.uta.edu/`~cook/aa/lectures/l10/node14.html` will be mapped to D1F1 and D1F2 where D1 is the value assigned for ranger.uta.edu/`~cook/aa/` lectures/l10/, F1 and F2 are values assigned when mapping file names L10.htm and node14.htm.

Directory-based approach: In this approach the path of the URL is replicated for the directory structure. For example for the URL ranger.uta.edu/ `~cook/aa/lectures/l10/L10.html`the directory structure is maintained the same upto the file name. In some cases (e.g., dynamically generated pages) the length of the file name can be very large. As the operating system imposes a restriction on the length of the directory name, we cannot create directories for files having large names. Hence we use hash-based approach to generate a unique mapping for only the file name i.e., for the previous example the file L10.html is mapped to F1 and a directory is created.

Experimental evaluation: We evaluate the two approaches based on the time taken to construct the mapping and time taken to reconstruct the

mapping (in case of recovery) plus time to retrieve the page given its URL. For the hash-based approach we have used the java hash provided in Java 1.4. The selection of data sets for the experiments were based on the length of the path (excluding the file name) of the URL denoted as depth and the number of URL's. The data sets are represented as L# where # denotes the number of URL's with depth varying between 1 to 3 and M# with depth varying between 4 and 6. Figure 10 shows the time taken to build the mapping for each of

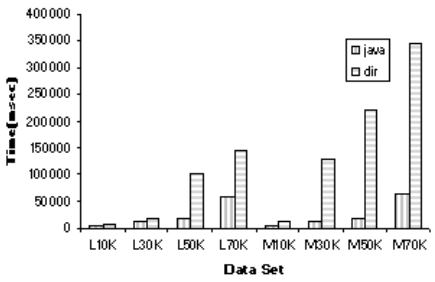

Fig. 10. Build Time

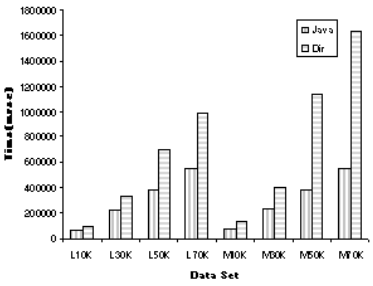

Fig. 11. Reconstruction + Seek Time

the data sets. The directory-based approach takes more time than hash-based approach as more I/O is involved. Figure 11 shows the time taken to rebuild the mapping plus time taken to retrieve the page given the URL. Since the mapping is persisted as and when unique URL's are hashed (during the build time), during reconstruction the unique mappings for persisted URL's are not regenerated, thus saving time. It can be observed that even though there is no reconstruction required in directory-based approach except for the file mapping, the time taken is much large. This increase in time is because the time taken to retrieve the pages increases with increase in depth, as more numbers of directories have to be traversed (apparent from Figure 11).

4.6 Change Presentation

Change presentation is the last phase of web monitoring where the changes detected, as outlined in the previous sections, are presented to the user. For meaningful interpretation of the presented changes, we have investigated three ways to present it to the user:

- *Only Change Approach:* Showing only the changes and omitting the common objects of the two pages is advantageous for pages of large size but will make interpretation intricate. This approach can be meaningful for

hand-held devises to conserve the amount of data transmitted over a limited bandwidth.

- *Single Frame Approach:* Produce a single document by merging the two documents summarizing all inserted, deleted and common objects. The advantage lies in displaying the common objects just once, but with the draw back of possibly changing the page structure.
- *Dual Frame Approach:* Showing both the documents side-by-side in different frames and highlighting the changes between the documents has the advantage of easy interpretation of the changes presented. When the number of changes to be presented is large, this approach may make it difficult to interpret the changes. This can be remedied by presenting parts of the pages at a time to limit the number of changes displayed in each instalment.

In WebVigiL we intend to use all of the three presentation schemes summarized above in a selected combination depending upon the type of change being presented. For example we plan on testing the Dual Frame Approach for presenting changes to phrases and keywords. For displaying changes to images, we plan on using the Single Frame approach (showing both the old and new image). Finally for the change type any-change, based on the number of changes detected, we use a heuristic cost model for choosing the presentation mechanism between the Dual Frame Approach, Only Change and Single Frame Approach for displaying changes. And for displaying inserted text, we are currently using colors and is highlighted.

5 Conclusions and Future Work

WebVigiL is a change monitoring system for the web that supports specification, management of Sentinels and provides presentation of detected changes in multiple ways (batch, interactive, for multiple devices). The first prototype has been completed and includes the following features: web-based sentinel specification [21], ECA rule based fetch that includes learning [21] to reduces the number of times a page is fetched, population of the knowledge base, detection of changes to HTML and XML pages [22]. A simple presentation module for the schemes briefly outlined in this paper has been implemented.

Currently, the individual modules are being integrated to instrument the first version of a complete WebVigiL system. The first release of WebVigiL with the above features is expected to be ready by March. The performance evaluation of change detection algorithms and their comparison with other approaches are currently underway.

5.1 Open Problems

WebVigiL has brought out a number of problems that are not yet completely addressed. The implementation of WebVigiL has raised a number of system-

related issues, such as where to use active capability effectively, persistence of large number of pages, scalability issues, and dealing with a number of devices for presentation and notification. A number of synchronization issues between the modules came into picture during the prototype development. On the conceptual side, a number of problems, such as change detection of dynamic pages, position-independent change detection, adaptive algorithms to minimize fetch, change detection on multiple pages, to name a few. Currently, a sentinel has a single life-span. Future work will include supporting multiple life spans for a sentinel.

References

1. Chakravarthy, S., et al.(1989) HiPAC: A research project in active, time-constrained database management. Tech. Report (89-02), Xerox Advanced Information Technology: Cambridge.
2. Diaz O,Paton N,Gray P (1991) Rule Management in Object Oriented Databases: A Unified Approach, in the Proceedings of 17th International Conference on Very Large DataBases, 1991. Barcelona, Spain.
3. Stonebraker M,Kemnitz G (1991) The Postgres Next-Generation Database Management System. Communications of the ACM, 1991. 34(10): p. 78–92.
4. Gatziu S,Dittrich K R (1992) SAMOS: an Active, Object-Oriented Database System. in IEEE Quarterly Bulletin on Data Engineering, 15(1-4): p. 23–26.
5. Anwar E,Maugis L,Chakravarthy S (1993) A New Perspective on Rule Support for Object-Oriented Databases, ACM SIGMOD Conf. on Management of Data. 1993: Washington D.C. p.99-108.
6. Gatziu S,Dittrich K R (1993) Events in an Active Object-Oriented System, in Rules in Database Systems., N. Paton and M. Williams, Editors. Springer. p. 127–142.
7. Chakravarthy S,Mishra D (1994) Snoop: An Expressive Event Specification Language for Active Databases. Data and Knowledge Engineering, 1994. 14(10): p. 1–26.
8. Chakravarthy S,Anwar E,Maugis L,Mishra D (1994) Design of Sentinel: An Object-Oriented DBMS with Event-Based Rules. Information and Software Technology. 36(9): p. 559–568.
9. Chakravarthy S,Krishnaprasad V,Anwar E,Kim S K (1994) Composite Events for Active Databases: Semantics, Contexts and Detection, in Proc. Int'l. Conf. on Very Large Data Bases VLDB. Santiago,Chile. p. 606–617.
10. Alexander,Urban S D,Dietrich S W (1996) PEARD: A Prototype Environment for Active Rule Debugging. Intelligent Information Systems : Integrating Artificial Intelligence and Database Technologies, 7(2).
11. Hanson E (1996) The Ariel Project, in Active Database Systems - Triggers and Rules For Advanced Database Processing. Morgan Kaufman Publishers Inc. p.63–86.
12. Widom J (1996) The Starburst Rule System, in Active Database Systems - Triggers and Rules For Advanced Database Processing. 1996, Morgan Kaufman Publishers Inc. p.87–110.

13. Fred Douglis,Thomas Ball,Yin-Farn Chen,Eleftherio (1998) The ATT Internet Difference Engine: Tracking and Viewing Changes on the Web,World Wide Web. Baltzer Science Publishers. p. 27-44.
14. Dasari R (1999) Design and Implementation of a Local Event Detector in Java, MS Thesis in CISE Department, Univ. of Florida:Gainesville.
15. Li L,Chakravarthy S (1999) An Agent-Based Approach to Extending the Native Active Capability of Relational Database Systems. ICDE Australia:IEEE.
16. Ling Liu,Calton Pu,Wei Tang (2000) WebCQ: Detecting and Delivering Information Changes on the Web. in Proceedings of International Conference on Information and Knowledge Management (CIKM). Washington D.C: ACM Press.
17. Benjamin Nguyen,Serge Abiteboul,Grgory Cobena,Miha Preda (2001) Monitoring XML Data on the Web. in Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data.
18. Deolasee,P., et al. (2001) Adaptive Push-Pull: Disseminating Dynamic Web Data. in Proceeding of the 10th International WWW Conference. Hong Kong.
19. Weera Tanpisut (2001) Design and Implementation of Event based subscription/notification paradigm for distributed environments. 2001, The University of Texas at Arlington.
20. Cobena G,Abiteboul S,Marian A (2002) Detecting Changes in XML Documents. Data Engineering.
21. Chakravarthy S,Pandrangi N,Jacob J,Sanka A (2003) WebVigiL: Architecture and Functionality of a Web Monitoring System, http://itlab.uta.edu/sharma/Projects/WebVigil/files/WWW2003.pdf.
22. Pandrangi, N., et al. (2003) WebVigiL: User-Profile Based Change Detection for HTML/XML Documents. in Proceedings of Twentieth British National Conference on Databases Coventry, UK.
23. HTML-Parser, http://www.quiotix.com/downloads/html-parser/.
24. Document Object Model., http://www.w3.org/DOM/.
25. Xerces-J, http://xml.apache.org/xerces2-j/index.html.
26. Changedetection http://www.changedetection.com.
27. Mind-it http://www.netmind.com/.
28. Xyleme http://www.xyleme.com/.